Motion Planning With Dynamic Obstacles Using Convexified Control Barrier Functions

Varun V. P.[1], Abraham P. Vinod[2], and Shishir Kolathaya[3]

***Abstract*— Model Predictive Control (MPC) is a popular approach used for motion planning in dynamical systems. Given a finite horizon cost, we seek an optimal control law subject to safety constraints. However, in the presence of obstacles, existing MPC formulations are often slow and may lead to infeasibility. We propose a real-time implementable MPC formulation using control barrier functions (CBF) and successive convexification. We represent the non-convex obstacle avoidance constraints using CBFs that ensure that a feasible solution always exists. We then reformulate the non-convex optimal control problem using successive convexification to enable the use of computationally-efficient conic solvers. Our approach enables controller synthesis at real-time, which is difficult with existing approaches that rely on nonlinear solvers. We demonstrate the method in simulation, where we navigate a UAV to a target while avoiding dynamic obstacles in the environment.**

I. Introduction

Model-predictive control (MPC), originating from the process industries [9], has now seen success in a variety of applications, including autonomous ground vehicles [12], robot manipulators [16] and unmanned aerial vehicles (UAVs) [10]. MPC enables trajectory planning for a desirable time horizon, while satisfying safety-critical constraints. Practical implementations of MPCs largely involve minimization of quadratic cost formulations subject to linear constraints. As a result, efficient optimization schemes leveraging advances in quadratic programming have permitted the use of low compute platforms like Raspberry Pi for real-time implementation [8]. However, these successes have not been extended to safety-critical operations like obstacle avoidance, due to the underlying non-convexity and nonlinear constraints.

MPC problem formulations with nonlinear constraints have been studied extensively [2], [17]. These works reduce the nonlinear constraints to simpler forms typically via linearizations and yield approximate solutions. However, these solutions may not be feasible for the original problem. In the context of obstacle avoidance, successive convexification has provided a fast alternative [15]. Here, the authors provide an iterative algorithm that at each step projects the current trajectory solution on to the obstacles, convexifies the nonconvex obstacle avoidance constraints via linearization about the projection points, and then solves the resulting convex program. Convex programs can be solved efficiently using off-the-shelf solvers like ECOS [7]. In the context of unmanned aerial vehicles (UAVs, see Fig. 1), the obstacle avoidance constraint can be reformulated to yield a conic constraint in every iteration, which is then evaluated successively to yield a fast locally optimal solution. This makes it attractive for use on onboard compute platforms, often constrained by power and memory requirements [14].

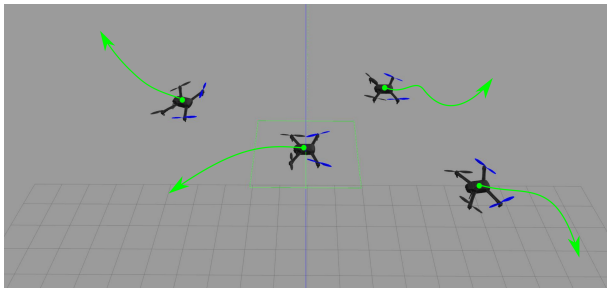


Fig. 1: Figure showing the typical UAV setup going to various destination points. The goal is to plan collision-free trajectories in real-time using model-predictive control (MPC). We propose to use convexified control barrier functions (CBFs) as MPC constraints to improve feasibility and robustness. The software-in-the-loop simulation using Gazebo can be seen here `https://youtu.be/NX4CBHGFbQA` .

Despite the fast computation times, the method of successive convexification in its current form has two practical issues that prevent us from effectively using them in real-time. First, the modeling errors in the robot and the error in predicted obstacle motion can lead to unsafe behaviors and infeasibility in future time steps. Second, the initial state of the robot itself may be in violation with one of the constraints in the MPC formulation, which may prevent the successive convexification from producing a feasible trajectory. While introducing a margin of error in the constraints can partially address the first issue, the second issue is currently solved by introducing control barrier functions (CBFs) [18]. CBF admits violations in a limited sense, i.e., it ensures that safety constraints are eventually not violated, even when starting from initial conditions that violate safety. However, a direct incorporation of CBF into an MPC formulation is hard due to the underlying non-convexity. *We utilize successive convexification, specifically difference-of-convex programming, to enforce CBF-based constraints in MPC.*

Control barrier functions (CBFs) were introduced in mid

This project is supported by Wipro IISc Research and Innovation Network (WIRIN) Initiative and the Robert Bosch Centre for Cyber Physical Systems.

[1]Varun V. P. is associated with the Robert Bosch Centre for Cyber-Physical Systems, IISc, India

[2]A. P. Vinod is with Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA, 02139 USA. Email: `aby.vinod@gmail.com`

[3]S. Kolathaya is associated with the Robert Bosch Centre for Cyber-Physical Systems and the department of Computer Science and Automation, IISc, India

2010's as a means to address safety-critical control problems via optimization based control laws. For the problem of obstacle avoidance, these functions are like artificial potential fields [11], with the added property of 1) asymptotic convergence to the safe region and 2) optimality of control laws. Potential fields are usually associated with a flag which is activated whenever the system is very close to the obstacle. CBFs do not require such conditional activations, and the constraints are well defined for states outside the safety set. Furthermore, CBFs not only ensure forward invariance of a specified safety set (super-level set generated by the constraint), but also allow asymptotic convergence to this set. This makes the CBFs more robust to constraint violations, which, in realistic scenarios, occur very often. Traditionally, CBFs have been used as a filter that eliminates the safety-violating decisions given by the higher level planner (say MPC) [1], [3]. This "short-sighted" approach, even though guarantees safety, does not yield a goal-bound trajectory, i.e., the filtered commands are in conflict with the main objective of reaching a destination. Therefore, we propose to reformulate the CBFs and use it in the higher level MPC planner. In order to improve feasibilty and speed of computation, we will approximate the nonconvex part to yield a convex sub-problem. This can be iteratively evaluated until convergence. This results in real-time generation of locally optimal trajectories avoiding obstacles and reaching the destination.

The main contribution of the paper is with the convexification of the control barrier functions in MPC optimization problems. We will first formulate the obstacle avoidance constraints in a standard MPC formulation for constrained motion planning using CBF-based constraints. The resulting constraints are nonlinear and non-convex. In particular, these constraints have two terms, a convex and a concave constraint. By the method of successive convexification for difference-of-convex programming, we repeatedly solve a convex optimization problem until convergence. At convergence, we obtain a locally-optimal, dynamically-feasible trajectory that satisfies the obstacle avoidance constraints. We demonstrate the proposed solution in two UAV obstacle avoidance problems.

The paper is structured as follows. We discuss the mathematical preliminaries and the problem formulation in Section II. We introduce our main approach in the paper, i.e., successive convexification of control barrier functions for solving nonlinear MPC optimization problems, in Section III. We finally discuss the results on a simulated UAV platform in Section IV.

II. Preliminaries and problem formulation

Notation: For any vectors $x; y$, $x \quad y$ and $x \quad y$ imply elementwise inequality. We denote the set of integers between (and including) $a; b \ 2 \ \mathbb{N}$ by $\mathbb{N}_{[a;b]}$.

A. Control barrier functions

In this subsection, we will review *control barrier functions* for discrete time systems, and also state their relationships with forward invariance of a set (see [1] for a detailed account of discrete-time barrier functions). Consider a system of the form:

$$x_{k+1} = f(x_k; u_k); \tag{1}$$

where $x_k \ 2 \ \mathbb{R}^n$, $u_k \ 2 \ \mathbb{R}^m$, $f : \mathbb{R}^n \quad \mathbb{R}^m \ ! \ \mathbb{R}^n$ is locally Lipschitz. Given an initial condition $x_0 \ 2 \ \mathbb{R}^n$, and a feedback control law $u_k = v(x_k)$, $v : \mathbb{R}^n \ ! \ \mathbb{R}^m$ for some control objective (say tracking), there exists a maximum sequence of k's $I(x_0) = f0; 1; 2; : : : ; Tg$ such that x_k is the unique solution to (1) on $I(x_0)$; in the case when (1) is forward complete, $T = 1$. A set $S \quad \mathbb{R}^n$ is forward invariant w.r.t. (1) if for every $x_0 \ 2 \ S$, $x_k \ 2 \ S$ for all $k \ 2 \ I(x_0)$. If S is forward invariant, then we call the set S *safe*.

Given a closed set $C \quad \mathbb{R}^n$ (which is a strict subset of $\mathbb{R}^n$), we determine conditions such that it is forward invariant. C is defined as

$$C = fx_k \ 2 \ \mathbb{R}^n : h(x_k) \quad 0g; \tag{2}$$

$$@C = fx_k \ 2 \ \mathbb{R}^n : h(x_k) = 0g; \tag{3}$$

$$\mathrm{Int}(C) = fx_k \ 2 \ \mathbb{R}^n : h(x_k) > 0g; \tag{4}$$

where $h : \mathbb{R}^n \ ! \ \mathbb{R}$ is a continuous function. It is also assumed that $\mathrm{Int}(C)$ is non-empty and C has no isolated points, i.e., $\mathrm{Int}(C) \ 6= \ ;$, and $\overline{\mathrm{Int}(C)} = C$. We are interested in a feedback control law that ensures forward invariance of C. We call this controller a *safeguarding* controller w.r.t. the set C. We can obtain a suitable safeguarding controller via control barrier functions (CBFs).

Initially defined in [3, Section II.B], the goal was to not only realize forward invariance of C, but also to realize asymptotic convergence of C. In other words, if $x_0 \ 2 \ D$, with D being a slightly larger set such that $C \quad D \quad \mathbb{R}^n$, we wish to have an inequality of the form:

$$h(x_{k+1}) \quad \quad h(x_k); \tag{5}$$

for some $\quad 2 \ (0; 1)$. The above inequality ensures that $h(x_k) \quad 0$ for all k, if $h(x_0) \quad 0$. In addition, If the system (1) is forward complete, we have that $h(x_0) < 0 \quad =)$ $\lim_{k \ ! \ 1} h(x_k) \quad 0$. If there exists a control law such that (5) is satisfied, then we call this type of function a *control barrier function* (BF). We have the following formal definition of a CBF [1].

Definition 1. *Given a set $C \quad \mathbb{R}^n$ defined by* (2)-(4) *for a continuous function $h : \mathbb{R}^n \ ! \ \mathbb{R}$, the function h is called a* **control barrier function** *(CBF) defined on the open set D with $C \quad D \quad \mathbb{R}^n$, if there exists a set of controls $\mathbb{U}$, and a $\ 2 \ (0; 1)$ such that for all $x_k \ 2 \ D$,*

$$\sup_{u_k 2 \mathbb{U}} h(f(x_k; u_k)) \quad \quad h(x_k): \tag{6}$$

It is important to note that the CBF defined above is usually a nonlinear function of the states and inputs. CBFs found a lot of success for continuous time systems, where the derivative of the CBFs contained an affine structure [3]. Robotic systems largely fall under this category, wherein safeguarding control laws can be directly formulated as

quadratic programs (QPs). This type of QP formulation uses only the inputs as decision variables. Since our interests are more towards realizing optimal control laws for the next N steps, our optimization requires both the inputs and the states for N steps as decision variables. This can be achieved via model predictive control.

B. Problem formulation

We consider the motion planning problem with linear dynamics for the UAVs,

$$x_{k+1} = Ax_k + Bu_k; \tag{7}$$

with state $x \in \mathbb{R}^n$, input $u \in U \subset \mathbb{R}^m$ for some convex and compact set U denoting the actuation constraints, and matrices $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$. We use N to denote the prediction horizon used in planning, and we wish to drive the UAV to a target state $x_f \in \mathbb{R}^n$ while staying with a convex and compact set $X \subset \mathbb{R}^n$.

We also wish to avoid $N_O \in \mathbb{N}$ dynamic obstacles (see Fig. 2). However, we typically do not have access to the future trajectory of these obstacles. In practice, we typically rely on simple models like constant velocity [5] to predict the obstacle motion. However, such simple models will cause infeasibility in the optimal control formulations, due to the deviation between the predicted and realized obstacle trajectories.

The control barrier functions, discussed in Section II-A, enables enforcement of the obstacle avoidance as a soft constraint, which mitigates the problem of infeasibility. Specifically, at each time step k, let $\{c_{ik}\}_{k=0}^{N}$ and r_i for $i \in \mathbb{N}_{[1;N_O]}$ be the known predicted motion of the dynamic obstacle and the desired clearance respectively. Consequently, we cast the obstacle avoidance constraint,

$$\|x_k - c_{ik}\| \geq r_i \tag{8}$$

for every $k \in \mathbb{N}_{[1;N]}$ and $i \in \mathbb{N}_{[1;N_O]}$ as follows,

$$h_i(x_{k+1}; c_{ik}; r_i) \geq h_i(x_k; c_{ik}; r_i) \tag{9}$$

where $h_i : \mathbb{R}^n \to \mathbb{R}$ for each $i \in \mathbb{N}_{[1;N_O]}$ is defined as follows,

$$h_i(x; c_i; r_i) = \|x - c_i\|^2 - r_i^2. \tag{10}$$

Note that (10) is positive provided x and c is more than r-separated, which is aligned with safety definition for obstacle avoidance (8).

We obtain the following receding horizon optimal control problem with the current state x_0 of the UAV,

$$\underset{\substack{x_1, x_2, \ldots, x_N \\ u_0, u_1, \ldots, u_{N-1}}}{\text{minimize}} \quad \sum_{k=0}^{N-1} (x_k - x_f)^T Q (x_k - x_f) + u_k^T R u_k \tag{11a}$$

$$\text{subject to} \quad x_0 = x_0 \tag{11b}$$

$$\forall k \in \mathbb{N}_{[0;N-1]} \quad x_{k+1} = Ax_k + Bu_k; \tag{11c}$$

$$\forall k \in \mathbb{N}_{[1;N]} \quad u_k \in U; x_k \in X; \tag{11d}$$

$$\forall k \in \mathbb{N}_{[0;N-1]}, \forall i \in \mathbb{N}_{[1;N_O]} \quad h_i(x_{k+1}; c_{ik}; r_i) \geq h_i(x_k; c_{ik}; r_i); \tag{11e}$$

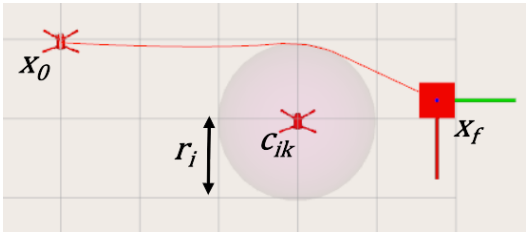


Fig. 2: x_0 is the starting state, x_f is the final state, $(c_{ik}; r_i)$ is the predicted motion of the i^{th} dynamic obstacle at time k with the corresponding desired clearance r_i

While h_i is a convex function, (11d) is a non-convex constraint. Consequently, we can not solve (11) in its current form in real-time using convex optimization.

Problem 1. *Design a real-time implementable algorithm that computes a feasible solution to* (11) *using successive convexification.*

III. Motion planning using successive convexification of CBFs

A naive approach to solve (11) is to linearize the constraint (11d) with respect to the decision variables about a solution guess, and repeatedly solve the resulting quadratic program. However, such an approach may not converge to a feasible solution. Instead, we exploit the *difference-of-convex* structure in (11d) to propose a successive convexification technique inspired by [13] to solve (11).

Recall that a function $g : \mathbb{R}^n \to \mathbb{R}$ is a *difference-of-convex* function if there exists two convex functions $g_1, g_2 : \mathbb{R}^n \to \mathbb{R}$ such that $g = g_1 - g_2$. The class of difference-of-convex functions is quite broad, since it includes all twice-differentiable functions with bounded Hessians [13]. The constraint $g \geq 0$ is a difference-of-convex constraint, when g is difference-of-convex.

Lemma 1. (11d) *is a difference-of-convex constraint.*

Proof. We can express (11d) for each $i \in \mathbb{N}_{[1;N_O]}$ and $k \in \mathbb{N}_{[0;N-1]}$ as follows

$$h_i(x_k; c_{ik}; r_i) - h_i(x_{k+1}; c_{ik}; r_i) \leq 0. \tag{12}$$

The convexity of $-h_i(x_k; c_{ik}; r_i)$ over $x_k \in \mathbb{R}^n$ follows from the convexity of h_i over x by (10) and the fact that -0 [4, Sec 3.2.1]. Similarly, the convexity of $h_i(x_{k+1}; c_{ik}; r_i)$ follows from the observation the convexity is preserved under affine transformation [4, Sec 3.2.2]. □

While difference-of-convex constraints are still non-convex constraints, one can convexify them without over-approximating the feasible space [13]. Recall that the first-order linear approximation of a convex function is a global underapproximant of the function [4, Sec 3.1.3]. In other words, for any $x_0 \in \mathbb{R}^n$, we have

$$f_1(x) - f_2(x) \geq f_1(x) - f_2(x_0) - \nabla f_2(x_0)(x - x_0);$$

Consequently, for any $x_0 \in \mathbb{R}^n$, we have

$$\{x : f_1(x) \leq (f_2(x_0) + \nabla f_2(x_0)^T (x - x_0))\} \subseteq \{x : g(x) \leq 0\}. \tag{13}$$

Lemma 2. *For any collection of states $\{z_k\}_{k=0}^N \in \mathbb{R}^{Nn}$ and the predicted obstacle information $\{(c_{ik}, r_i)\}_{k=0}^N$ for every $i \in \mathbb{N}_{[1;N_O]}$, every feasible solution satisfying the following collection of convex quadratic constraints for $k \in \mathbb{N}_{[0;N-1]}$ and $i \in \mathbb{N}_{[1;N_O]}$*

$$\begin{aligned} \|x_k - c_{ik}\|^2 \geq\ & \|z_{k+1} - c_{ik}\|^2 \\ & + 2(z_{k+1} - c_{ik})^T (x_{k+1} - z_{k+1}) \\ & + r_i^2(\gamma - 1) \end{aligned}; \tag{14}$$

also satisfy (11d).

Proof. We see that (14) inner-approximates the original constraint (11d) by (13). We observe that (14) is a quadratic function in the decision variables x_k and x_{k+1}. □

Using Lemma 2, we obtain the following convexification of the original problem (11),

$$\begin{aligned} \underset{\substack{x_1, x_2, \ldots, x_N \\ u_0, u_1, \ldots, u_{N-1}}}{\text{minimize}} \quad & \text{(11a)} \\ \text{subject to} \quad & \text{(11b); (11c); (14)} \end{aligned} \tag{15}$$

We note that (15) is a convex, quadratically-constrained quadratic program, and can be efficiently solved using off-the-shelf solvers like ECOS [7]. We next tackle the problem of identifying the states about which we linearize the non-convex constraint (11d) to obtain (14).

The feasible space associated with (14) can be empty since (14) is an inner-approximation of the constraints (11d). To avoid such a scenario, we need to carefully choose the states $\{z_k\}_{k=0}^N$ about which we linearize (11d).

In [13, Alg. 1], the authors ensure that the feasible space of the convexified subproblem of a general difference-of-convex program is non-empty by assuming access to an initial feasible solution guess. However, obtaining feasible solutions is (11) is hard due to its non-convexity. Therefore, we utilize the penalty-based approach proposed in [13, Alg. 2], which accommodates an infeasible start. Specifically, we relax the constraints (14) via slack variables, which we subsequently penalize in the objective.

We summarize the approach to solve (11) in Algorithm 1. We first solve a convex quadratic program that ignores the non-convex obstacle avoidance constraint (11d). Next, we iteratively solve a relaxed version of (15) where the convex quadratic constraints from Lemma 2 are relaxed by slack variables, that are penalized in the objective. Recall that any feasible solution to (17) with slack variables zero automatically becomes feasible for (11) due to the difference-of-convex structure (13). Therefore, we progressively increase the penalty suffered by non-zero slack in the objective across iterations, and terminate when the sum of the non-negative slack variables becomes sufficiently low and the cost converges across iterations, as prescribed in [13].

Algorithm 1: Successive convexification using difference-of-convex CBF constraints

Data: UAV dynamics (A, B), UAV state and input constraints $\mathcal{X}$ and $\mathcal{U}$, UAV current state x_0, UAV target state x_f, UAV quadratic costs Q, R, obstacle predicted motion $\{(c_{ik}, r_i)\}_{k=0}^N$, penalty scaling $\gamma > 1$, maximum scaling $\gamma_{\max}$, thresholds for convergence $\epsilon_{\text{viol}}, \epsilon_{\text{cost}}$

Result: A feasible solution (possibly sub-optimal) to (11)

Compute $\{z_k\}_{k=0}^N$ by solving the following quadratic program (ignore obstacle avoidance constraints and set $z_0 = x_0$):

$$\begin{aligned} \{z_k\}_{k=1}^N, \{v_k\}_{k=0}^{N-1} \in \arg\min_{\substack{x_1, x_2, \ldots, x_N \\ u_0, u_1, \ldots, u_{N-1}}} \quad & \sum_{k=0}^{N-1} (x_k - x_f)^T Q (x_k - x_f) + u_k^T R u_k \\ \text{subject to} \quad & \\ \forall k \in \mathbb{N}_{[0;N-1]} \quad & x_{k+1} = A x_k + B u_k, \\ \forall k \in \mathbb{N}_{[1;N]} \quad & u_k \in \mathcal{U}, x_k \in \mathcal{X}, \end{aligned} \tag{16}$$

$\texttt{sum\_slacks} \leftarrow 1, \texttt{cost} \leftarrow 1, \texttt{cost}_\text{p} \leftarrow 1$

while $\texttt{sum\_slacks} \geq \epsilon_{\text{viol}}$ *and* $\texttt{cost} \geq \epsilon_{\text{cost}}$ **do**

  Compute $\{x_k\}_{k=0}^N$, $\{u_k\}_{k=0}^N$, and current cost estimate $\texttt{cost}$ by solving the following QCQP:

$$\begin{aligned} \underset{\substack{x_1, x_2, \ldots, x_N \\ u_0, u_1, \ldots, u_{N-1}}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} (x_k - x_f)^T Q (x_k - x_f) + u_k^T R u_k + \rho \sum_{ik} s_{ik} \\ \text{subject to} \quad & \\ \forall k \in \mathbb{N}_{[0;N-1]} \quad & x_{k+1} = A x_k + B u_k, \\ \forall k \in \mathbb{N}_{[1;N]} \quad & u_k \in \mathcal{U}, x_k \in \mathcal{X}, \\ \forall k \in \mathbb{N}_{[0;N-1]}; \ \forall i \in \mathbb{N}_{[1;N_O]} \quad & \|x_k - c_{ik}\|^2 \geq \|z_{k+1} - c_{ik}\|^2 + 2(z_{k+1} - c_{ik})^T (x_{k+1} - z_{k+1}) + r_i^2(\gamma - 1) - s_{ik}, \ s_{ik} \geq 0 \end{aligned} \tag{17}$$

  $\gamma \leftarrow \min(\gamma \cdot \alpha, \gamma_{\max})$

  $\texttt{cost} \leftarrow \texttt{cost} - \rho \cdot \texttt{cost}_\text{p}, \texttt{cost}_\text{p} \leftarrow z_N - x_N$

  $\texttt{sum\_slacks} \leftarrow \sum_{i \in \mathbb{N}_{[1;N_O]}, k \in \mathbb{N}_{[0;N-1]}} s_{ik}$

end

IV. Results

The proposed algorithm is tested on UAVs in the Gazebo physics simulator, implemented using the ROS (Robotic Operating System) framework. The UAVs use the PX4 SITL (Software In The Loop) autopilot. Here the control outputs are implemented realistically as it would on a real UAV, by passing through the low level controller of the SITL autopilot. The algorithm is coded using the CVXPY modeling language [6], which allows us to implement the objectives and constraints close to its natural form. The simulation computer runs Ubuntu 18 on an Intel i5 7300HQ CPU with 16 GB of RAM, and an NVIDIA GTX1050 GPU.

In the implementation of the algorithm, four UAVs are

used. Each UAV executes an instance of the MPC obstacle avoidance code. Each UAV treats the other UAVs as moving obstacles. It is assumed that the obstacle location and velocity is known for each UAV in real-time. The velocity information is used to extrapolate the obstacle trajectory for the entire horizon (shown by red straight lines in Fig. 3a and 4). An imaginary radius of the obstacle is also set. A background program collects the position and velocity of each UAV and publishes all of them on a single topic, which is subscribed to by all the UAVs. It also renders markers (shown as coloured squares) which can be dragged to set the target position for each UAV in Rviz, a graphics interface. Each UAV, having received the target position and the obstacle states, plans its path accordingly.

We will first demonstrate the controller with a single obstacle, and then extend it for three more obstacles. In both cases, the following simulation parameters are used. The prediction horizon $N = 15$, and each step is 0.1 s long. The velocity limits for all UAVs are set to 1 m/s. Higher speeds are possible, but the simulating computer was constrained in terms of CPU and memory by simulating four UAVs. Each obstacle is depicted by a circle of radius $r_i = 1.5$ m, which is the desired clearance between all the UAVs. The robot and the obstacles use linear discrete point mass dynamics. We set the other parameters $= 4$; $_{max} = 10^5$; $_{cost} = 0.1$; $_{viol} = 0.001$ based on the discussion in [13]. We chose the values of ; $_{max}$, and $_{cost}$ to obtain a fast convergence to a local optimum for real-time implementation, and the values of $_{viol}$ to ensure reasonable constraint satisfaction of the resulting trajectory.

A. Single obstacle avoidance ($N_O = 1$)

Here, the UAV has to cross the obstacle at $(0, 4)$ to reach the destination at $(0, 9)$. Fig. 3a shows the setup. The rise in number of iterations and solve time in Fig. 3b correspond to the moments where the UAV is making its way around the obstacle. The iterations and solve times reduce significantly after the UAV clears the obstacle.

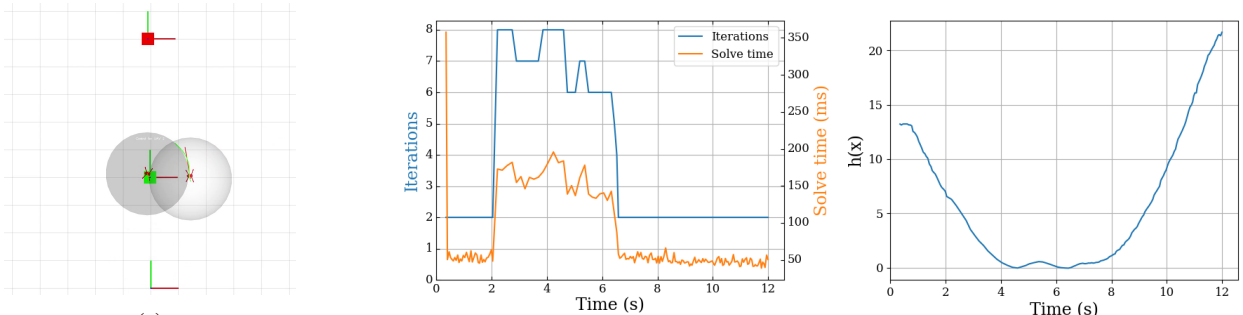
Fig. 3c shows the variation in h as the UAV goes towards the target. We can see that h is mostly positive. This is acceptable as the CBF constraint is soft, eventually allowing for recovery in the subsequent timesteps. Hence the CBF prevents the solver from failing due to the momentary constraint violation. This is important, considering that the simplified models used for both the UAV and the obstacle do not account for inertial effects, and there might be real world instances where the UAV might find itself inside another obstacle briefly.

B. Multiple obstacle avoidance ($N_O = 3$)

For the multiple obstacle simulations, we have four UAVs, each running Algorithm 1. For the sake of brevity, we analyze the iteration and solve times of a single UAV that has to make its way through the three obstacles. From Fig. 4, we can see that the UAV is successfully able to navigate around the obstacles dynamically. While the solver time increases to 600 ms in the worst case scenario (Fig. 5), when dealing with multiple obstacles at once, it drops to around 100 ms after avoiding. This can be compensated by increasing the control horizon, so that there exist some feasible control inputs that the UAV can execute when dealing with longer solve times.

V. Conclusion

In this work, we demonstrate how dynamically moving obstacles can be avoided in real-time by using control barrier functions (CBFs). In particular, since the CBF constraint formulations are non-convex in nature, the method of successive convexifications accelerates the process of obtaining safe trajectories in real-time. We demonstrate our approach on UAVs in a Gazebo simulation framework. We show that the UAVs are able to effectively navigate around the obstacles, with a maximum solve time of 600 ms. Given that the computer is rendering, simulating and executing all four control algorithms satisfactorily in real-time, the individual MPC avoidance control algorithm shows promise to run on mobile compute platforms within a reasonable solve time, which will be the subject of a future work.



(a) (b) (c)

Fig. 3: Figures showing obstacle avoidance for a single obstacle. (a) shows the UAV (white shaded circle) at $t = 5$ s avoiding the obstacle (gray shaded circle). (b) shows the number of successive convexification iterations and the associated solve time. The time peaks to around 350 ms when the UAV gets very close to the obstacle. (c) shows the CBF varying as a function of time. It can be verified that the CBF has very few violations all throughout the trajectory.