UNIVERSITY OF CALIFORNIA SAN DIEGO

# Validation of Refining Control Barrier Functions for Hardware Applications

A thesis submitted in partial satisfaction of the
requirements for the degree Master of Science

in

Engineering Sciences (Mechanical Engineering)

by

Nathan Cusson-Nadeau

Committee in charge:

Professor Sylvia Herbert, Chair
Professor Nikolay Atansov
Professor Jorge Cortes

2023

The thesis of Nathan Cusson-Nadeau is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGEMENTS

I would like to acknowledge the numerous (too many to name) people from the Safety for Autonomous Systems Lab and Contextual Robotics Institute for the immense amount of help and support they offered throughout my thesis. Advice on ROS development and dealing with research hurdles were paramount to my success during this work. Specific shout outs to my advisor Professor Sylvia Herbert and mentor Sander Tonkens for their endless patience, teachings, and for the opportunities they provided me for my thesis.

ABSTRACT OF THE THESIS

Validation of Refining Control Barrier Functions for Hardware Applications

by

Nathan Cusson-Nadeau

Master of Science in Engineering Sciences (Mechanical Engineering)

University of California San Diego, 2023

Professor Sylvia Herbert, Chair

Control Barrier Functions (CBFs) have gained rapid popularity in the recent years as a method to verify and enforce safety properties in safety-critical controllers for autonomous systems. However, developing a valid CBF that is not overly conservative can prove to be a non-trivial task in conjunction with input constraints. Using a recently developed algorithm called RefineCBF, this task can be made easier by providing a constructive method that iteratively constructs a valid CBF using dynamic programming (DP) based reachability analysis. This work seeks to validate that RefineCBF can be used with hardware-in-the-loop by demonstrating the algorithm successfully enforcing safety online for a robotic agent. We successfully demonstrate this by showing that a three degree of freedom robot can safely reach a goal pose in the presence

of obstacle with minimal violations to safety using a safety filter whose constraint is informed from RefineCBF. Additionally, we demonstrate that in scenarios where the obstacles change in time in a non-adversarial way, RefineCBF can be used to adaptively enlarge the safe set online.

# Chapter 1

# Introduction

With the ever-increasing integration of autonomous systems into society, it is crucial to maximize the safety of these systems while minimally compromising their utility and efficiency. Previous research in the field of safe controls has developed robust safety tools based on Hamilton-Jacobi Reachability (HJR) [1] and Control Barrier Functions (CBF) [2]. Both methods provide a safe set for the system along with a control policy that keeps the system within the safe set. However, these methodologies have certain limitations. Firstly, although HJR offers a constructive approach to ensuring safety, it encounters exponentially increasing computational complexity as the dimension of the state-space grows—an issue often referred to as the "curse of dimensionality". While CBFs overcome this computational challenge by relying on analytical or data-driven methods, finding a valid CBF that is not excessively conservative or merely an approximation can be quite difficult. To address these issues, Choi aimed to unify these two theories through their work on Control Barrier Value Functions (CBVFs) [3]. CBVFs utilize HJR to construct valid and less conservative CBFs. However, this method still faces the same "curse of dimensionality" as HJR and the initial use of a constraint function to begin construction of a valid safe set can lead to unsafe behavior if used online before convergence.

Recently, Tonkens and Herbert introduced an algorithm called RefineCBF, which aims to overcome the shortcomings of CBVF [4]. RefineCBF demonstrates that safety can be guaranteed and the effects of the "curse of dimensionality" associated with DP-based approaches can be

mitigated by warmstarting the DP recursion of HJR with a candidate CBF instead of the typical constraint function.

Although the results of RefineCBF carry significant implications, the utility of the algorithm within a real-time system has not yet been demonstrated or evaluated. This work aims to showcase the practical value of the algorithm by implementing it in a real-time physics simulation environment and on physical hardware using a differential drive-based robot. Furthermore, this in-the-loop implementation allows us to assess the effectiveness of the algorithm in the face of changing constraint sets that are non-adversarial[1].

[1]Adversarial in this sense would mean that the obstacles would intersect with the current safe set.

# Chapter 2

# Related Works

## 2.1 An Efficient Reachability-Based Framework for Provably Safe Autonomous Navigation in Unknown Environments

Previous to our work, an online enforcement of safety for autonomous systems using reachability-based methods has been demonstrated by [5]. In this work, all unknown space is considered to be an obstacle and HJ reachability is use to compute the backward reachable set (BRS). The complement of the BRS is the safe set for the system. Safety is enforced by using the least restrictive safety controller when the system would enter an unsafe state, otherwise, it will use a nominal planner. This results in jerky behavior, and may frequently violate safety when there are time delays or measurement noise.

To circumvent the BRS computational challenge at run time, the algorithm only locally update the BRS when new environment information is uncovered by the sensors. This results in a faster update of the safe set in response to exploration of the environment.

## 2.2 Scalable Learning of Safety Guarantees for Autonomous Systems Using HJ Reachability

In their work, HJR is used to update safety analysis online upon receiving new information from the environment, system dynamics, or the predictions of agents in the operating environment.

They circumvent the computational complexity associated with HJR using decomposition, warm-starting, and adaptive grids, which allows safe sets to be updated by one or more orders of magnitude faster. These results were demonstrated on simulated high dimensional 10D quadcopters, but never tested on real-time hardware [6].

## 2.3 Governor-parameterized barrier function for safe output tracking with locally sensed constraints

In this work, the authors introduce a governor-parameterized barrier function (PBF) that can change in real-time as the governor state, system state, and sensor measurements change. The PBF defines a local safe set by quantifying the trade-off between safety and the system's stability. This can be used to achieve output-tracking with formal safety and stability guarantees [7]. The ability for this methodology to enable safe autonomous navigation in an *a priori* unknown unstructured environment was demonstrated using a simulated omnidirectional robot.

# Chapter 3

# Preliminaries

The following sections present the foundational theory underlying the RefineCBF algorithm, as well as the theory behind the algorithm itself. These preliminary explanations are necessary because the hardware application will directly rely on the following theoretical framework.

## 3.1 Dynamical System

The dynamics of a general nonlinear control-affine (linear in the control input $u$) system can be expressed by the following ordinary differential equation:

$$\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})u \tag{3.1}$$

where $\dot{\mathbf{x}}$ is the rate of change of the system state $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$, $u \in \mathcal{U} \subseteq \mathbb{R}^m$ is the system's current control input, $f : \mathbb{R}^n \to \mathbb{R}^n$ is a vector-valued function that describes the drift of the system (or open-loop dynamics), and $g : \mathbb{R}^n \to \mathbb{R}^{n \times m}$ is a matrix-value function that represents the control influence on the system. All of which are assumed to be locally Lipschitz continuous on their respective domains.

### 3.1.1 Nominal Policy

If the control signal is defined by a locally Lipschitz continuous on its domain nominal policy $\pi : \mathbb{R}^n \to \mathbb{R}^m$, the resulting dynamics will be:

$$\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})\pi \tag{3.2}$$

The Lipschitz continuity of $f$, $g$, and $\pi$ provide a sufficient condition for the existence and uniqueness of the solutions to (3.2). This is required for the following theoretical results to hold.

## 3.2 Constraint Set

### 3.2.1 Obstacle Set

We define the set of states that result in immediate failure of a system as the Minkowski Sum of:

$$\mathcal{O}_k^* = A + \mathcal{O}_k \tag{3.3}$$

where $A \in \mathbb{R}^p$ is the set of vertices that comprise the boundary of some shape, and similarly, $\mathcal{O}_k \in \mathbb{R}^p$ is the set of vertices that comprise the boundary of the shapes of an obstacle of interest at a discrete time $k$. $\mathcal{O}_k^*$ denotes the new inflated obstacle set vertices at this same time $k$. This can be viewed as inflating or enlarging the obstacles by the maximal dimensions of set $A$. This is useful, as it allows a physical system with shape $A$ to check for collisions with the shape defined by the original obstacle set.

### 3.2.2 Constraint Set

Let $\mathcal{L} \subseteq \mathcal{X}$, be defined as a 0-superlevel set of a bounded Lipschitz continuous function $\ell : \mathcal{X} \to \mathbb{R}$. $\ell(\mathbf{x})$ will be referred to as the constraint function, which will provide a way of

measuring the distance from a set of failure states, $\mathcal{L}^c$, the complement of $\mathcal{L}$. A state contained in $\mathcal{L}$ can be viewed as *safe*, so a trajectory staying within $\mathcal{L}$, can also be viewed as safe. Throughout this work, $\ell(\mathbf{x})$ will be equivalent to the signed distance function from the closest state contained in $\mathcal{L}^c$. And, $\mathcal{L}^c$ will be identical to the current inflated obstacle set $\mathcal{O}_k^*$ (i.e. $\mathcal{L}^c = \mathcal{O}_k^*$), as the robotic system will be tasked to avoid collision with obstacles. Therefore:

$$\ell(\mathbf{x}) = \begin{cases} -d(\mathbf{x}, \partial\mathcal{O}_k^*) & \text{if } \mathbf{x} \in \mathcal{O}_k^* \\ d(\mathbf{x}, \partial\mathcal{O}_k^*) & \text{if } \mathbf{x} \in \mathcal{O}_k^{*c} \end{cases} \tag{3.4}$$

where $d(\mathbf{x}, \partial\mathcal{O}_k^*)$ is the signed distance function from the closest boundary of the current inflated obstacle set. Thus our constraint and failure sets will be:

$$\mathcal{L} := \{\mathbf{x} \in \mathcal{X} : \ell(\mathbf{x}) > 0\} \tag{3.5}$$

$$\mathcal{L}^c := \{\mathbf{x} \in \mathcal{X} : \ell(\mathbf{x}) \leq 0\} \tag{3.6}$$

### 3.2.3 Viability Kernel

The viability kernel, $\mathcal{S}(t)$ is defined as the largest control invariant subset of $\mathcal{L}$:

$$\mathcal{S}(t) := \{\mathbf{x} \in \mathcal{L} : \exists u(\cdot) \in \mathcal{U}\, s.t.\, \xi_{\mathbf{x},t}^u(s) \in \mathcal{L}\ \forall s \in [t, 0]\} \tag{3.7}$$

$\mathcal{S}(t) \subseteq \mathcal{L}$ can be interpreted as the set of all initial states from which there exists an admissible control input which keeps the system contained within the constraint set for some set time duration $t$. The infinite-time ($t = \infty$) viability kernel is denoted as $\mathcal{S}^*$.

## 3.3 Control Barrier Functions (CBFs)

Recently popularized by [2], a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$ is a Control Barrier Function (CBF) for the control affine system (3.1) if there exists an extended

class $\mathcal{K}_\infty$ function $\alpha$ such that the following inequality is satisfied:

$$\sup_{u \in \mathcal{U}} \dot{h}(\mathbf{x}) := \sup_{u \in \mathcal{U}} [\, \mathcal{L}_f h(\mathbf{x}) + \mathcal{L}_g h(\mathbf{x}) u \,] \geq -\alpha(h(\mathbf{x})) \tag{3.8}$$

A continuous function $\alpha : (-b, a) \to (-\infty, \infty)$ belongs to extended class $\mathcal{K}_\infty$ for some $a, b > 0$ if it is monotonically increasing and is zero at zero ($\alpha(0) = 0$). Typically, a linear function $\alpha(z) = \gamma z$ is used, where $\gamma \in \mathbb{R}_+$ is the maximal discount rate of $h(\mathbf{x})$.

We define the 0-superlevel set of $h(x)$ as:

$$\mathcal{C}_h = \{\mathbf{x} \in \mathcal{X} \,|\, h(\mathbf{x}) \geq 0\} \tag{3.9}$$

### 3.3.1 Candidate CBFs and Valid CBFs

#### Candidate CBF

A *candidate* CBF is a continuously differentiable scalar function $h : \mathbb{R}^n \to \mathbb{R}$ for a closed set $\mathcal{C} \subseteq \mathcal{X}$ if it satisfies:

1. $h(\mathbf{x}) > 0 \Leftrightarrow \mathbf{x} \in Int(\mathcal{C})$, positive inside the set
2. $h(\mathbf{x}) = 0 \Leftrightarrow \mathbf{x} \in \partial\mathcal{C}$, zero at the boundary of the set

which are necessary but not sufficient conditions for (3.8). Hence, a candidate CBF may not be able to guarantee safety for all states within its 0-superlevel set, $\mathcal{C}_h$.

#### Valid CBF

In order for a candidate CBF to also be a *valid* CBF it must additionally satisfy (3.8) $\forall \mathbf{x} \in \mathcal{C}$ and $\mathcal{C} \subset \mathcal{S}^*$. A valid CBF is able to be used to impose a constraint on the input of the system which can assure the forward invariance of $\mathcal{C}_h$ (i.e. guarantee safety).

From [2], let $h$ be a CBF on $\mathcal{X}$, then any Lipschitz continuous control policy $\pi : \mathcal{X} \to \mathcal{U}$ such that $\pi(\mathbf{x}) \in K_h(\mathbf{x})$ where:

$$K_h(\mathbf{x}) := \{u \in \mathcal{U} \,|\, \dot{h}(\mathbf{x}) \geq -\gamma h(\mathbf{x})\} \tag{3.10}$$

will render $\mathcal{C}_h$ forward invariant. In addition, a valid CBF instills asymptotic stability in $\mathcal{C}_h$. When $h(\mathbf{x}) < 0$, a valid CBF describes a Control Lyapunov Function (CLF) and thus enforces an exponential return to safety.

### 3.3.2 Safety-Critical Control

A valid CBF can be used for safety-critical control by formulating an optimization problem called a CBF Quadratic Program (CBF-QP). To formulate this problem, we first define the set of all control values $K_h(\mathbf{x})$ that can render $\mathcal{C}_h$ forward invariant (i.e. safe) (3.10). Then by using the derivative requirement (3.8) on this set of controls as a constraint, and the fact that this constraint is linear in $u$, we can design a minimally invasive safety-filter for a safety-agnostic nominal policy $\hat{\pi}(\mathbf{x})$ by solving the following quadratic program (CBF-QP):

$$\begin{aligned} & u^*(\mathbf{x}) = \underset{u \in \mathbb{R}^m}{\operatorname{argmin}} \, ||u - \hat{\pi}(x)||_R^2 \\ \text{s.t.} \quad & \mathcal{L}_f h(\mathbf{x}) + \mathcal{L}_g h(\mathbf{x}) u \geq -\gamma h(\mathbf{x}) \\ & u \in \mathcal{U} \end{aligned} \tag{3.11}$$

where $u^*(\mathbf{x}) \in \mathbb{R}^m$ is the resulting safe control signal, $\hat{\pi}(\mathbf{x}) \in \mathbb{R}^m$ is a nominal (safety-agnostic) control policy, $\mathcal{L}_f$ is the Lie Derivative with respect to $f$, the open-loop dynamics and similarly, $\mathcal{L}_g$ is the Lie Derivative along the vector field $g$ (the vector field which dictates how the dynamics will evolve based on the control influence). Of note, $\mathcal{U}$ must be a convex polytope in order for (3.11) to be a quadratic program.

Solving this optimization problem will yield the safest control possible while minimizing the deviation from the nominal control. Notably, even when not provided a nominal policy to filter, (3.11) returns a safety preserving control signal. Critically, the (3.11) form as a QP allows it to be solved extremely quickly online, making it a viable safety enforcement tool in online

control applications.

The use of a candidate CBF instead of a valid CBF in (3.11) does not mean that a system cannot remain safe during operation. But, it will not provide safety guarantees and thus should be avoided when possible.

## 3.4 Hamilton-Jacobi Reachability (HJR)

Given an initial constraint function, HJR provides a constructive method that can be used to generate a value function, $V(\mathbf{x},t)$, where the function's 0-superlevel set defines the set of state's that can be reached within in a specified time frame [1].

The value function can be found by solving the Hamilton-Jacobi Bellman partial differential equation (HJB-PDE):

$$-\frac{\partial V}{\partial t} = H(\mathbf{x}, \nabla V) \tag{3.12}$$

where $H(\mathbf{x}, \nabla V)$ is called the Hamiltonian and is defined as:

$$H(\mathbf{x}, \nabla V) = \inf_{u \in \mathcal{U}} {}^{1} \langle \nabla V, f(\mathbf{x}, u) \rangle \tag{3.13}$$

However, solving the HJB-PDE analytically can prove quite challenging, one popular way to solve it numerically is through the dynamic programming (DP) principle. This is typically done through the following:

1. Discretize the state space into a grid.

2. Initialize the value function with some initial value at every grid point. This is typically initialized as the constraint function (3.4) defined as the signed distance function from some failure sets.

$$V(x,0) = \ell(\mathbf{x}) \tag{3.14}$$

[1] If instead we wanted to avoid a set of target states, the $\sup_{u \in \mathcal{U}}$, would be used.

3. Value iteration over the grid points until the value function converges.

In solving for $V(x,t)$, the optimal control policy $u^*(x)$ can be retrieved:

$$u^*(\mathbf{x}) = \underset{u \in \mathcal{U}}{\operatorname{argmin}}^2 \langle \nabla V, f(\mathbf{x}, u) \rangle \tag{3.15}$$

**Constraint Function Choice**

Commonly in safe control problem formulations, where the task at hand is to *avoid* an obstacle set ($\mathcal{O}$), the constraint function is equivalent to the SDF from (3.4). In a *reach* problem formulation, where the task at hand is to *reach* a target set of states, $\mathcal{G} \subseteq \mathbb{R}^n$, the constraint function is similarly the SDF from (3.4) but the instead of an obstacle set, the distance function measures from the current state to the goal state:

$$\ell(\mathbf{x}) = d(\mathbf{x}, \mathcal{G}) \tag{3.16}$$

## 3.5 Control Barrier Value Function (CBVF)

Control Barrier Value Functions (CBVFs) can be viewed as a unification of CBFs and HJR. A Control Barrier-Value Function $B_\lambda : \mathcal{X} \times (-\infty, 0] \to \mathbb{R}$ [3] is defined as:

$$B_\lambda(\mathbf{x}, t) := \max_{u \in \mathcal{U}_{[t,0]}} \min_{s \in [t,0]} e^{\lambda(s-t)} \ell(\xi^u_{x,t}(s)) \tag{3.17}$$

for some $\lambda \in \mathbb{R}_+$ and $\forall t \leq 0$, with initial condition $B_\lambda(x,0) = \ell(\mathbf{x})$.

Given $\mathcal{C}_{B_\lambda}(t) := \{\mathbf{x} \,|\, B_\gamma(\mathbf{x}, t) \geq 0\}$, a CBVF recovers the viability kernel, $\forall t \leq 0$, $\lambda \in \mathbb{R}_+$, $\mathcal{C}_{B_\lambda}(t) = \mathcal{S}(t)$. As such, a CBVF recovers the largest (time-varying) control invariant set for maintaining safety, similar to the converged HJR value function from the solution of (3.12) in an avoid case. To practically implement this, we would begin by spatially discretizing the state space into a grid and solving for $B_\lambda$ using dynamic programming recursion [8]:

[2] For an avoid problem, this would be the $\operatorname{argmax}_{u \in \mathcal{U}}$

$$B_\lambda(\mathbf{x},t) := \max_{u\in U} \min\{\min_{s\in[t,t+\delta]} e^{\lambda(s-t)}\ell(\xi^u_{\mathbf{x},t}(s)), e^{\lambda\delta}B_\lambda(\xi^u_{\mathbf{x},t}(t+\delta),t+\delta)\} \tag{3.18}$$

with the initial condition $B_\lambda(\mathbf{x},0) = \ell(\mathbf{x})$. This requirement of the initial condition as the constraint function is a key requirement for CBVFs.

As opposed to a standard HJR value function, using a CBVF in a safety filter (3.11) (i.e. $h(\mathbf{x}) = B_\lambda(\mathbf{x},t)$) allows for resulting trajectories to approach the boundary of the safe set, which is preferable in many robotic applications as it allows for more efficient goal reaching.

Analogous to (3.10), a control set satisfying the CBF derivative inequality (3.8) for a CBVF is described by:

$$K^\gamma_{B_\lambda}(\mathbf{x},s) := \{u\in\mathcal{U} \,|\, \dot{B}_\lambda(\mathbf{x},s) \geq -\gamma B_\lambda(\mathbf{x},s)\} \tag{3.19}$$

where $\lambda$ is the discount rate for the *offline* DP-recursion, and $\gamma$ is the maximal discount rate for the *online* CBVF constraint (3.8).

As a final remark, $\mathcal{C}_{B_\lambda}(t) \subseteq \mathcal{S}(t)$, therefore $B_\lambda$ satisfies the requirements to be a valid CBF almost everywhere.[3]

## 3.6 Refine CBF

Thus we finally arrive at the core recent theoretical development to be applied in this work, RefineCBF. In essence, RefineCBF warmstarts the dynamic programming recursion of (3.18) with a *candidate* CBF $h(\mathbf{x})$ instead of the constraint function $\ell(\mathbf{x})$ (i.e. $B_\lambda(\mathbf{x},0) = h(\mathbf{x})$ **not** $B_\lambda(\mathbf{x},0) = \ell(\mathbf{x})$). The idea being, an initial candidate CBF will be a better initial condition for the DP-recursion of (3.18). This also allows for expert crafted and data-driven CBFs to be utilized effectively - as they make excellent initial candidate CBFs to warmstart the DP-recursion.

[3] Because a CBVF an have points of non-differentiability, (3.8) cannot always be satisfied.

In order to practically implement and leverage the theoretical guarantees from warmstarting developed for HJR [9], it was proposed that applying a maximal discount rate of $\gamma$ in the safety-filter (3.11) online for a CBVF constructed from (3.17) maintains control invariance of the safe set [4].

In [4], theorem 2 is key to enabling the online implementation of this work, as it states that each iteration of RefineCBF will either maintain the current level of safety[4] or become less unsafe[5]. We refer the reader to [4], for details and proofs regarding the theorem. The implications of this theorem are that at every iteration of RefineCBF, the safe set is contained in the union of the viability kernel and its previous iteration:

$$\mathcal{C}_h(t-\delta) \subseteq \mathcal{S}^* \cup \mathcal{C}_h(t) \tag{3.20}$$

$\forall t \leq 0$, $\delta \in \mathbb{R}_{>0}$, where $\delta$ is a time step. In other words, the safe set is at all times contained within the safe set of the union of the viablity kernel and *candidate* CBF. This also means that if the initial candidate CBF is contained within the viability kernel, then the refined CBFs safe set will be contained within the viability kernel at all times. Inversely, if the viability kernel is contained within the initial candidate CBF, proceeding refined CBFs will be contained within the prior iterations safe set. Importantly, this means that if at any point during the DP recursive process the CBF is valid, it will remain valid for every further recursion!

As well as improving or keeping the current level of safety between iterations, RefineCBF similarly alters the conservativeness of the current iteration of the CBF. Each iteration of the current CBF will be as conservative or less conservative than the prior. Conservativeness is measured by how much coverage the safe set $\mathcal{C}_h$ has within $\mathcal{X}$. Figure 3.1 depicts a high-level visual of the provisions of RefineCBF.

[4] If the 0-superlevel set of the candidate CBF is conservative compared to the viability kernel.
[5] If the original candidate CBF was *not* valid.

### 3.6.1 Practically Implementing Refine CBF

In [4], steps were provided to implement RefineCBF in an offline or online setting. In this work, we will be following these steps explicitly. The steps are as follows:

1. Spatially discretize the initial analytical CBF $h_o(\mathbf{x})$, which will require the evaluation of the CBF for every point across this grid.

2. Apply RefineCBF (see algorithm 1 for full breakdown):

   (a) Use $\lambda = 0$ to update the CBVF (3.17) and use $\gamma \geq 0$ for the safety filter (3.11). Larger values of $\gamma$ in the safety filter will result in a faster exponential decrease to the boundary of the safe set.

   (b) Initialize the recursion process of (3.17) with the current discretized $h(\mathbf{x})$. Theorem 2 from [4] guarantees that recursive updates do not become more unsafe while converging to a valid CBVF.

3. The safety filter from (3.11) will be implemented using spatial finite differences for computing the Lie derivatives at the grid points and interpolation to determine the CBF and its time derivative.

**Algorithm 1.** RefineCBF Algorithm

**Require:** State space discretization $\mathcal{X}$, control space $\mathcal{U}$, time step $\delta$, discount factor $\gamma = 0$, iteration count $k$, initial candidate CBF $h_o(\mathbf{x})$

Initialize $B_{\gamma,0}(\mathbf{x},0) = h_o(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$
**while** not converged **do**
  **if** new $\ell(\mathbf{x})$ **then**
    update $\ell(\mathbf{x})$
  **end if**
  **for all** $x \in \mathcal{X}$ **do**
    HJ Reachability Value Update:
    Compute the gradient $\nabla_{\mathbf{x}} B_{\lambda,k}$
    Find the optimal control $u^* = \text{argmin}_{u \in \mathcal{U}} H(\mathbf{x}, \nabla_{\mathbf{x}} B_{\lambda,k})$
    Compute the Hamiltonian $H^*(\mathbf{x}, \nabla_{\mathbf{x}} B_{\lambda,k}) = \langle \nabla V, f(\mathbf{x}, u^*) \rangle$
    Update the value function $B_{\gamma,k+1}(\mathbf{x},t) = \max(\ell(\mathbf{x}), B_{\gamma,k}(\mathbf{x},t) + \delta H^*(\mathbf{x}, \nabla_{\mathbf{x}} B_{\lambda,k})$
  **end for**
  Update $B_{\gamma,k} \leftarrow B_{\gamma,k+1}$
  Update iteration count $k = k+1$
**end while**
**return** $B_{\gamma,k}$

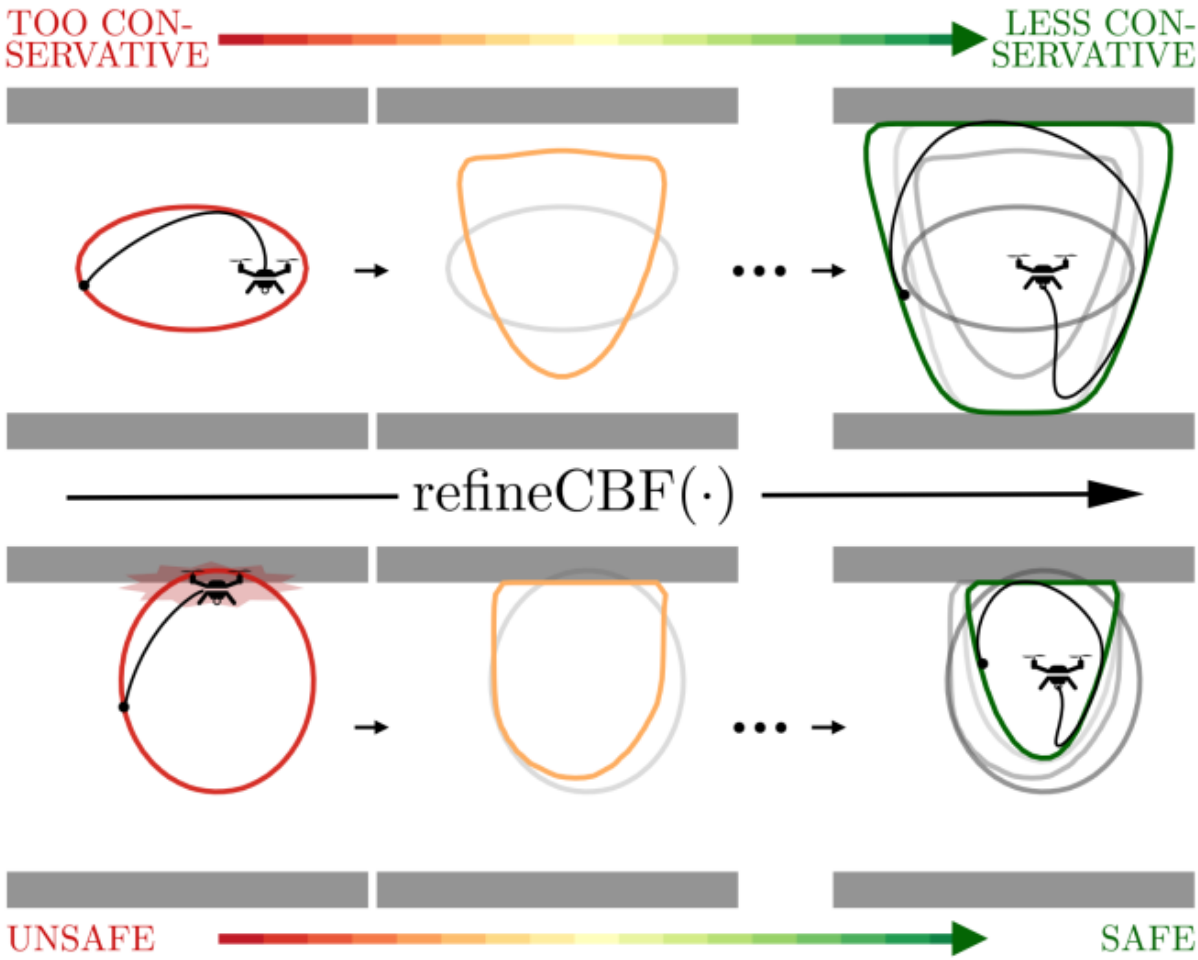



**Figure 3.1.** Refine CBF High-Level Representation [4]. In the top left is an initial 0-levelset of an overly conservative CBF. Through applying RefineCBF, this initial CBF can be refined to be less conservative. Thee 0-levelset of the refined CBF is shown in yellow and then green from left to right. Similarly, starting from the bottom right, an initial unsafe CBF can be refined to become more safe, as shown by the 0-levelsets of the yellow and green refined CBFs from left to right on the bottom.

# Chapter 4

# Refine CBF ROS Implementation

To provide modularity and ease of use of the software packaged developed for this project, we elected to implement RefineCBF in the Robot Operating System (ROS). Specifically, with the Ubuntu 20.04 LTS distribution of ROS2, Foxy Fitzroy. This was chosen due to its current live support and modernity[1].

ROS is an open-source middleware software package that allows computer program scripts (written in C++ or Python) to communicate with one another asynchronously. It is widely used in the robotics community in both academia and industry which allows the work developed here to be easily shared and utilized. [10]

ROS achieves a high degree of its modularity through how robotics software packages can be structured within it. Typically, different processes of the robot are written into distinct *nodes*, or programming scripts, which can exchange data with one another over *topics*.

## 4.1 RefineCBF: ROS Package

The core Refine CBF implementation in ROS can be seen as three distinct interfacing nodes in a network, these nodes being: **Dynamic Programming**, **Safety Filter**, **Nominal Controller**. Nodes communicate to each other over topics through a publisher/subscriber model. The rate at which the publishers and subscribers send or receive messages over topics is tunable

[1] ROS2 is becoming the *de facto* standard version across academia and industry.

by the user[2].

**Table 4.1.** RefineCBF ROS Publishers and Subscribers. *Because the State Estimation node will be different depending on if Gazebo or the Vicon arena is used, a general node name State Estimation and topic name /odom is used.

| Node | Published | Subscription 1 | Subscription 2 |
|---|---|---|---|
| Dynamic Programming | /cbf_availability | - | - |
| Nominal Controller | /nom_policy | /odom* | - |
| Safety Filter | /cmd_vel | /odom* | /cbf_availability |
| TurtleBot3 | - | /cmd_vel | - |
| State Estimation* | /odom* | - | - |

### 4.1.1 Dynamic Programming

This node is the core of RefineCBF algorithm the implementation. Initialized with a discretized candidate CBF over a predefined grid, the node executes the DP-recursion of (3.18) with a provided $\delta$, admissible control set $u \in \mathcal{U}$, and the dynamics of the system $\dot{\mathbf{x}}$ of the designer's choice. The CBF is iteratively updated indefinitely according to algorithm 1.

Upon completion of every iteration, the new CBF is saved to the remote PC. After saving, a boolean message of True is published over the topic named /cbf_availability.[3]

### 4.1.2 Nominal Policy

This node contains the nominal controller logic for the package. The choice of nominal controller does not matter, and is the designer's choice. If feedback is required, this node subscribes to a predefined /odom topic of ROS navigation message type Odom. The computed nominal policy will be published over the /nom_policy ROS geometry message type, Twist.

[2]This is only to a limit. If computation time within a node is too large, the frequency will be bottle-necked at a lower limit by the computation time.

[3]This was done because ROS2's default message types do not support arrays of dimensions typically used to represent the discretized CBF. Through the clever crafting of a custom message type or packaging of the array, it may be possible to avoid the saving CBF process, but this implementation presented no issues during our work.

### 4.1.3 Safety Filter

This node is the software implementation of the CBF-QP as seen in (3.11). It subscribes to the current system state through the topic /odom, nominal control topic /nom_policy, and finally /cbf_availability. Upon receiving a True from /cbf_availability, the node will load the most recent CBF for use in the constraint of the CBF-QP.

With this information, it indefinitely computes the minimally invasive safe control by solving (3.11) with the most recent values of the CBF and nominal policy. $h(\mathbf{x})$ will be interpolated if the current state does not lie excatly on a grid point. This safe control is then published over the ROS geometry message type, Twist, as the topic /cmd_vel.

# Chapter 5

# Experiment

## 5.1 Hardware Platform: Turtlebot3 Burger

To validate RefineCBF, we elect to use the Robotis Turtlebot3 Burger robot. The robot's specifications can be found in Table 5.1. Notable specifications are the maximum linear and angular velocities and the radius, as these will be utilized in the applied theory discussed later.

### 5.1.1 Modifications

We chose to slightly modify the tires of the Turtlebot by stretching rubberbands around them. This was reduce slippage in the wheels by increasing the coefficient of friction of the tires on our surface. Figure 5.4 shows these modifications at various angles.

**Table 5.1.** Turtlebot3 Burger Specifications

| **Turtlebot3 Burger Specifications** | |
|---|---|
| Maximum Linear Velocity (m/s) | 0.22 |
| Maximum Angular Velocity (rad/s) | 2.84 |
| Radius (mm) | 105 |
| Size (L x W x H) (mm) | 138 x 178 x 192 |
| Weight (SBC + Battery + Sensors) (kg) | 1 |
| SBC (Single Board Computers) | Raspberry Pi 3 Model B |
| MCU (OpenCR) | 32bit ARM Cortex-M7 w/ FPU |
| Actuators | XL430-W250 |
| Battery | Lithium polymer 11.1V 1800mAh / 19.98Wh 5C |

### 5.1.2 Differential Drive Dynamics

The Turtlebot3 Burger's dynamics can be modeled as a differential drive robot (sometimes also referred to as a bicycle model), which is a control-affine system (3.1) that can be expressed as:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{5.1}$$

where $\mathbf{x} \in \mathbb{R}^3$ is defined as $\mathbf{x} := (x, y, \theta)$ where $x$ and $y$ are the systems $x$ and $y$ coordinates respectively, and $\theta$ is the heading angle of the robot. The control input $u \in \mathbb{R}^2$ is defined as $u := (v, \omega)$ and is composed of the robot's linear velocity $v$ and angular velocity $\omega$. The $f(x)$ term from (3.1) does not appear because this system does not contain any drift (i.e. $f(x) = \mathbf{0} \in \mathbb{R}^3$). Lastly, $g(x) \in \mathbb{R}^{3\times 2}$ is defined as:

$$g(x) := \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix}$$

## 5.2 Physics Simulation: Gazebo

Gazebo, an open-source 3D robotics simulator, was used for the physics simulation environment. Gazebo accurately simulates real-world physics, providing a high-fidelity simulation environment. It enables developers to quickly test algorithms and design robots within digital environments. In our work, we utilize Gazebo for sim-to-real comparison purposes by subjecting the Turtlebot3 to identical tasks in both simulated and real-world settings. Conveniently, Robotis offers a Turtlebot3 package specifically designed for Gazebo, which includes a physical model

of the robot with accurately scaled visual and collision geometry. Additionally, this package provides precise state estimation for our robot. An example of the Turtlebot3 model in a simulated environment is shown in Figure (5.1).

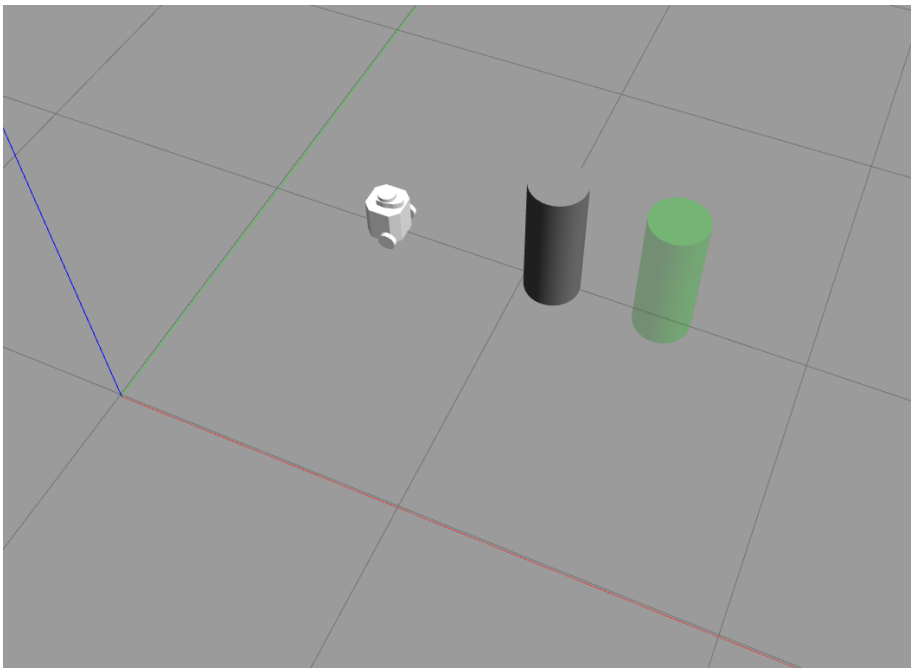


**Figure 5.1.** Turtlebot3 in Gazebo World. The positive y-axis is represented in green, positive x-axis in red, and positive z in blue. Each grid square is 1 meter by 1 meter. The Turtlebot3 Burger is in white. The residual obstacle is represented by the dark gray cylindrical obstacle. The goal set of states $\mathcal{G}$ is represented by the green semi-translucent cylinder.

## 5.3 Hardware: State Estimation

In order to get near perfect state estimation of the Turtlebot3 during hardware operation, we elected to use a Vicon motion capture camera system. By attaching light and reflective motion capture balls[1] about the robot asymmetrically, and using up to 30 infrared cameras to track the robot (5.4), the Vicon system can get a very precise[2] estimate of where the robot's pose is in relation to a predefined world frame origin. The defined body frame origin of the robot is

[1] Pearl Markers manufactured from B&L Engineering

[2] Within millimeters of the true position.

determined by the centroid of the object formed by the motion capture balls, depicted in Figure 5.3.

**Figure 5.2.** Hardware Setup. A black mat was laid out within the UCSD aerodrome which contains 30 IR cameras. The orange dotted lines represent 1 meter by 1 meter squares. The positive y-axis is in the direction of the green arrow at the bottom right and the positive x-axis the red. The Turtlebot3 Burger is the black wheeled robot within the red circle. The red circle represents the 0-levelset of the initial candidate CBF. The final obstacle set is represented by the blue lines and chalk bucket in the center. The green circle represents the goal set $\mathcal{G}$.

### 5.3.1 Defining the World Frame Origin

To define the $x$,$y$,$z$ origin for the world frame of reference, the Vicon system uses an infrared (IR) laser cross (see Figure 5.5). While laying IR laser side up, the long and short handles of the wand point in the positive $y$, and positive $x$ axes directions respectively. Following

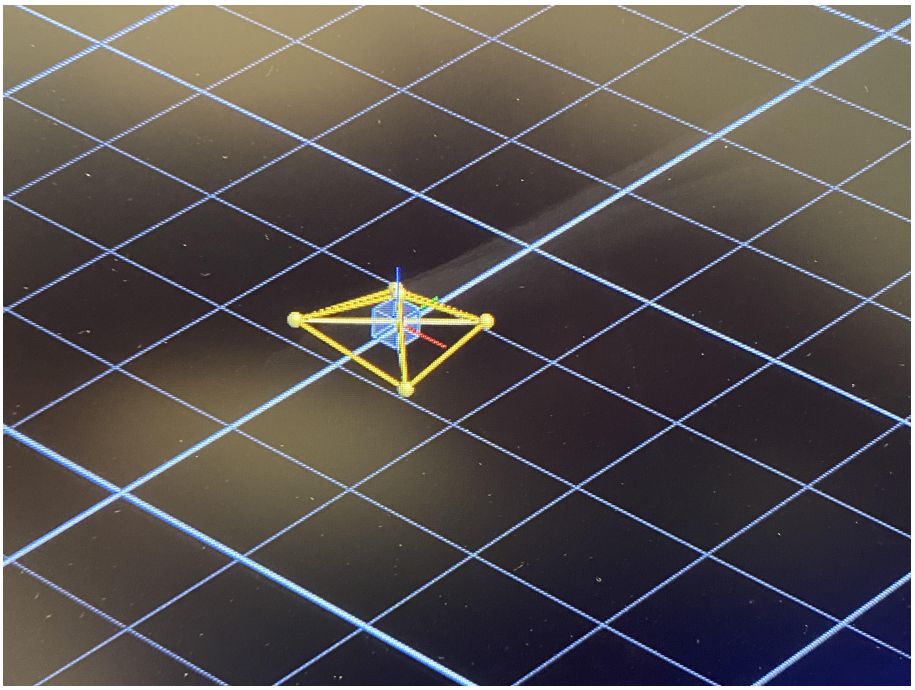
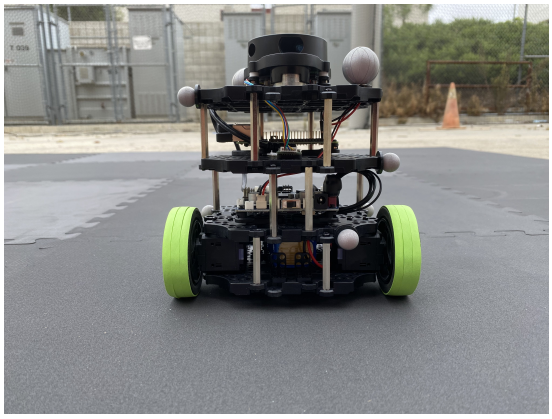


**Figure 5.3.** Turtlebot3 Centroid In Vicon. The Turtlebot3 object in the Vicon software. Orange spheres are detected MoCap balls, red-green-blue coordinate axis stems from the centroid of all of the balls. Coordinate axis colors follow typical convention, red for +x, green for +y, blue for +z with origin centered at the body frame origin. (x,y,z) coordinate is determined by body frame origin in relation to the world frame origin.

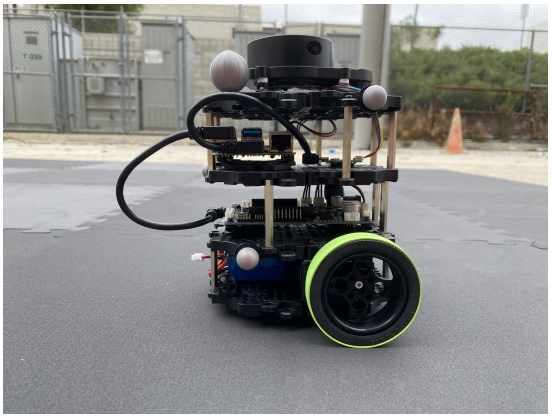
the right hand rule, the positive $z$ axis is defined as coming up from the ground the wand is lying on. The wand must be placed as level with the ground as possible to make the $x-y$ plane as parallel as possible with the ground the robot is to traverse.

## 5.4 Experimental Objective

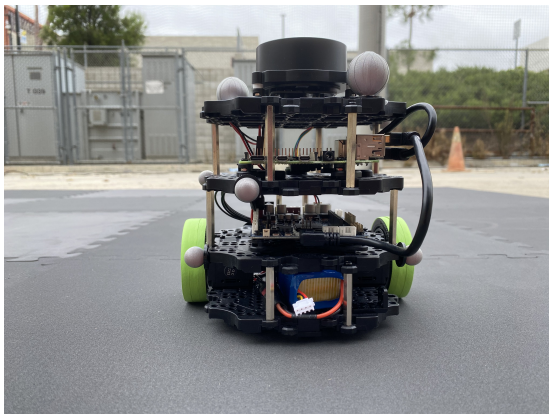
We task a grounded robot agent to navigate in a known environment from an initial state $\mathbf{x}_o$, to a goal state $\mathbf{x}^*$. The agent will attempt to complete this objective safely (i.e. never being in an unsafe state $\mathbf{x} \notin \mathcal{L}^c$) using the same nominal policy $\hat{\pi}(\mathbf{x})$ filtered by a CBF-QP safety-filter with constraints defined by the control limits $u \in \mathcal{U}$ and the current CBF $h(\mathbf{x})$ using the same initial candidate CBF $h_o(\mathbf{x})$ under three different scenarios:



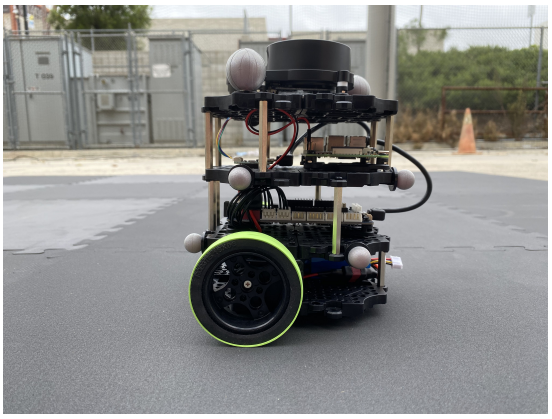
**(a)** Front



**(b)** Right



**(c)** Back



**(d)** Left

**Figure 5.4.** Turtlebot3 Burger With Motion Capture Balls. Balls are manufactured from BL Engineering, and are coated in an IR reflective material which makes the balls easily discernible by the Vicon cameras for near perfect state estimation from the Vicon system. Balls arranged asymmetrically about the center of the robot for the robot's orientation to be easily determined by the system.

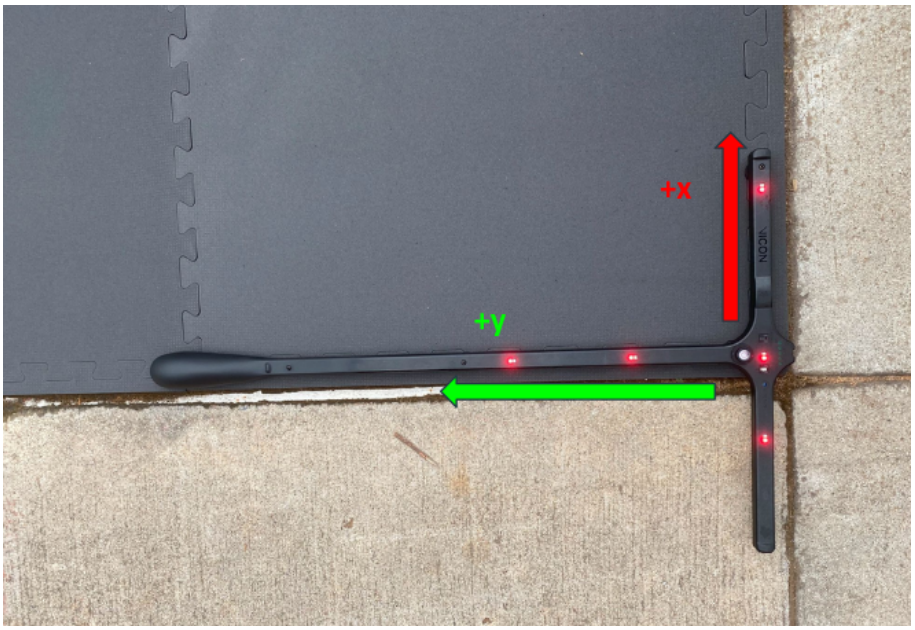



**Figure 5.5.** Vicon Wand Defining The World Origin. Short end of wand determines the x-axis, long handle points in the positive y-axis direction. Following the right hand rule, positive z is up from the ground, in the direction of the IR sensors.

1. Without updating the constraint function $\ell(\mathbf{x})$ and use RefineCBF to iteratively update $h(\mathbf{x})$. **Motivation:** This is to serve as a stress test for the algorithm online. Because the robot cannot ever reach its goal, this allows us to see how well safety can preserved on the boundary of the safe set.

2. Update the constraint function $\ell(\mathbf{x})$ at 5 predetermined Refine CBF iterations $k$, but **do not** use RefineCBF to iteratively update $h(\mathbf{x})$.[3] **Motivation:** This case is to serve as a baseline comparison for the algorithm - a naive approach just taking into account the complement of the obstacles as the safe set.

3. Update the constraint function $\ell(\mathbf{x})$ at 5 predetermined Refine CBF iterations $k$, and use RefineCBF to iteratively update $h(\mathbf{x})$. **Motivation:** A successful navigation to the goal in this scenario would demonstrate that RefineCBF can successfully be used in the loop *and*

[3]Because RefineCBF isn't used for this experiment, $k$ will refer to an artificially incremented $k$, where it is incremented regularly at intervals of 0.5 seconds. This very roughly matches the time $k$ would increment when using RefineCBF.

in an environment with a changing constraint set.[4]

### 5.4.1 Nominal Policy of Choice

To construct the nominal policy for this experiment, we elected to use a precomputed nominal policy table of dimension equal to the dimension of the grid the CBF is discretized into.[5] The table was computed from deriving the optimal control at each point over the grid using reachability based methods as shown in (3.15). This table was saved to the remote PC and loaded upon launch of the nominal controller node. To query the table, the nominal controller node takes the current known state $\mathbf{x}$ and interpolates between grid points if this state does not lie exactly on a grid point.

## 5.5 Experimentals

### 5.5.1 Environment

The robot is to navigate in a $2 \times 2$ meter space where bounding walls lie on the perimeter of the space. The robot's initial pose is:

$$\mathbf{x}_o = \begin{bmatrix} 0.5 \\ 1.0 \\ 0 \end{bmatrix} \tag{5.2}$$

where $x$ and $y$ are expressed in meters and $\theta$ expressed in radians. These units for these state components will remain the same throughout the remainder of this work.

The goal pose of this robot is:

[4] albeit this constraint set must be changing favorably - i.e. updated obstacles do not occupy states of the current safe set

[5] The table's dimensions could have been larger, but computational time of the policy would scale exponentially with diminishing returns.

$$\mathbf{x}^* = \begin{bmatrix} 1.5 \\ 1.0 \\ 0 \end{bmatrix} \tag{5.3}$$

with a tolerance padding of $\pm 0.1$, $\pm 0.1$, and $\pm\pi$ for the $x^*$,$y^*$ and $\theta^*$ states respectively. Therefore, the robot's goal will be a set within a circle of radius 0.1 at any orientation. We will call this set of goal states $\mathcal{G}$. A depiction of the initial starting environment can be seen in Figure 5.5.1. This same environment for the hardware experiment can be seen in Figure 5.3. The simulation initial configuration can be seen in the prior Figure 5.1.

In the hardware experiment, the orange tape represents a grid composed of cells of $1 \times 1$ meter in dimension. The red circle depicts the initial 0-superlevel set of the initial candidate CBF. The green circle depicts the target pose with padding, $\mathcal{G}$. Finally, the light blue lines depict the final obstacle set without padding, $\mathcal{O}_{k=40}$. The bucket in the center is to represent a residual object after crowd of people disperses. Similarly, in the simulation environment, the gray cylinder represents the residual object and the green cylinder the goal pose with padding, $\mathcal{G}$.

Within the environment, the robot is presented with obstacles that make it impossible for the robot to traverse safely to the goal using safety-agnostic controls. For our experiments we attempt to simulate scenarios where the goal is blocked initially by a crowd of people that cover the entire right half side of the space. In scenario 1, this crowd never disperses, thus $\ell(\mathbf{x})$ never updates and the robot can never safely traverse to its goal (see Figure 5.5.1). In scenarios 2 and 3, the crowd disperses and the obstacle set is updated at discrete iterations $k$, where $k$ is the number of iterations the RefineCBF algorithm has underwent.[6] Obstacles are assumed to be static at every iteration, and eventually disperse enough to provide a safe path to the goal.

[6]This was done for ease of implementation, but means that the obstacles could be introduced at different times into an experiment run depending on the $\delta$ chosen. A higher $\delta$ would mean iterations take longer so obstacle updates would also take longer.

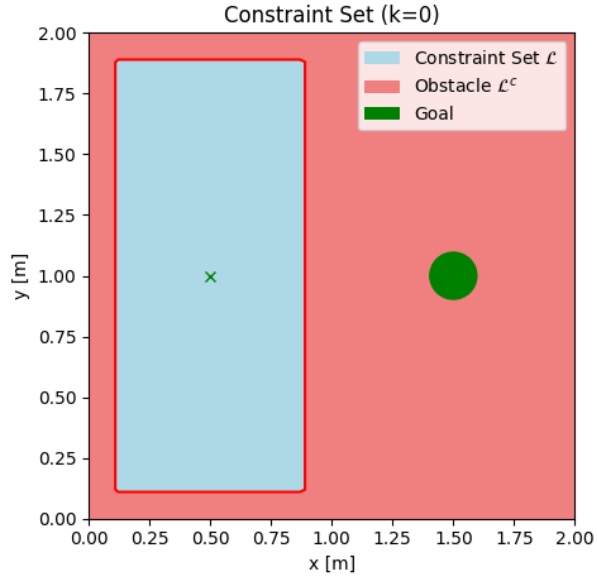



**Figure 5.6.** Initial Constraint Set In Relation to Starting Position and $\mathcal{G}$. In light blue is the constraint set $\mathcal{L}$ for our experiments. It is determined from the 0-superlevel set of the constraint function, $\ell(\mathbf{x})$ (the signed distance function), this is the set of states that would not immediately result in failure. In pink is this sets complement, the failure set $\mathcal{L}^c$ which, because of our choice of $\ell(\mathbf{x})$, is exactly the inflated obstacle set $\mathcal{O}_0^*$. The goal set, $\mathcal{G}$, is represented by the green circle in the right hand side of the environment. The initial position of the robot is denoted by the green x. Notably, the goal is contained within $\mathcal{L}^c$ making it unsafe to traverse to initially.

### Obstacle Inflation

In order to consider the robot as a point traversing through the space, it was necessary to inflate the obstacle set by an appropriate amount to accommodate the maximal radius of the Turtlebot. Using a Minkowski Sum, the obstacle set was inflated by the Turtlebot3's maximal radius plus a small amount of buffer (as shown in (3.3)). We will refer to this obstacle padding value as $p$. Using the maximal radius of 105mm from the Turtlebot3 Burger's specifications (5.7) and a small buffer of 5mm, we define this padding as $p = 0.11$. See Figure (5.8) for visualization of this inflation for the final constraint set of scenarios 2 and 3.

Table 5.3 provides the definition of these sets at the iterations that they are updated and

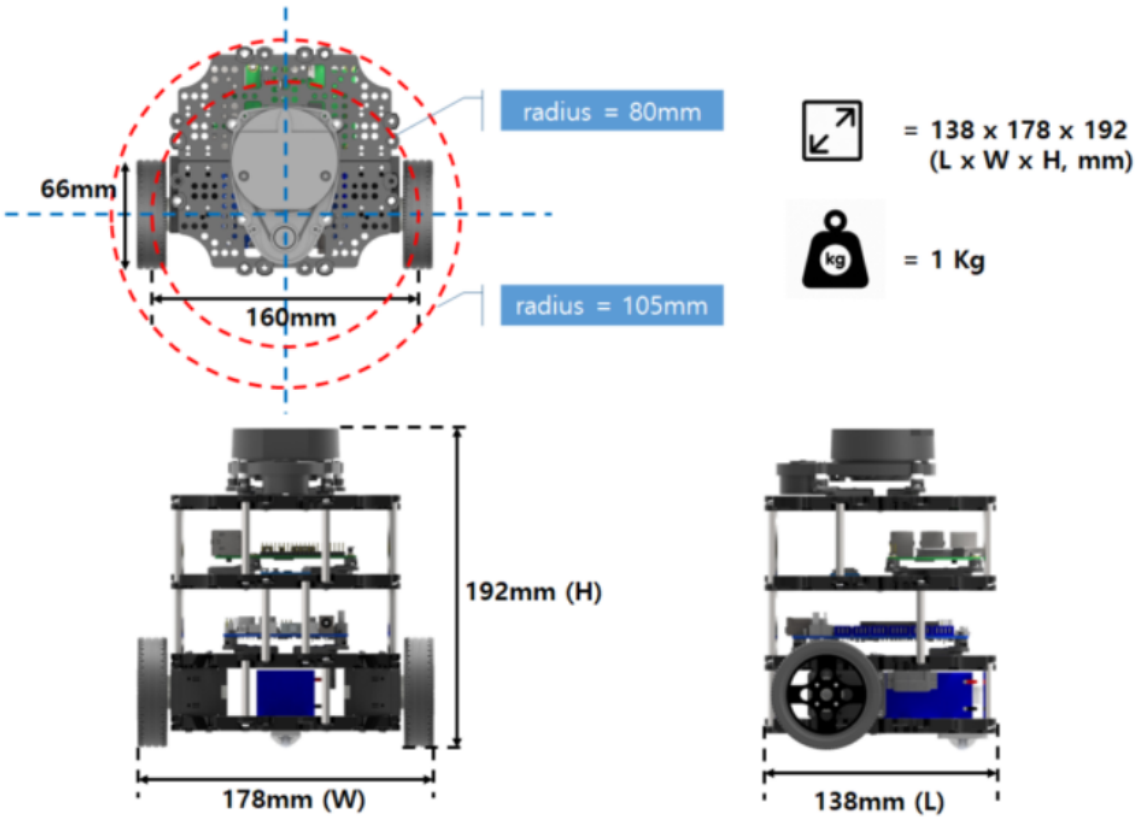



**Figure 5.7.** Turtlebot3 Burger's Dimensions and Weight. The maximal radius of 105mm will be used to compute the padding used to inflate the obstacles.

Figures 5.9 and 5.10 will aid in visualizing what these will look like in the environment. The following section describes what each obstacle set is composed of.

**Obstacle Set Compositions**

Scenario 1, bound to the left hand side of $2 \times 2$ environment:

$$\mathcal{O}_{0\to\infty} = \{(x,y) \mid x \notin [0,1], y \notin [0,2]\}$$

With padding this becomes:

$$\mathcal{O}^*_{0\to\infty} = \{(x,y) \mid x \notin [p,1-p], y \notin [p,2-p]\}$$

**Table 5.2.** Padding $p$ components and final padding value the obstacles will be inflated by.

| **Distances** | **Value** |
|---|---|
| Turtlebot3 Burger Radius (mm) | 105 |
| Buffer (mm) | 5 |
| **Padding ($p$) (mm)** | **110** |

Scenarios 2 and 3, initially bound to the left hand side of the $2 \times 2$ environment but obstacle set updates at discrete $k$ iterations:

The bounding box of the entire environment:

$$R = \{(x,y) \mid x \notin [0,2], y \notin [0,2]\}$$

with padding:

$$R^* = \{(x,y) \mid x \notin [0+p, 2-p], y \notin [0+p, 2-p]\}$$

Initial bounding box:

$$B = \mathcal{O}_{0\to\infty} = \{(x,y) \mid x \notin [0,1], y \notin [0,2]\}$$

with padding:

$$B^* = \mathcal{O}^*_{0\to\infty} = \{(x,y) \mid x \notin [p,1-p], y \notin [p,2-p]\}$$

Circular object:

$$C_1 = \{(x,y) \mid \mathbf{x} \in (x-x_c)^2 + (y-y_c)^2 \leq r^2\}$$

with padding:

$$C_1^* = \{(x,y) \mid \mathbf{x} \in (x-x_c)^2 + (y-y_c)^2 \leq (r+p)^2\}$$

Dispersing people at different $k's$. $C_{2,k}$ and $C_{3,k}$ are the upper and lower circular groups of people respectively:

$$C_{2,10} = \{(x,y) \mid \mathbf{x} \in (x-1.75)^2 + (y-1.75)^2 \leq (0.75)^2\}$$

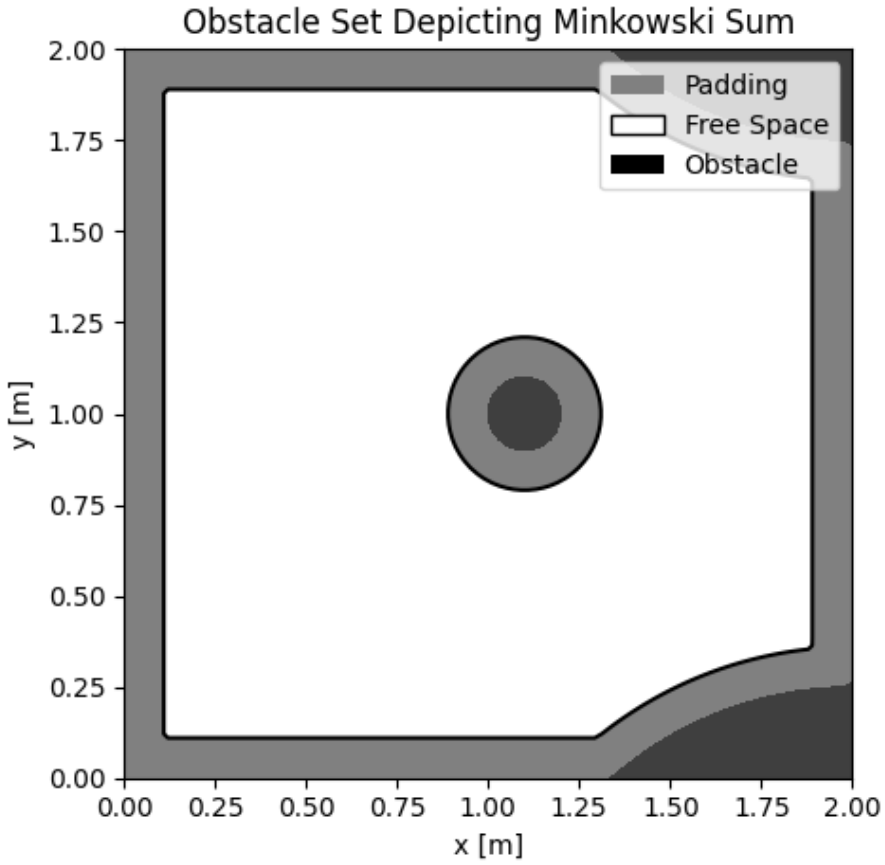



**Figure 5.8.** Scenario 2 and 3 $k = 40$ Inflated Obstacle Sets. The inflation is determined by the Minkowski sum using the Turtlebot3 Burgers radius and some buffer distance.

$$C_{3,10} = \{(x,y) \mid \mathbf{x} \in (x-1.75)^2 + (y-0.25)^2 \leq (0.75)^2\}$$

$$C_{2,20} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y-2.0)^2 \leq (1.0)^2\}$$

$$C_{3,20} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y-0)^2 \leq (1.0)^2\}$$

$$C_{2,30} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y-2.25)^2 \leq (1.0)^2\}$$

$$C_{3,30} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y+0.25)^2 \leq (1.0)^2\}$$

$$C_{2,40} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y-2.25)^2 \leq (1.0)^2\}$$

$$C_{3,40} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y+0.25)^2 \leq (1.0)^2\}$$

with padding:

$$C^*_{2,10} = \{(x,y) \mid \mathbf{x} \in (x-1.75)^2 + (y-1.75)^2 \leq (0.75+p)^2\}$$

$$C^*_{3,10} = \{(x,y) \mid \mathbf{x} \in (x-1.75)^2 + (y-0.25)^2 \leq (0.75+p)^2\}$$

$$C^*_{2,20} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y-2.0)^2 \leq (1.0+p)^2\}$$

$$C^*_{3,20} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y-0)^2 \leq (1.0+p)^2\}$$

$$C^*_{2,30} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y-2.25)^2 \leq (1.0+p)^2\}$$

$$C^*_{3,30} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y+0.25)^2 \leq (1.0+p)^2\}$$

$$C^*_{2,40} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y-2.25)^2 \leq (1.0+p)^2\}$$

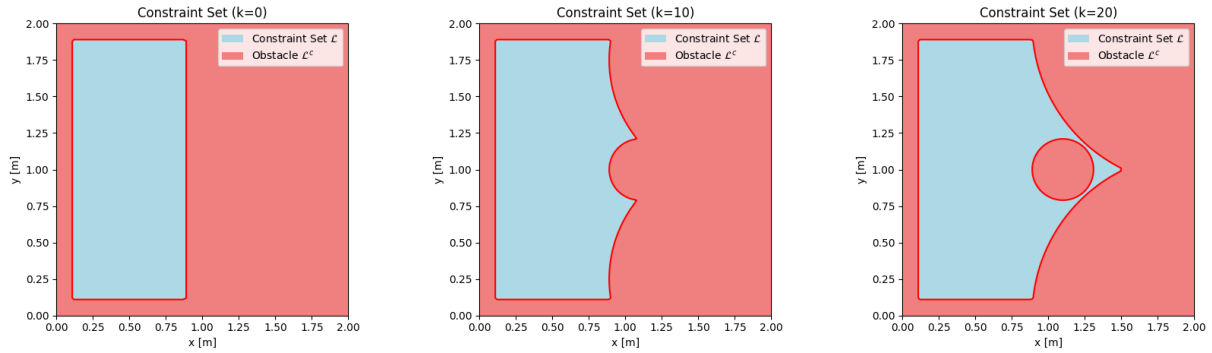
$$C^*_{3,40} = \{(x,y) \mid \mathbf{x} \in (x-2.0)^2 + (y+0.25)^2 \leq (1.0+p)^2\}$$

### 5.5.2 Initial Candidate CBF

Initially, we warmstart the DP recursion of the RefineCBF algorithm using a *conservative* and *unsafe* candidate CBF. Conservative in the sense that the resulting 0-superlevel set of the CBF, $\mathcal{C}_h$, does not cover a large set of states in the environment and unsafe in the sense that we

**Table 5.3.** Obstacle Sets at Iterations $k$. Because the signed distance function is $\ell(\mathbf{x})$, the failure set will be exactly obstacle set. The obstacle sets will be a union of shapes defined above in the Obstacle Set Compositions section. And these obstacle sets will change in their definition based on what iteration of the RefineCBF algorithm is at.

| Iterations | Obstacle Set |
|---|---|
| $k = 0$ | $\mathcal{O}_0^* = B^*$ |
| $k = 10$ | $\mathcal{O}_{10}^* = R^* \cup C_1^* \cup C_{2,10}^* \cup C_{3,10}^*$ |
| $k = 20$ | $\mathcal{O}_{20}^* = R^* \cup C_1^* \cup C_{2,20}^* \cup C_{3,20}^*$ |
| $k = 30$ | $\mathcal{O}_{30}^* = R^* \cup C_1^* \cup C_{2,30}^* \cup C_{3,30}^*$ |
| $k = 40$ | $\mathcal{O}_{40}^* = R^* \cup C_1^* \cup C_{2,40}^* \cup C_{3,40}^*$ |




**(a)** k=0 **(b)** k=10 **(c)** k=20

**(d)** k=30 **(e)** k=40

**Figure 5.9.** Constraint Sets at Different Iterations For Scenarios 2 and 3. $k = 0$ set also represents scenario 1's constraint set for all time. The failure set's evolution is meant to simulate a crowd of people dispersing and leaving behind a residual obstacle represented by the pink circle near the center.

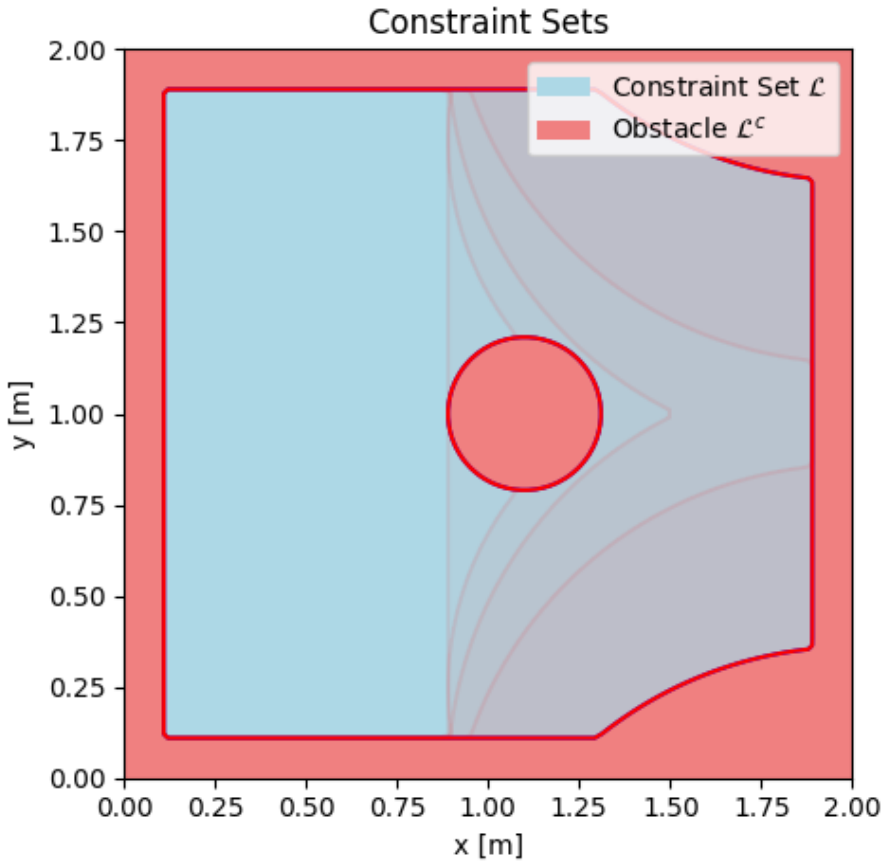



**Figure 5.10.** All Constraint Sets Overlayed. This depicts the evolution of the obstacle sets to show how the people disperse.

know this initial CBF is only a **candidate**. Thus, without refining the safety filtered trajectories would collide with the obstacle set if the defined safe set were close enough to the boundary of $\mathcal{L}$. This initial CBF choice would be akin to a user defining some region in their operating space they know will be largely free of obstacles.

Our formal definition of the initial candidate CBF is as follows:

$$h_o(\mathbf{x}) := 0.33^2 - (x-0.5)^2 - (y-1.0)^2 \tag{5.4}$$

This function describes a circular paraboloid $f(\mathbf{x}) := r^2 - (x-x_c)^2 - (y-y_c)^2$ centered at $(x_c, y_c) = (0.5, 1.0)$ with a radius of $r = 0.33$. This initial CBF is depicted in (Figure 5.11).

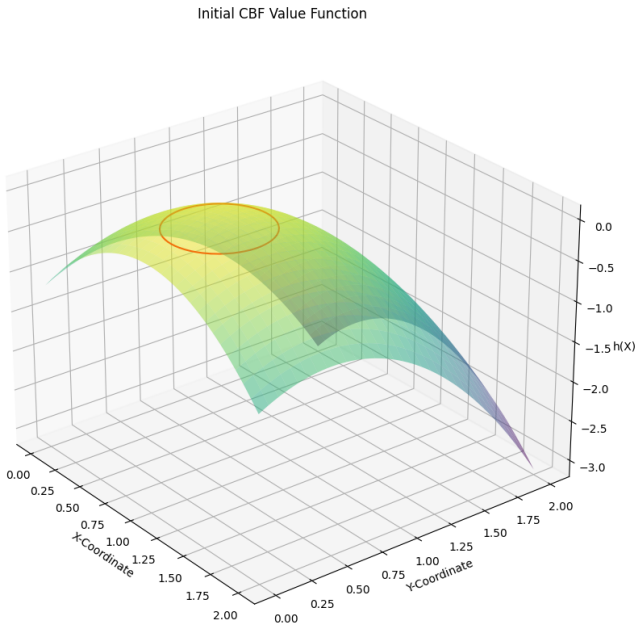



**Figure 5.11.** Initial Control Barrier Function. The 0-levelset of this function is depicted by the red circle, and is centered at (0.5, 1.0) with a radius of 0.33. This initial CBF is conservative and unsafe. It is conservative because it does not allow for traversal of much of the safe parts of the state space when using a safety filter. It unsafe because a circular set will have unsafe states at the edge corresponding to the current heading angle. (see Figure 5.12)

While this initial CBF is only a candidate, it will quickly converge to a valid one. Because the radius of $\mathcal{C}_h$ is greater than the minimal turning radius of the robot when using a linear velocity of 0.1 (the lower bound) and angular velocity of 1.3 (maximum), the robot can certainly stay within the defined circle of $C_{ho}$ for all time - except for a handful of states at a corresponding side of the circle based on the current heading angle. In other words, a similarly sized safe-set will resemble a peach-like shape as shown in figure 5.12. Thus, in very few iterations of RefineCBF this type of shape will be achieved. So, although the initial CBF will be a candidate, we can expect in very few iterations of RefineCBF, the CBF will become valid. Therefore, online, we should expect to see a persistent preservation of safety shortly after initialization.

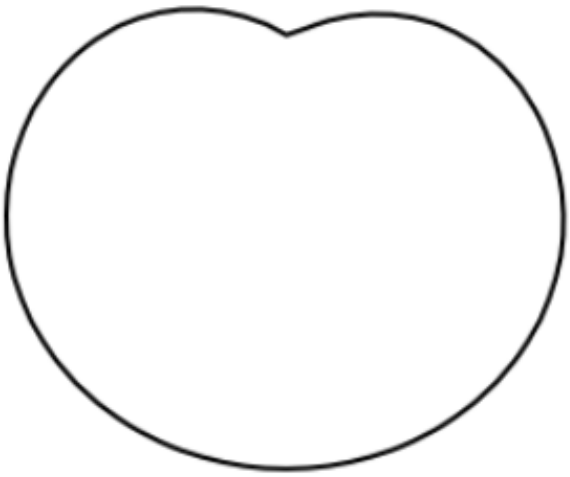


**Figure 5.12.** Valid Control Barrier Function 0-levelset for Differential Drive Robot. Corresponds to a heading angle pointing upwards. We observe the resulting 0-superlevel sets of the refined CBFs take on this shape after very few iterations. Iterations thereon will be guaranteed to be safe because of the preserving safety guarantees of RefineCBF.

### 5.5.3 RefineCBF Term Definitions

From the experiment we can now define the corresponding mathematical terms to be used in the RefineCBF algorithm. Given that the robot's operating environment is a $2 \times 2$ grid, we define the state space as $\mathcal{X} = \{(x, y, \theta) \mid x, y \in [0, 2], |\theta| \leq \pi\} \in \mathbb{R}^3$. Our control space must operate within the limits defined by the limits of the Turtlebot3 Burger (5.1). In order to induce an interesting safe control problem[7], we do not allow the robot to stop and define a minimum linear velocity of 0.1 m/s. Therefore, our control space is given by $\mathcal{U} = \{(v, \omega) \mid v \in [0.1, 0.21], |\omega| \leq 1.3\} \in \mathbb{R}^2$.[8]

[7] If we did not make the minimum speed greater than 0 m/s, the robot could stop instantaneously, making safety a trivial task.

[8] The keen reader will be quick to notice that the maximal bounds in both the linear velocity and do not match those specified as the maximums in Table 5.1. The linear velocity maximum velocity cannot actually reach the specified value on the physical hardware as noted by Robotis developers. While the angular velocity can indeed reach the maximal defined limits, rapidly switching angular velocity controls cause erroneous behavior on hardware. To avoid this it is necessary to restrict the maximal jump in $\omega$ to 2.6 rad/s

Across all experiments, $\gamma = 0.25$ for the safety filter is used. We found this to be a nice middle ground between efficiency and cautious behavior for the system. Finally, the time step chosen for each DP iteration is $\delta = 0.15$ seconds. A longer $\delta$ would result in a larger computation time between each iteration, thus the reactivity to a changing constraint set would be slower. A smaller $\delta$ would result in higher reactivity, but is gated by read/write speeds in our implementation. This time value, after some tuning, seemed to provide a fast update speed without compromising reactivity.

## 5.6 Results

For each scenario, 3 runs were completed for analysis. These scenarios were completed in simulation and on the physical hardware. Multiple runs were done on the simulation because there is still small inherent elements of randomness encoded into each run. For example, computation time can very slightly depending on things such as the current CPU temperature, quickly creating cumulative deviations from a different run.

For each of these runs, there was an accommodating RVIZ[9] video which depicts the trajectory of the robot (in yellow), the current inflated obstacle set $\mathcal{O}_k^*$ (white), the 0-superlevel set of the initial candidate CBF (red) and finally the current 0-superlevel set of the refined CBF at the theta slice of the current heading angle of the robot (green). A keen viewer will notice that even upon convergence of the refined CBF (i.e. the viability kernel), the set will appear to change. This, as mentioned just prior, is caused by the change in heading angle of the robot. See Figure 5.6 for an example of RVIZ plots during a run.

Figures (5.6.2 through 5.6.3) provide visuals of the results for a run of interest for each of the scenarios. All other runs can be found in the appendix.

[9]RVIZ is "ROS" Visualization. It is a software for visualizing ROS topics.

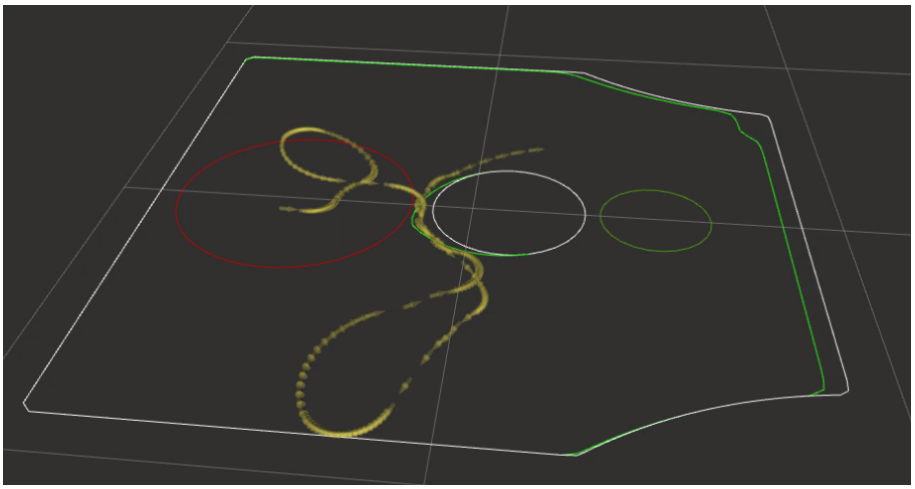


**Figure 5.13.** RVIZ example figure. Red circle represents the 0-levelset of the initial candidate CBF for the experiment. The yellow series of arrows depicts the trajectory of the robot through time during the experiment. The light green line depicts the 0-levelset of the current refined CBF at the current theta slice corresponding to the heading angle of the robot. The dark green circle represents the boundary of the goal set $\mathcal{G}$. The white lines represent the boundaries of the current obstacle inflated obstacle set. Each grid square here is 1 meter by 1 meter. Besides the trajectory, the lines shown are approximate to the real corresponding values.

### 5.6.1 Videos

Due to the heavily qualitative nature of this work, a set of accompanying videos was created for analysis. The associated videos for each scenario and hardware and simulation can be found in the Youtube playlist: https://youtube.com/playlist?list=PL-_gS2FerBEOP_CrxeBosCAPUTwxcKlte

Each hardware experiment is divided into 2 videos, the RVIZ visualization and the actual hardware video in the UCSD Vicon aerodrome.

### 5.6.2 Software Package

The software package developed for this work can be found here: https://github.com/ncussonn/turtwig.

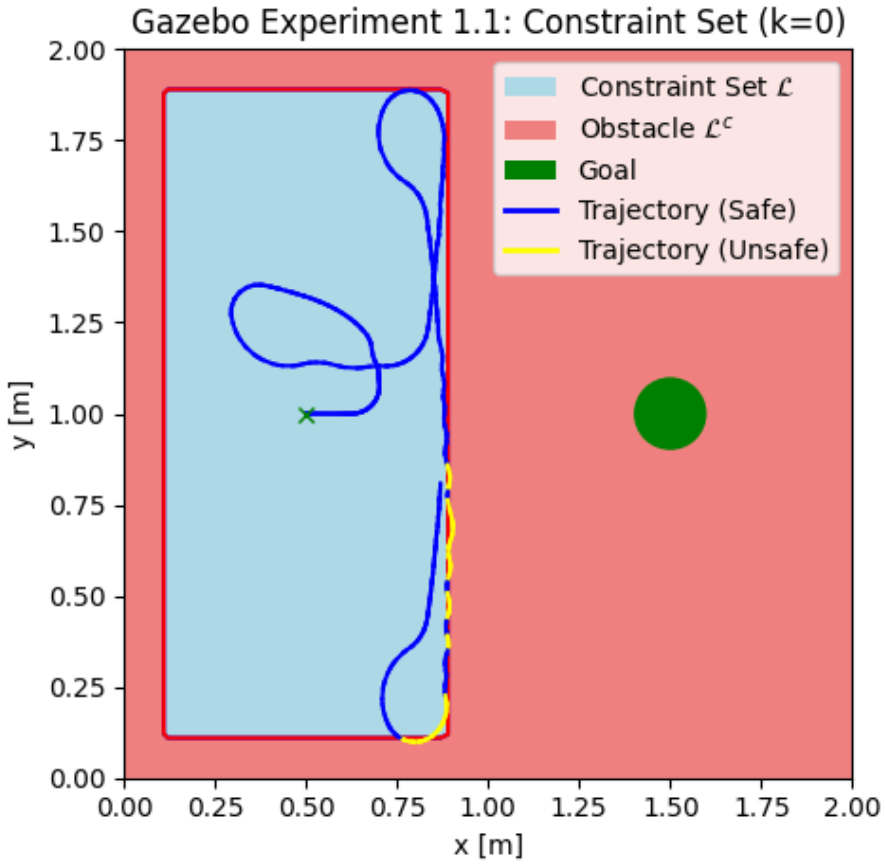



**Figure 5.14.** Gazebo Simulation Trajectory for scenario 1 run 1. The blue line represents a safe trajectory, while the yellow is when the trajectory is unsafe, or in other words, takes on a 0 or negative safety value informed from the CBF $h(\mathbf{x})$.

### 5.6.3 Observations

**Scenario 1**

In scenario 1, during all runs on simulation and hardware we observe the safe set swiftly expanding to take up all space within $\mathcal{L}$ with small backwards reachable tubes at the boundaries corresponding to the current heading angle. The robot also swiftly reaches the closest edge of the safe set to the goal, as the nominal policy attempts to steer the robot towards the goal. However, because the goal is completely obstructed, the robot cannot safely progress to the goal and the

safety filter steers it to the left or right. From here, the robot would bounce off the barrier closest to the goal, slightly violating safety at each bounce. This would continue until the robot reached the wall on the top or bottom of the state space. At this point it would turn around and repeat the barrier bounce.

While not surprising the robot did not reach the goal in this scenario, it is nonetheless important to highlight the importance of how RefineCBF must be integrated with an update to the constraint set to extract its maximal utility. Additionally, the repeated violations to safety here are of note - as theoretically this should not occur.

**Scenario 2**

With updates to the obstacle set now being acknowledged by the robot, it is now able to steer to the goal. However, while we see the robot traverse to the goal eventually, it frequently violates safety. In both simulation and on hardware we observe this behavior. Additionally, the hardware experiments exhibit greater safety violations than the simulation.

This scenario illustrates why simply using the SDF for a safety constraint is not a viable solution to ensure safety. The SDF will *always* be a candidate CBF and thus cannot ensure forward set invariance of its 0-superlevel set. Because of this, it highlights the importance of needing alternative methods to adjust the safe set depending on environmental changes while also remaining safe.

**Scenario 3**

Now combining both RefineCBf and updating the constraint set, in theory the robot should be able to reach to the goal with minimal safety violations. And in fact, this is what we observe. For both the simulation and hardware experiment, the trajectories can[10] safely navigate to the goal region even in the face of a changing constraint set. As such, we can solidly conclude that RefineCBF can be used for hardware in-the-loop and in the face of a changing constraint set!

[10] In some runs on hardware, we see slight violations to safety (see Figure (5.6.3)) This will be discussed in the conclusion.

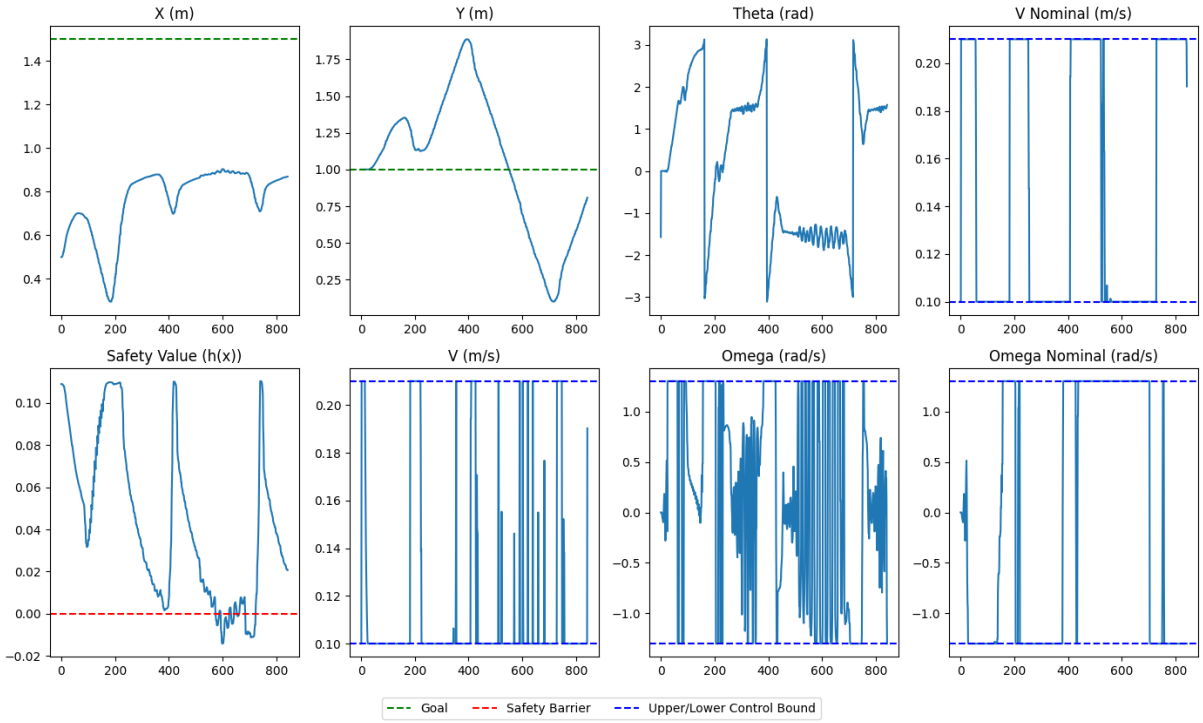



**Figure 5.15.** Parameters for scenario 1 run 1 on hardware. Depiction of all of the parameters for the experiments in time. From left to right, top to bottom: X is the x position of the robot (in meters), Y is the y position of the robot (in meters), Theta is the heading angle of the robot in radians which loops at $\pm 2\pi$, V Nominal is the safety-agnostic linear velocity control given to the robot in meters per second where the upper and lower bounds are shown with a dashed blue line. Safety Value h(x) is the level of safety based on the current state where the red dashed line represents where safety is 0, V is the safety filtered linear velocity with the same bounds as V Nominal, Omega is the safety filtered nominal angular velocity in radians per second with upper and lower bounds shown by the blue dashed line, and finally, Omega Nominal is the safety-agnostic safety filtered control with the same bounds.

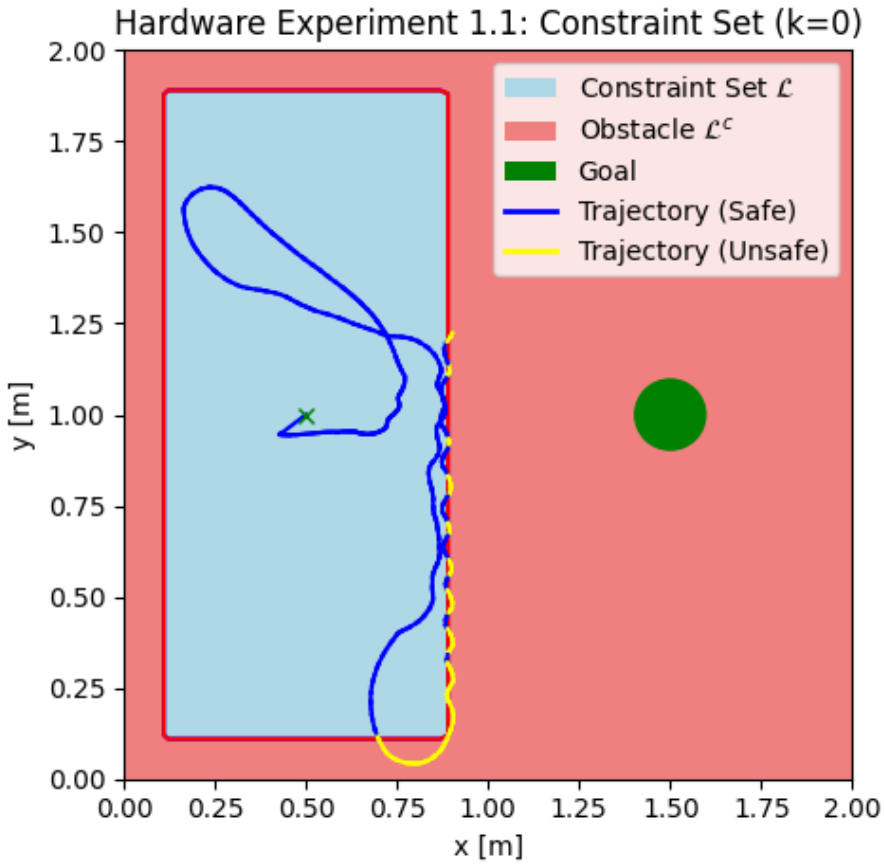



**Figure 5.16.** Hardware Trajectory for scenario 1 run 1. See figures (5.6.2) and (5.5.1) captions for full description of visuals in image.

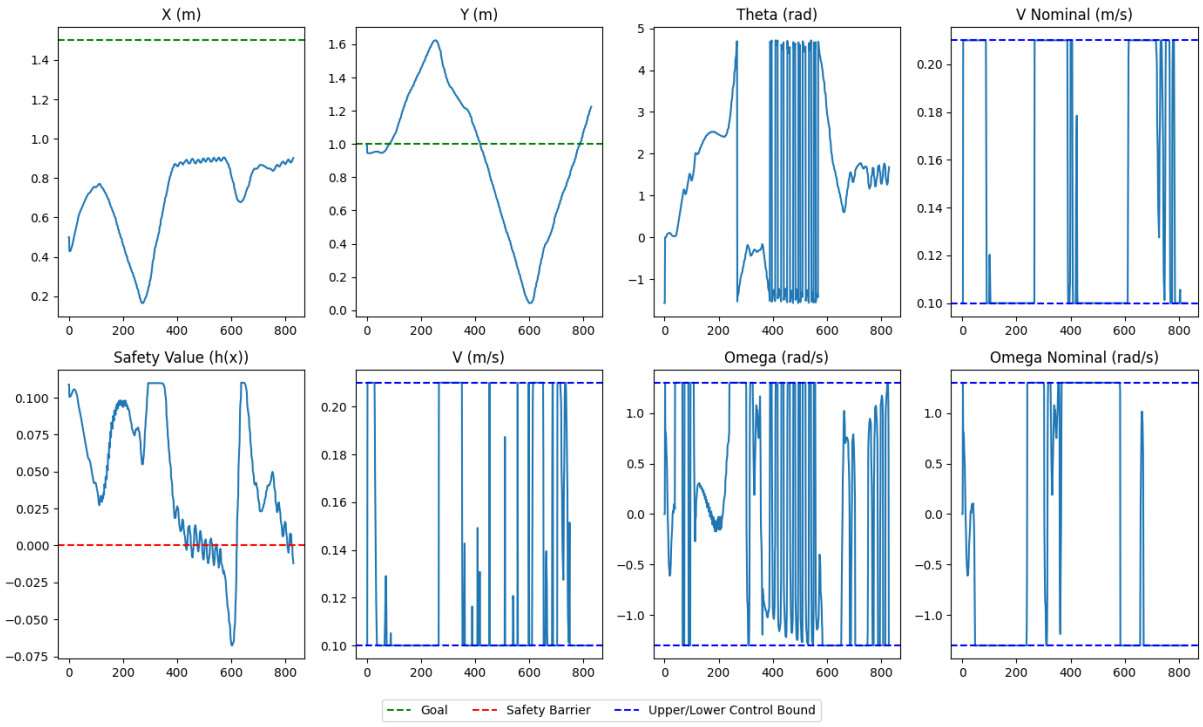



**Figure 5.17.** Parameters for scenario 1 run 1 on hardware. See the caption of Figure (5.6.2) for general description of graphs. Notable differences from the simulation graphs is that the Theta value loops at different values due to intricacies associated with quaternion conversions when receiving information from the Vicon system.

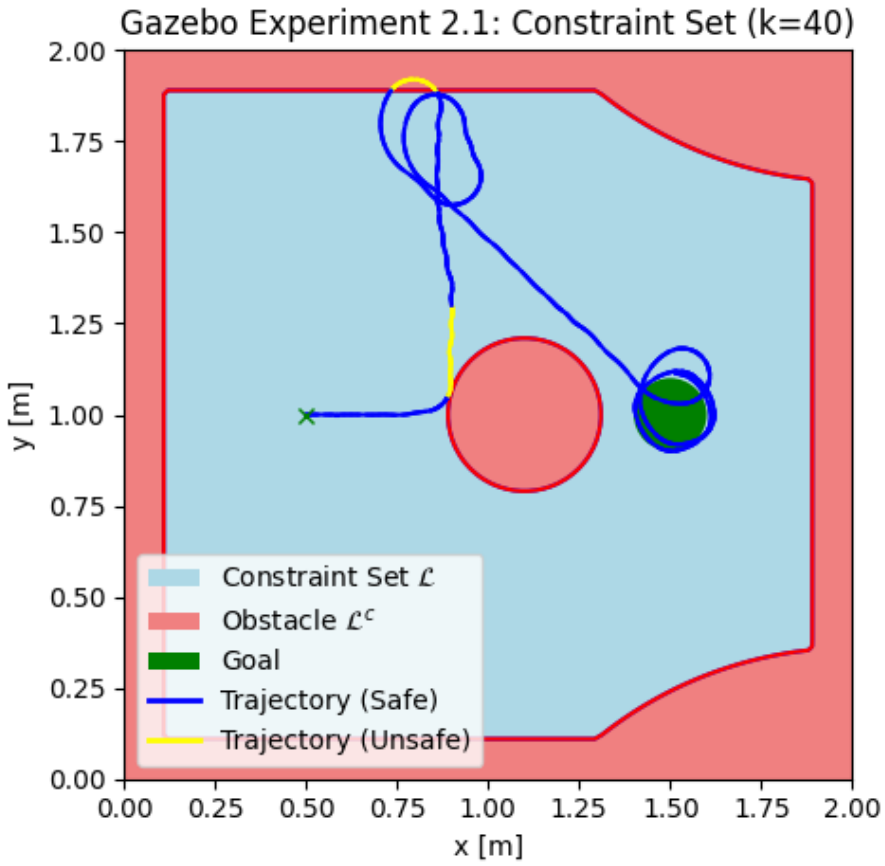



**Figure 5.18.** Gazebo Simulation Trajectory for scenario 2 run 1. See figures (5.6.2) and (5.5.1) captions for full description of visuals in image.

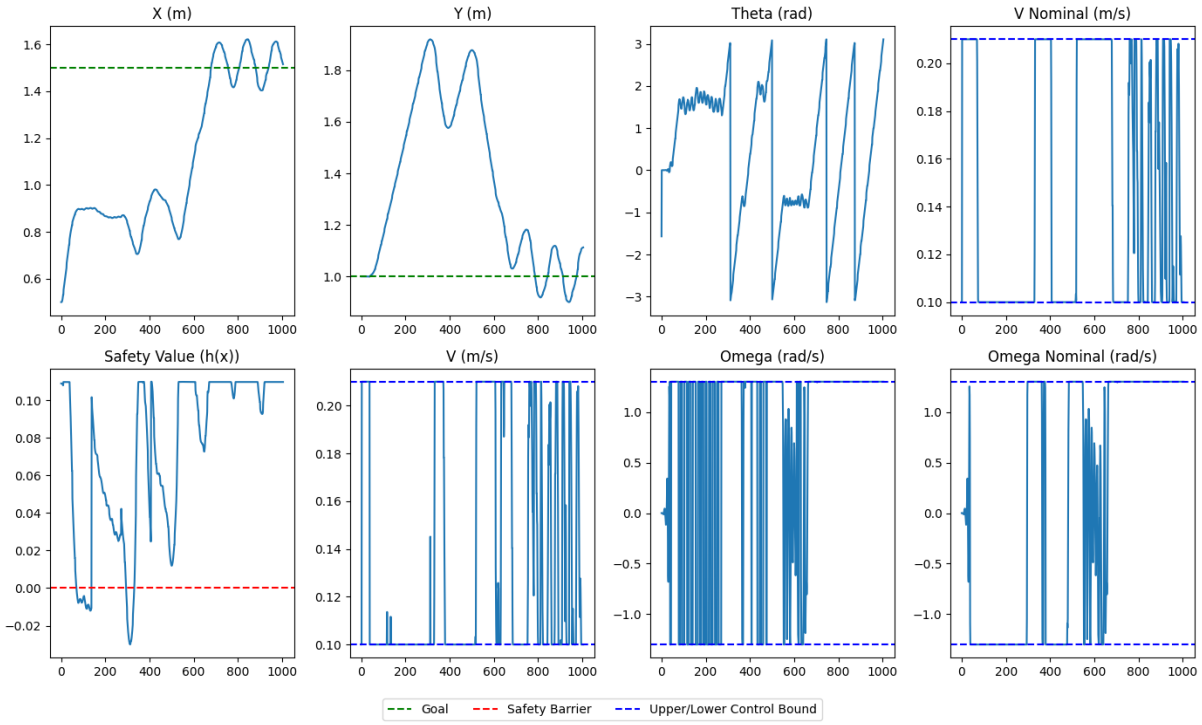



**Figure 5.19.** Parameters for scenario 2 run 1 in Gazebo simulation.

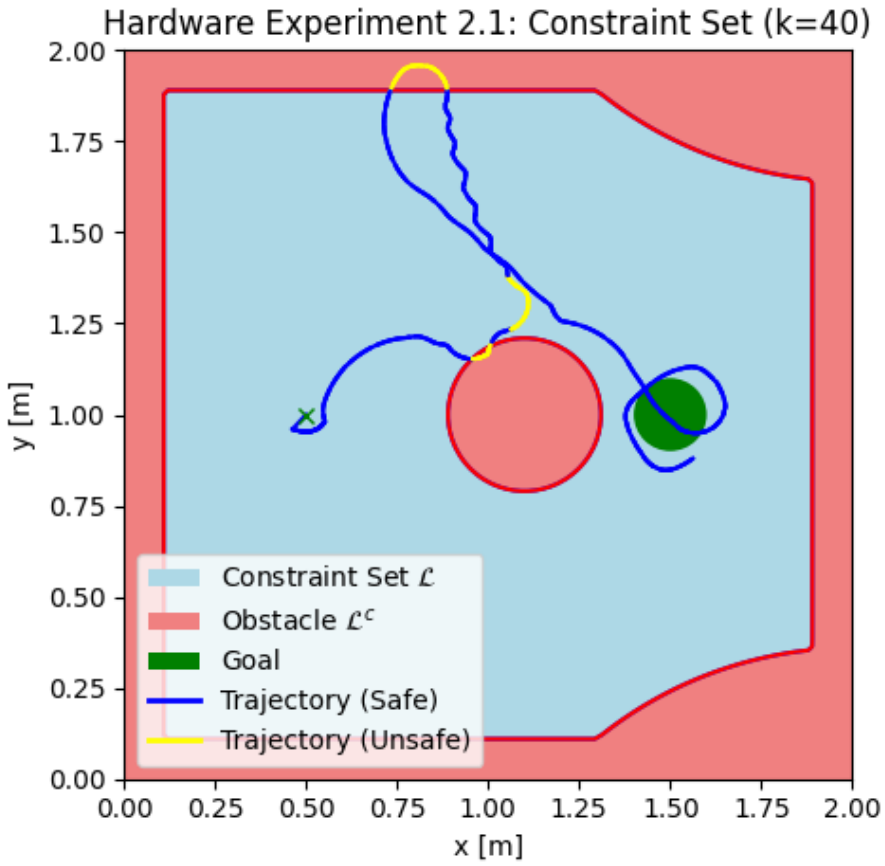



**Figure 5.20.** Hardware Trajectory for scenario 2 run 1. See figures (5.6.2) and (5.5.1) captions for full description of visuals in image.

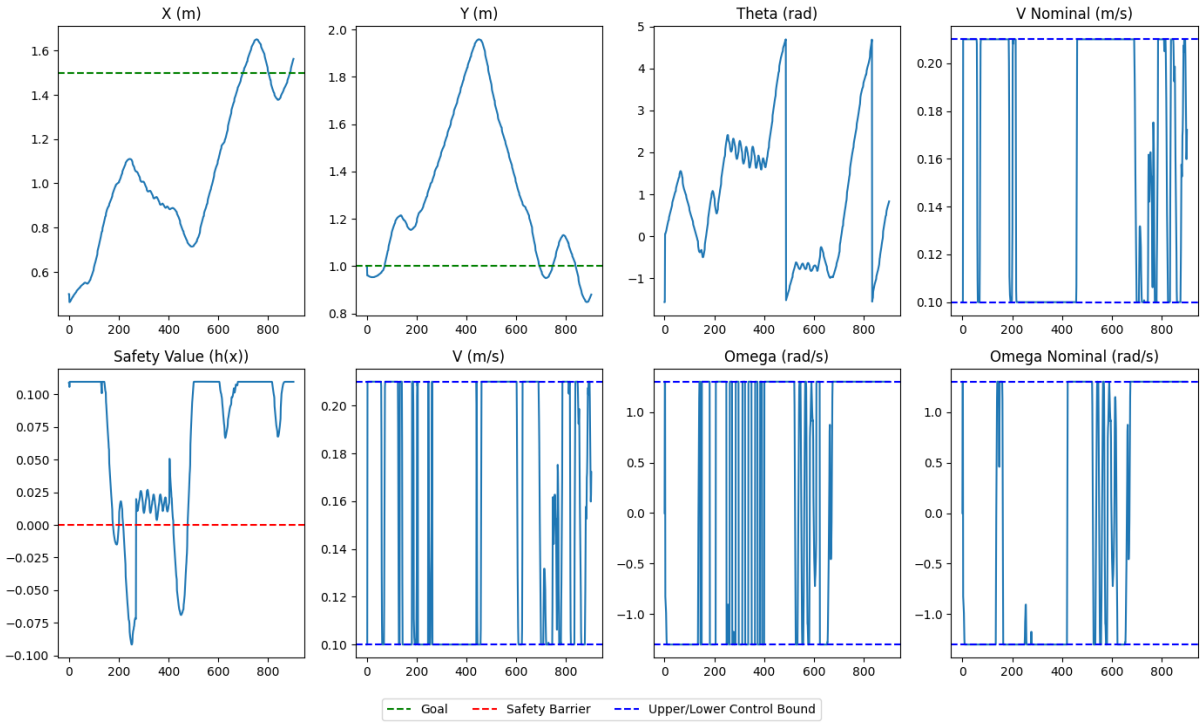



**Figure 5.21.** Parameters for scenario 2 run 1 on hardware.

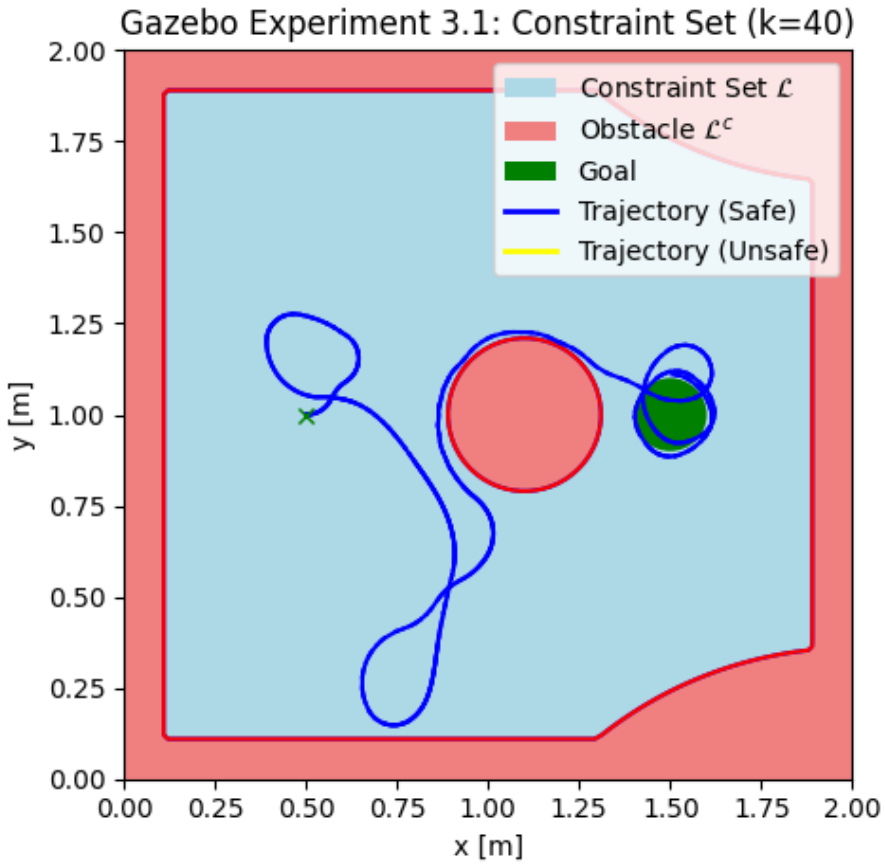



**Figure 5.22.** Gazebo Simulation Trajectory for scenario 3 run 1. See figures (5.6.2) and (5.5.1) captions for full description of visuals in image.

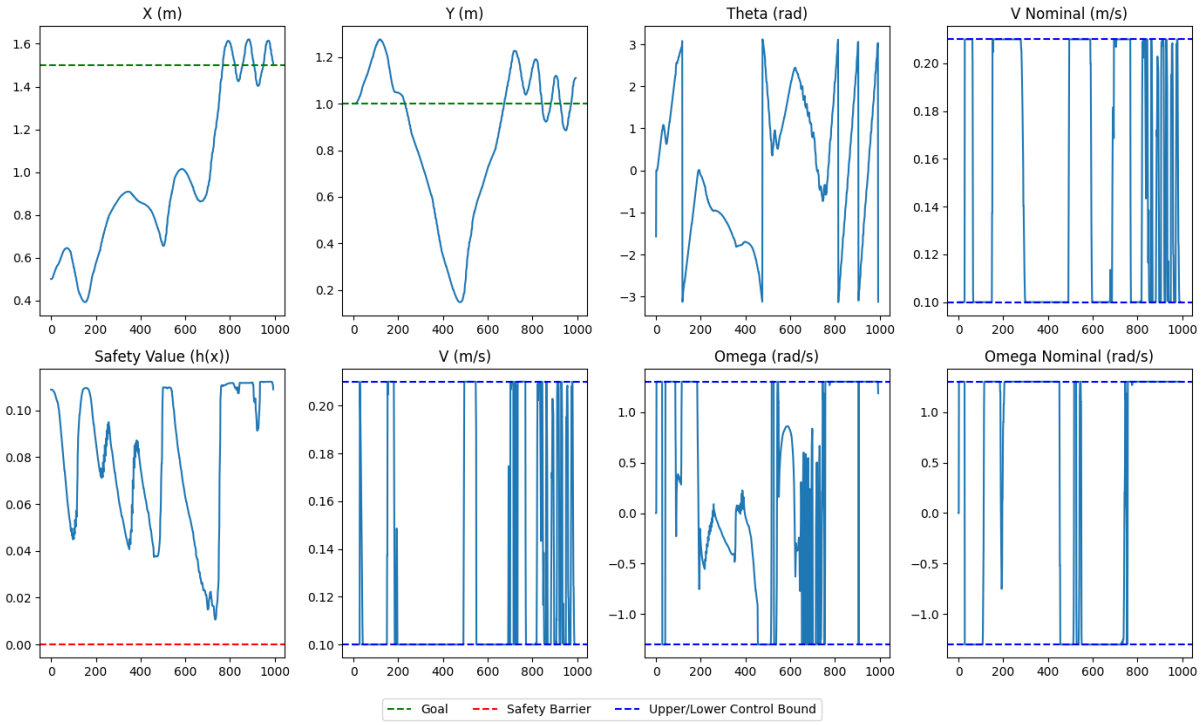



**Figure 5.23.** Parameters for scenario 3 run 1 in simulation. See the caption of Figure (5.6.2) for general description of graphs.

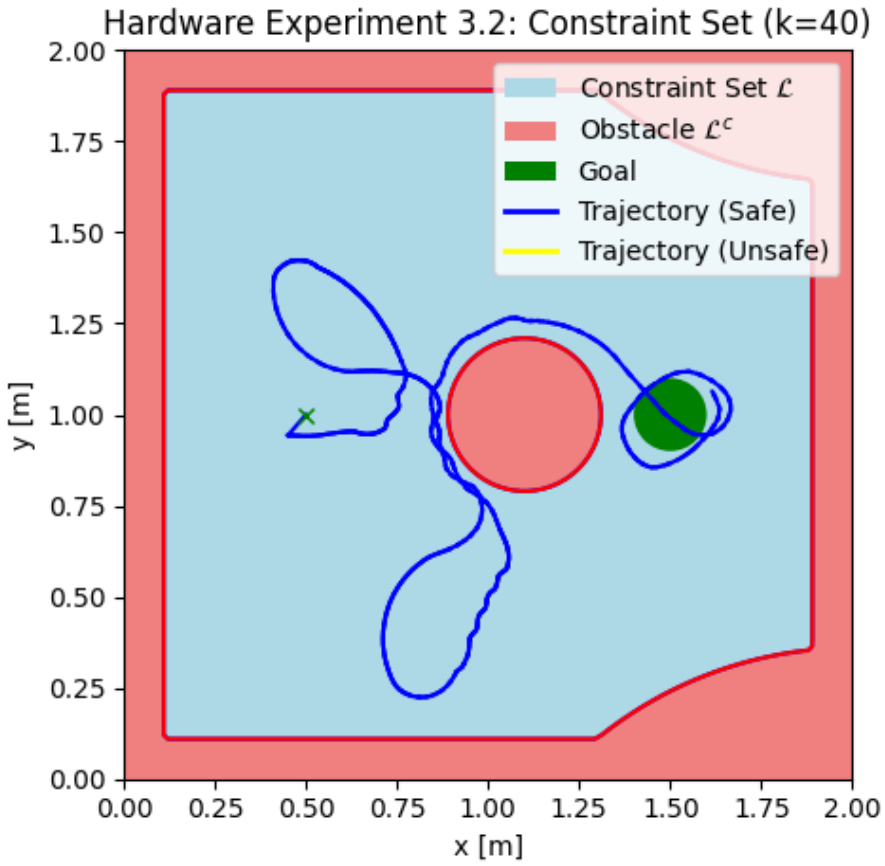



**Figure 5.24.** Hardware Trajectory for scenario 3 run 2. See figures (5.6.2) and (5.5.1) captions for full description of visuals in image.

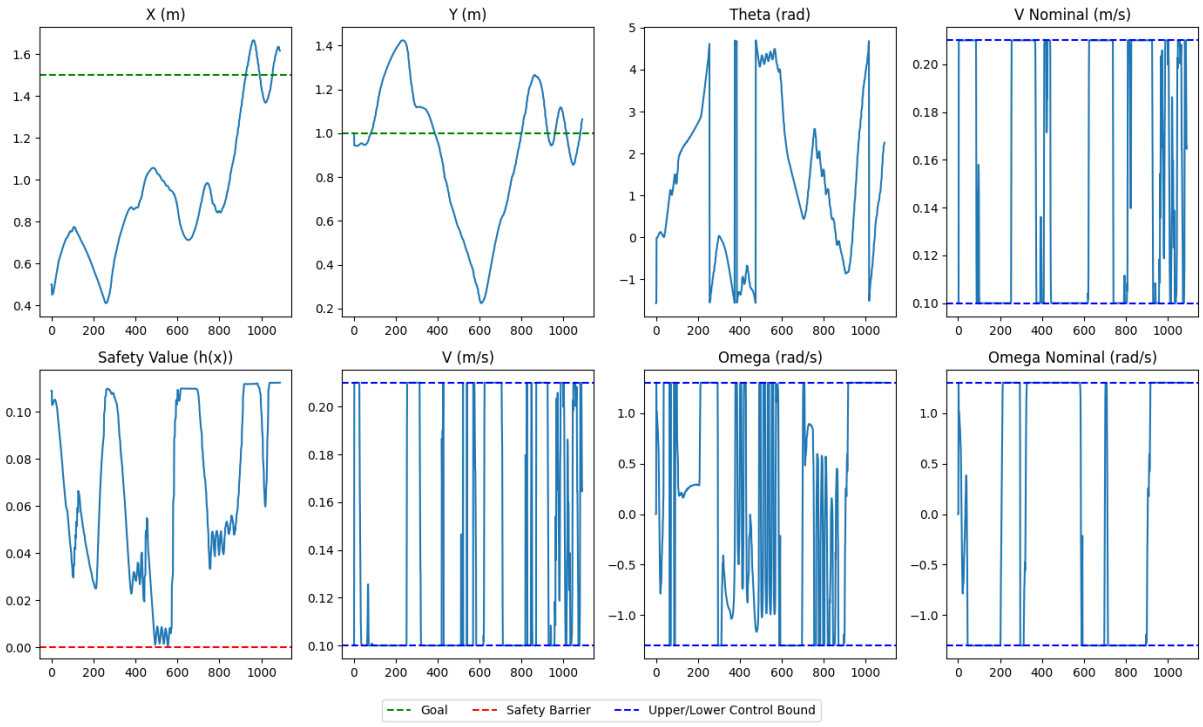



**Figure 5.25.** Parameters for scenario 3 run 1 on hardware. See the caption of Figure (5.6.2) for general description of graphs. Notable differences from the simulation graphs is that the Theta value loops at different values due to intricacies associated with quaternion conversions when receiving information from the Vicon system.

# Chapter 6

# Discussion and Conclusion

## 6.1 Discussion

### 6.1.1 Issues

The primary issue encountered with this implementation was evident during scenarios 1 and certain runs of scenario 3. During these scenarios, we observed repeated safety violations at the edge of the safe set with oscillatory behavior. This behavior we believe could be attributed to a few factors for both simulation and hardware:

1. Time delays in the inputs caused by latency in the system.
2. Numerical inaccuracies as the system trajectory gets too close to the safe set boundary.
3. Grid density. With a denser grid, the accuracy of the interpolation of the current safety value would also increase. In turn, this would allow for more accurate safety preserving controls.

and for strictly hardware:

1. Additional time delays associated with the hardware experimental setup.
2. A mismatch between the differential drive model and the real dynamics of the Turtlebot3 Burger.

### 6.1.2 Time Delays

Most of the aforementioned issues cannot be addressed without causing some sort of other issue. For instance, the numerical inaccuracies and grid density could be addressed through means which would require additional computational power, but this would increase the time between CBF computations which is not ideal. Possibly, the model of the robot could have been improved upon, but, besides being a nontrivial thing to improve upon, if it involves adding more states to the space to get a closer model to reality, this could also pose a computationally problematic issue. So this leaves us with time delays.

The time delays in the control input are a practical challenge for any theoretical to pragmatic implementation of a control problem. It is almost impossible for state information to be updating instantaneously as is assumed in theory. In both the simulation and on hardware, this delay in state information is present. In Gazebo, although minute, the state information was not obtained instantaneously, it only updated at a frequency of 20Hz. While fast, this is still enough time to partake in unsafe control actions at the boundary of the safe set, as any slight deviation in the current heading angle would steer the system towards unsafety. This issue is made even worse on hardware, where there are 6 contributors to latency between the observed state and the control action based on that state being executed. These contributors are:

1. The Vicon system processing the robot's pose by the camera information.
2. The state information being sent from the router to the Remote PC wirelessly.
3. The safety-preserving control action being computed based on the received state.
4. Sending this control action back to the router wirelessly.
5. The router wirelessly sending the control action to the Turtlebot3 Burger.
6. The Turtlebot3 Burger interpreting this control action and performing it.

So clearly, there is much more at play with the hardware experiment in terms of latency. While circumventing this issue is not possible with this current implementation of RefineCBF, potential ways to address this problem will be discussed in the Future Directions section 6.2.1.

## 6.2 Conclusion

In this work, we have verified the applicability of RefineCBF in real-time hardware setting. By employing a Turtlebot3 Burger as the base hardware platform and packaging the RefineCBF algorithm in a ROS2 software package, we were able to demonstrate an enforcement to safety online with minimal safety violations. Using a CBF constraint based safety-filter whose constraints were updated dynamically from RefineCBF, we have also shown that in real-time the algorithm can be robust to updates in its environment and remain safe.

### 6.2.1 Future Directions

However, this implementation is nowhere near perfect, and there are seemingly endless future research directions to go in from here. A few of the most interesting are mentioned here:

1. Test RefineCBF on higher dimensional systems such as Ackermann Drive (4D) or quad-copter (6D+). This would stress test how the slower iteration speed will effect the efficiency of the system.

2. Addressing time-delays. Work like that from [11] provides safety guarantees for CBF based safety-critical system in the presence of time-delays. By incorporating their work in the control loop, it may be possible to mitigate the latency issue mentioned prior. This would require to model the time-delay on the system and using a state predictor.

3. Update the constraint set using real sensor information, rather than artificial obstacles. This would present a more realistic scenario to test RefineCBF in the loop with.

4. Validate RefineCBF's robustness in online scenarios where the dynamics, control space, and disturbance change during operation.

5. Present the robot with adversarial obstacles. In these series of experiment we only introduce neutral or beneficial obstacle changes. In other words, the obstacles will either keep the viability kernel the same size and shape, or allow it to expand. However, if an obstacle were to move into the safe set it would be interesting to see how safety could be recovered. Clearly, if the obstacle was too adversarial and moved on top of the current system state, there is no preserving safety, but nonetheless through some leeway built into the constraint function and the inherent reactivity of RefineCBF (i.e. the new viability kernel will be converged to), safe navigation is still possible.

We are excited to see where this core implementation is expanded upon and utilized in the future, and hope this work can be used to facilitate research for a safer autonomous world.

# Bibliography


[1] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J. Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances, 2017.

[2] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications, 2019.

[3] Jason J. Choi, Donggun Lee, Koushil Sreenath, Claire J. Tomlin, and Sylvia L. Herbert. Robust control barrier-value functions for safety-critical control, 2021.

[4] Sander Tonkens and Sylvia Herbert. Refining control barrier functions through hamilton-jacobi reachability, 2022.

[5] Andrea Bajcsy, Somil Bansal, Eli Bronstein, Varun Tolani, and Claire J. Tomlin. An efficient reachability-based framework for provably safe autonomous navigation in unknown environments. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 1758–1765, 2019.

[6] Sylvia Herbert, Jason J. Choi, Suvansh Sanjeev, Marsalis Gibson, Koushil Sreenath, and Claire J. Tomlin. Scalable learning of safety guarantees for autonomous systems using hamilton-jacobi reachability, 2021.

[7] Zhichao Li and Nikolay Atanasov. Governor-parameterized barrier function for safe output tracking with locally sensed constraints. *Automatica*, 152:110996, 2023.

[8] Sylvia Herbert. *Safe Real-World Autonomy in Uncertain and Unstructured Environments*. PhD thesis, EECS Department, University of California, Berkeley, Aug 2020.

[9] Sylvia L. Herbert, Shromona Ghosh, Somil Bansal, and Claire J. Tomlin. Reachability-based safety guarantees using efficient initializations, 2019.

[10] Morgan Quigley. Ros: an open-source robot operating system. In *IEEE International Conference on Robotics and Automation*, 2009.

[11] Imoleayo Abel, Miroslav Krstić, and Mrdjan Janković. Safety-critical control of systems with time-varying input delay∗∗this work was supported by the national science foundation and ford motor company. *IFAC-PapersOnLine*, 54(18):169–174, 2021. 16th IFAC Workshop on Time Delay Systems TDS 2021.

# Appendix A

# Additional Result Figures

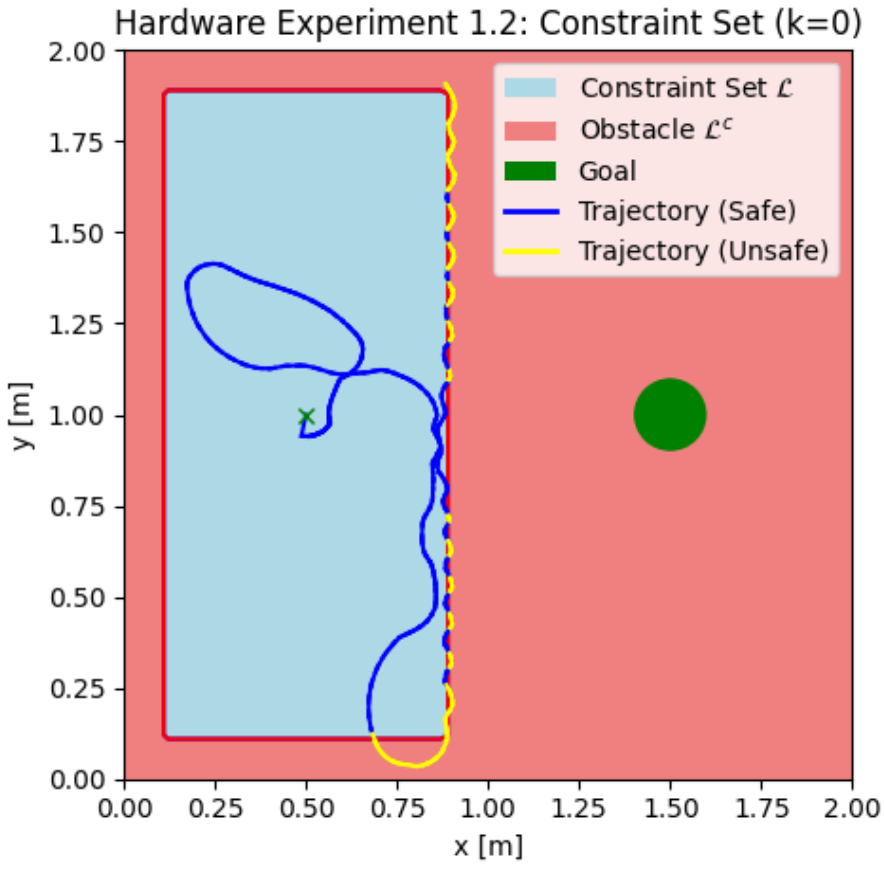



**Figure A.1.** Hardware Trajectory for scenario 1 run 2.

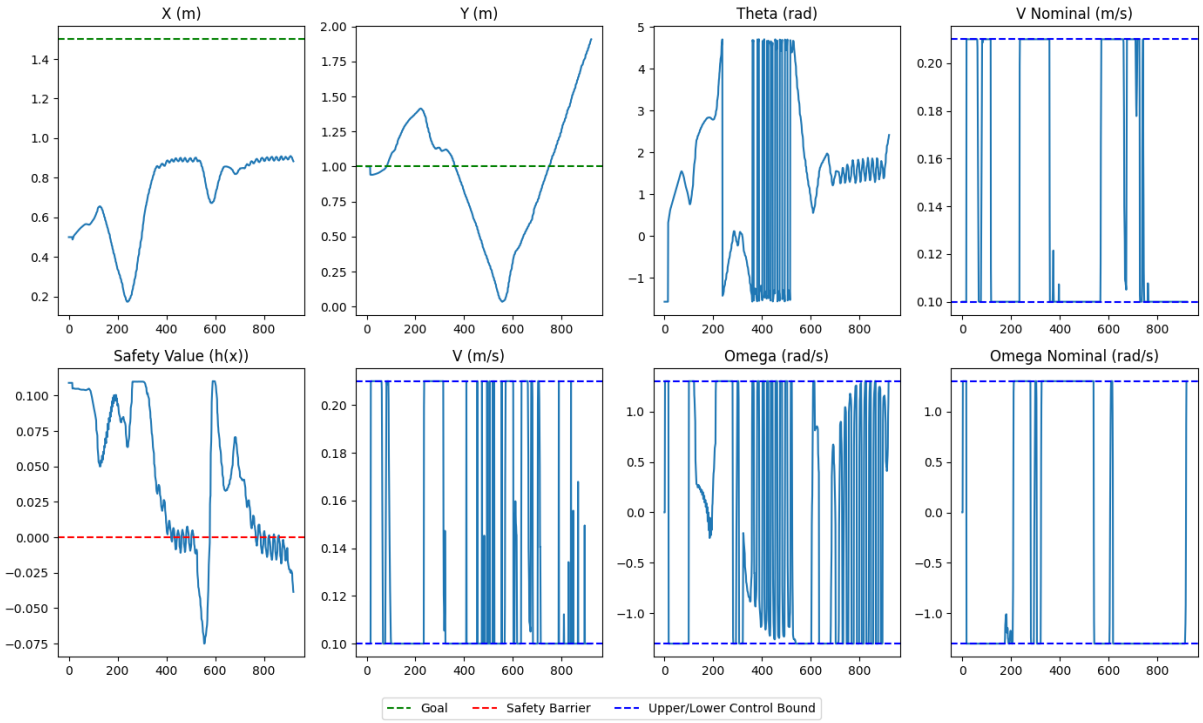



**Figure A.2.** Hardware for scenario 1 run 2 on hardware.

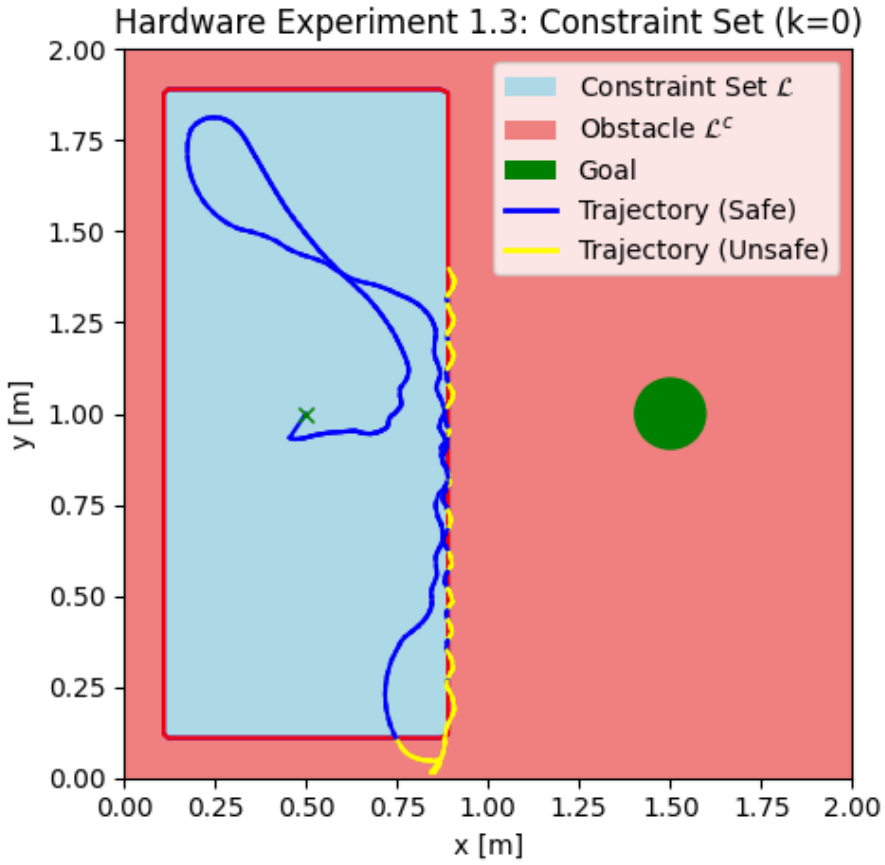



**Figure A.3.** Hardware Trajectory for scenario 1 run 3.

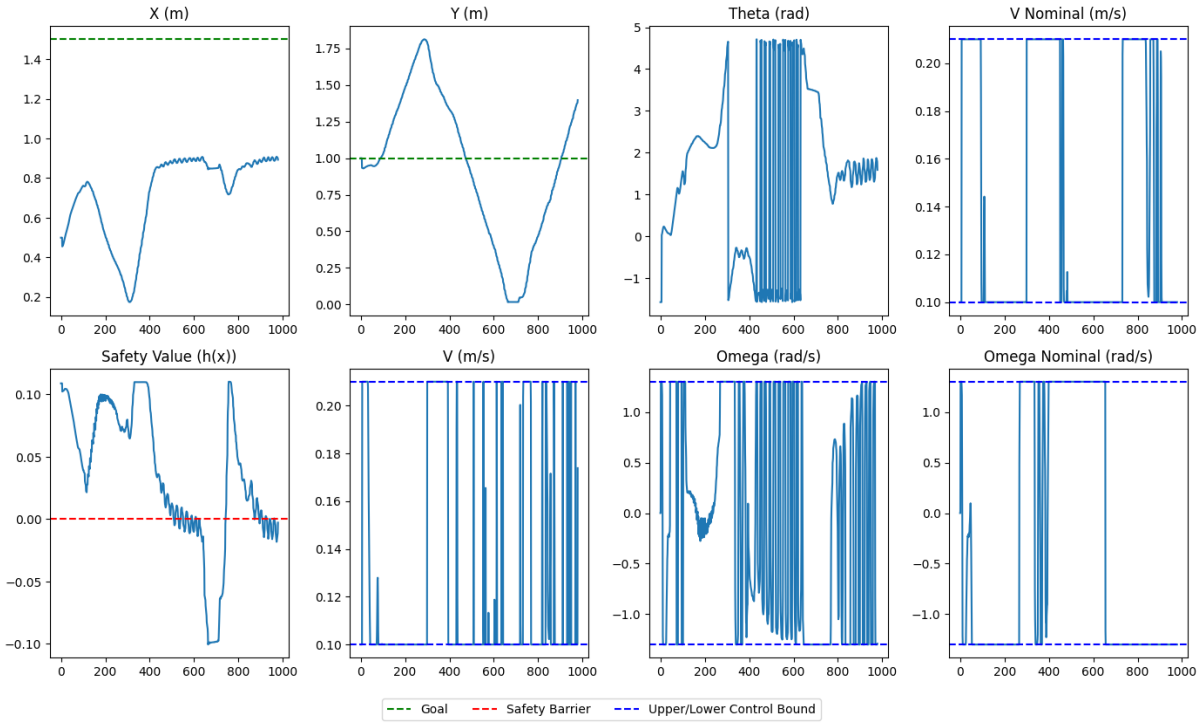



**Figure A.4.** Parameters for scenario 1 run 3 on hardware.

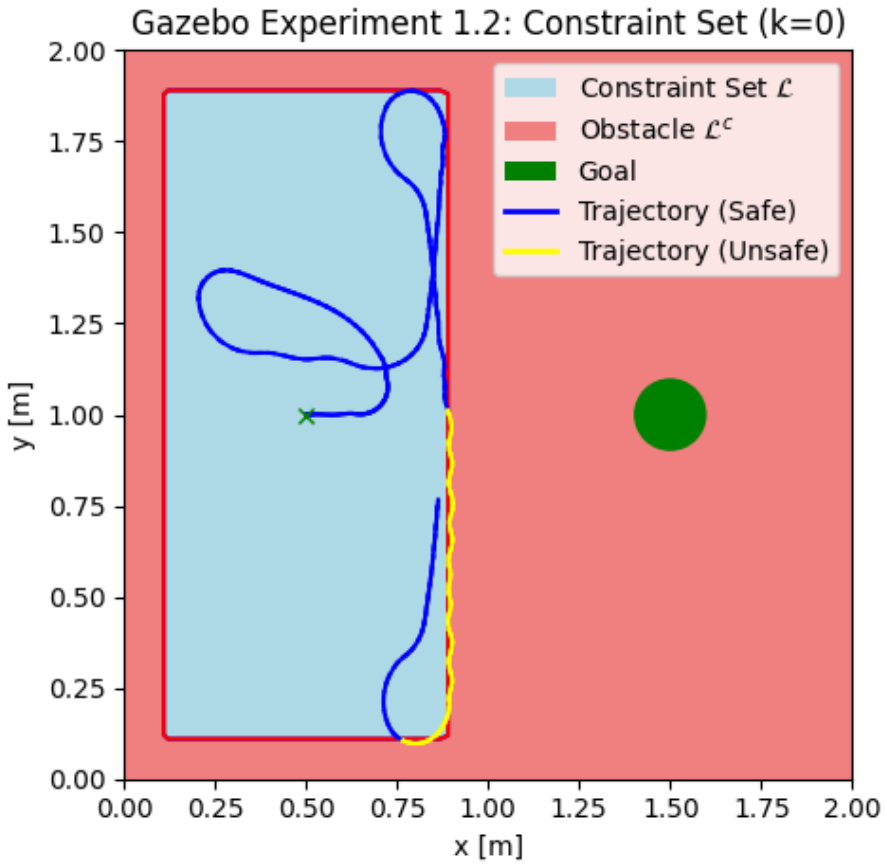



**Figure A.5.** Simulation Trajectory for scenario 1 run 2.

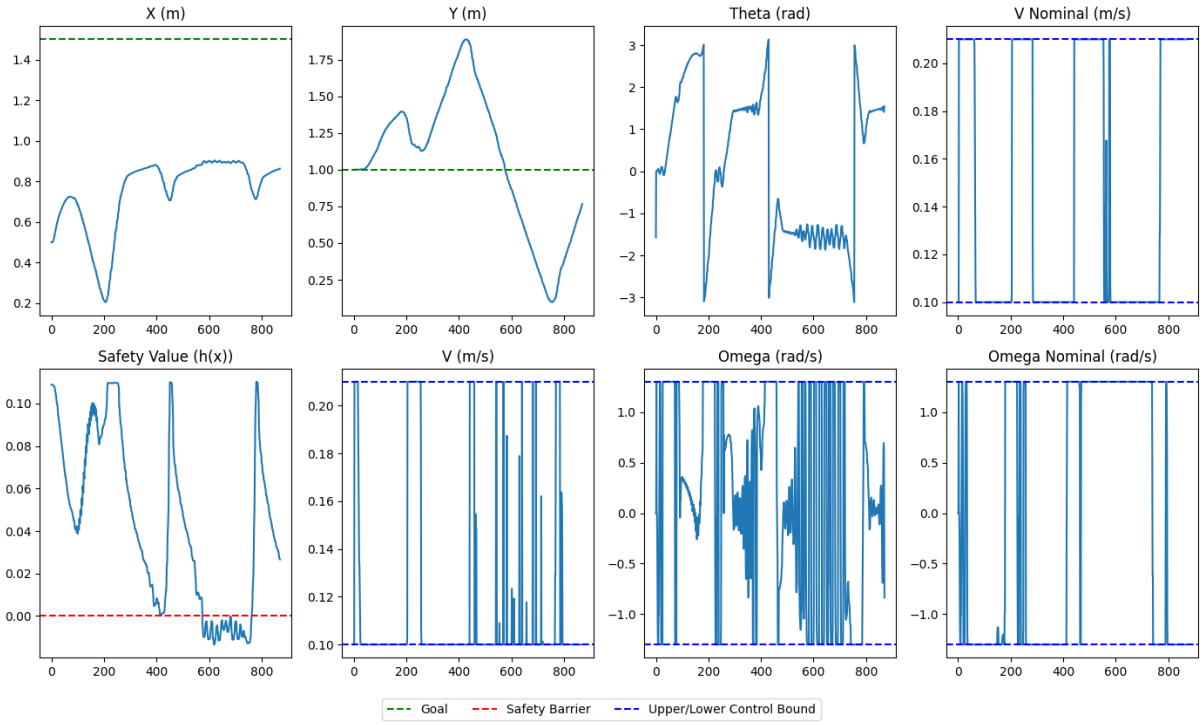



**Figure A.6.** Parameters for scenario 1 run 2 in simulation.

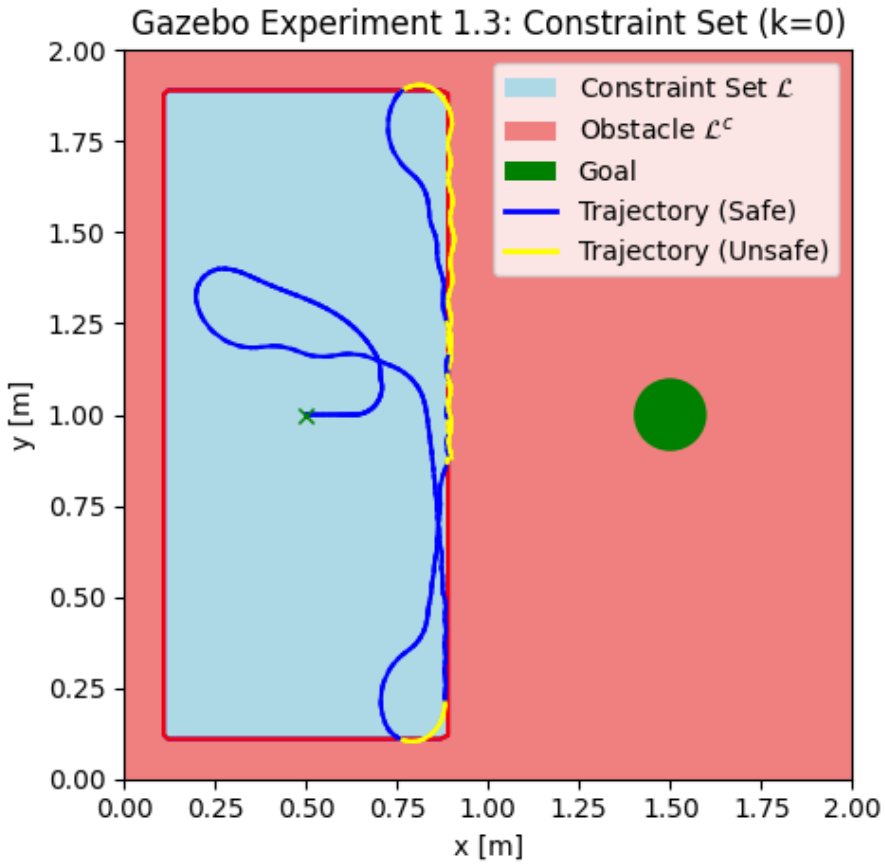



**Figure A.7.** Gazebo Simulation Trajectory for scenario 1 run 3.

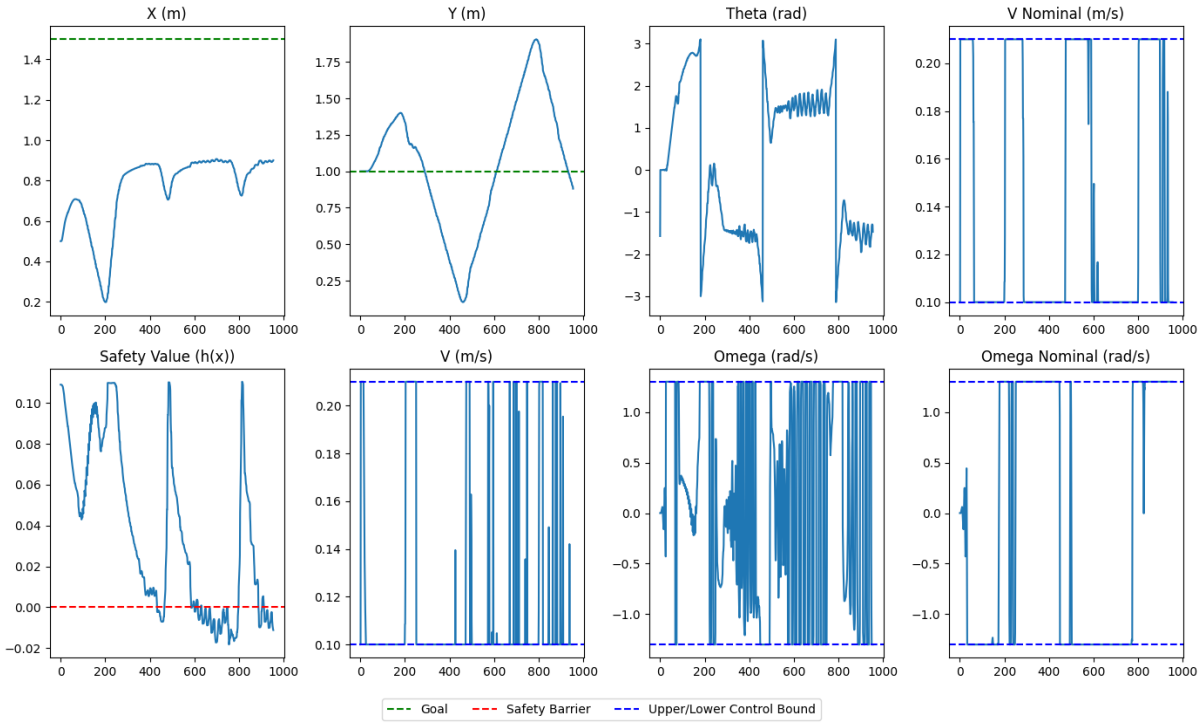



**Figure A.8.** Parameters for scenario 1 run 3 in simulation.

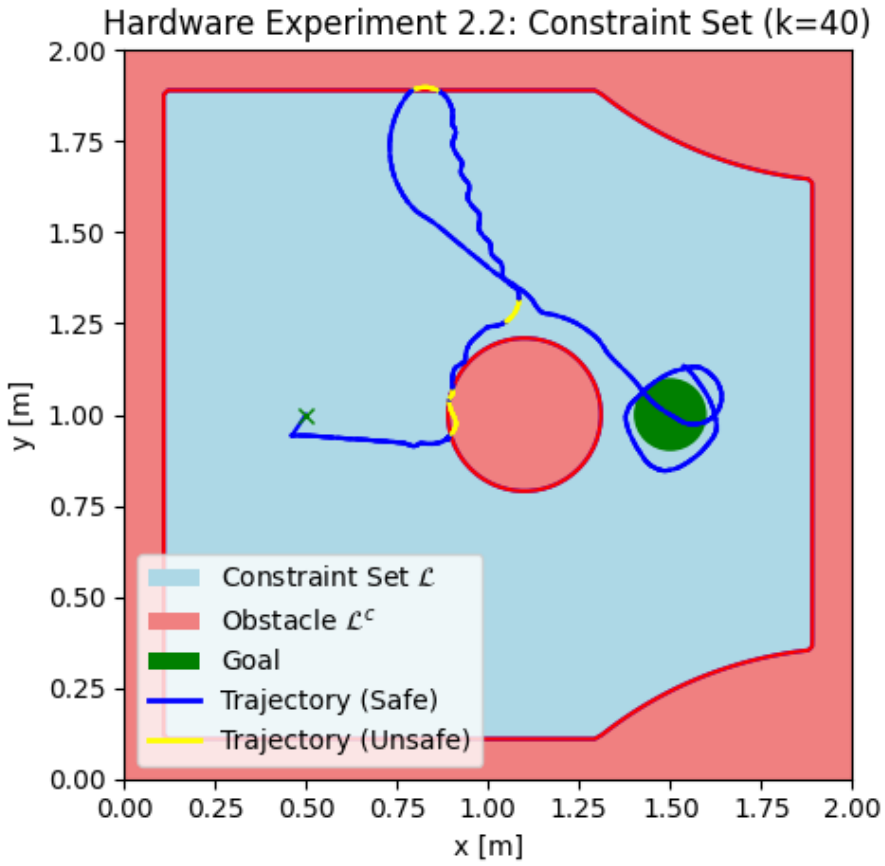



**Figure A.9.** Hardware Trajectory for scenario 2 run 2.

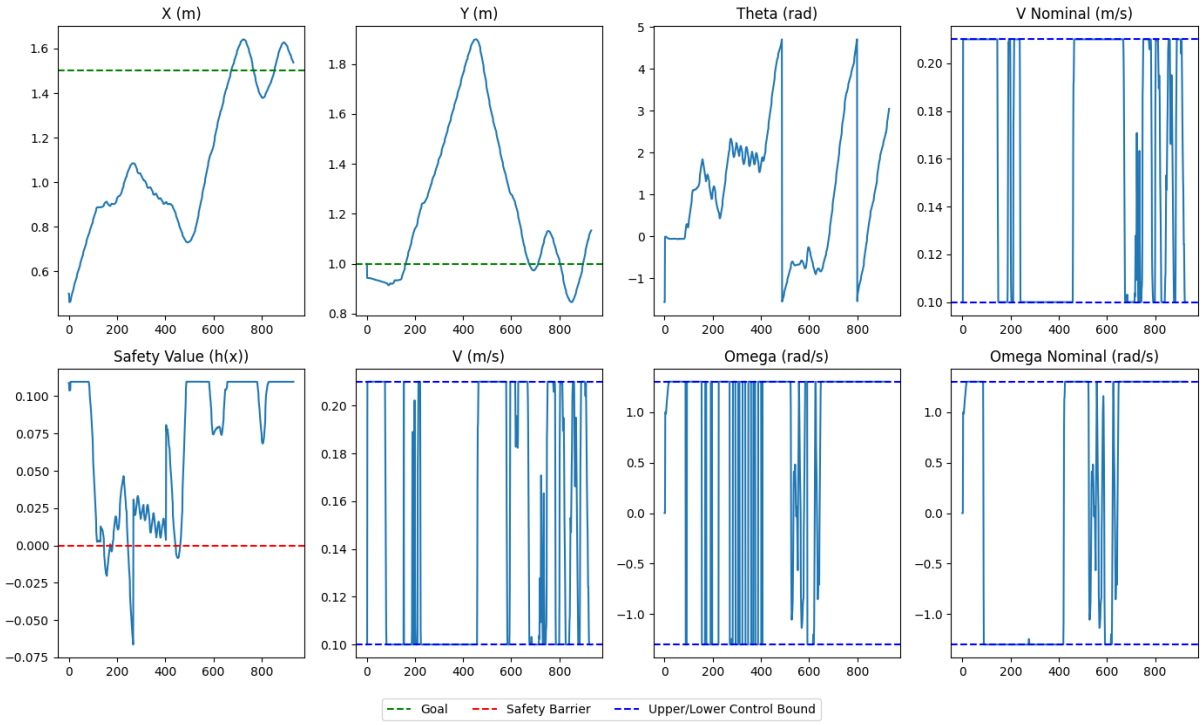



**Figure A.10.** Parameters for scenario 2 run 2 on hardware.

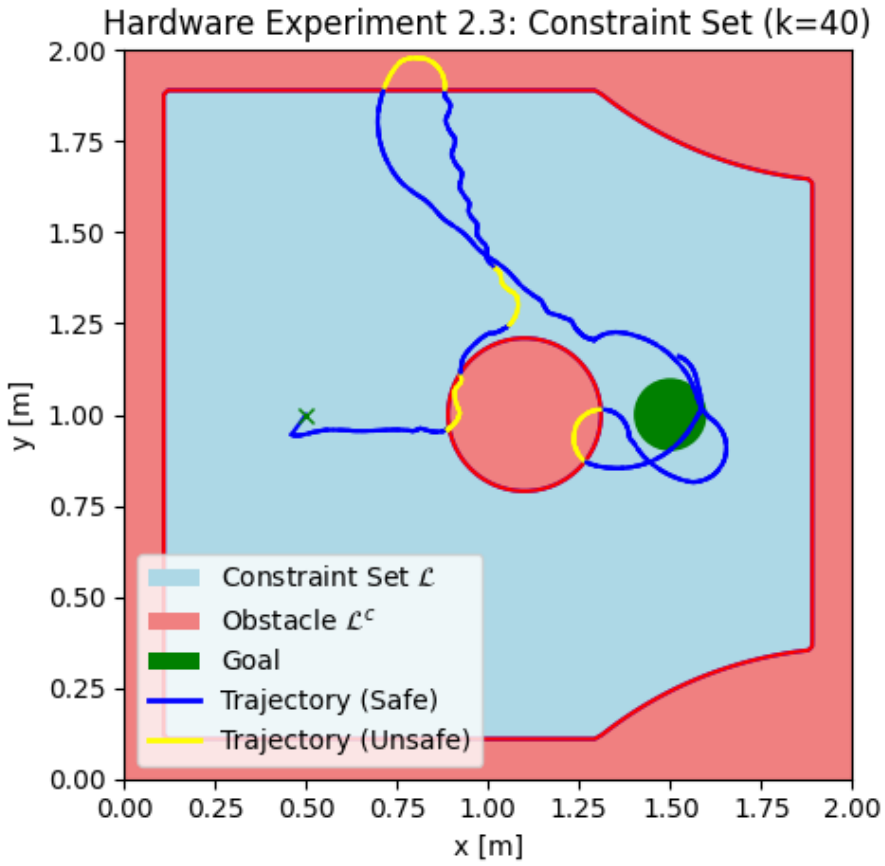



**Figure A.11.** Hardware Trajectory for scenario 2 run 3.

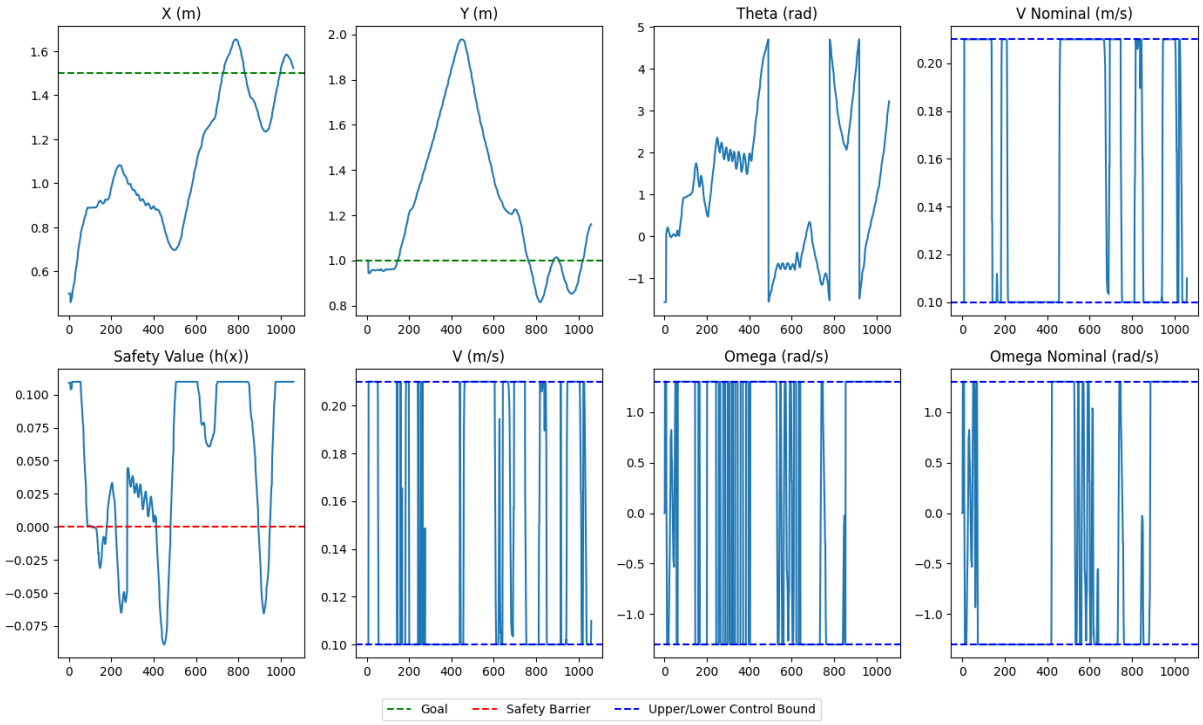



**Figure A.12.** Parameters for scenario 2 run 3 on hardware.

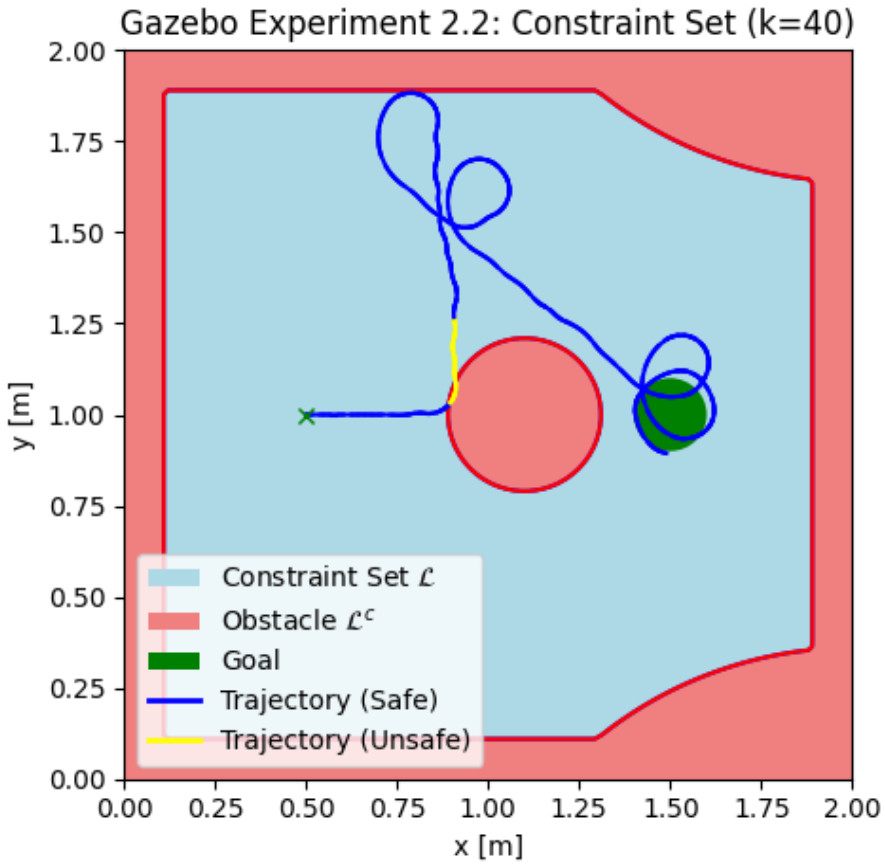



**Figure A.13.** Gazebo Simulation Trajectory for scenario 2 run 2.

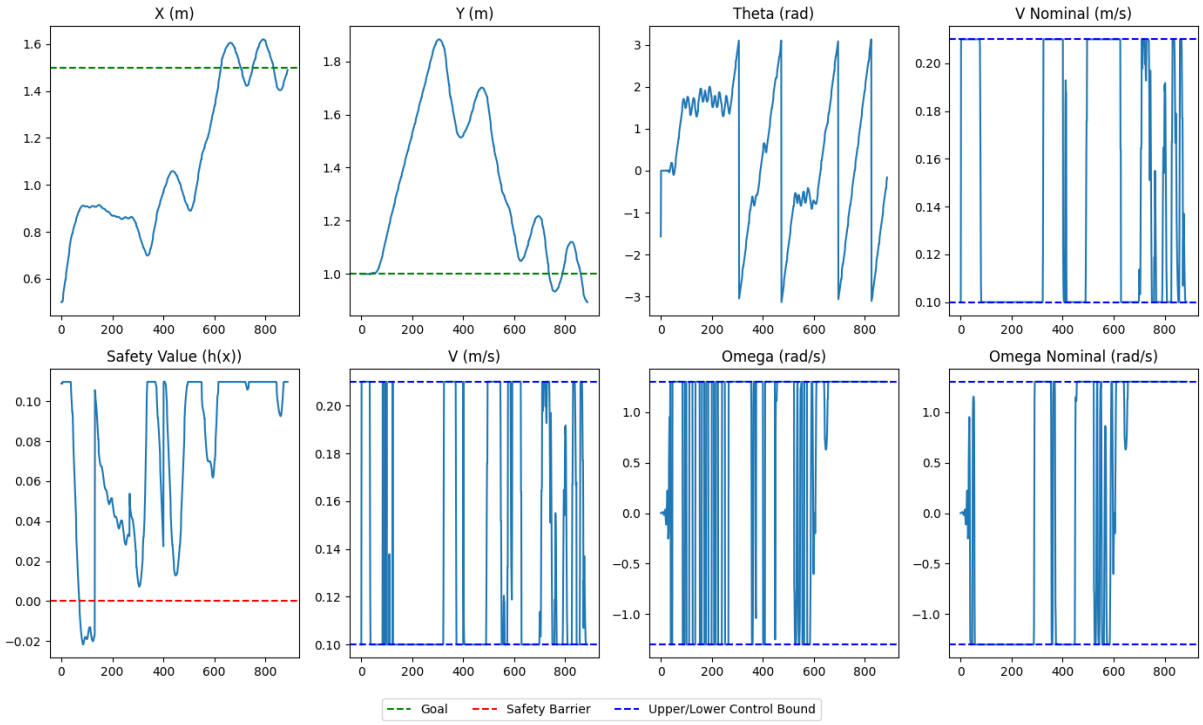



**Figure A.14.** Parameters for scenario 2 run 2 in simulation.

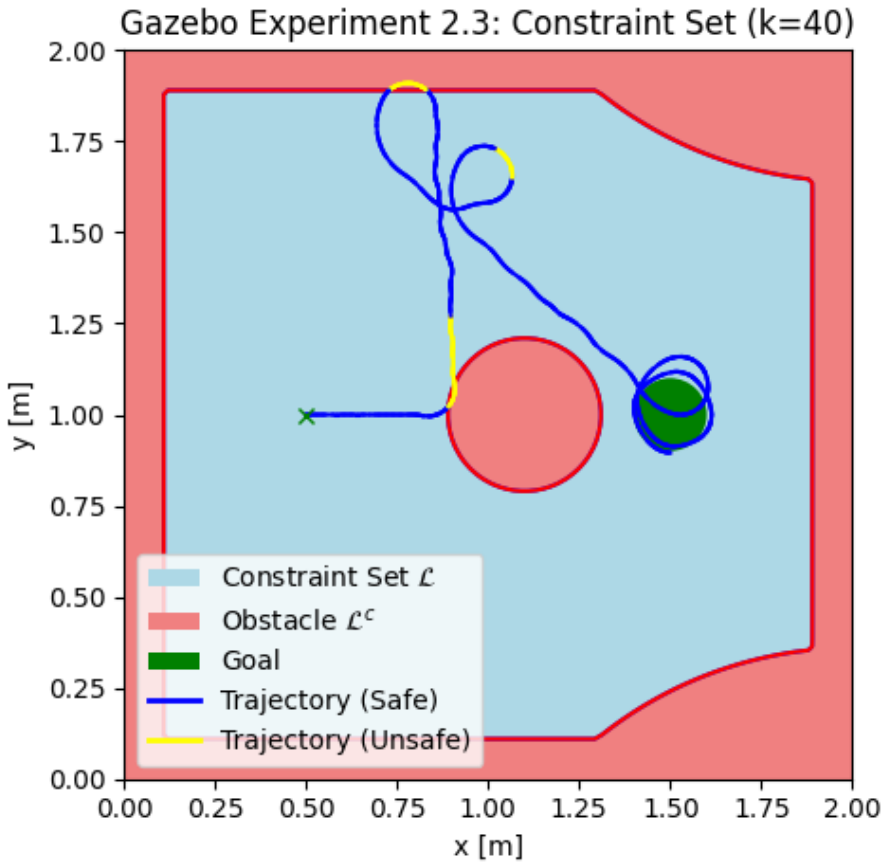



**Figure A.15.** Gazebo Simulation Trajectory for scenario 2 run 3.

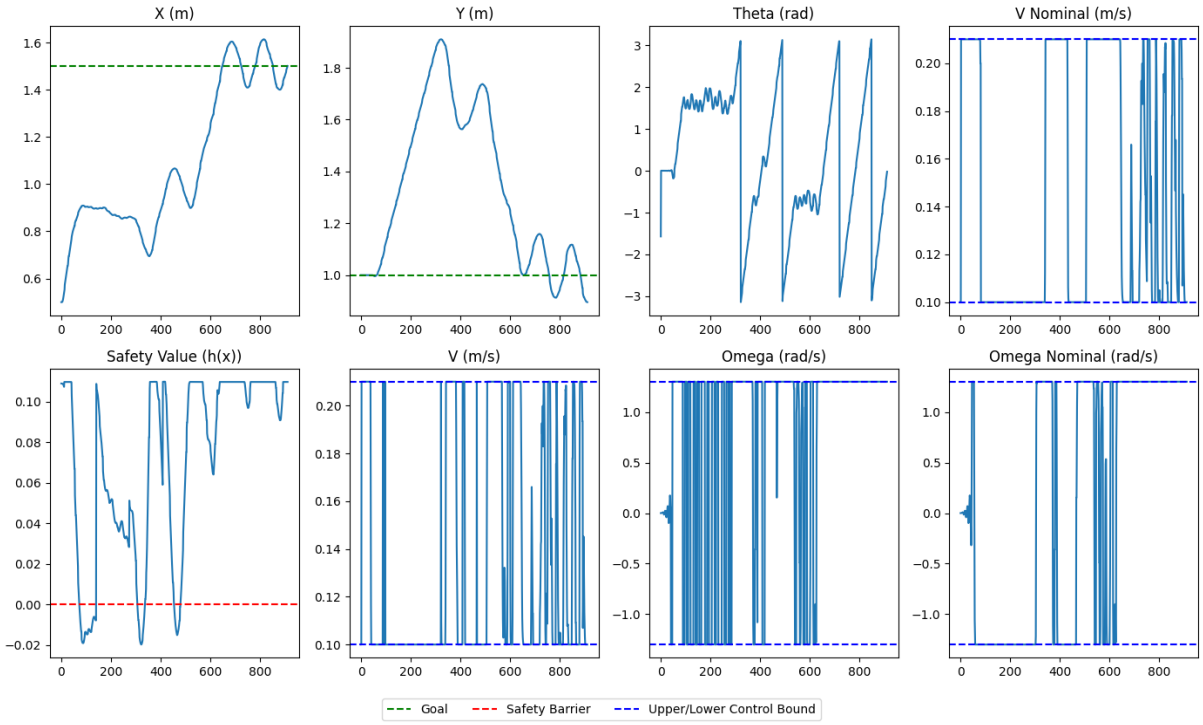



**Figure A.16.** Parameters for scenario 2 run 3 in simulation.

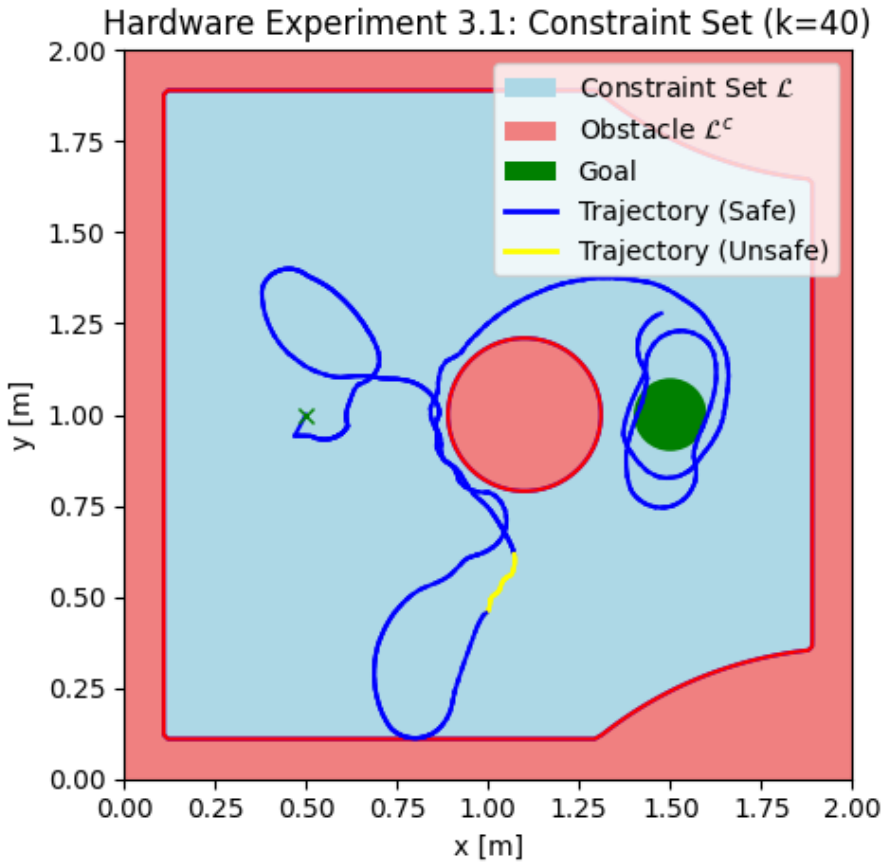



**Figure A.17.** Hardware Trajectory for scenario 3 run 1.

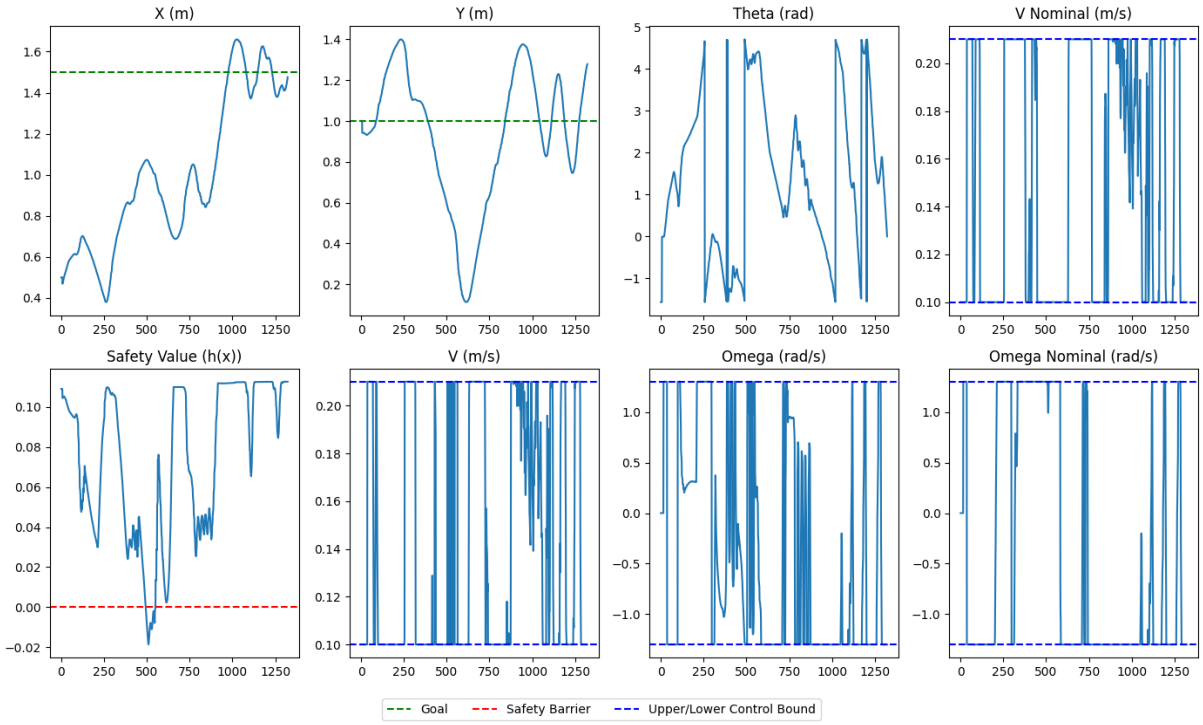



**Figure A.18.** Parameters for scenario 3 run 1 on hardware.

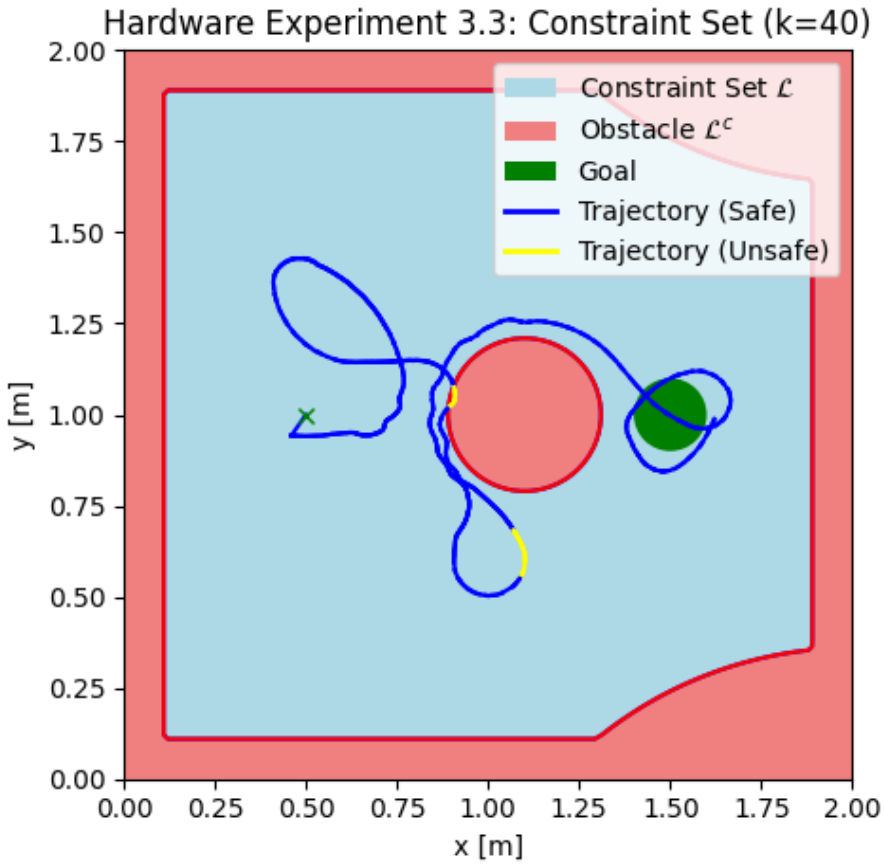



**Figure A.19.** Hardware Trajectory for scenario 3 run 3.

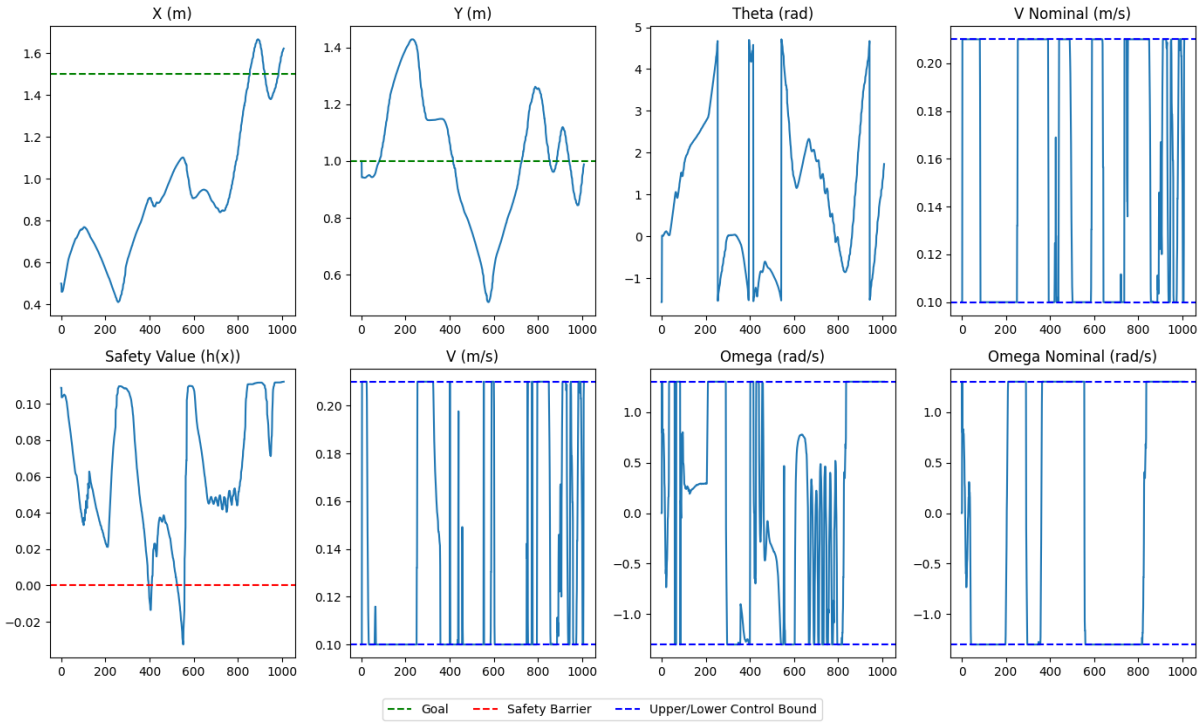



**Figure A.20.** Parameters for scenario 3 run 3 on hardware.

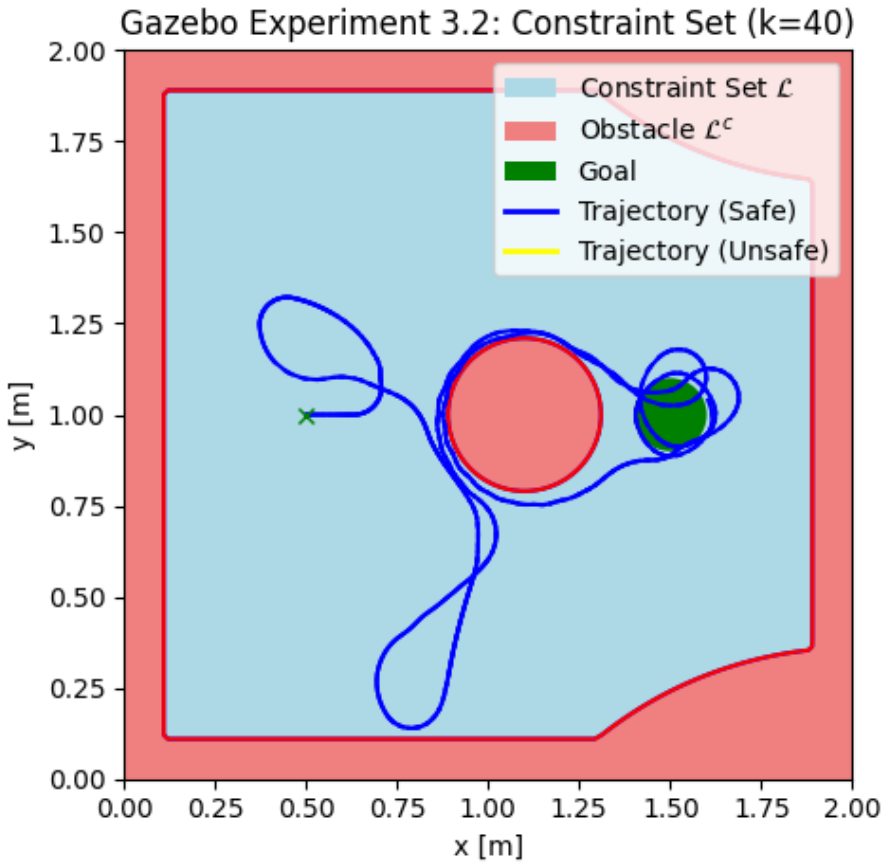



**Figure A.21.** Gazebo Simulation Trajectory for scenario 3 run 2.

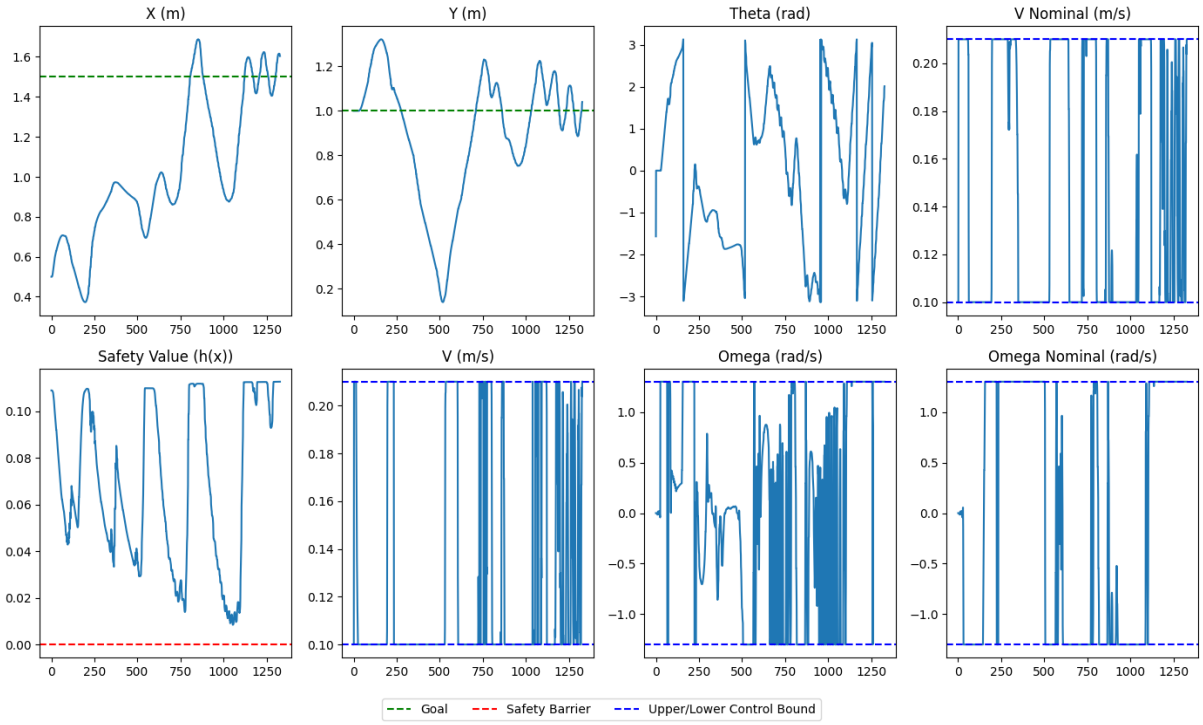



**Figure A.22.** Parameters for scenario 3 run 2 in simulation.

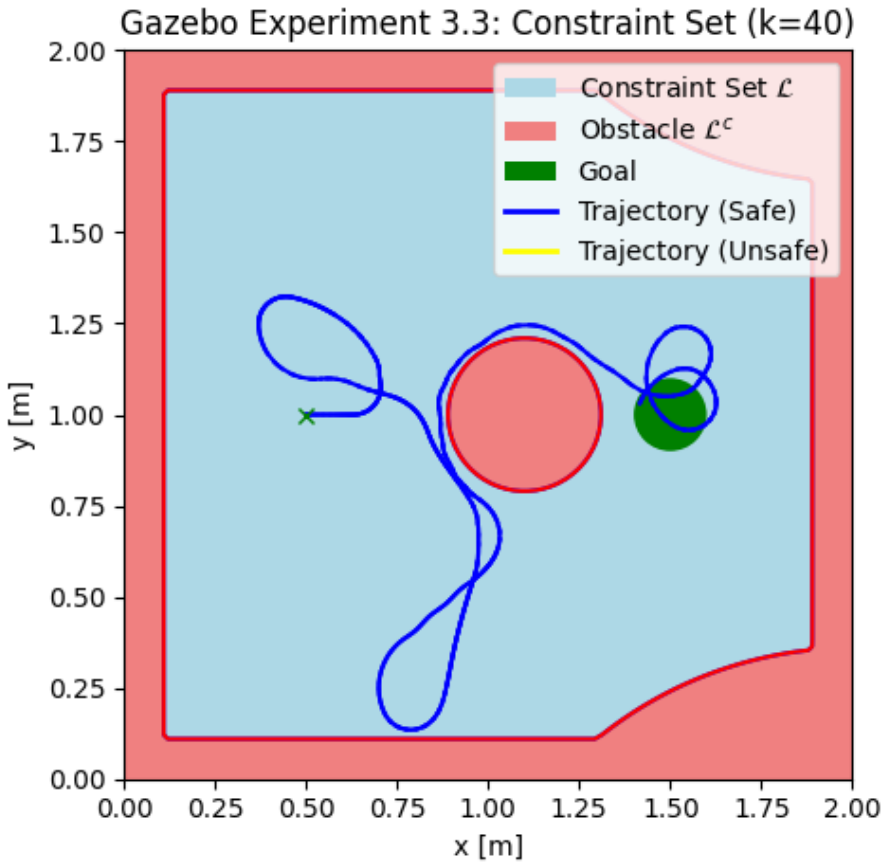



**Figure A.23.** Gazebo Simulation Trajectory for scenario 3 run 3.

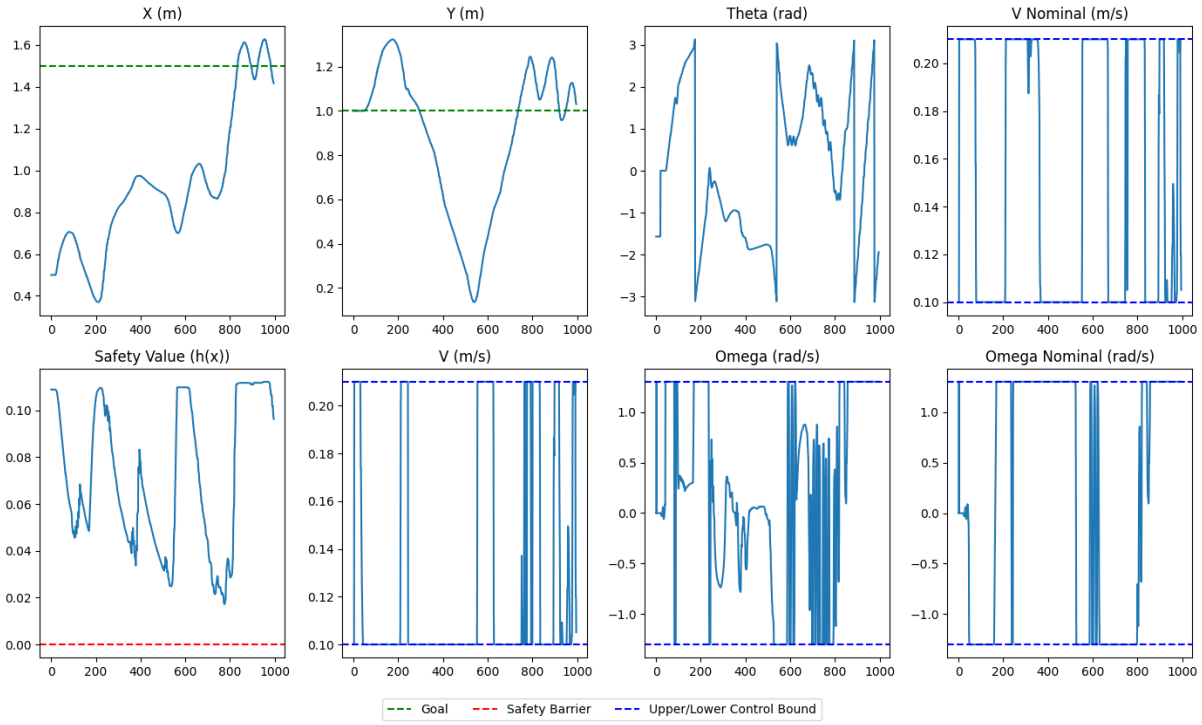



**Figure A.24.** Parameters for scenario 3 run 3 in simulation.