UNIVERSITY OF CALIFORNIA SAN DIEGO

Safe Autonomous Robot Navigation in Dynamic Environments using Reference Governor and Control Barrier Function Techniques

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

in

Engineering Sciences (Mechanical Engineering)

by

Zhuolin Niu

Committee in charge:

Professor Nikolay Atanasov, Chair
Professor Sicun Gao
Professor Sylvia Herbert

2023

University of California San Diego

2023

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Nikolay Atanasov for his support as the chair of my committee. Through multiple drafts and many long nights, his guidance has proved to be invaluable.

I would also like to acknowledge Zhichao Li and Yinzhuang Yi of ERL-lab, without whom my research would have no doubt taken fives times as long. It is their support that helped me in an immeasurable way.

Chapter 5, in part is currently being prepared for submission for publication of the material. Li, Zhichao; Yi, Yinzhuang; Atanasov, Nikolay. The thesis author was the primary investigator and author of this material.

ABSTRACT OF THE THESIS

Safe Autonomous Robot Navigation in Dynamic Environments using Reference Governor and Control Barrier Function Techniques

by

Zhuolin Niu

Master of Science in Engineering Sciences (Mechanical Engineering)

University of California San Diego, 2023

Professor Nikolay Atanasov, Chair

This thesis considers the problem of safe navigation for autonomous mobile robots working in partially known environments with static and non-adversarial dynamic obstacles. A virtual reference governor system is designed to serve as a local regulation point for the real robot system. We calculate a safe set of the robot-governor system by actively measuring the distance from surrounding static obstacles. The motion of the governor is designed to guide the robot towards the goal, while remaining within the safe set. To avoid dynamic obstacles, a control barrier function is built to further constrain the governor-robot system away from dynamic obstacles. A quadratically constrained quadratic program (QCQP) is formulated to minimally modify the governor control input to ensure dynamic obstacle avoidance. Combining these two techniques, the robot can navigate autonomously in unknown environments – slowing down when approaching obstacles, speeding up in free space, and reacting to the position and velocity of the surrounding dynamic obstacles. Our techniques are demonstrated in simulations and real-world experiments using a ground robot equipped with LiDAR to navigate among a cluttered environment in the presence of humans.

Chapter 1

Introduction

1.1 Motivation

The vehicle automated driving system (ADS) has been used in multiple leading car manufacturers, and this has brought people's interest in improving driving safety and the driver's operating experience by continuously upgrading ADS to make it smarter and more helpful. This fact also accelerated the emergence of Advanced Driver Assistant System (ADAS), which is in a higher level of automation and can complete more complex tasks. Moreover, thanks to the significant improvement of sensor and computation capabilities, more and more autonomous driving techniques has been deployed into the real world robots and eventually converted to real life applications. All these facts lead to an increasing need for research concentration on the safety of robot system self-navigation. Recent work has demonstrated both theoretical and practical guarantees for autonomous robot navigation among static obstacles but much fewer results exist for dynamic environments. Surely, there are various well developed or innovative obstacle avoiding methods at the planning level, including some learning-based planning algorithms which can also deal with dynamic obstacles. However, it is also valuable to explore possibilities for solving this safety critical problem or refining the robot's performance by smoothing the planned path from the low-level control side, as a good control policy can be more adaptive to varying environments and requires less computation power.

1.2 Contributions

The main contribution of this thesis can be recognized as an integrated Control Barrier Function (CBF) and Reference Governor (RG) method for safe autonomous robot control that can successfully avoid static and dynamic obstacles. The control policy can both drive the robot along a given path to the goal, and take necessary avoiding motions while encountering dynamic obstacles. The safe control algorithm was validated in various of simulation environments with different dynamic obstacles as well as in real-world experiments in a controlled dynamic environment.

1.3 Related Work

1.3.1 Learning-based path planning

Using Deep Reinforcement Learning (DRL) for mobile robot indoor path planning has become a prominent trend in recent years, and it has done so with great success [2]. Although its large training period and sensitivity to initial network parameters can be a downside, DRL provides noticeable strength on learning capacity, model-free advantages and independence on sensor accuracy. As traditional planning algorithms can only solve obstacle avoidance problems in a static environment, DRL became a powerful tool to solve dynamic obstacle avoidance problems benefiting from its map-free advantages and simultaneously interacting with the environment while updating its learning results. For example, the A3C algorithm's performance in maze navigation was proposed to be enhanced by employing unsupervised auxiliary tasks by Jaderberg M, and the proposed algorithm increased the convergence speed, resilience, and success rate [2]. In [19], the authors combined an artificial neural network (ANN) with Q-reinforcement learning method to find an optimal path for mobile robot navigation.

1.3.2 Model Predictive Control (MPC)

MPC is an optimized control-dependent approach for selecting control inputs by minimizing an objective function [14]. It uses an internal model with disturbances to generate estimations of predicted future states, then calculate an optimal control input based on the prediction. As such method can actively react to the internal and external dynamics, MPC became a popular theoretical-based technique to handle obstacle avoidance problems in a dynamic environment. For applications like overtaking maneuver, researchers turned to use MPC as a real-time motion generator. For example, an overtaking motion planner was proposed in [16], using MPC to deal with real-time scenarios for autonomous vehicles. However, the overtaking motion generation is considered a non-linear and non-convex problem [18], thus the non-convex optimal control problem typically arises when using the MPC strategy to solve the overtaking problem [9]. To improve the performance and further study the convexification of the problem, variants such as nonlinear MPC, Model predictive contour control, adaptive MPC, and distributed MPC emerged base on the classic MPC method.

Chapter 2

Background

2.1 System Dynamics and State Estimation

2.1.1 Ground Wheeled Robot Models

The design of system control requires a good understanding of system dynamics. We introduce two widely used vehicle kinematic models here - the bicycle model and the unicycle model. The bicycle model is usually used for steering control design, it is a kinematic model simplified from the ASG (Ackerman's Steering Geometry). The unicycle model simplifies the vehicle into a rectangular object and generate the motion model based on its location coordinates and heading angle in the world frame.

Bicycle Model

The bicycle kinematic model is generated under assumptions of both the front and rare wheels are steerable and with "zero slip angles", which is reasonable with low speed (less than 5m/s). To describe the motion of a vehicle in the world frame, three variables $x,\ y,\ \phi$ are used to represent the location and orientation of the vehicle's c.g. (center of gravity). The overall equations of motion are given by

$$
\begin{aligned}
\dot{x} &= v\ cos(\phi+\beta) \\
\dot{y} &= v\ sin(\phi+\beta) \\
\dot{\phi} &= \frac{v\ cos\beta}{l_f+l_r}(tan\delta_f - tan\delta_r)
\end{aligned}
\tag{2.1}
$$

where l_f and l_r denotes the distance from front and rare tire center to the vehicle's g.c., and β represents the vehicle's body side slip angle, i.e. the heading difference between the vehicle's longitudinal axis and its velocity V at the g.c.. δ_f, δ_r and v are three inputs of the model, with δ_f and δ_r give the front and rare tires steering angle respectively, and v being an external variable which can be picked as a constant value or generated from a longitudinal vehicle model[17].

However, at a higher vehicle speed, the ”zero slip angle” assumption can no longer be held. A dynamic model for lateral vehicle motion then was developed from the ”bicycle” model, with the vehicle lateral position y and yaw angle ϕ being two degrees of freedom. Using Newton's second law and moment balance about z axis, the equations for the lateral motion and yaw dynamics can be obtained as

$$
\begin{aligned}
m(\ddot{y}+\phi v_x) &= F_{yf}+F_{yr} \\
I_z\ddot{\phi} &= l_f F_{yf} - l_r F_{yr}
\end{aligned}
\tag{2.2}
$$

where F_{yf} and F_{yr} are the lateral forces on the front and rare tires, I_z is the vehicle's rotational inertia of z axis. To achieve automatic lane keeping, an error dynamic model, using small angle approximation to eliminate nonlinear terms, can be developed with a ”desired path”. Assume a constant longitudinal velocity v_x, it is then possible to obtain an LTI (linear time invariant) state space model in terms of error variables for steering control system, with the fact that F_{yf} and F_{yr} are proportional to the individual front and rare tire slip angles while they are small. The proportionality constant $C_{\alpha f}$ and $C_{\alpha r}$ here are called the tire cornering stiffness. [17].

$$\frac{d}{dt}\begin{bmatrix} e_1 \\ \dot{e_1} \\ e_2 \\ \dot{e_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{\alpha f}+2C_{\alpha r}}{mv_x} & \frac{2C_{\alpha f}+2C_{\alpha r}}{m} & \frac{-2C_{\alpha f}l_f+2C_{\alpha r}l_r}{mv_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2C_{\alpha f}l_f-2C_{\alpha r}l_r}{I_z v_x} & \frac{2C_{\alpha f}l_f-2C_{\alpha r}l_r}{I_z} & -\frac{2C_{\alpha f}l_f^2+2C_{\alpha r}l_r^2}{I_z v_x} \end{bmatrix} \begin{bmatrix} e_1 \\ \dot{e_1} \\ e_2 \\ \dot{e_2} \end{bmatrix} \\ + \begin{bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2C_{\alpha f}l_f}{I_z} \end{bmatrix} \delta + \begin{bmatrix} 0 \\ -\frac{2C_{\alpha f}l_f-2C_{\alpha r}l_r}{mv_x} - v_x \\ 0 \\ -\frac{2C_{\alpha f}l_f^2+2C_{\alpha r}l_r^2}{I_z v_x} \end{bmatrix} \dot{\psi}_{des} \tag{2.3}$$

The model can also be LPV (linear parameter varying) if V_x becomes time varying[17].

Unicycle Model

To get a simple unicycle kinematic model, the vehicle is usually assumed to be a rectangular object with its COM (center of mass) located on its rare wheel axle center. This allows the vehicle's velocity vector **v** has the same heading as the vehicle based on ASG motion. If using differential-drive robot, the robot can be further simplified into a rectangular with its COM lying on the geometric center. Suppose the robot 2D pose in the world frame can be denoted as $[x,\ y,\ \theta]^T$, the motion equations are

$$\begin{aligned} \dot{x} &= v\ sin\theta \\ \dot{y} &= v\ cos\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{2.4}$$

where control inputs v is the linear velocity, and ω denotes the angular velocity.

2.1.2 Simultaneous Localization and Mapping (SLAM)

While applied on a real world system, the performance of a feedback controller can be significantly influenced by the state estimation quality. For a ground wheeled robot autonomous navigation problem, the low-level controller requires the feedback states as the robot's current 2D pose. In an unknown environment, the estimation of the environment's map and the robot's states are dependent. Thus, the SLAM method was developed to generate robot 2D pose estimation while creating a single consistent map model of the environment by integrating partially observed surrounding features continuously.[7] A typical SLAM parameter estimation problem can be described as:

Problem 1. *in time* $[0,\ T]$*, given a dataset of the robot inputs* $\mathbf{u}_{0:T-1} = \{\mathbf{u}_0,\ \mathbf{u}_1,\ ...,\ \mathbf{u}_T\}$ *and observations* $\mathbf{s}_{0:T} = \{\mathbf{s}_0,\ \mathbf{s}_1,\ ...,\ \mathbf{s}_T\}$*, estimate the environment map* $\mathbf{m}$ *and the robot state trajectory* $\mathbf{x}_{0:T} = \{\mathbf{x}_0,\ \mathbf{x}_1,\ ...,\ \mathbf{x}_T\}$.

In probabilistic form, the problem requires that the distribution

$$P(\mathbf{x}_t,\ \mathbf{m}|\mathbf{s}_{0:t},\ \mathbf{u}_{0:t},\ \mathbf{x}_0) \tag{2.5}$$

be computed for all times $t \in [0,\ T]$. A recursive solution is then desired, starting with computing $P(\mathbf{x}_{t-1},\ \mathbf{m}|\mathbf{s}_{0:t},\ \mathbf{u}_{0:t})$ at time $t-1$.[6] Possible approaches for obtaining the estimation including, using Maximum Likelihood Estimation (MLE) to maximize the data likelihood conditioned on the parameters:

$$\max_{\mathbf{x}_{0:T},\mathbf{m}} logp(\mathbf{z}_{0:T},\mathbf{u}_{0:T-1}|\mathbf{x}_{0:T},\mathbf{m}), \tag{2.6}$$

or using Maximum A Posterior (MAP) to maximize the posterior pdf (probability density function) of the parameters given the data:

$$\max_{\mathbf{x}_{0:T},\mathbf{m}} logp(\mathbf{x}_{0:T},\mathbf{m}|\mathbf{z}_{0:T},\mathbf{u}_{0:T-1}), \tag{2.7}$$

then maintain a posterior pdf for the parameters given the data:

$$p(\mathbf{x}_{0:T},\mathbf{m}|\mathbf{z}_{0:T},\mathbf{u}_{0:T-1}), \tag{2.8}$$

with Bayesian Inference (BI). The joint posterior pdf can be decomposed into:

$$p(\mathbf{x}_{0:T}, \mathbf{m}, \mathbf{s}_{0:T}, \mathbf{u}_{0:T-1}) = p_0(\mathbf{x}_0, \mathbf{m}) \prod_{t=0}^{T} p_h(\mathbf{s}_t \mid \mathbf{x}_t, \mathbf{m}) \prod_{t=1}^{T} p_f(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \prod_{t=0}^{T} p(\mathbf{u}_t \mid \mathbf{x}_t) \quad (2.9)$$

due to the Markov assumptions, with $p_f(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ representing the motion model of the robot and $p_h(\mathbf{s}_t \mid \mathbf{x}_t, \mathbf{m})$ representing the observation model of the sensor.

Solutions to the probabilistic SLAM problem depends on the particular representation of the motion model and observation model. Two popular solution methods are EKF-SLAM and Fast-SLAM, with EKF-SLAM represents the robot's motion as a state-space model with additive Gaussian noise, while Fast-SLAM describes it as a set of samples with a more general non-Gaussian distribution[15]. The Fast-SLAM algorithm is also a popular dense SLAM algorithm, since it uses occupancy grid map for the environment estimation. There's also SLAM algorithms based on a sparse map model which represents the environment with landmarks, point cloud or surfels. Newer alternatives also emerged with much potential on reducing computational complexity as introduced in [4].

2.2 Safe Control

2.2.1 Lower-level Controllers

A low level controller generates control inputs effect directly on the vehicle, and is desired to drive the vehicle to achieve a local goal efficiently and stably. For a differential-drive robot with unicycle motion model, multiple feedback controllers were developed in previous research works like [20] and [1] to complete the goal tracking task.

Closed Loop Steering Goal Alignment Controller

Suppose the robot is located at $\mathbf{z}_p = [z_x,\ z_y]^T$ with heading z_θ in the world frame, to steer the robot from current 2D pose $\mathbf{z} = [z_x,\ z_y,\ z_\theta]^T$ to eventually reach the 2D goal pose $\mathbf{g} = [g_x,\ g_y,\ g_\theta]^T$, a feedback control law constructed from a simple Lyapunov function based on an error motion model was proposed in [1]. The error model is generated by projecting the robot heading in the world frame $\{W\}$ into the goal frame $\{G\}$, and compute both the location tracking error $[e,\ e_\phi]^T$ and the heading alignment error e_δ in frame $\{G\}$ as shown in Fig. 2.1.

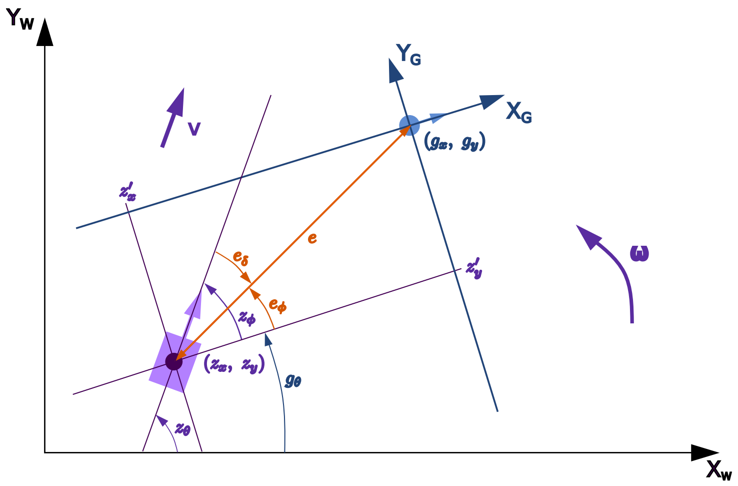


Figure 2.1. Definition of location tracking distance error e, location tracking heading error e_δ, and heading alignment error e_ϕ display in the world frame $\{W\}$ and goal frame $\{G\}$.

Suppose the robot 2D pose can be wrote as $\mathbf{z}' = [z'_x,\ z'_y,\ z_\phi]^T$ in frame $\{G\}$, and use the motion equations for a differential drive robot based on the unicycle model introduced in Sec. 2.1, the robot kinematic equations in frame $\{G\}$ are

$$\begin{aligned} \dot{z}'_x &= v\cos z_\phi \\ \dot{z}'_y &= v\sin z_\phi \\ \dot{z}'_\phi &= \omega \end{aligned} \tag{2.10}$$

Compute the errors as

$$\begin{aligned} e\ &= \|[z'_x,\ z'_y]^T\| \\ e_\phi &= \arctan 2(-z'_y, -z'_x) \\ e_\delta &= e_\phi - z_\phi \end{aligned} \tag{2.11}$$

and the motion equations then can be wrote in terms of the error states as

$$\begin{aligned} \dot{e} &= -v\cos e_\delta \\ \dot{e}_\phi &= v\frac{\sin e_\delta}{e} \\ \dot{z}_\phi &= \omega \end{aligned} \tag{2.12}$$

Construct the Lyapunov function as

$$V = V_1 + V_2 = \frac{1}{2}\lambda e^2 + \frac{1}{2}(e_\delta^2 + he_\phi^2); \quad \lambda > 0, h > 0, \tag{2.13}$$

with V_1 and V_2 represent the weighted squared norm of tracking distance error and heading alignment errors respectively. From (2.12),

$$\dot{e}_\delta = \dot{e}_\phi - \dot{z}_\phi = v\frac{\sin e_\delta}{e} - \omega, \tag{2.14}$$

consider the time derivative of V:

$$\dot{V} = \dot{V}_1 + \dot{V}_2 = -\lambda ev\cos e_\delta + e_\delta[-\omega + v\frac{\sin e_\delta}{e_\delta}\frac{(e_\delta + he_\phi}{e}] \tag{2.15}$$

Design the control law to stabilise the system, i.e. to make $\dot{V}_1 \leq 0$ and $\dot{V}_2 \leq 0$, and get

$$\begin{aligned} v &= k_v e\cos e_\delta \\ \omega &= k_\delta e_\delta + k_v\frac{\cos e_\delta \sin e_\delta}{e_\delta}(e_\delta + he_\phi) \end{aligned} \tag{2.16}$$

where $k_v > 0$ and $k_\delta > 0$ are the control gains. As the control law assures the Lyapunov function non-increasing in time, while it also has a lower bound as zero, it guarantees the state trajectory asymptotically converges to a finite non-negative limit over time corresponding to any initial condition. A simulation example of different initial conditions converging to the same goal pose is demonstrated in Fig. 2.3a.

Goal Position Tracking Controller

When the heading alignment is not required, the tracking task becomes less complex. A goal position tracking controller was proposed in [20] generated from an error motion model extracted from the projected goal position in the robot's body frame $\{B\}$.

Suppose the robot's current 2D pose is $\mathbf{z} = [z_x,\ z_y,\ z_\theta]^T$ in the world frame $\{W\}$, and rewrite the motion equations of the differential drive robot unicycle model introduced in Sec. 2.1 as

$$\dot{\mathbf{z}}_\mathbf{p} = v \begin{bmatrix} \cos z_\theta \\ \sin z_\theta \end{bmatrix} \qquad\qquad (2.17)$$
$$\dot{z_\theta} = \omega$$

with $\mathbf{z_p} = [z_x,\ z_y]^T$. Assume the robot doesn't have any lateral motion ($\begin{bmatrix} -\sin z_\theta \\ \cos z_\theta \end{bmatrix}^T \dot{\mathbf{z}}_\mathbf{p} = 0$) and is only allowed to move forward due to perception limits ($v \geq 0$). To complete a goal position tracking task, we need to construct the forward motion controller $u(\mathbf{z}) = [v(\mathbf{z}),\ \omega(\mathbf{z})]^T$ that drives the robot towards any given 2D goal position $\mathbf{g} = [g_x,\ g_y]^T$ in frame $\{W\}$. Thus the controller can be formulated based on an error dynamic stabilizer.

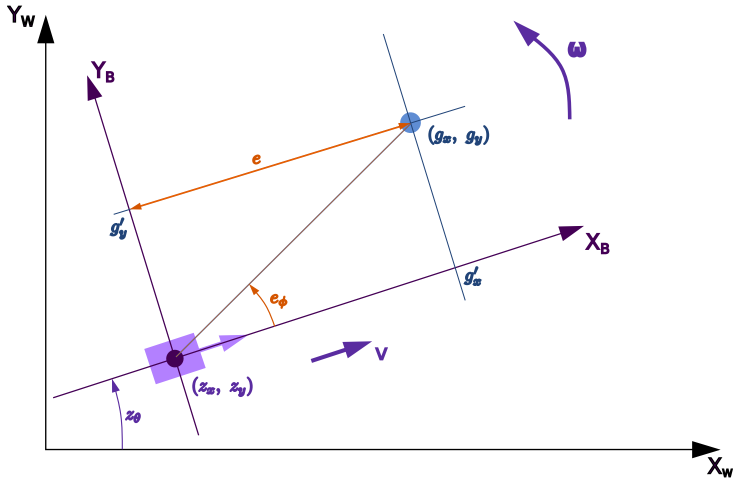


Figure 2.2. Definition of tracking distance error e and heading error e_ϕ displayed in frame $\{W\}$.

Compute the tracking distance error e and heading error e_ϕ by projecting $\mathbf{g}$ into frame

$\{B\}$ to get

$$\mathbf{g}' = R(-z_\theta)(\mathbf{g} - \mathbf{z_p}), \tag{2.18}$$

with $\mathbf{g}' = [g'_x,\ g'_y]^T$ and $R(-z_\theta) = \begin{bmatrix} \cos z_\theta & \sin z_\theta \\ -\sin z_\theta & \cos z_\theta \end{bmatrix}$.

Get the error states

$$\begin{aligned} e &= \max(0,\ g'_x) \\ e_\phi &= \arctan 2(g'_y,\ g'_x) \end{aligned} \tag{2.19}$$

as shown in Fig. 2.2.

Then pick the control law as

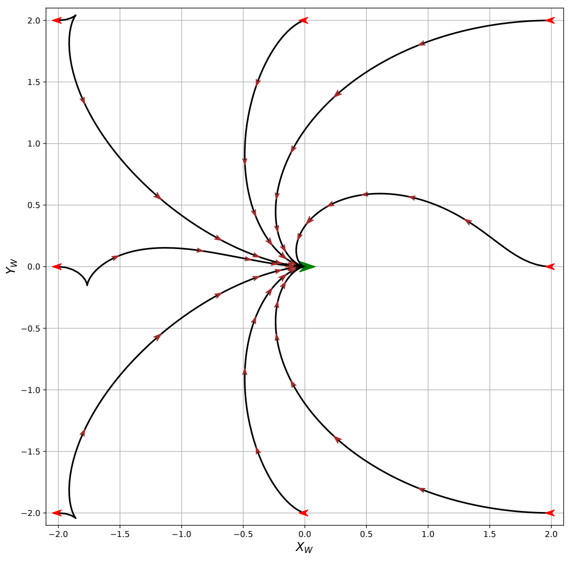
$$\begin{aligned} v &= k_v e = k_v \max\left(0,\ \begin{bmatrix} np.pi \cos z_\theta \\ \sin z_\theta \end{bmatrix}^T (\mathbf{g} - \mathbf{z_p})\right) \\ \omega &= k_\omega e_\phi = k_\omega\ \arctan 2\left(\begin{bmatrix} -\sin z_\theta \\ \cos z_\theta \end{bmatrix}^T (\mathbf{g} - \mathbf{z_p}),\ \begin{bmatrix} \cos z_\theta \\ \sin z_\theta \end{bmatrix}^T (\mathbf{g} - \mathbf{z_p})\right) \end{aligned} \tag{2.20}$$

where $k_v > 0$ and $k_\omega > 0$ are positive scalar control gains.

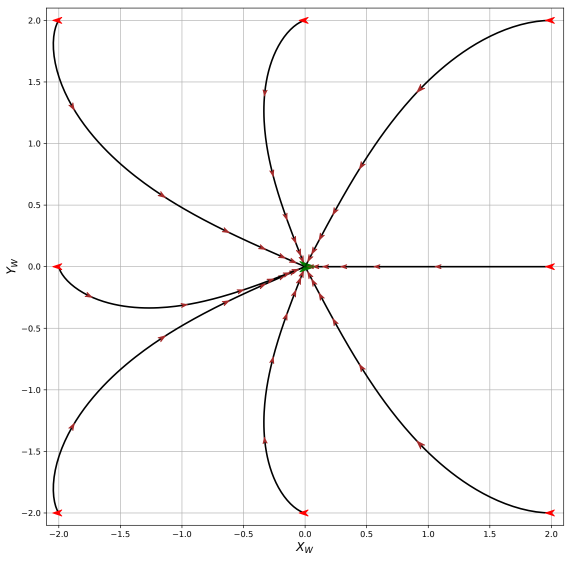
This forward motion controller can be proved as globally asymptotically stable, decreasing the errors over time and converge the errors asymptotically to zero in finite time. A simulation example of the controller driving the robot from different initial positions to the same goal position is demonstrated in Fig. 2.3b.

2.2.2 Reference Governor

The reference governor, as its name suggests, emerged as an add-on scheme for a well-designed low-level feedback controller to satisfy certain limits. It is one of the methods that augment a high performance controller for small signals with constraint handling capability for large signals. Various of reference governors were proposed for linear and non-linear systems,



(a) Goal Alignment Controller



(b) Goal Position Tracking Controller

Figure 2.3. Trajectories of eight different initial poses (red arrow) converging to the same goal pose (green arrow) under two goal tracking controllers.

and were applied in multiple fields as introduced in [11]. Fig. 2.4 is a demonstration of how the reference governor can be augmented into the system. In a typical autonomous robot driving scenario, the reference governor is used to pass the upper level reference generated from, for example a motion planning algorithm, to a lower level controller while handling constraints regarding navigation safety. Using reference governor can lead to a computationally simple implementation of both linear and non-linear systems with complex constraints.

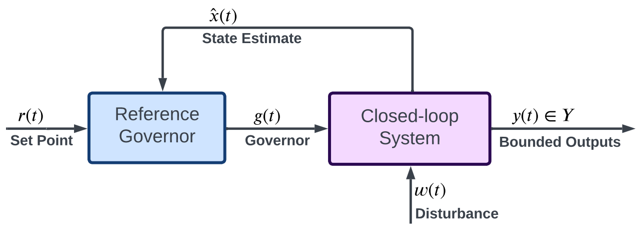


Figure 2.4. Reference governor applied in a closed-loop system with constraints and disturbances.

2.2.3 Control Barrier Function (CBF)

There's a long history of using barrier functions as safety conditions for dynamical systems to avoid undesirable regions. The main idea was to define a safe set

$$\mathscr{C} = \{x \in D \subset \mathbb{R}^n : \ h(x) \geq 0\}, \tag{2.21}$$

$$\partial\mathscr{C} = \{x \in D \subset \mathbb{R}^n : \ h(x) = 0\}, \tag{2.22}$$

$$\mathrm{Int}(\mathscr{C}) = \{x \in D \subset \mathbb{R}^n : \ h(x) > 0\}, \tag{2.23}$$

which is the super-level set of a continuously differentiable function $h : \ D \subset \mathbb{R}^n \to \mathbb{R}$, for a given dynamical system

$$\dot{x} = f(x), \quad x \in D \subset \mathbb{R}^n. \tag{2.24}$$

Definition 1 (forward invariant [5])**.** *A set $\mathscr{C}$ is forward invariant with respect to system (2.24) if, for every initial state $x(t_0) \in \mathscr{C}$, its solutions remain within $\mathscr{C}$, i.e. $x(t) \in \mathscr{C}$ for all $t \geq t_0$.*

As an extension from ensuring safety at the boundary, "Lyapunov-like" methods[3] were used to get invariant level sets and allow the safe conditions to be applied on the entire set. When comes to a control system given as

$$\dot{x} = f(x) + g(x)u, \quad u \in U \subset \mathbb{R}^m, \tag{2.25}$$

it requires a transformation from invariant sets to controlled invariant sets. More specifically, the conditions yield as:

$$\exists\, u \ \text{ s.t. } \quad \dot{h}(x,u) \geq 0 \quad \Rightarrow \quad \mathscr{C} \ \text{ is invariant} \tag{2.26}$$

Although these conditions can jointly guarantee safety and stability as they combined Lyapunov and barrier functions, they are too conservative for safety-critical control problems. As a result, the widely used modern control barrier function form expanded the barriers to the entire safe set by relaxing the restriction to:

$$\exists\, u \ \text{ s.t.} \quad \dot{h}(x,u) \geq -\alpha(h(x)) \quad \Rightarrow \quad \mathscr{C} \ \text{ is invariant} \tag{2.27}$$

with α to be an (extended) class κ function. More importantly, this less strong condition is evidently necessary and sufficient for safety-critical control policy design[3].

Definition 2 (CBF [5])**.** *Let $\mathscr{C} \subset D \subset \mathbb{R}^n$ be the super-level set of a continuously differentiable function $h:\ D \rightarrow \mathbb{R}$, then h is a control barrier function (CBF) if there exists an extended class κ_∞ function α such that for the control system (2.25):*

$$\sup_{u \in U} \left[L_f h(x) + L_g h(x)u\right] \geq -\alpha(h(x)), \tag{2.28}$$

for all $x \in D$.

Safe Control via CBFs

Suppose a feedback controller was given as $u = k(x)$ for system (2.25), the closed loop dynamical system became:

$$\dot{x} = f(x) + g(x)k(x) \tag{2.29}$$

To satisfy the conditions in (2.27) and guarantee safety of the system, we would need the control low $u = k(x)$ to render the safe set $\mathscr{C}$ forward invariant. Thus, a set contents all control inputs that keep the system safe[3]:

$$K_{cbf}(x) = \{u \in U \subset \mathbb{R}^m :\ L_f h(x) + L_g h(x)u + \alpha(h(x)) \geq 0\} \tag{2.30}$$

As a result, for a control affine system (2.25), if there exist a proper CBF h, any control law $u(x) \in K_{cbf}(x)$ can make $\mathscr{C}$ forward invariant, thus keep the system safe.

CBF composition

In cases of multi-agent control problems, the safe control set can be generated from CBFs based on different sets along time. Previous works have been done to combine barrier functions with a set of Boolean operators[8]:

$$h(x) = \min\{h_1(x), h_2(x)\} := h_1 \wedge h_2 \tag{2.31}$$

$$h(x) = \max\{h_1(x), h_2(x)\} := h_1 \vee h_2 \tag{2.32}$$

$$h(x) = -h_1(x) := \neg h_1 \tag{2.33}$$

and resulting in a Boolen Nonsmooth Control Barrier Function (BNCBF).

Chapter 3

Problem Formulation

Consider a robot navigating in an unknown environment $\mathscr{W} \subset \mathbb{R}^2$ with both static and dynamic obstacles. Denote the static obstacle space by a closed set $\mathscr{O}$, and the static-obstacle-free space by an open set $\mathscr{F}_S := \mathscr{W} \setminus \mathscr{O}$. Suppose we can model the obstacles' dynamics as an independent, linear, time-invariant system as:

$$\dot{\mathbf{p}}_\mathbf{i} = \mathbf{v}_\mathbf{i} \tag{3.1}$$

where the state $\mathbf{x}_\mathbf{i} := (\mathbf{p}_\mathbf{i}, \mathbf{v}_\mathbf{i}) \in \mathbb{R}^4$, with $\mathbf{p}_\mathbf{i}(t) \in \mathbb{R}^2$ and $\mathbf{v}_\mathbf{i}(t) \in \mathbb{R}^2$ denoting its center position and velocity in the workspace, respectively. Then abstract each dynamic obstacle set as an ellipsoid $\mathscr{D}_i = \{\mathbf{q} \in \mathbb{R}^2 \mid (\mathbf{q} - \mathbf{p}_\mathbf{i})^T \mathbf{A}_\mathbf{i} (\mathbf{q} - \mathbf{p}_\mathbf{i}) \leq 1\}$ with its shape defined by $\mathbf{A}_\mathbf{i} \in \mathbb{R}^{2\times 2}$. Define a combined space of k dynamic obstacles as $\mathscr{D}(t) := \bigcup_{i=1}^{k} \mathscr{D}_i(t)$, and the absolute-free space as $\mathscr{F}(t) := \mathscr{W} \setminus (\mathscr{O} \cup \mathscr{D}(t))$.

As higher-order robot dynamics can be controlled by a fast low-level control loop, we model the robot as a fully actuated acceleration controlled system:

$$\dot{\mathbf{p}} = \mathbf{v} \quad \dot{\mathbf{v}} = \mathbf{u} \tag{3.2}$$

where $\mathbf{u}(t) \in \mathbb{R}^2$ is the control input; $\mathbf{x} := (\mathbf{p}, \mathbf{v}) \in \mathbb{R}^4$ is the robot state, with $\mathbf{p}(t) \in \mathbb{R}^2$ the robot's position, and $\mathbf{v}(t) \in \mathbb{R}^2$ its velocity in the workspace. The goal is to design a closed-loop control policy $\mathbf{u}(t)$ that can accomplish a path following task while ensuring avoidance of all obstacles.

Problem 2. *Given a planar curve $\mathscr{P} : [0,1] \mapsto \mathscr{F}_S$ that connects the robot's start position $\mathscr{P}(0)$ and goal position $\mathscr{P}(1)$ in the free space $\mathscr{F}_S$, design a control policy $\mathbf{u}(t)$ that can drive the robot in* (3.2) *asymptotically to $\mathscr{P}(1)$, while remaining collision-free, i.e., $\mathbf{p}(t) \in \mathscr{F}(t)$ for all $t \geqslant 0$.*

Chapter 4

Optimal Control Policy Design

4.1 Robot-Governor System

4.1.1 Robot Position Convergence

To simplify the collision checking problem with complex kinematic systems, we introduce a robot-governor system with state $(\tilde{\mathbf{x}}, \mathbf{g})$ [13], as $\tilde{\mathbf{x}}$ being a modified robot state which shifts the robot system's equilibrium point off the origin. This augmented system includes a virtual first-order reference governor system:

$$\dot{\mathbf{g}} = \mathbf{u}_{\mathbf{g}} \tag{4.1}$$

with state $\mathbf{g}$ and control input $\mathbf{u}_{\mathbf{g}}$. We use a proportional-differential (PD) controller:

$$\mathbf{u}(t) = -k_p(\mathbf{g}(t) - \mathbf{p}(t)) - k_d \mathbf{v}(t) = -\mathbf{K}\tilde{\mathbf{x}}(t) \tag{4.2}$$

with two positive scalar control gains, $k_p > 0$ and $k_d > 0$, to generate feedback control inputs. The closed-loop system of the robot can be constructed as:

$$\dot{\tilde{\mathbf{x}}} = \bar{\mathbf{A}}\tilde{\mathbf{x}} \tag{4.3}$$

where $\bar{\mathbf{A}} := (\mathbf{A} - \mathbf{BK})$ is Hurwitz, and ensures any initial state $\tilde{\mathbf{x}}_0$ exponentially converge to the equilibrium point at the origin. It is then possible to make $(\mathbf{g}, \mathbf{0})$ an equilibrium point to the robot local system by picking $\tilde{\mathbf{x}} = (\mathbf{p} - \mathbf{g}, \mathbf{v})$, which allows the robot position $\mathbf{p}(t)$ always track the governor position $\mathbf{g}(t)$ for all $t \geq 0$ [13].

4.1.2 Robot Prediction Set

To look further into safety of the robot, we first need to get a prediction set $\mathscr{R}(t)$, which bounds the trajectory of robot tracking a static governor $\mathbf{g}(t)$ from an initial position $\mathbf{x}(t)$, to generate robot's safe set with respect to all surrounding obstacles for the governor control policy design. In this thesis, we simply define the robot motion prediction set as an Euclidean ball centered at $\mathbf{g}(t)$ with radius $\delta(t)$, i.e.

$$\mathscr{R}(t) = \{\mathbf{q} \in \mathbb{R}^2 | \ \|\mathbf{q} - \mathbf{g}(t)\|^2 \leq \delta(t)\} \tag{4.4}$$

where $\delta(t)$ is an upper bound on the largest deviation of robot position $\mathbf{p}(t)$ from the governor state $\mathbf{g}(t)$, i.e.:

$$\delta(t) \geq \max_{\tau \geq t} \|\mathbf{p}(\tau) - \mathbf{g}(\tau)\|^2 \tag{4.5}$$

The value of $\delta(\mathbf{x}(t), \mathbf{g}(t))$ can be obtained in different ways, while the specific expression of $\delta(t)$ depends on the definition of robot model and feedback controller. One simple method is to construct a Lyapunov function as:

$$V(\mathbf{x}, \mathbf{g}) = \|\mathbf{p} - \mathbf{g}\|^2 + \frac{1}{2}\|\mathbf{v}\|^2 \tag{4.6}$$

which bounds the robot trajectory with

$$\delta_s(t) = \|\mathbf{p}(t) - \mathbf{g}(t)\|^2 + \frac{1}{2}\|\mathbf{v}(t)\|^2 \tag{4.7}$$

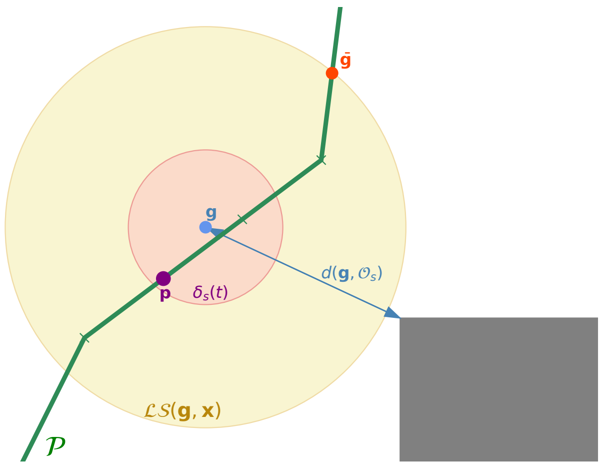
We picked this bound function to construct safety constraints for static obstacle avoidance as its easier to compute, however, this estimation of robot's prediction set is relatively conservative.

Other than simply using an Euclidean ball, there are methods proposed by other researchers that can generate a more adaptive prediction set with respect to specific robot models and controllers. For example, Vandermonde simplexes of initial robot state was derived as a more accurate prediction set compared with Lyapunov ellipsoid sets for safety evaluation in [20], corresponding to a proportional-high-order-derivative (PhD) feedback controller designed to complete safe path-following tasks based on a parameterized reference path and the collision distance. There is also work done in [12] and [13] that found two methods of obtaining tight output peak bounds of a control-affine non-linear robot system, by solving an semi-definite programming (SDP) or a series of Lyapunov-equations constructed with positively invariant ellipsoids for the linearized dynamical system. Moreover, an (chunked) ice-cream cone shaped motion set for a goal-alignment controller, which can drive a unicycle robot from any initial position towards a given goal position in the front direction, was proposed in [21].

With a proper robot prediction set selected, we can then design the governor's control policy to ensure safety of the augmented robot-governor system, which also implies the safety of the robot. To make the control policy more adaptive, we consider static obstacles and dynamic obstacles separately. This allows us to get an adaptive local safe set for the robot based on safety measuring functions combined the static environment and the augmented system's state, then optimize the control policy generated from this local safe set to also take dynamic obstacles' states in count.

4.2 Governor Control Policy with Static Obstacles

To navigate the robot to avoid static obstacles, a popular method of generating control inputs is to develop a function that measures the robot motion's safety level with respect to free space boundaries, and use this function as a feedback for the control policy.



$\bar{\mathbf{g}}$
$\mathbf{g}$
$d(\mathbf{g}, \mathcal{O}_s)$
$\mathbf{p}$
$\delta_s(t)$
$\mathcal{LS}(\mathbf{g}, \mathbf{x})$
$\mathcal{P}$

Figure 4.1. Geometric relations between robot-governor positions, local-safe-zone and governor local projected goal projected in 2d environment.

Here we follow the process in [13] and first define an energy function

$$\Delta E(\mathbf{x}, \mathbf{g}) := d^2(\mathbf{g}, \mathscr{O}_S) - \delta(\mathbf{x}, \mathbf{g}) \tag{4.8}$$

to measure the leeway to safety violation, where $d(\mathbf{g}, \mathscr{O}_S)$ denotes the minimum distance from governor's position to the closest static obstacle, and pick $\delta(\mathbf{x}, \mathbf{g}) = \delta_s(t)$ as defined in Sec. 4.1.2. Then define a robot local safe zone as:

$$LS(\mathbf{x}, \mathbf{g}) := \{\mathbf{q} \in \mathscr{F}_s \mid d^2(\mathbf{q}, \mathbf{g}) \leq \max(0, \Delta E(\mathbf{x}, \mathbf{g}))\} \tag{4.9}$$

Use the time-varying local safe zone to generate an adaptive control policy for the virtual governor to achieve safe path-following, while its speed varies with current detected surroundings. Thus, we pick a local projected goal for the governor as:

$$\bar{\mathbf{g}} := \mathscr{P}(\alpha^*) \tag{4.10}$$

$$\alpha^* := \max_{\alpha \in [0,1]} \{\alpha \mid \mathscr{P}(\alpha) \in LS(\mathbf{x}, \mathbf{g})\} \tag{4.11}$$

where $\mathscr{P}(\alpha^*)$ represents the furthest intersection of the path and the local safe zone as the robot driving from $\mathscr{P}(0)$ to $\mathscr{P}(1)$ (shown in Fig. 4.1). With this well defined governor goal, we can simply get a governor feedback control policy as:

$$\bar{\mathbf{u}}_{\mathbf{g}} = -k_g(\mathbf{g} - \bar{\mathbf{g}}(\mathbf{x}, \mathbf{g})) \tag{4.12}$$

where $kg > 0$ is a control gain for the feedback governor controller.

4.3 CBF for Dynamic Obstacles

On the opposite of keeping the robot in a local safe zone to avoid static obstacles, when come to avoiding dynamic obstacles, we need to make sure the robot stay away from the obstacle ellipsoid sets $\mathscr{D}(t)$ as we defined in Chap. 3 while tracking the virtual governor. It is also important to assure the robot can always converge to the governor position, so we modify the governor's control policy $\bar{\mathbf{u}}_{\mathbf{g}}$ designed in Sec. 4.2 to get an optimized governor control policy $\mathbf{u}_{\mathbf{g}}^*$ instead of changing the robot control input directly.

To reduce the complexity of dynamic collision check in this section, we ignore the relative position or direction from the robot-governor system to moving obstacle sets. As to say, we can define the robot-governor set as an time varying Euclidean ball as

$$\mathscr{B}(t) := \{\mathbf{b} \mid \|\mathbf{b} - \mathbf{g}\| \leq \|\mathbf{p} - \mathbf{g}\| + r_b\} \tag{4.13}$$

where r_b is the radius of the robot object's circumscribed circle. Such that we can check the safe level for the whole robot-governor set, not only the point of the robot position, w.r.t. surrounding moving obstacles. Here we pick each dynamic obstacle set $\mathscr{O}_{mi}$ as a 2D circle with radius r_{oi}, and define

$$d_{hi} := \|\mathbf{g} - \mathbf{p_i}\| - \|\mathbf{g} - \mathbf{p}\| - R_i \tag{4.14}$$

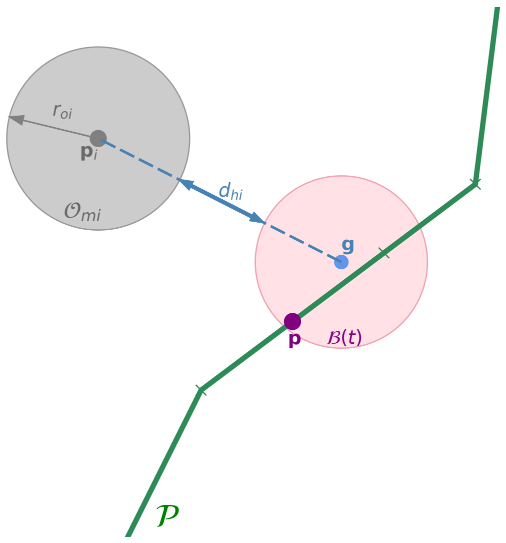


Figure 4.2. Geometric relations between robot-governor set and dynamic obstacle set in 2d environment used to construct CBF.

as the minimal distance between the i-th dynamic obstacle set and the robot-governor set as shown in Fig. 4.2, with $R_i = r_i + r_b$.

Define a positive invariant safe set $\mathscr{C}_i$ relating the augmented governor system's states and each dynamic obstacle' states as:

$$\mathscr{C}_i := \{(\mathbf{g}, \mathbf{p}, \mathbf{p_i}) \in \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \ : \ h_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i}) \geq 0\} \tag{4.15}$$

then derive the control barrier function (CBF) as:

$$h_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i}) := (\mathbf{g} - \mathbf{p_i})^T \mathbf{A}_i (\mathbf{g} - \mathbf{p_i}) - (\mathbf{g} - \mathbf{p})^T \mathbf{A}_i (\mathbf{g} - \mathbf{p}) - 1 - \frac{2\|\mathbf{g} - \mathbf{p}\|}{R_i} \tag{4.16}$$

with $A_i = I\frac{1}{R_i^2} \in \mathbb{R}^{2\times 2}$. So that $h_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i}) \geq 0$ implies the robot-governor set $\mathscr{B}(t)$ stays non-intersect with the dynamic obstacle set $\mathscr{O}_{mi}(t)$, thus ensure safety of the robot w.r.t. the i-th dynamic obstacle (shown in Fig. 4.2). To get a control policy for the first-order governor system, we can bridge the safe condition with the governor control input by taking the first derivative of

$h_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i})$ to get

$$\dot{h}_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i}, \mathbf{u_g}) = \frac{\partial h_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i})}{\partial \mathbf{g}} \mathbf{u_g} + \frac{\partial h_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i})}{\partial \mathbf{p}} \mathbf{v} + \frac{\partial h_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i})}{\partial \mathbf{p_i}} \mathbf{v_i} \tag{4.17}$$

Since it is desired the system avoids collision with all of the dynamic obstacles, we must compose individual barrier functions using the Boolean conjunction $\wedge$ (AND) operation [8] [5] to get

$$h_m = \bigwedge_{i=1}^{k} h_i = \min(h_1, ..., h_k) \tag{4.18}$$

to get the safe condition of avoiding k dynamic obstacles. Our goal here then becomes to find a governor control policy $\mathbf{u_g}$, such that $h'_m + \alpha(h_m) \geq 0$ for a selected extended class $\mathscr{K}_\infty$ function $\alpha(h)$.

4.4 Control Synthesis via Quadratic Programs

To achieve the final goal of avoiding both static and dynamic obstacles, we can combine safe conditions obtained in Sec. 4.2 and Sec. 4.3 to get an optimized governor control policy. Although avoiding dynamic obstacles is critical, we still need the robot to stay on the given path as more as possible so that it can achieve the global goal. It is also important to make sure the robot won't run into any static obstacle while avoiding the dynamic ones. Thus, the static-obstacle-safe governor control policy $\bar{\mathbf{u}}_\mathbf{g} = -k_g(\mathbf{g} - \bar{\mathbf{g}})$ designed in Sec. 4.2 can be used as a nominal control input for the optimization problem, while

$$\dot{h}_m + \alpha(h_m) \geq 0 \tag{4.19}$$

$$\bar{\mathbf{g}}^*_m \in LS(\mathbf{x}, \mathbf{g}) \tag{4.20}$$

can be used as two constrains while searching for the solution $\bar{\mathbf{g}}^*_m$ of dynamic-obstacle-safe local projected goal $\bar{\mathbf{g}}_m$, which can be used to generate the dynamic-obstacle-safe governor control policy $\mathbf{u_g} = -k_g(\mathbf{g} - \bar{\mathbf{g}}_m)$.

Since it's simpler and more intuitive to construct the optimization problem on $\bar{\mathbf{g}}$, we reformat (4.17) into

$$\dot{h}_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i}) = -k_g \frac{\partial h_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i})}{\partial \mathbf{g}}(\mathbf{g} - \bar{\mathbf{g}}_m) + \frac{\partial h_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i})}{\partial \mathbf{p}}\mathbf{v} + \frac{\partial h_i(\mathbf{g}, \mathbf{p}, \mathbf{p_i})}{\partial \mathbf{p_i}}\mathbf{v_i} \tag{4.21}$$

and use it with (4.20) as the constraints to synthesize a dynamic-obstacle-safe governor local projected goal $\bar{\mathbf{g}}_m$ via the following quadratically constrained quadratic program (QCQP):

$$\begin{aligned} &\min_{\bar{\mathbf{g}}_m \in \mathbb{R}^2} \; \|\bar{\mathbf{g}}_m - \bar{\mathbf{g}}\|^2 \\ \text{s.t.} \quad &\dot{h}_m + \alpha(h_m) \geq 0, \; \|\mathbf{g} - \bar{\mathbf{g}}_m\|^2 \leq \|\mathbf{g} - \bar{\mathbf{g}}\|^2 \end{aligned} \tag{4.22}$$

The solution $\bar{\mathbf{g}}_m^*$ will lead to a governor control law $\mathbf{u_g} = -k_g(\mathbf{g} - \bar{\mathbf{g}}_m)$ that can avoid collision with both static and dynamic obstacles while still tracking the given global goal. An example of how $\bar{\mathbf{g}}$ was modified to $\bar{\mathbf{g}}_m^*$ in the sense of dynamic obstacle avoidance is displayed in Fig. 4.3.

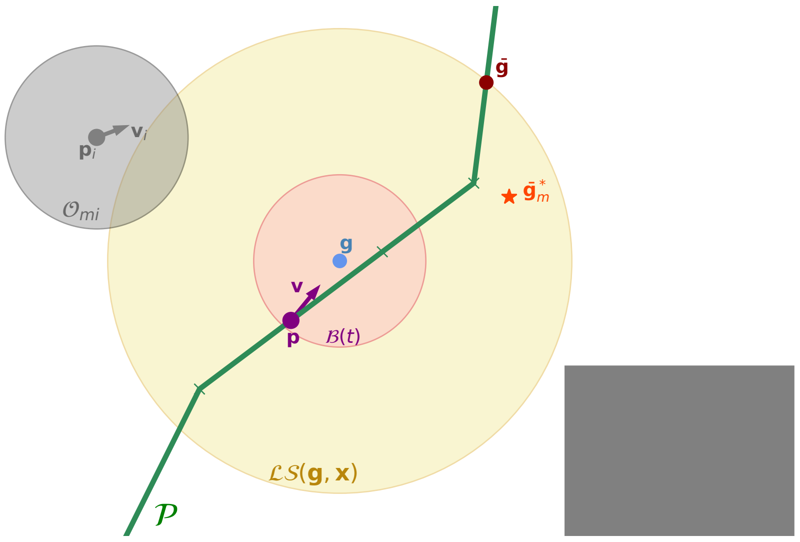


Figure 4.3. Original governor local projected goal $\bar{\mathbf{g}}$ generated from the sense of static environment versus optimized governor local projected goal $\bar{\mathbf{g}}_m^*$ generated in a dynamic environment.

4.5 Robot Position Convergence in Dynamic Environment

As proved in Sec. 4.1.1, the robot position is asymptotically converge to the governor's position. As the governor's local projected goal was defined in (4.10) and (4.11) as an intersection of the planned path and the local safe zone, the governor position can be assured to eventually converge to a global goal. However, when the dynamic obstacles come to the environment, the governor control inputs can be shifted by the dynamic obstacles' motion and deviate from the predefined path. In this case, we need additional assumptions on the dynamic obstacles' motion to make sure the robot-governor system can reach the global goal position. First, the dynamic obstacles can only pass by but not block the final goal position over time. Second, we assume the dynamic obstacles are non-aggressive, as to say, they are traveling with natural motions without continuously pushing the navigation agent away from its goal or towards a corner space.

4.6 Low-level Controller Design

To apply the robot-governor system on a wheeled mobile robot, we need to integrate the governor system with a unicycle model as described in Sec. 2.1 with a better designed low-level controller other than the simple PD controller (4.2) we picked in Sec. 4.1.1. Here we modified the two goal tracking controllers introduced in Sec. 2.2.1 to add bi-directional motions to make the tracking trajectory more smooth and to better accommodate with all-direction sensors like LiDAR.

Bi-directional Goal Alignment Controller

Use the same notation as in Sec. 2.2.1, the current robot pose and the goal pose were denoted as $\mathbf{z} = [z_x,\ z_y,\ z_\theta]^T$ and $\mathbf{g} = [g_x,\ g_y,\ g_\theta]^T$ in frame $\{W\}$, with $\mathbf{z}_p = [z_x,\ z_y]^T$ and $\mathbf{g}_p = [g_x,\ g_y]^T$. To allow the vehicle to act fully backward motion, instead of aligning the vehicle's front heading to the goal pose's front heading ($z_\theta = g_\theta$), when the goal position located in the backwards half-plane of the vehicle ($[\cos z_\theta, \sin z_\theta](\mathbf{g}_p - \mathbf{z}_p) < 0$), as displayed in Fig.

we set a virtual goal pose as $\hat{\mathbf{g}} = [g_x,\ g_y,\ \hat{g}_\theta]^T$ with $\hat{g}_\theta = g_\theta + \pi$, so the modified controller aligns the vehicle's back heading to the goal pose's back heading ($z_\theta + \pi = \hat{g}_\theta$). In this case, while computing the heading alignment error $\hat{e}_\phi$ w.r.t. the virtual goal frame $\hat{G}$, we project the robot pose $\mathbf{z}$ into frame $\hat{G}$ to get $\hat{\mathbf{z}}' = [\hat{z}'_x,\ \hat{z}'_y,\ \hat{z}_\phi]^T$ and $\hat{e}_\phi = \arctan 2(-\hat{z}'_y, -\hat{z}'_x)$. Similar as (2.10), (2.11) and (2.12), we can get the robot kinematic equations in frame $\{\hat{G}\}$ as

$$\begin{aligned}
\dot{\hat{z}}'_x &= v\cos\hat{z}_\phi \\
\dot{\hat{z}}'_y &= v\sin\hat{z}_\phi \\
\dot{\hat{z}}'_\phi &= \omega
\end{aligned} \tag{4.23}$$

compute the errors as

$$\begin{aligned}
e \ &= \|[\hat{z}'_x,\ \hat{z}'_y]^T\| \\
\hat{e}_\phi &= \arctan 2(-\hat{z}'_y, -\hat{z}'_x) \\
\hat{e}_\delta &= \hat{e}_\phi - \hat{z}_\phi
\end{aligned} \tag{4.24}$$

and the motion equations then can be wrote in terms of the error states as

$$\begin{aligned}
\dot{e} \ &= -v\cos\hat{e}_\delta \\
\dot{\hat{e}}_\phi &= v\frac{\sin\hat{e}_\delta}{e} \\
\dot{\hat{z}}_\phi &= \omega
\end{aligned} \tag{4.25}$$

Using the same Lyapunov function as (2.13), the control law for backwards motion then can be obtained as

$$\begin{aligned}
v &= k_v e\cos\hat{e}_\delta \\
\omega &= k_\delta\hat{e}_\delta + k_v\frac{\cos\hat{e}_\delta\sin\hat{e}_\delta}{\hat{e}_\delta}(\hat{e}_\delta + h\hat{e}_\phi)
\end{aligned} \tag{4.26}$$

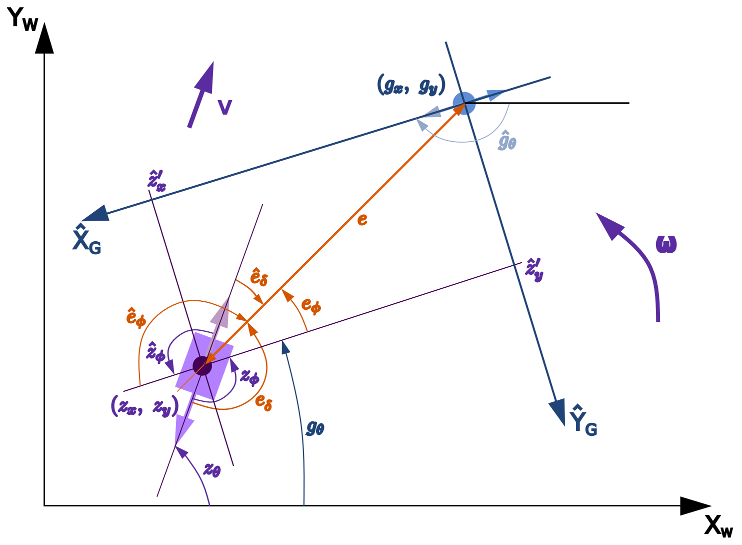


Figure 4.4. Definition of location tracking distance error e, location tracking heading error e_δ, and heading alignment error e_ϕ display in the world frame $\{W\}$ and the virtual goal frame $\{\hat{G}\}$.

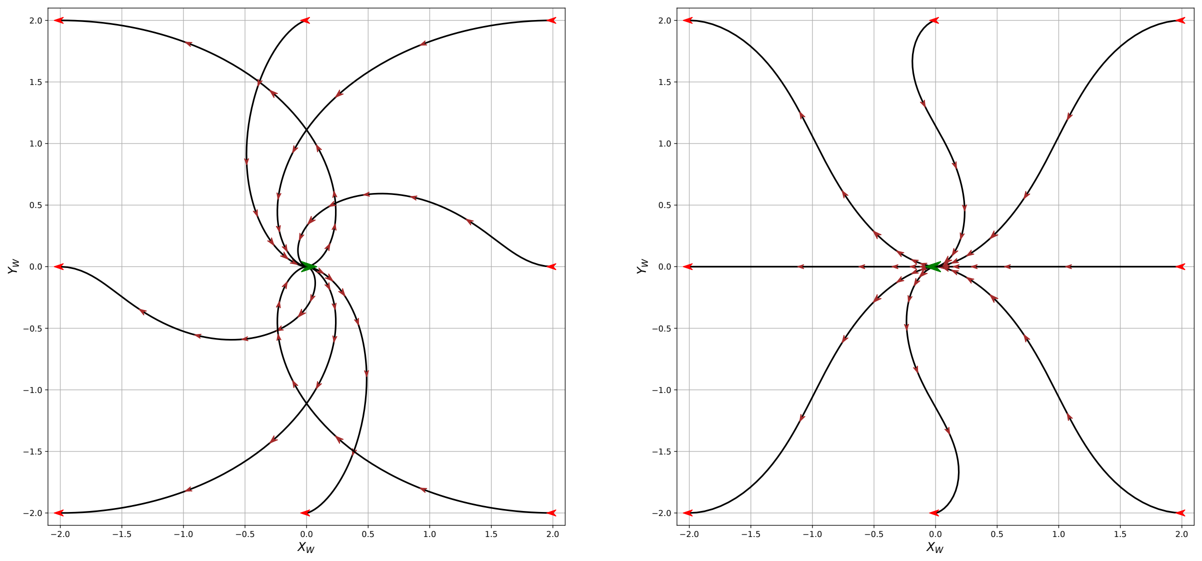
The integrated bi-directional control law then was given as

$$\begin{aligned} v &= \begin{cases} k_v e \cos \hat{e}_\delta, & [\cos z_\theta, \sin z_\theta](\mathbf{g}_p - \mathbf{z}_p) < 0 \\ k_v e \cos e_\delta, & otherwise \end{cases} \\ \omega &= \begin{cases} k_\delta \hat{e}_\delta + k_v \frac{\cos \hat{e}_\delta \sin \hat{e}_\delta}{\hat{e}_\delta}(\hat{e}_\delta + h\hat{e}_\phi), & [\cos z_\theta, \sin z_\theta](\mathbf{g}_p - \mathbf{z}_p) < 0 \\ k_\delta e_\delta + k_v \frac{\cos e_\delta \sin e_\delta}{e_\delta}(e_\delta + h e_\phi), & otherwise \end{cases} \end{aligned} \tag{4.27}$$

Examples of different initial conditions converging to the same goal pose under the control of this modified controller were displayed in Fig. 4.5.

Bi-directional Goal Position Tracking Controller

Based on the forward-motion-only position tracking controller introduced in Sec. 2.2.1, we modified the controller by eliminating the limit of $v \geq 0$ and confine the tracking heading error range from $[-\pi, \pi]$ to $[-\frac{\pi}{2}, \frac{\pi}{2}]$. This was done because adding the backward driving motion allows the robot to track goal positions on its back-half plane by directly heading back, while with the forward-motion-only controller, the robot needs to first turning in place until the



(a) Goal Heading at 0 rad. **(b)** Goal Heading at π rad.

Figure 4.5. Trajectories of eight different initial poses (red arrow) converging to the same goal pose (green arrow) under the bi-directional goal pose alignment controller.

goal position can be projected on its front-half plane, and then move towards the goal.

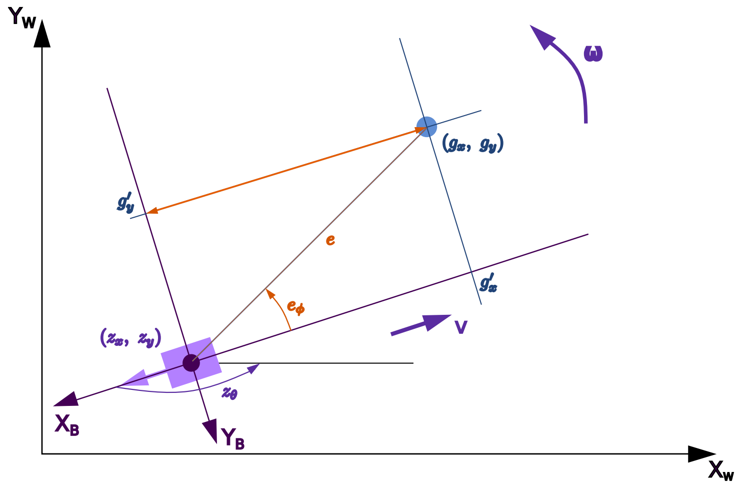


Figure 4.6. Definition of tracking distance error e and heading error e_ϕ for bi-directional motion control.

Follow the same notation as in Sec. 2.2.1 and under the same assumption that there's no lateral motion of the robot. Similar as (2.19), the modified controller computed the tracking errors also by first projecting the goal position $\mathbf{g} = [g_x,\ g_y]^T$ into the robot's body frame B to get

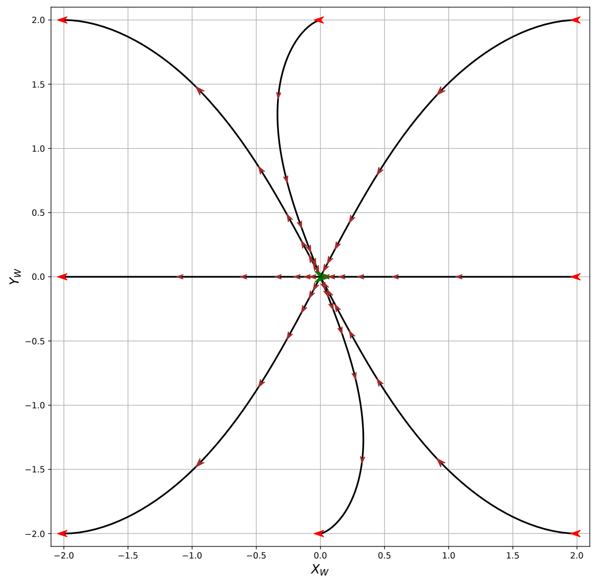


Figure 4.7. Trajectories of eight different initial positions (red arrow) converging to the same goal position (green arrow) under the bi-directional goal position tracking controller.

$\mathbf{g}' = R(-z_\theta)(\mathbf{g} - \mathbf{z_p})$. Compute the error states

$$\begin{aligned} e &= g'_x \\ e_\phi &= \begin{cases} 0, & g'_x = g'_y = 0 \\ \arctan(\frac{g'_x}{g'_y}), & otherwise \end{cases} \end{aligned} \tag{4.28}$$

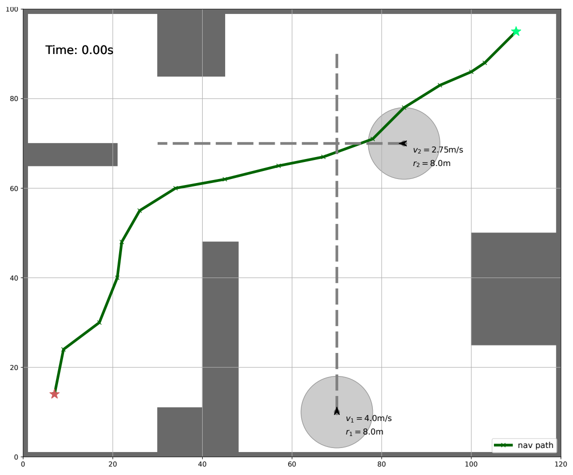
as shown in Fig. 4.6. The control law stays the same as (2.20), an example of different initial conditions converging to the same goal position using this modified position tracking controller were displayed in Fig. 4.7.

Chapter 5

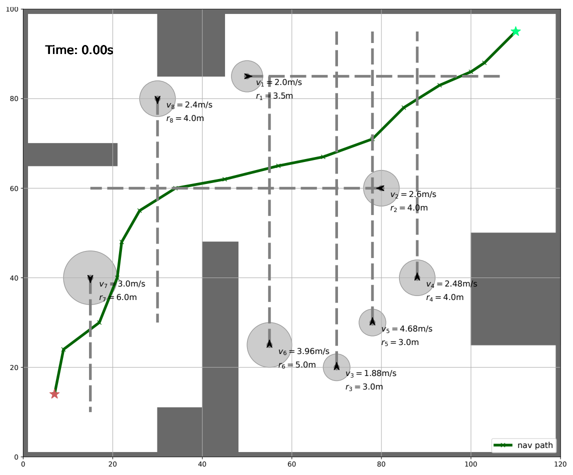
Evaluation

5.1 Virtual Environment Simulation

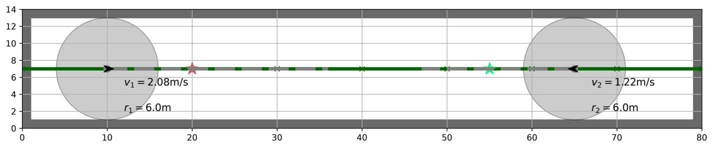
5.1.1 Environment Setup



(a) Simple case with two dynamic obstacles.



(b) Complex case with eight dynamic obstacles.



(c) Extreme situation in a corridor.

Figure 5.1. Simulation environment layout.

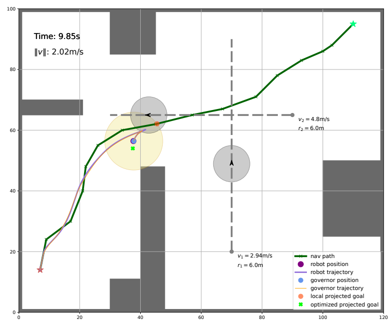
To evaluate the performance of the control policy designed in Chap. 4, we first set up a simulated environment with a 2D grid map including a few static obstacles and used an open source rapidly-exploring random tree (RRT*) path planning package to generate the path $\mathscr{P}$ from the start position to the goal position for the robot to track. For a simple case, we added two dynamic obstacles as shown in Fig. 5.1a, and for a complex case, we put eight dynamic obstacles with different sizes onto the map as shown in Fig. 5.1b. Additionally, to validate the performance of this control law under an extreme situation where there's dynamic obstacles moving towards the system in a corridor, we set up a small map with two large dynamic obstacles blocking the whole free zone on the system's travel direction as shown in Fig. 5.1c. The simulation environment includes static obstacles (dim-gray rectangles), dynamic obstacles (gray circles) and a given path $\mathscr{P}$ (dark-green line segments) connecting the start position (red star) and goal position (green star). The gray dash-line represents the individual dynamic obstacle's trajectory and the black arrow points at its moving direction, its radius and linear velocity are noted next to its initial position (gray dot).

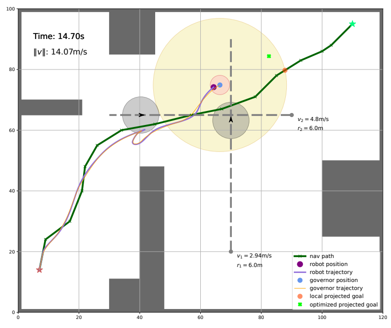
5.1.2 Simulation with Double Integrator Model

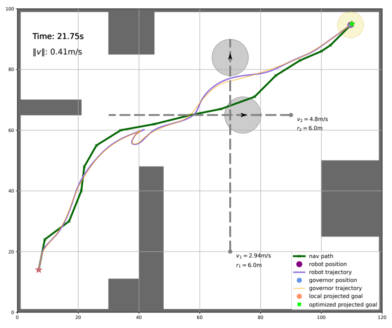
Simple environment

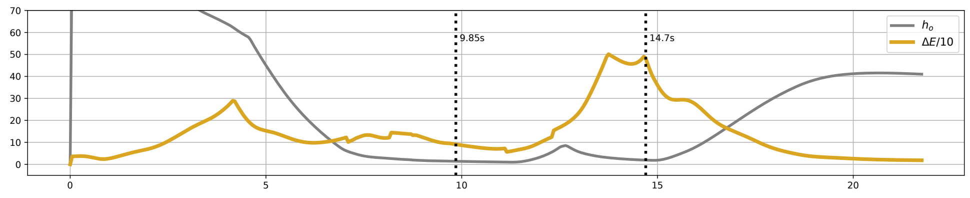
The first simulation was intended to show the control policy can achieve the goal of dynamic obstacle avoidance. Two medium size dynamic obstacles was placed in the environment, moving back and forth with a constant speed. The simulation results in Fig. 5.2 shows the robot's and governor's trajectories while avoiding dynamic obstacles, and the real-time robot velocity was noted on the figure as well.

In this simulation, we used a simple double integrator model to represent the robot system and a PD controller as the low level controller as described in Sec. 3, we also ignored any potential control inputs' constraints that a typical vehicle may have in practice. The simulation results showed that the robot-governor system run into the first dynamic obstacle moving toward it in Fig. 5.2a, and slowed down while steered away from the path until the obstacle totally past the path; then it encountered the second dynamic obstacle from its side in Fig. 5.2b, and turned away from the obstacle's moving direction to avoid the obstacle with an overtaking from its front


(a) Avoiding 1st Dynamic Obstacle


(b) Avoiding 2nd Dynamic Obstacle


(c) Goal Reached



(d) Safe Metrics Value over Time

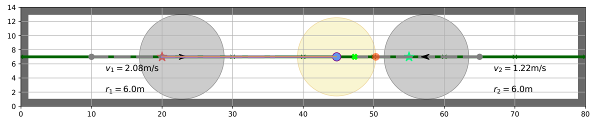
Figure 5.2. Snapshots of robot driving along the predefined path while avoiding collision of both static and dynamic obstacles by tracking the reference governor in a simple environment, the safe metrics value are also displayed.

then drove back to the path. The system eventually reached the selected goal position in Fig. 5.2c and performed a successful dynamic obstacle avoidance while remain collision free with static obstacles. Safe metrics values in Fig. 5.2d stay positive thru the whole process, indicating no violation of safe conditions.

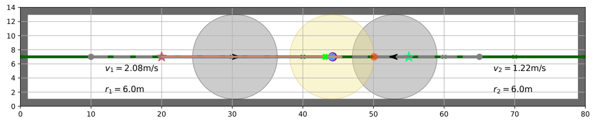
Extreme 1d path validation

To test the control policy's limits, we put the system into a corridor with two dynamic obstacles whose diameters can fully cover the width. As to say, while encountering with the dynamic obstacle, there's no side ways to take. Moreover, one of the dynamic obstacle' trajectory was set to pass the robot's goal position, so that the robot couldn't reach the goal without running into any dynamic obstacle.

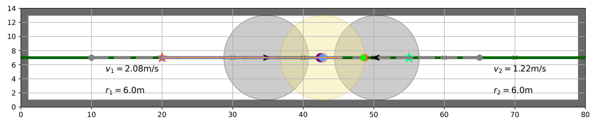
The results demonstrated that the system can drive in the opposite direction from the goal while necessary, but may get confused at the moments when surrounding dynamic obstacles came to similar distances due to the composition of CBFs in Sec. 4.4. It also proved the policy can assure safety of the system in the "worst case", i.e. the obstacle is traveling towards the



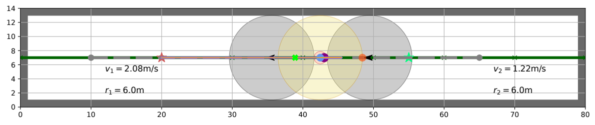
(a) Slowing down when encountered the obstacle



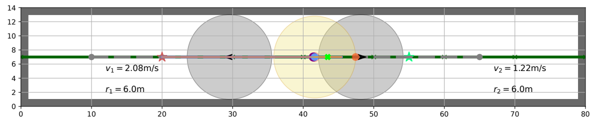
(b) Stepping back when pushed by the obstacle



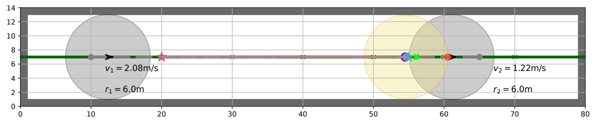
(c) Picking the back obstacle as the closest



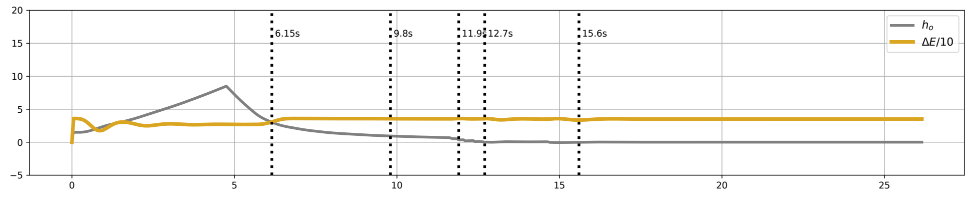
(d) Picking the front obstacle as the closest



(e) Following the obstacle at a low speed to track the goal



(f) Goal Reached



(g) Safe Metrics Value over Time

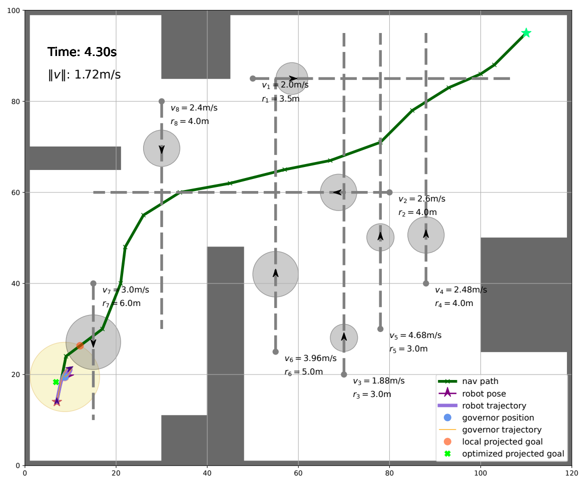
Figure 5.3. Snapshots of robot driving along a 1d path while avoiding collision aggressive dynamic obstacles in a corridor, the safe metrics value are also displayed.

robot-governor system, with the robot position lies on the line segment that connects the governor and the obstacle's center. Fig. 5.3c and Fig. 5.3d showed two moments of the optimized governor projected goal fell into the other obstacle set while avoiding the closest one, but it was able to switch fast enough to keep the safety metric valid. Fig. 5.3d shows how the robot-governor system tried to track the goal as fast as possible while its speed got significantly reduced by the obstacle's speed.

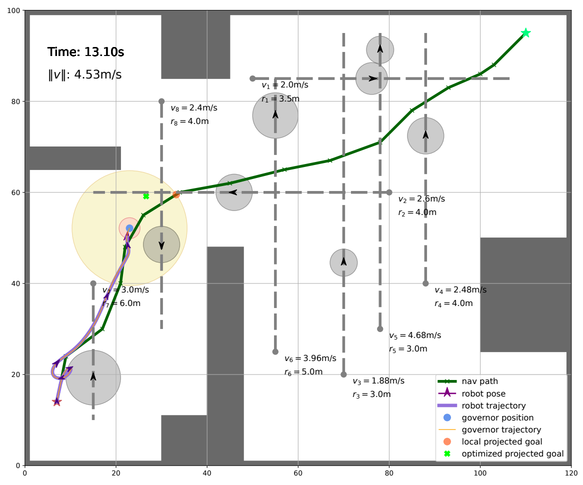
5.1.3 Simulation with Unicycle Model

Complex environment with goal position tracking controller

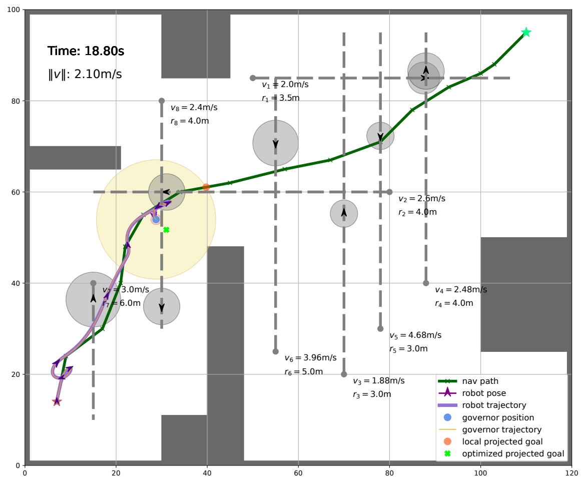
This simulation set a more complex environment with eight dynamic obstacles in different sizes moving around with different constant speeds. To make the simulation closer to a practical case, we used a differential drive unicycle model as introduced in Sec. 2.1. So the velocity vector became $\mathbf{v} = [v\cos z_\theta, v\sin z_\theta]^T$. We paired the model with the position tracking controller (2.20) introduced in Sec. 2.2.1 and added control inputs constraints as $|v| \leq 10\ m/s$ and $|\omega| \leq 5\ rad/s$ for the linear and angular velocities respectively. Selected snapshots are displayed in Fig. 5.4 showing the dynamic obstacle avoidance motions with the safe metric values over time.



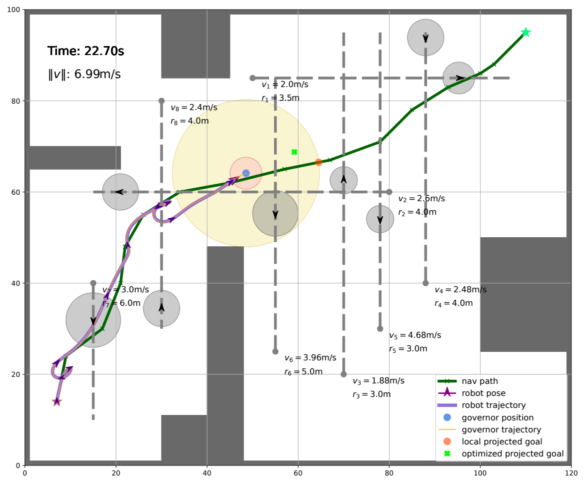
(a) 1st Avoidance



(b) 2nd Avoidance



(c) 3rd Avoidance

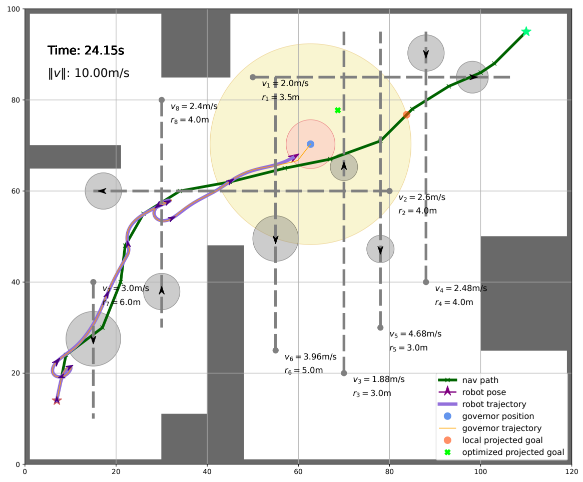


(d) 4th Avoidance

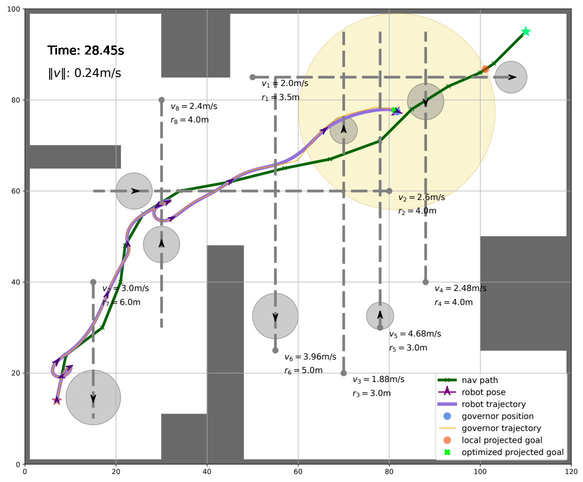
Figure 5.4. Snapshots of robot driving along the predefined path while avoiding collision of both static and dynamic obstacles by tracking the reference governor in a more complex environment with a ggoal position tracking low-level controller, the safe metrics value are also displayed.

Out of eight dynamic obstacles put in the environment, the robot-governor system had interactions with seven of them and was able to avoid collision with them successfully. The dynamic obstacles were traveling with different linear velocities and moving towards the system

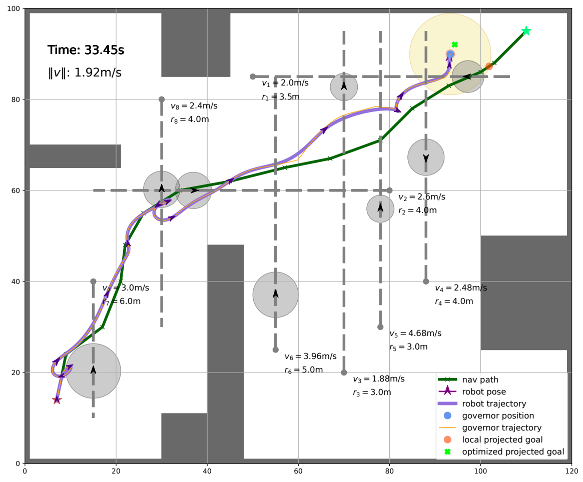
Figure 5.4. Continued.



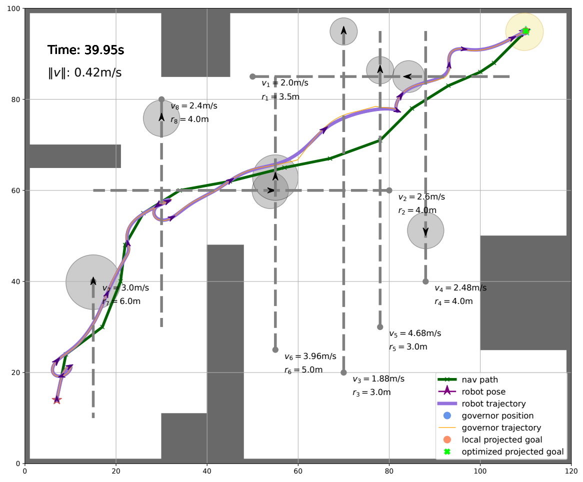
(e) 5th Avoidance



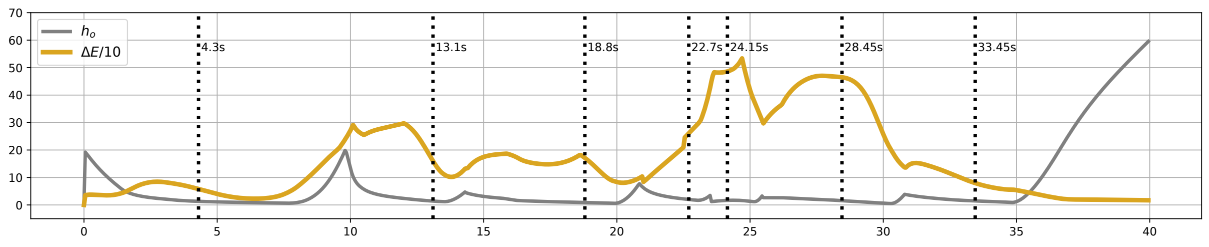
(f) 6th Avoidance



(g) 7th Avoidance



(h) Goal Reached

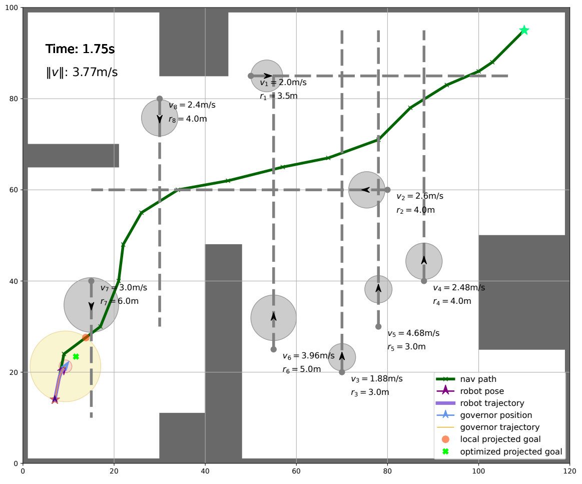


(i) Safe Metrics Value over Time

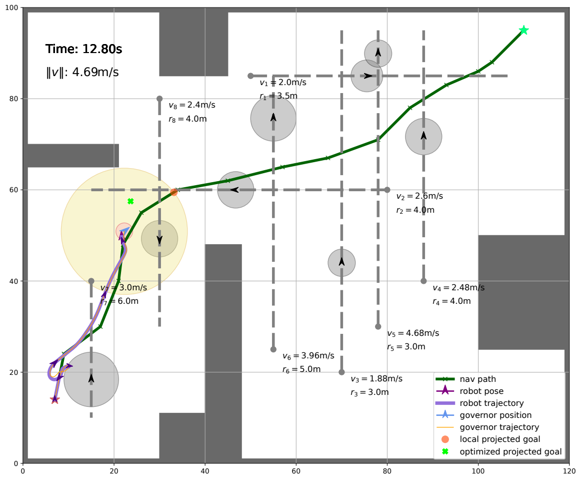
from different directions. The system acted as ”slowing down”, ”stepping back” or ”overtaking” while running into dynamic obstacles from its front, left or right side.

Complex environment with goal pose alignment controller

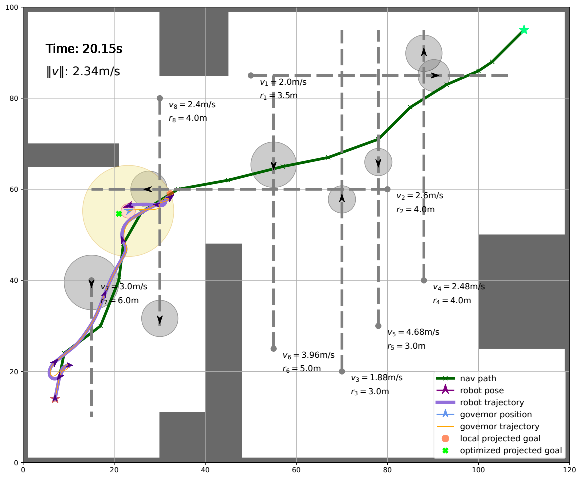
Another simulation we did used the same environment and input constraints, but the robot was driven by the goal alignment controller (2.16) introduced in Sec. 2.2.1. The governor heading was set to always pointing at the static local projected goal, i.e. $g_\theta = \arctan 2(\bar{\mathbf{u}}_{\mathbf{g}y}, \bar{\mathbf{u}}_{\mathbf{g}x})$. Selected snapshots are displayed in Fig. 5.5 with the safe metric values over time, showing the dynamic obstacle avoidance motions.



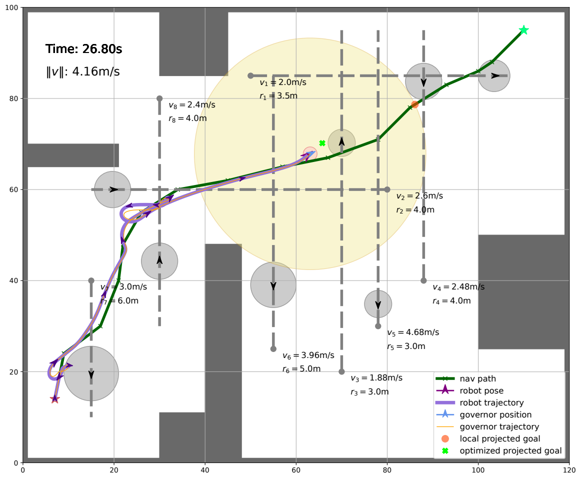
(a) 1st Avoidance



(b) 2nd Avoidance



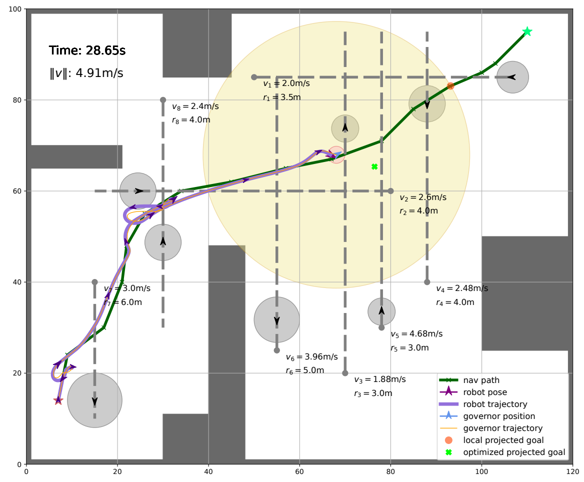
(c) 3rd Avoidance



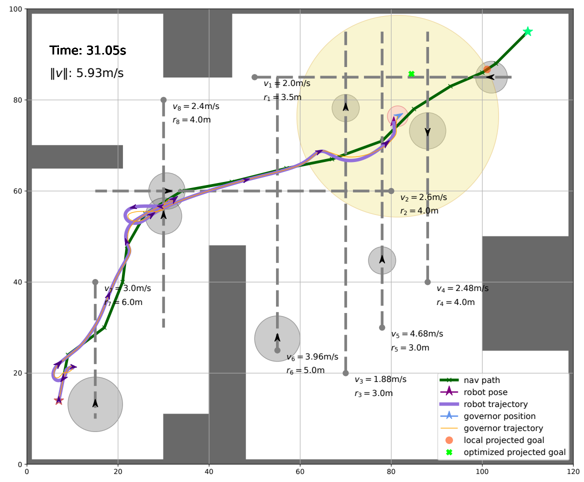
(d) 4th Avoidance

Figure 5.5. Snapshots of robot driving along the predefined path while avoiding collision of both static and dynamic obstacles by tracking the reference governor in a more complex environment with a goal pose alignment low-level controller, the safe metrics value are also displayed.

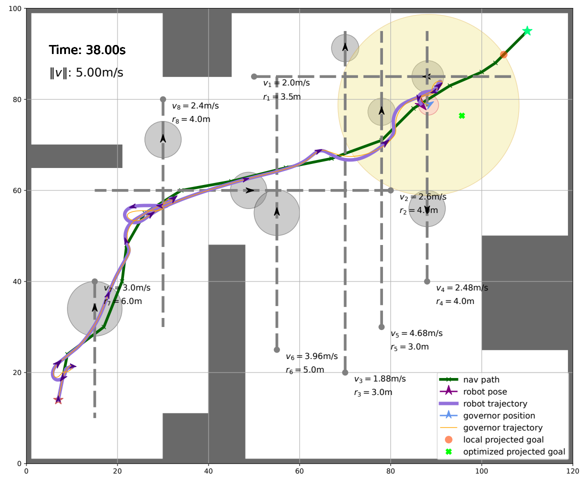
Figure 5.5. Continued.



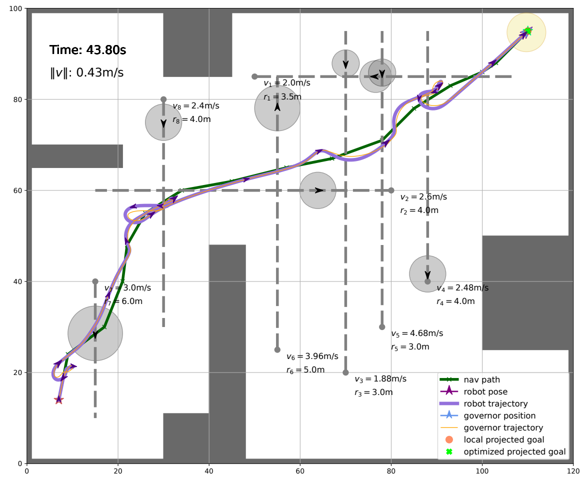
(e) 5th Avoidance



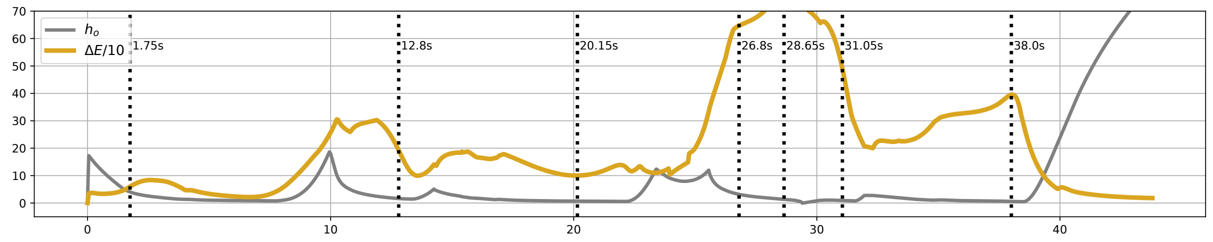
(f) 6th Avoidance



(g) 7th Avoidance



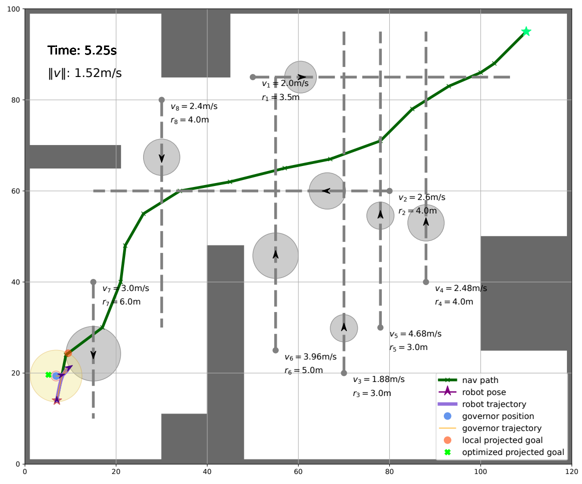
(h) Goal Reached



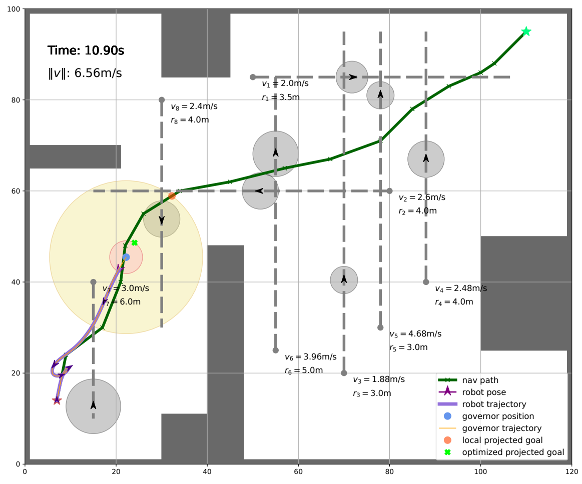
(i) Safe Metrics Value over Time

As demonstrated in Fig. 5.5, the robot-governor system could also successfully complete the dynamic obstacle avoidance task. With the goal pose alignment controller, the robot was actively tracking the goal heading. This action may be useful when the given path was generated from a $SE(2)$ planner, i.e. the waypoints along the path are formed as 2D poses other than only 2D positions.

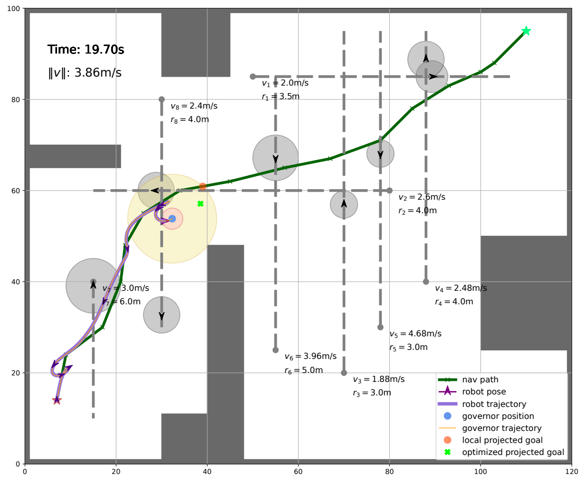
Complex environment with bi-directional controllers



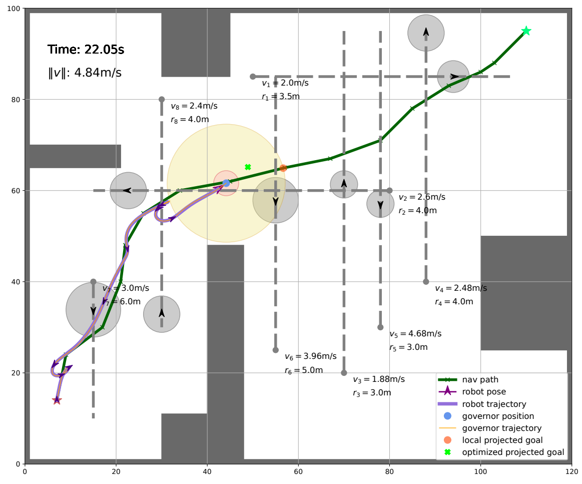
(a) 1st Avoidance



(b) 2nd Avoidance



(c) 3rd Avoidance

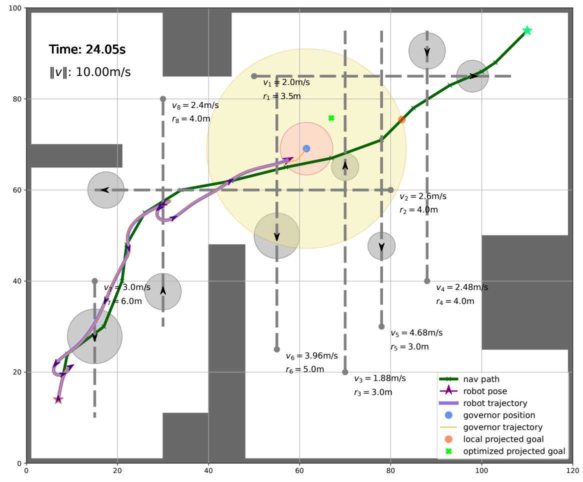


(d) 4th Avoidance

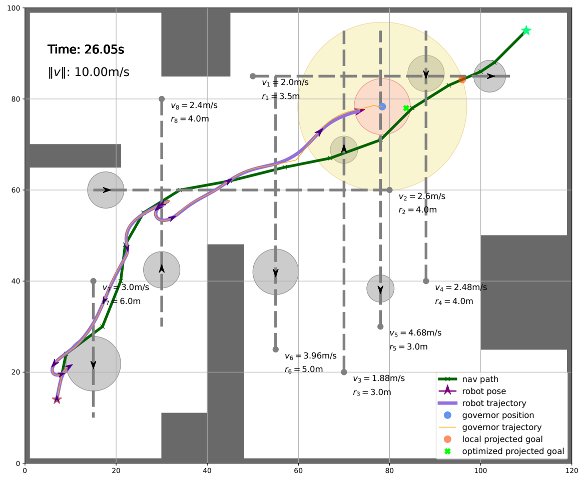
Figure 5.6. Snapshots of robot driving along the predefined path while avoiding collision of both static and dynamic obstacles by tracking the reference governor in a more complex environment with a bi-directional goal position tracking low-level controller, the safe metrics value are also displayed.

To make the goal tracking more efficient and its trajectory more smooth, we designed the low-level controller to be bi-directional in Sec. 4.6. The simulation results with a bi-directional goal position tracking controller is displayed in Fig 5.6, and the results with the bi-directional

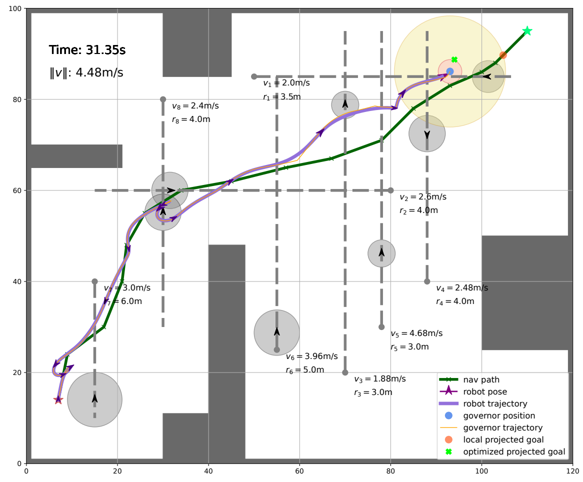
Figure 5.6. Continued.



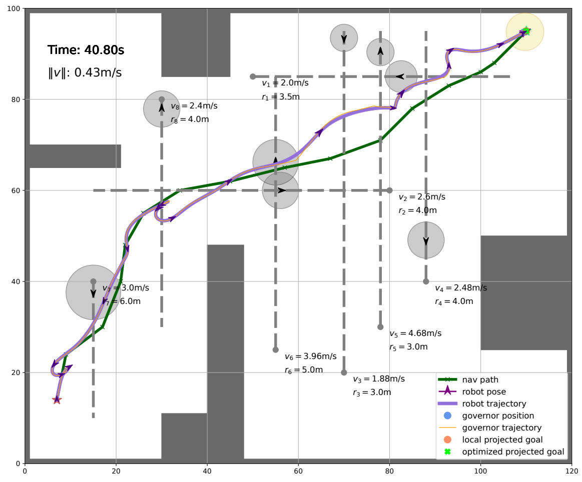
(e) 5th Avoidance



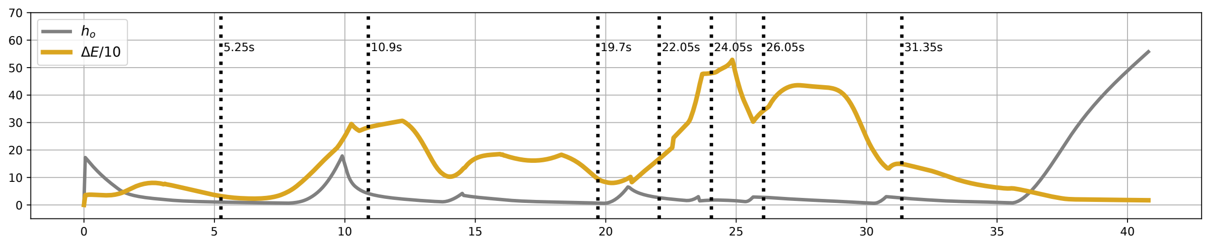
(f) 6th Avoidance



(g) 7th Avoidance

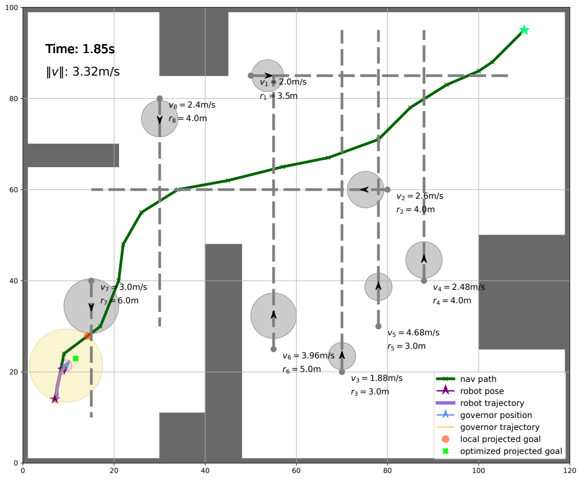


(h) Goal Reached

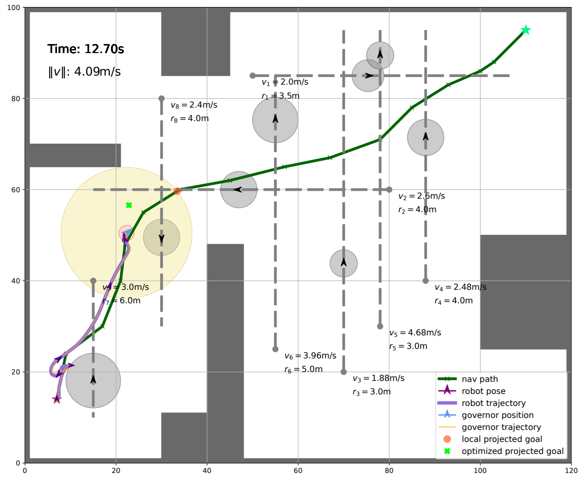


(i) Safe Metrics Value over Time

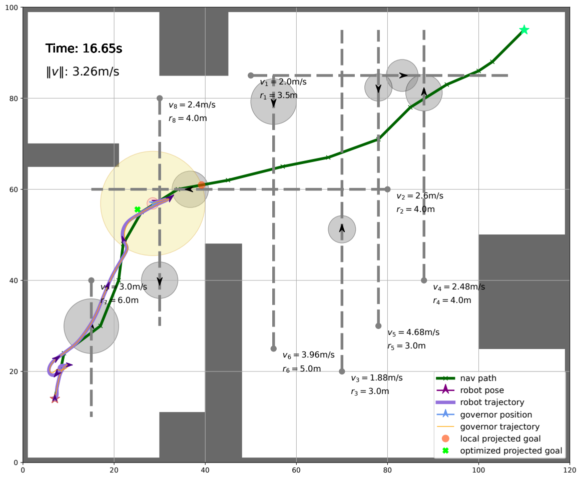
goal pose alignment controller is displayed in Fig 5.7



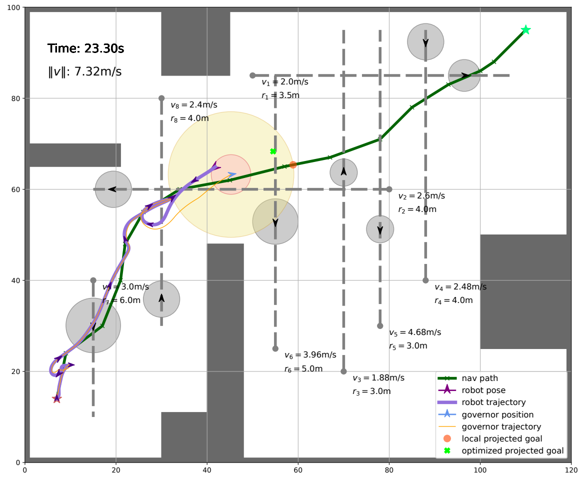
(a) 1st Avoidance



(b) 2nd Avoidance



(c) 3rd Avoidance



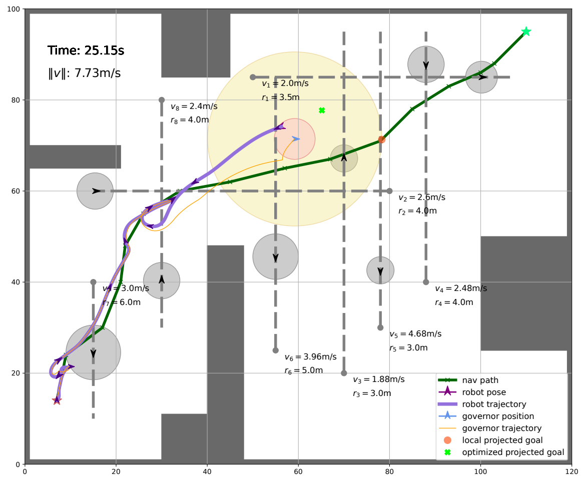
(d) 4th Avoidance

Figure 5.7. Snapshots of robot driving along the predefined path while avoiding collision of both static and dynamic obstacles by tracking the reference governor in a more complex environment with a bi-directional goal pose alignment low-level controller, the safe metrics value are also displayed.

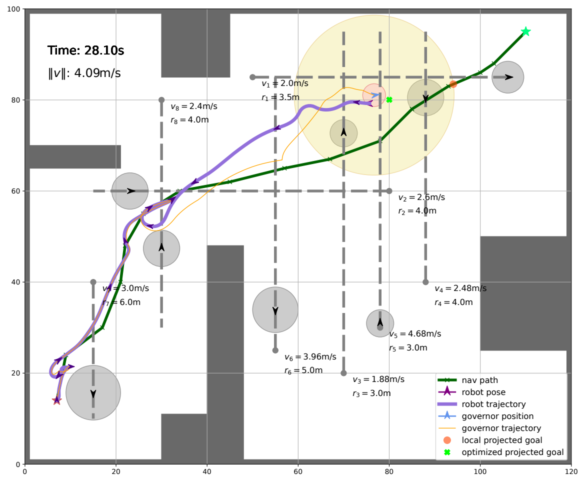
With the position tracking controller, the robot's driving direction was changed as the first (Fig. 5.6a) and third (Fig. 5.6c) avoidance happened, and backwards motion can be observed at the time period in between. While with the pose alignment controller, the direction change happened when reacted to the third dynamic obstacle (Fig. 5.7c), and the robot kept driving backwards until goal reached.

Adding the bi-directional motion mode can reduce failures like de-localization happening during the complete navigating process while working on the real robot or in a virtual simulation

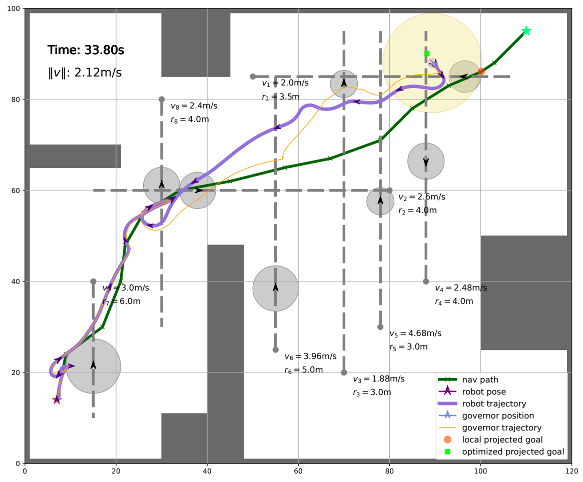
Figure 5.7. Continued.



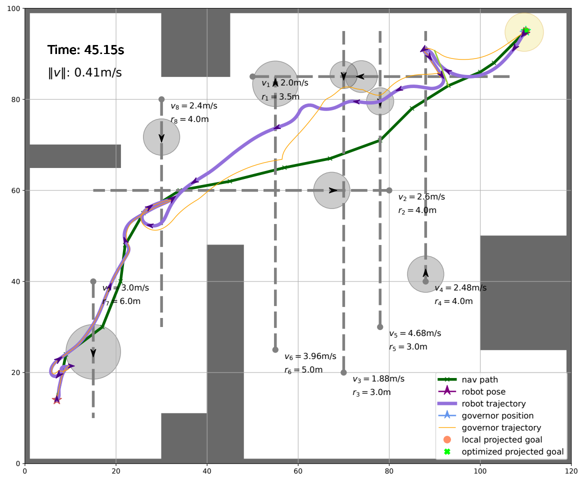
(e) 5th Avoidance



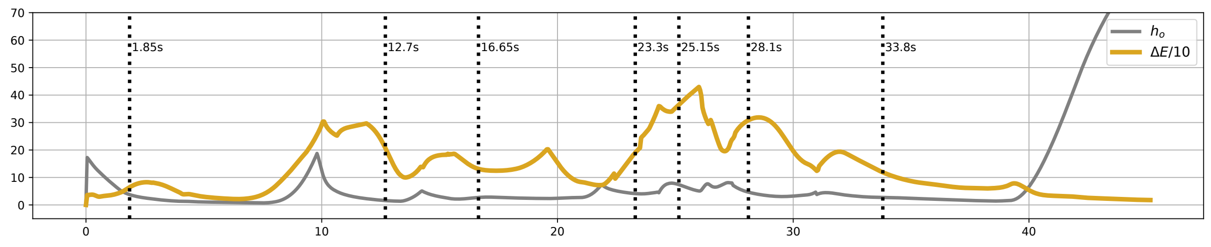
(f) 6th Avoidance



(g) 7th Avoidance



(h) Goal Reached



(i) Safe Metrics Value over Time

environment like Gazebo. Since it allows the robot to simply switch direction without spinning and thus prevent its wheels from sliding or slipping while the road surface condition is not ideal.

5.2 Real World Experiments

After getting the simulation results under different conditions, we deployed the control policy designed in this thesis with a series of other robot self-navigation packages on a wheeled mobile robot to evaluate its obstacle avoiding performance in the real world. This section introduces the experiment's hardware and software configuration, experiment setup, and details of the control module.

5.2.1 Hardware Overview

We chose *ClearPath* Jackal Unmanned Ground Vehicle (UGV) as our experimental platform. Jackal is a small field robotics research platform with onboard computer and customizable sensor configuration such as GPS, IMU and LiDAR. It is Robot Operating System (ROS) compatible, so we can quickly test and debug our algorithm in simulation and deploy it to hardware.

To simplify the perception module and get a more accurate dynamic obstacle information, we use the Aerodrome facility provided by *UCSD-Contextual Robotics Institute* to generate the odometry message of dynamic obstacles. It runs the Vicon motion capture system to catch the objects' current pose, with a group of markers attached on the objects' surfaces.

Mobile platform

Jackal UGV is a differential drive robot with four equivalent wheels, two on each side running at same speed. The external dimensions are $508 \times 430 \times 250$ mm. It weighs about 17 kg and can run at top speed 2.0 m/sec.

Computation, sensing and communication

Our customized Jackal is equipped with an Intel i7-9700TE CPU with 32GB RAM, an Ouster OS1-32 LiDAR and a UM7 9-axis IMU. This supplies sufficient computation power and sensing capability for autonomous navigation tasks. The UGV can be manually controlled by a Bluetooth joystick or remotely accessed by onboard Wi-Fi. We use the joystick to trigger our

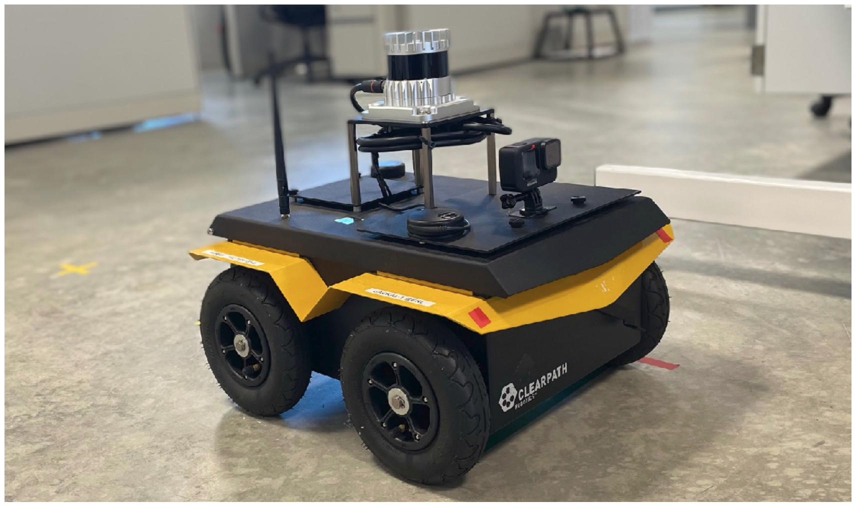


Figure 5.8. Jackal UGV.

problem at the beginning and use it as an emergency stop controller. A local network is setup via a standard router, so we can use visualization tools that come with ROS development kit to monitor running status of our algorithm and save ROS bag for replay and debugging. The Vicon system runs in the same local network, so that it can broadcast dynamic obstacles' odometry messages to Jackal. In this way, the dynamic obstacles' motions can also be monitored by the same visualization interface as for other features of the program.

5.2.2 Software Architecture

Other than the control module we introduced above in this thesis, to complete the autonomous navigation experiment, there's more software modules used. The overall software components can be grouped into three categories: Localization and Mapping, Planning and Control. We use Robot Operation System (ROS) as the framework for all high-level tasks running on the robot. Fig. 5.9 is an overview of our system architecture.

We also use the out-of-box low-level velocity controller provided by *Clearpath Robotics* which takes linear and angular velocity command and transforms them to motor input signals.

Localization and Mapping

For localization and mapping task, we use an open-source LiDAR-based 2D SLAM algorithm called Hector SLAM [10]. This algorithm takes 2D laser scan as inputs and outputs robot's poses (position and orientation) in an online constructed occupancy grid map. We use an ROS package named `pointcloud_to_laserscan` to convert 3D pointcloud from Ouster LiDAR to 2D laser scan.

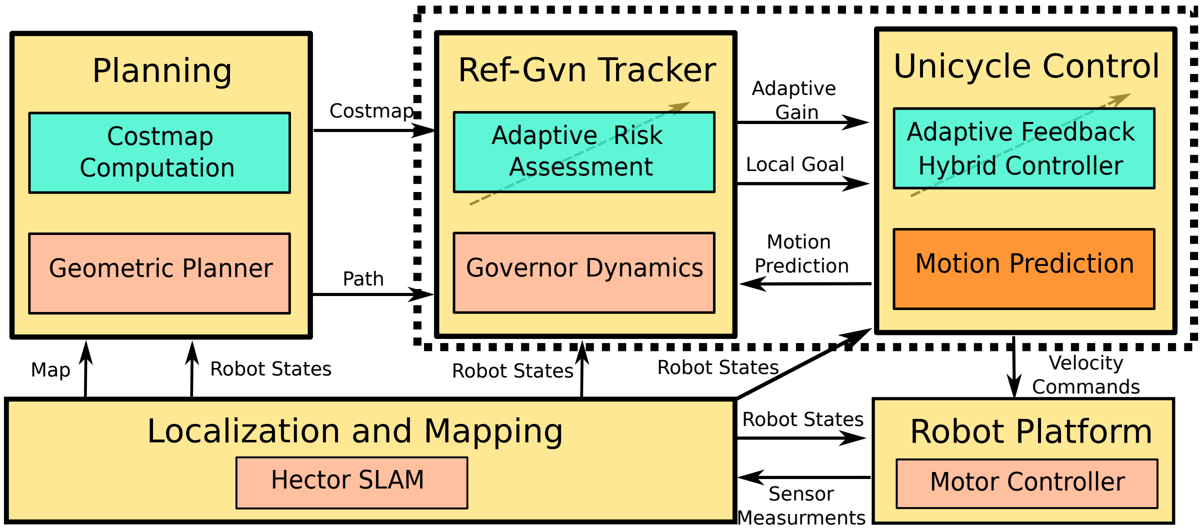


Figure 5.9. System architecture overview.

Planning

The planning module consists of two parts, a costmap computation block and a standard geometric planner. We choose a classical searched-based planning algorithm named A^* as our 2D geometric planner. Instead of searching for an optimal path (in terms of travel distance) over binary occupancy grid map, we create a distance-field based costmap as input for the planner. The customizable costmap can be easily tuned to generate optimal paths from max-clearance path to shortest travel distance path, leaving practitioner the freedom to balance safety and efficiency.

5.2.3 Experiment Setup Overview

Environment setup

The experiment was conducted on the Aerodrome test ground, it is a 45’ x 55’ x 35’ open space located on the UCSD campus. We put different sizes of boxes on the ground to stand as static obstacles Fig. 5.10 shows a layout of the static environment and the origin axes, also the robot’s initial position, we started with. **G1** to **G4** denoted four goal positions we picked.

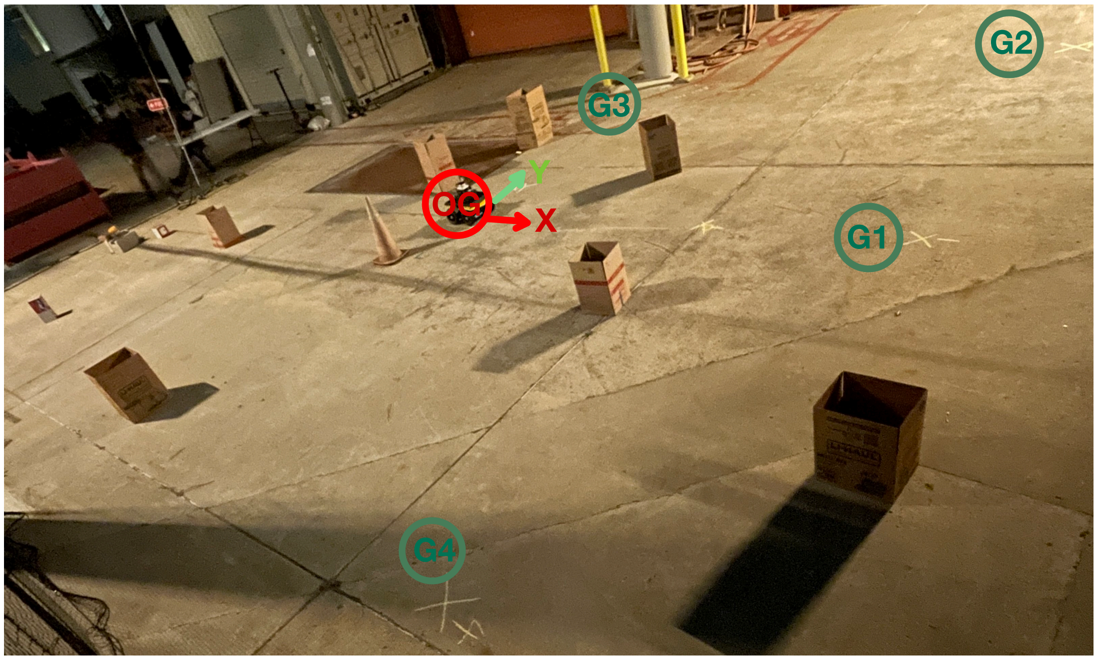
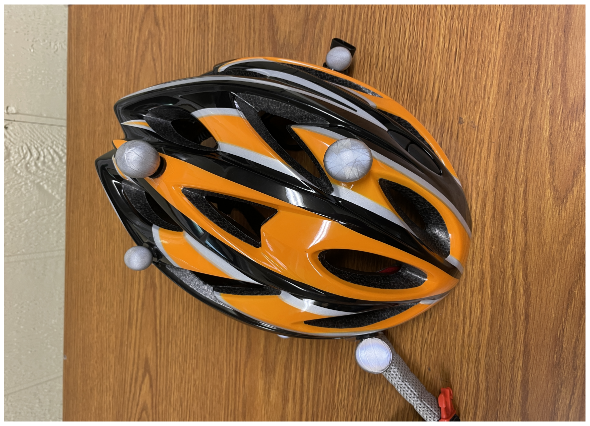


Figure 5.10. Layout of real world experiment static environment.

For dynamic obstacles, we used helmets to carry the markers and invited some people to wear them to move around in the Aerodrome region so they can get detected by the Vicon system. To represent a high speed dynamic obstacle, we asked people to ride on a scooter also with the helmet on and drive around. An example of marked helmet and how people used it to play dynamic obstacles on the test ground is displayed in Fig. 5.11

Experiment design

We prepared two tests with the same static environment but different dynamic obstacle actions. For each test, the whole navigation process was broke down to five segments, and can


(a) Helmet with markers.

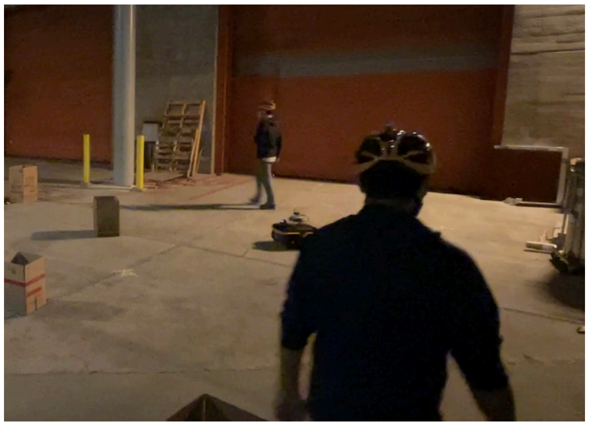

(b) People in helmet.

Figure 5.11. Example of dynamic obstacle setup.

be represented as:

$$\mathbf{OG} \rightarrow \mathbf{G1} \rightarrow \mathbf{G2} \rightarrow \mathbf{G3} \rightarrow \mathbf{G4} \rightarrow \mathbf{OG} \tag{5.1}$$

For the first test, we have two people walk around with different speed, and for the second test, we want one people to ride with the scooter and the other people to run so we can get high speed dynamic obstacles. In both of the tests, all the dynamic obstacles were randomly passing the robot on its path from different directions with random angles. We want to keep the dynamic obstacles' motion mostly natural but can still create some moments that the robot can run into the obstacles right in face.

5.2.4 Control Module

The final implemented control algorithm in the experiment includes two parts. One is the reference governor system as introduced in Sec. 4.1, but use a different robot prediction set to generate the local safe zone. The other part is a safety filer used on the governor's local projected goal to actively search an optimized projected goal that can respond to surrounding dynamic obstacles. The filter is constructed with the QCQP based on CBF constraints as introduced in Sec. 4.3 and Sec. 4.4.

Safety metric with ice-cream cone prediction set

Following the same notion as in Sec. 4, for a robot state $\mathbf{x}(t)$, while heading to the governor position $\mathbf{g}(t)$, its predicted trajectory bound can be defined as a time-varying set $\delta(\mathbf{x}(t), \mathbf{g}(t))$. Here instead of a Euclidean ball computed via a Lyapunov function (4.6), we used an ice-cream cone shape prediction set $\mathscr{M}(\mathbf{x}, \mathbf{g})$ as introduced in [21]. The corresponding safety energy function $\Delta E(\mathbf{x}, \mathbf{g})$ hereby can be changed to a measurement from the prediction set to its surrounding static obstacles. However, as the noisy estimation of the environment may cause potential collision at the obstacles' boundaries on the robot navigation process. The original obstacle set $\mathscr{O}$ was inflated into $\mathscr{O}^+$ with stepped cost values as explained in Sec. 5.2.2, and the safety margin is defined as:

$$\Delta E(\mathbf{x}, \mathbf{g}) := d(\mathscr{M}(\mathbf{x}, \mathbf{g}), \mathscr{O}^+) \tag{5.2}$$

where $d(\mathscr{A}, \mathscr{B})$ defines the distance between two sets as:

$$d(\mathscr{A}, \mathscr{B}) := \inf_{a \in \mathscr{A}, b \in \mathscr{B}} \|a - b\| \tag{5.3}$$

Hyper-parameter tuning

As the surface of the test ground is pretty rough, we need to limit the robot's linear and angular accelerations as well as velocities to reduce the possibility of wheel slipping, and hence prevent problems like localization failed from happening. However, the speed limit can not be too conservative as the large saturation could limit the robot's obstacle avoidance ability. Thus we constrain the governor speed $\dot{\mathbf{g}}$ by limiting the distance between the governor goal $\bar{\mathbf{g}}_m$ and the governor $\mathbf{g}$ within one meter. Moreover, we limit the distance between the governor position and the robot position also to be within one meter, so that the size of the robot-governor set defined as (4.13) won't become too large and result in making the robot too sensitive to the dynamic obstacles.

5.2.5 Testing Results Visualization

As described in Sec. 5.2.3, we run two tests with different dynamic obstacle motions and observe the robot-governor system's avoidance actions through the visualization tools.

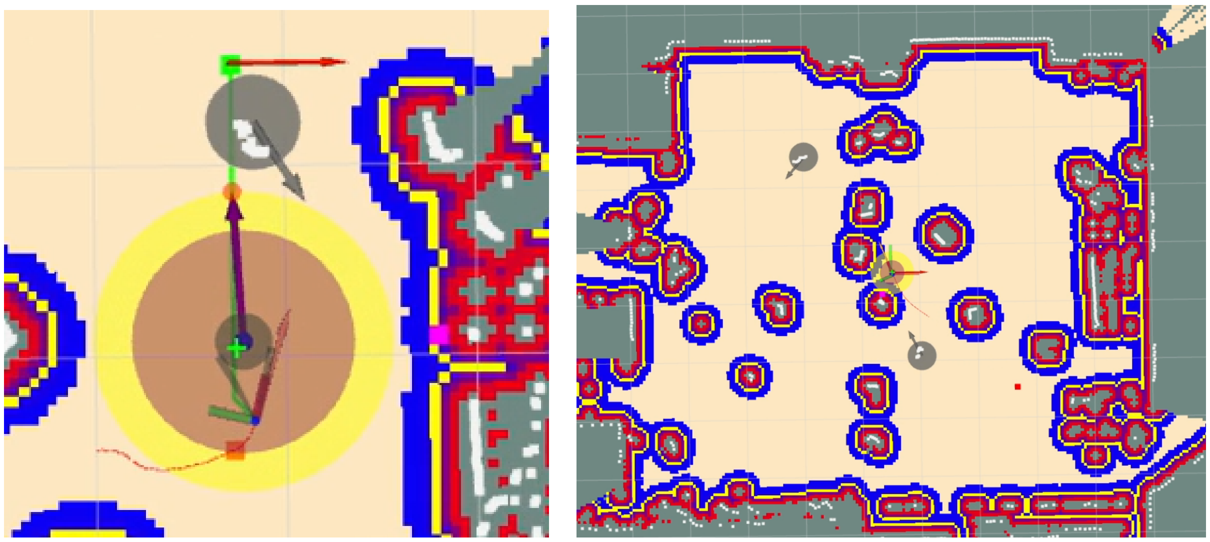

(a) System sets visualization. **(b)** Map visualization.

Figure 5.12. The robot-governor system and all sets visualized with the testing environment layout; the static environment was displayed as a costmap and dynamic obstacles were displayed as gray circles with arrows pointing their velocity headings.

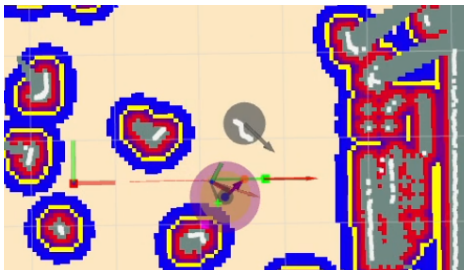
The robot-governor system and all the corresponding sets are displayed in Fig. 5.12a. The green cone shape group marker denotes the ice-cream cone shaped robot prediction set, the purple circle represents the robot-governor set, and the yellow circle shows the local safe zone. As for individual states, the blue dot shows the governor position, the orange dot denotes the local projected goal generated from the local safe zone, and the light green cross represents the active projected goal corresponding to dynamic obstacles. The robot is presented as an axes and its short-time trajectory is shown with a thin red line. The light-green line segment represents the planner generated path. In 5.12b, the layout of the static environment was modeled as a costmap, with the gray cells being the actual obstacle covered cells and the light yellow cells being the free cells. Cells colored otherwise are inflated contours of the static obstacles assigned with different costs, varying with the cell's distance to obstacle bounds.

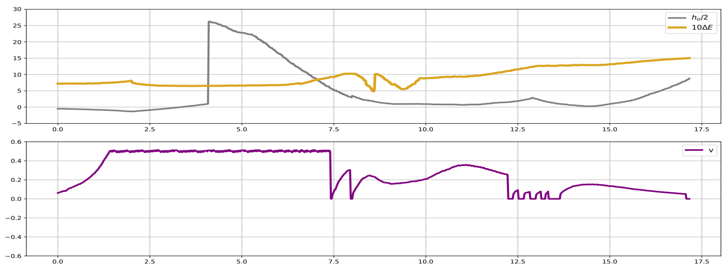
Low speed dynamic obstacles

In this test, two people played low speed dynamic obstacles with one of them walking in a ”busy mode” and another walking in a ”casual mode”. In the sense of CBF value computation, we define the radius of theses two dynamic obstacle sets both as 0.5 meter. Fig. 5.13 to Fig. 5.17 below display selected dynamic obstacle avoidance motions on each path segment as described in Sec. 5.2.3.

(a) Robot run into dynamic obstacle (before avoiding).

(b) Robot run into dynamic obstacle (after avoiding).

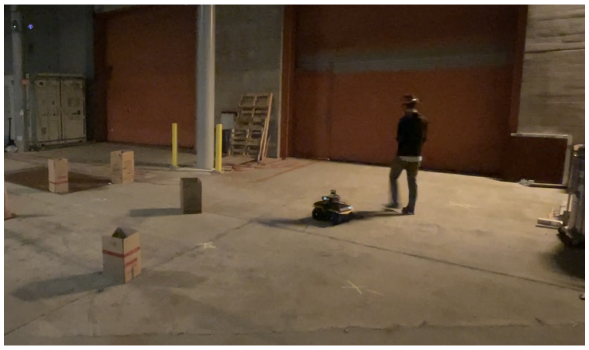

(c) Avoidance action shown in visualization tool.

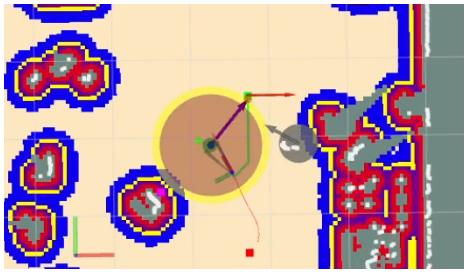


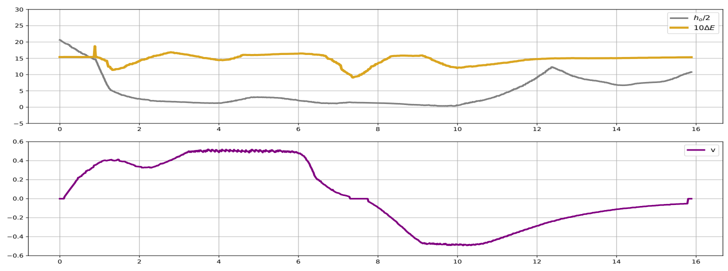
(d) Safety metric value and robot linear velocity over time.

Figure 5.13. Dynamic obstacle avoidance example on path from **OG** to **G1**.

(a) Robot run into dynamic obstacle (before avoiding).

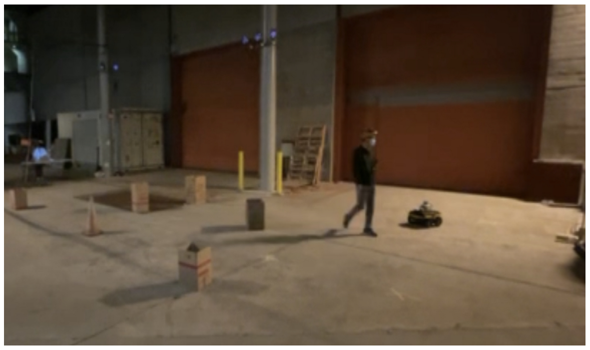

(b) Robot run into dynamic obstacle (after avoiding).


(c) Avoidance action shown in visualization tool.

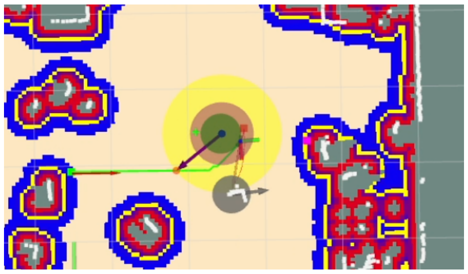


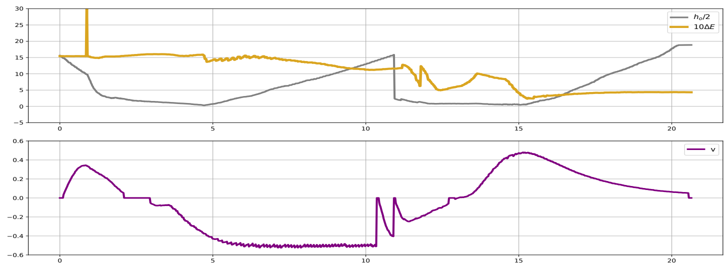
(d) Safety metric value and robot linear velocity over time.

Figure 5.14. Dynamic obstacle avoidance example on path from **G1** to **G2**.


(a) Robot run into dynamic obstacle (before avoiding).

(b) Robot run into dynamic obstacle (after avoiding).

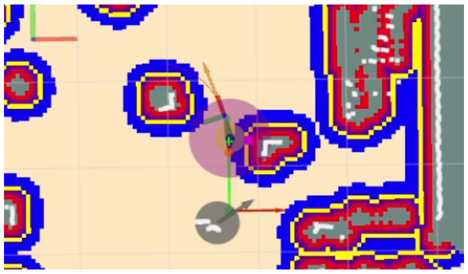

(c) Avoidance action shown in visualization tool.



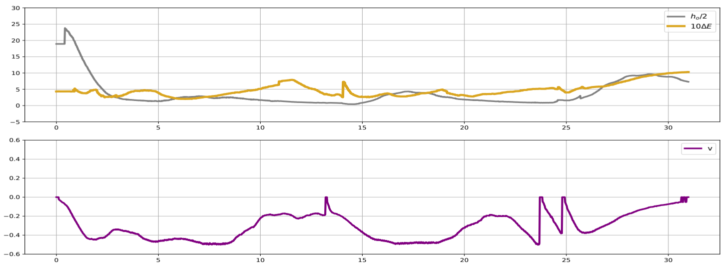
(d) Safety metric value and robot linear velocity over time.

Figure 5.15. Dynamic obstacle avoidance example on path from **G2** to **G3**.

(a) Robot run into dynamic obstacle (before avoiding).

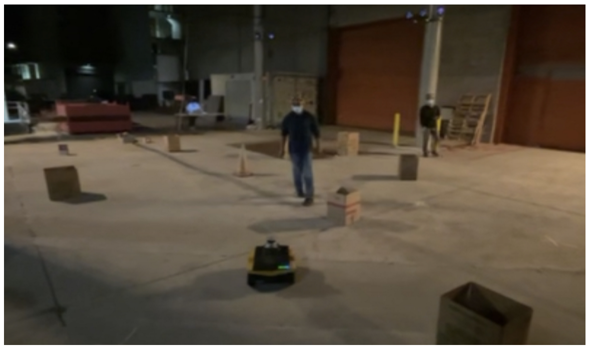


(b) Avoidance action shown in visualization tool.



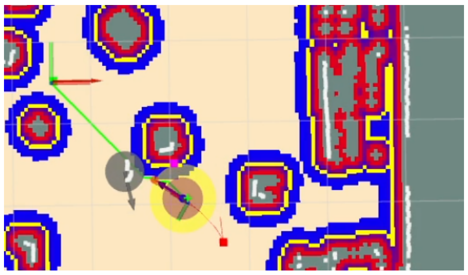
(c) Safety metric value and robot linear velocity over time.

Figure 5.16. Dynamic obstacle avoidance example on path from **G3** to **G4**.

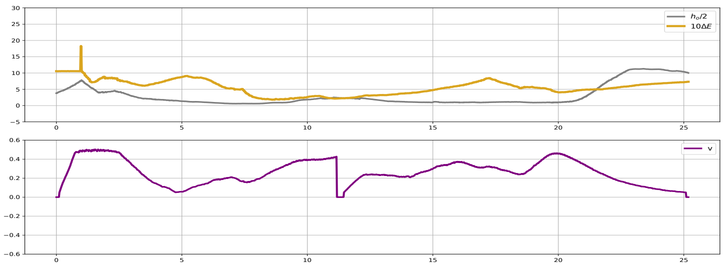


(a) Robot run into dynamic obstacle (before avoiding).

(b) Robot run into dynamic obstacle (after avoiding).



(c) Avoidance action shown in visualization tool.



(d) Safety metric value and robot linear velocity over time.

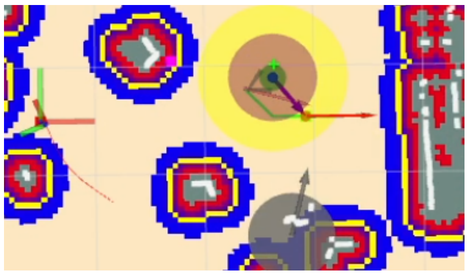
Figure 5.17. Dynamic obstacle avoidance example on path from **G4** to **OG**.

High speed dynamic obstacles

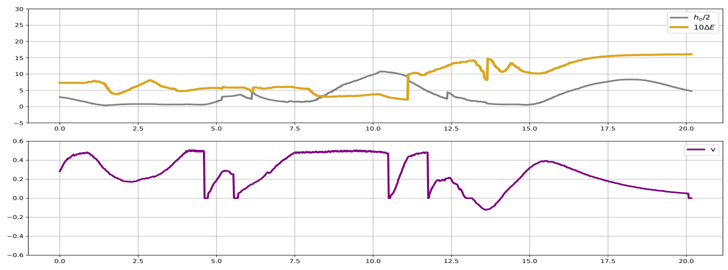
In this test, one people was jogging and the other one was riding a scooter. As the scooter covers more space than a single person, we change the radius of the second obstacle to 0.8m and the other one stays as 0.5m. Fig. 5.13 to Fig. 5.17 below display selected dynamic obstacle avoidance motions on each path segment as described in Sec. 5.2.3.

(a) Robot run into dynamic obstacle (before avoiding).

(b) Robot run into dynamic obstacle (after avoiding).



(c) Avoidance action shown in visualization tool.

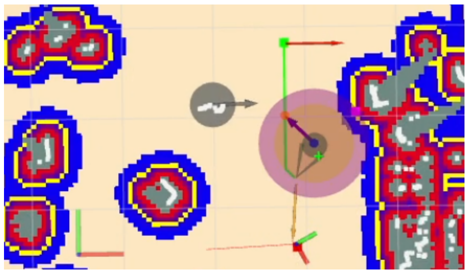


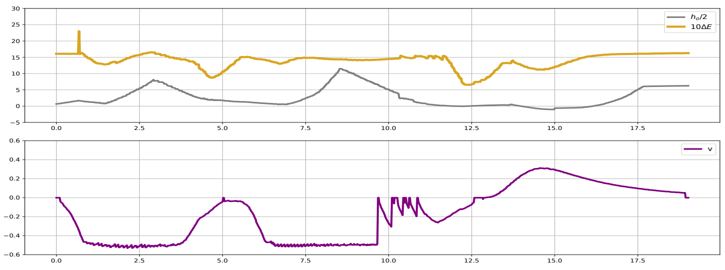
(d) Safety metric value and robot linear velocity over time.

Figure 5.18. Dynamic obstacle avoidance example on path from **OG** to **G1**.

(a) Robot run into dynamic obstacle (before avoiding).

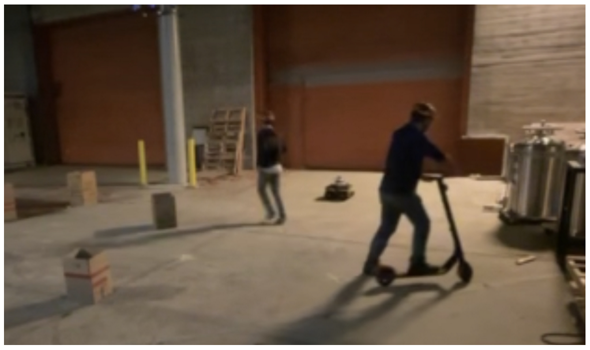
(b) Robot run into dynamic obstacle (after avoiding).


(c) Avoidance action shown in visualization tool.

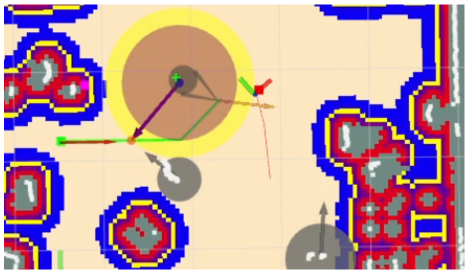


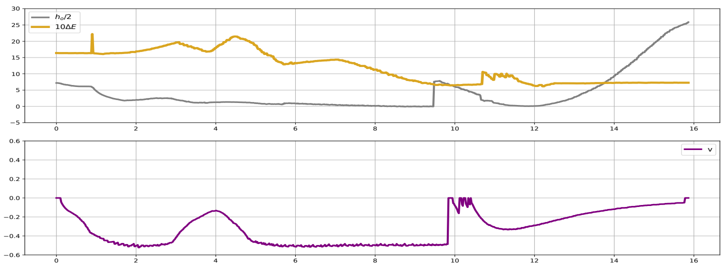
(d) Safety metric value and robot linear velocity over time.

Figure 5.19. Dynamic obstacle avoidance example on path from **G1** to **G2**.


(a) Robot run into dynamic obstacle (before avoiding).

(b) Robot run into dynamic obstacle (after avoiding).

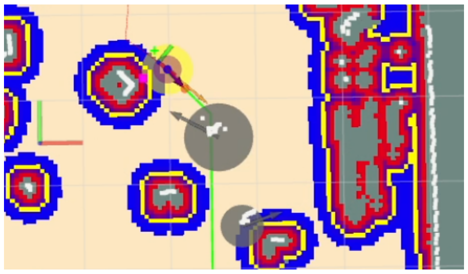

(c) Avoidance action shown in visualization tool.

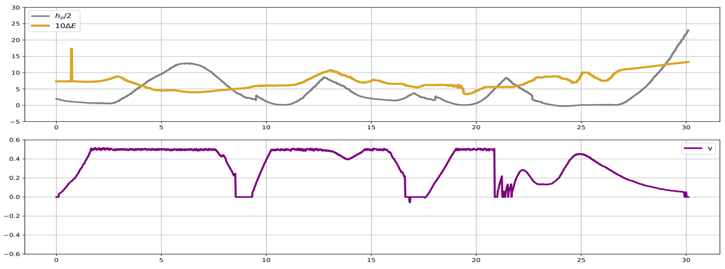


(d) Safety metric value and robot linear velocity over time.

Figure 5.20. Dynamic obstacle avoidance example on path from **G2** to **G3**.

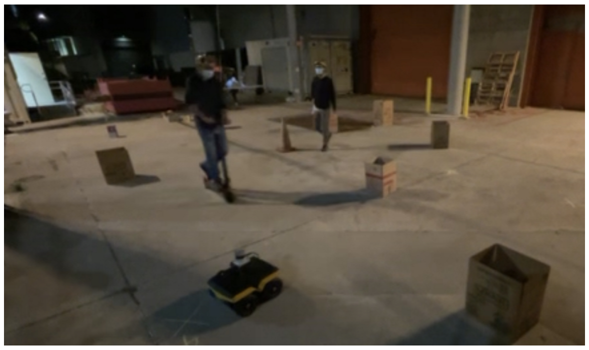
(a) Robot run into dynamic obstacle (before avoiding).


(b) Avoidance action shown in visualization tool.

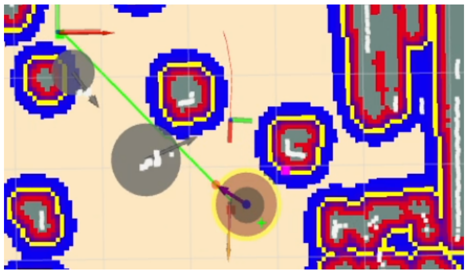


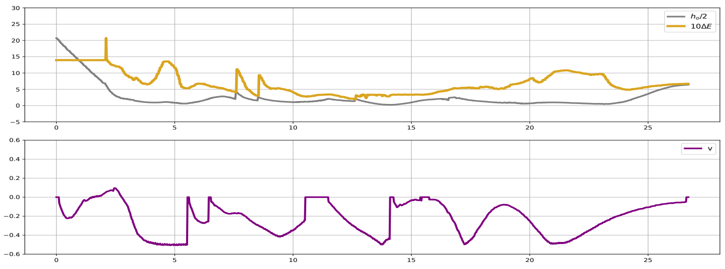
(c) Safety metric value and robot linear velocity over time.

Figure 5.21. Dynamic obstacle avoidance example on path from **G3** to **G4**.


(a) Robot run into dynamic obstacle (before avoiding).

(b) Robot run into dynamic obstacle (after avoiding).


(c) Avoidance action shown in visualization tool.



(d) Safety metric value and robot linear velocity over time.

Figure 5.22. Dynamic obstacle avoidance example on path from **G4** to **OG**.

Chapter 6

Concluding Remarks

6.1 Conclusion

In this thesis, we demonstrated a safe control policy designed from applying CBFs to a robot-governor augmented system, and got validated results both in the simulated environments and in a real world experiment. The algorithm was able to navigate the autonomous driving robot in a dynamic environment to follow a pre-generated path while stay safe. It also completed the task of reacting to diverse moving obstacles properly, though the convergence of the robot position to a global goal requires mild assumptions for the dynamic obstacles.

6.2 Future Work

There are potential extensions from the work we have done in this thesis. One of them can be to practice a more complex emergency maneuver control by modifying the QCQP's cost or CBFs' formulation. We are currently searching the active governor projected goal only in a ball set, and this ignores the directional information of the system's surrounding static obstacles, so it will be more helpful if the shape of the searching set can be refined to also include relative distance or direction measurements of different obstacle bounds. As a result, the controller can be improved with a preferred avoiding direction, which deviates from nearby static obstacles, while reacting to the coming dynamic obstacle. On the experiment side, the moving obstacle detection and odometry information generation task can be handled by an on-board sensor such as LiDAR and depth cameras. This will allow the robot to drive in an ordinary space and test with various natural scenarios. Besides, with the pose alignment low-level controller we studied, it can be beneficial to use it with an SE(2) planner to complete more complex path following tasks and eventually get a better performance while navigating in a clustered environment.

Bibliography

[1] M Aicardi, G Casalino, A Biechi, and A Balestrino. Closed loop steering of unicycle-like vehicles via.

[2] Khawla Almazrouei, Ibrahim Kamel, and Tamer Rabie. Dynamic obstacle avoidance and path planning through reinforcement learning. *Applied Sciences*, 13(14), 2023.

[3] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431, 2019.

[4] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006.

[5] Fernando S. Barbosa and Jana Tumova. Risk-aware navigation on smooth approximations of euclidean distance fields among dynamic obstacles. QC 20220112.

[6] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[7] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial intelligence review*, 43:55–81, 2015.

[8] Paul Glotfelter, Jorge Cortés, and Magnus Egerstedt. Boolean composability of constraints and control synthesis for multi-robot systems via nonsmooth control barrier functions. In *2018 IEEE Conference on Control Technology and Applications (CCTA)*, pages 897–902, 2018.

[9] Johan Karlsson, Nikolce Murgovski, and Jonas Sjöberg. Temporal vs. spatial formulation of autonomous overtaking algorithms. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1029–1034. IEEE, 2016.

[10] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, 2011.

[11] Ilya Kolmanovsky, Emanuele Garone, and Stefano Di Cairano. Reference and command governors: A tutorial on their theory and automotive applications. In *2014 American Control Conference*, pages 226–241, 2014.

[12] Zhichao Li, Thai Duong, and Nikolay Atanasov. Safe robot navigation in cluttered environments using invariant ellipsoids and a reference governor. *arXiv preprint arXiv:2005.06694*, 2020.

[13] Zhichao Li, Ömür Arslan, and N. Atanasov. Fast and Safe Path-Following Control using a State-Dependent Directional Metric. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[14] Shikhar Singh Lodhi, Neetesh Kumar, and Pradumn Kumar Pandey. Autonomous vehicular overtaking maneuver: A survey and taxonomy. *Vehicular Communications*, 42:100623, 2023.

[15] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[16] Makoto Obayashi, Keisuke Uto, and Gaku Takano. Appropriate overtaking motion generating method using predictive control with suitable car dynamics. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4992–4997, 2016.

[17] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[18] Ugo Rosolia, Stijn De Bruyne, and Andrew G Alleyne. Autonomous vehicle control: A nonconvex approach for obstacle avoidance. *IEEE Transactions on Control Systems Technology*, 25(2):469–484, 2016.

[19] Hairong Xiao, Li Liao, and Fengyu Zhou. Mobile robot path planning based on q-ann. In *2007 IEEE International Conference on Automation and Logistics*, pages 2650–2654, 2007.

[20] Ömür Arslan. Time governors for safe path-following control, 2022.

[21] Aykut İşleyen, Nathan van de Wouw, and Ömür Arslan. Feedback motion prediction for safe unicycle robot navigation, 2023.