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

A GP-based Robust Motion Planning Framework for Agile Autonomous Robot Navigation and Recovery in Unknown Environments

Nicholas Mohammad, Jacob Higgins, Nicola Bezzo

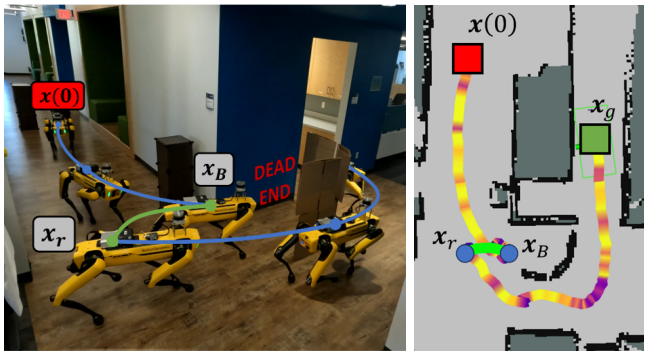
***Abstract*— For autonomous mobile robots, uncertainties in the environment and system model can lead to failure in the motion planning pipeline, resulting in potential collisions. In order to achieve a high level of robust autonomy, these robots should be able to proactively predict and recover from such failures. To this end, we propose a Gaussian Process (GP) based model for proactively detecting the risk of future motion planning failure. When this risk exceeds a certain threshold, a recovery behavior is triggered that leverages the same GP model to find a safe state from which the robot may continue towards the goal. The proposed approach is trained in simulation only and can generalize to real world environments on different robotic platforms. Simulations and physical experiments demonstrate that our framework is capable of both predicting planner failures and recovering the robot to states where planner success is likely, all while producing agile motion.**

I. Introduction

Robust motion planning for autonomous mobile robots (AMR) remains an open problem for the robotics community. One of the main challenges is to navigate through environments in the presence of uncertainty, like an unknown map a priori or inaccurate system models. For example, this lack of robustness was clearly evidenced at the ICRA BARN challenge [1], [2], in which no team was able to navigate a robot through an unknown, cluttered environment without any collisions[1]. Within the navigation stack, the cause of such runtime failures and possible collisions is typically attributed to the motion planning pipeline.

To prevent such situations, reactive approaches have been developed that detect potentially risky states as they occur [3]. These reactive approaches, however, suffer from poor performance because they are often tuned to be conservative and overly cautious, since it is better to actively avoid unsafe states before they occur. They also do not perform well for high-inertial systems which need an appropriately large reaction time in order to avoid collision. Alternatively, proactive approaches classify future robot states as safe or unsafe based on the current sensor readings and motion plan [4], [5]. These approaches often rely on complex deep learning models which require exhaustive real world training data to detect the safety of future states. Furthermore, these proactive approaches do not solve the problem of recovery after detecting such risky states in the planning horizon.

Consider the case in Fig. 1, where the robot is tasked with navigating quickly through an unknown, occluded environment. Without any proactive scheme, the robot suddenly recognizes the dead end, and does not have enough time to stop before collision. If these motion planning failures are proactively predicted for future states, then the robot can stop before crashing ($\boldsymbol{x}_B$ in Fig. 1), maneuver to a safe recovery point $\boldsymbol{x}_r$, then return to nominal planning towards the final goal. This exact test case will be discussed in Sec. VI, along with the experimental results.



(a) (b)

Fig. 1. Example in which an unexpected dead end could cause a motion planning failure. The proposed approach predicts the risk of such a failure and recovers before it can occur.

To achieve this behavior, in this work we propose a proactive- and recovery-focused approach that seeks to predict the risk of failure for a receding horizon, safe corridor motion planner, as well as recover from these potential failures. Such a planner is chosen due to its effectiveness at navigating unknown environments as well as its ubiquity within the robotics community. Additionally, we have found that such a planner requires a relatively small number of features to correctly classify potential failures. A Gaussian Process (GP) is trained on simulated data to predict failures along the planned receding horizon trajectory. When the predicted risk meets a certain threshold criterion, the robot is stopped and a recovery behavior is engaged. This process leverages the same GP to find a nearby safe state from which the robot can safely negotiate its immediate environment and continue motion towards its ultimate goal.

The contribution of this work is a complete and robust motion planning pipeline for robot navigation in unknown environments with two main innovations: 1) **a proactive planner failure detection scheme** in which a model agnostic, proactive GP-based approach detects and predicts future planning failures and their risk within a horizon, *without the need to retrain between simulation and the real world* and 2) **a robust recovery scheme** in which a GP-based, sampling-based recovery method drives the robot to a safe recovery point in order to continue with nominal planning.

II. Related Work

While motion planning is an active field of research within the robotics community, the problem of robust, agile

Nicholas Mohammad, Jacob Higgins, and Nicola Bezzo are with the Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22903, USA {nm9ur, jdh4je, nbezzo}@virginia.edu

[1]our team placed second in this competition

navigation through cluttered, unknown environments remains unsolved [1]. Many state-of-the-art motion planners impose hard constraints within a nonlinear optimization problem and use numerical solvers to generate the final trajectories within safe corridors [6], [7], [8]. However, random disturbances and occluded obstacles may cause constraint violations at runtime, leading to an inability to generate updated trajectories. [9] considers the potential for planner failure by generating an additional safe trajectory which stops within known free space at each planning iteration. However, they do not provide any recovery behaviors in case the vehicle is unable to find feasible trajectories at the stopping point. A popular alternative to the hard-constrained methods are soft-constrained planners, where the hard constraints are converted into differentiable terms and put into the cost function of an unconstrained nonlinear optimization problem [10], [11]. While the soft-constrained methods generate trajectories even when constraints aren't satisfied, conflicting terms within the cost function can lead to low quality solutions, i.e, unsafe or untrackable trajectories [12]. In this paper, we work with the hard-constrained motion planner paradigm and develop an algorithm to monitor for and recover from possible failures proactively at runtime.

Safety monitoring during runtime motion planning is a problem with a catalogue of potential solutions. One well studied technique is Hamilton-Jacobi-Isaacs (HJI) reachability analysis, where safe control is transformed into a formal verification method with theoretical safety guarantees. However, HJI reachability requires an accurate model of the system and suffers from the curse of dimensionality [13]. In order to overcome this problem, recent works have used machine learning techniques to approximate and learn from the generated reachable sets. [14] leverages Adaptive Kriging using a surrogate GP model and Monte Carlo sampling to approximate the sets at runtime. [15] uses a neural network trained on ground truth reachable sets to output binary safe/unsafe classifications for planned trajectories. While these works get around the intractability of runtime reachability analysis, they still rely on specific, accurate system models, limiting their generalizability.

Machine learning methods are also used to monitor vehicle safety outside of the reachability context, stopping the robot when anomalous states are detected. [4] and [5] proactively predict anomalous states which lead to collisions and stop the vehicle before reaching them. However, these works either implement trivial backup and rotate recovery behaviors with no consideration for planning success, or rely on humans to perform the recovery for them.

Our approach leverages machine learning techniques to monitor vehicle safety through planner failure detection. Specifically, we train on the distribution of failures over hand-selected input features which enable our approach to be model agnostic and require only training data from simulation. To the best of our knowledge, our work is the first to utilize a learning component to both proactively predict future planning failures and recover after prediction.

III. Problem Formulation

Given a mobile robot system tasked to navigate an unknown environment, let $\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}$ define the equations of motion for the system with state $\boldsymbol{x} \in \mathbb{R}^{n_x}$ and control inputs $\boldsymbol{u} \in \mathbb{R}^{n_u}$. These controls are produced by a low-level controller that is tracking a time-based trajectory $\boldsymbol{\tau}(t; t_0) \in \mathbb{R}^{n_x}$ generated at time t_0. The purpose of this trajectory is to provide a high-level path plan over a future planning horizon T_H from the current state of the robot $\boldsymbol{x}(t_0)$ towards a goal $\boldsymbol{x}_g$ while avoiding the state subset $\mathcal{X}_O(t_0)$ occupied by obstacles currently known to the robot. While tracking this trajectory, information about obstacles in the environment are updated at runtime so that, in general, $\mathcal{X}_O(t) \neq \mathcal{X}_O(t_0)$. This means the trajectory $\boldsymbol{\tau}(t; t_0)$ has the potential to collide with these newly discovered obstacles; if this is the case, then a new trajectory must be re-planned. Practical path planners, however, suffer from planning failures within certain situations due to infeasible constraints for the current planning iteration. While a single path-planning failure may not be fatal, several failures within the planning horizon could lead to unsafe situations for the robot.

Problem 1: ***Proactive Planner Failure Detection:*** Let $\{\hat{\boldsymbol{x}}_i\}$ be a set of predicted future states of the robot while tracking $\boldsymbol{\tau}$ over a control horizon $T_F \leq T_H$. For a given motion planning policy Π, define the random variable $Z_{\boldsymbol{\tau}} \in \mathbb{W}$ as the number of motion planning failures that occurs from t_0 to $t_0 + T_F$ while the robot tracks $\boldsymbol{\tau}$, with $P(Z_{\boldsymbol{\tau}})$ denoting their probabilities. We seek the creation of a risk metric $\rho_{\boldsymbol{\tau}} \in \mathbb{R}$ that maps from $Z_{\boldsymbol{\tau}}$ to a single real number that characterizes the risk of path planner failure over T_F.

Problem 2: ***Recovery After Failure Detection:*** We seek a recovery strategy $\Pi_r(\boldsymbol{x}_0)$ that, when the risk $\rho_{\boldsymbol{\tau}}$ exceeds a threshold ψ_ρ, stops the robot and performs a recovery behavior to reduce the risk of planner failure back down to an acceptable level. Specifically, define $Z_{\boldsymbol{x}} \in \{0, 1\}$ to represent the success (0) or failure (1) of Π from state $\boldsymbol{x}$. The objective of the recovery policy Π_r is to locate and control the vehicle to a nearby state $\boldsymbol{x}_r$ which maximizes the expected success of Π:

$$\boldsymbol{x}_r = \underset{\boldsymbol{x} \notin \mathcal{X}_O}{\arg\min}\, \mathbb{E}\left[Z_{\boldsymbol{x}}\right]. \tag{1}$$

In the following section, we discuss in detail the design of Π and Π_r, and demonstrate that proactively detecting planner failure and recovering after detection can be achieved by leveraging the same data-informed model.

IV. Approach

We propose a GP regression-based scheme to assess the risk of future motion planning failure while tracking a trajectory $\boldsymbol{\tau}(t)$. Data were collected from simulations that record motion planning successes and failures in various states that the robot may encounter during typical operation. This data were used to train a GP regression model to predict the probability of motion planning failure for individual states over a future horizon. Fig. 2 shows the outline of our approach. The front-end of the motion planner policy Π generates a corridor $\mathcal{C}$ of convex polytopes, illustrated in Fig. 3(a). The corridor is then sent to the back-end for final trajectory generation $\boldsymbol{\tau}(t)$ (see Fig. 3(b)). A Model Predictive Controller (MPC) is then used to generate the control signal to track $\boldsymbol{\tau}(t)$, generating a sequence of future robot states $\{\hat{\boldsymbol{x}}_i\}$ over horizon T_F. These states, along with the corridor $\mathcal{C}$, are used to predict the risk of motion planning

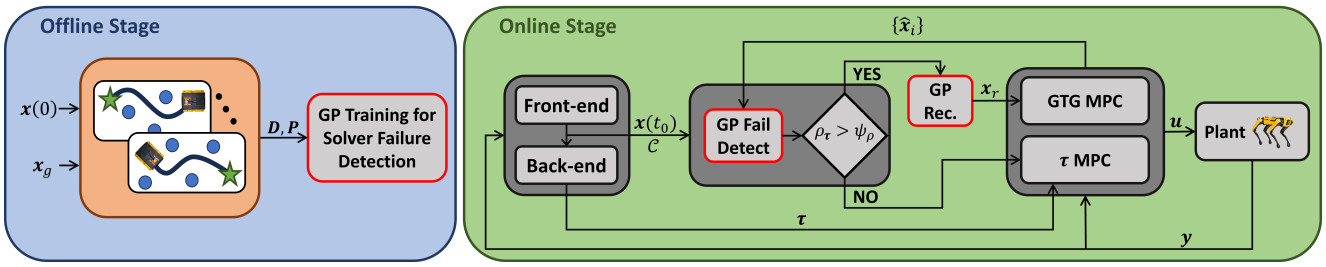
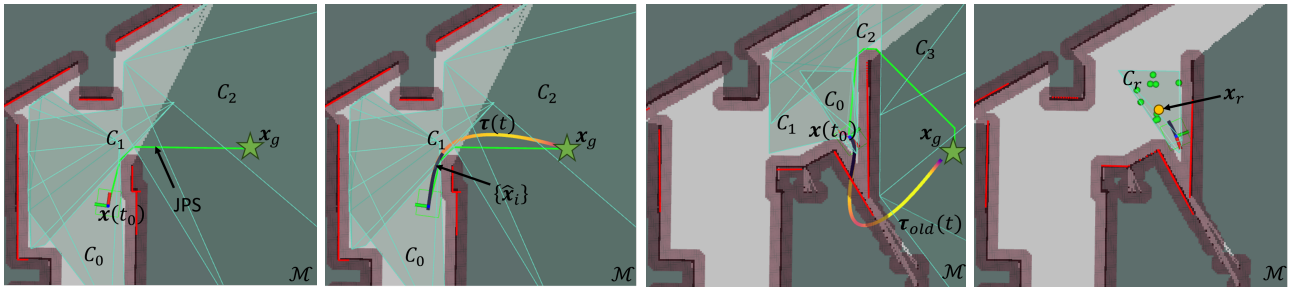


Fig. 2. Block diagram for both the offline training stage and online solver failure prediction and recovery framework.



(a) front-end (b) back-end (c) failure detection (d) recovery

Fig. 3. An example case study investigated in this work visualizing the complete navigation pipeline (a,b), in which the receding horizon, safe corridor motion planner fails (c), prompting the recovery pipeline to take over and recover the system (d). For $\boldsymbol{\tau}(t)$, the color gradient for $\boldsymbol{\tau}$ indicates the speed of the robot (from purple as the min to yellow as the max).

failure $\rho_{\boldsymbol{\tau}}$. Consider the case shown in Fig. 3(c), where the vehicle is driving towards a previously occluded dead-end. If the predicted risk from our GP-based failure detection model exceeds a user-defined threshold ψ_ρ, then the recovery behavior is triggered, and a recovery goal $\boldsymbol{x}_r$ is sent to our go-to-goal (GTG) MPC to bring the vehicle to a state where solver success is likely (Fig. 3(d)). In the following sections, we describe in detail our motion planner failure prediction and recovery framework, starting with a brief background of the base motion planner.

A. Motion Planner Preliminaries

1) *Planner Front-End.* The front-end starts with the global occupancy map $\mathcal{M}$, which is generated by fusing data from an onboard depth sensor, along with the current state of the vehicle, $\boldsymbol{x}(t_0)$, and the goal state $\boldsymbol{x}_g$. As shown in Fig. 3(a), an initial 0-order path within the free and unknown space of $\mathcal{M}$ is generated by using a graph-based, global planner. In this work, we use the Jump Point Search (JPS) algorithm [16], due to the reduced computational complexity when compared to other common algorithms like A* .

A corridor $\mathcal{C}$ of intersecting convex polytopes is then established along this generated initial path, in order to connect $\boldsymbol{x}(t_0)$ to $\boldsymbol{x}_g$. Each $C_i \in \mathcal{C}$ is represented as an H-polytope defined by a matrix $\boldsymbol{A}_i$ and vector $\boldsymbol{b}_i$ that define a convex set of points $\boldsymbol{p} \in \mathbb{R}^2$ in the xy plane

$$C_i = \{\boldsymbol{p} \in \mathbb{R}^2 \,|\, \boldsymbol{A_i p} \leq \boldsymbol{b_i}\}. \tag{2}$$

In order to generate each C_i of the corridor $\mathcal{C}$, we rely on the gradient-based optimization approach in [10]. With $\mathcal{C}$ constructed, the corridor is sent along with $\boldsymbol{x}(t_0)$ and $\boldsymbol{x}_g$ to the back-end optimization to find the final trajectory $\boldsymbol{\tau}(t)$.

2) *Planner Back-End.* We represent the trajectory $\boldsymbol{\tau}(t)$ (shown in Fig. 3(b)) as a collection of N cubic ($n = 3$) Bézier curves. We use these curves for the trajectory formulation as they are a commonly utilized basis with several salient properties for corridor-based motion planners [7]. One useful property of the Bézier curve $\boldsymbol{\tau}_j(t)$ is that it is fully contained within the simplex formed by the control points $\boldsymbol{q}_j^i$, $i \in [0, n]$. Thus, for $\boldsymbol{\tau}_j(t)$ to be contained within a convex polytope C, it is sufficient to ensure that $\boldsymbol{q}_j^i \in C$, $\forall i \in [0, n]$. To generate the final trajectory, we leverage the FASTER solver [9], altered to convert the Bézier control points to the MINVO basis [17] during optimization to improve solver success rate. Once $\boldsymbol{\tau}(t)$ has been found, it is sent to the tracking MPC to be executed on the robot.

B. Failure Modes: Front-End vs Back-End

There are two distinct failure modes of the motion planner described in Sec. IV-A, both of which will result in $\Pi = \emptyset$: (i) a front-end failure, in which an intersecting corridor $\mathcal{C}$ between $\boldsymbol{x}(t_0)$ and $\boldsymbol{x}_g$ cannot be found, or (ii) a back-end failure, in which the numerical solver fails to generate a trajectory along the JPS search path. Front-end failures can occur when a feasible search path doesn't exist (e.g., either $\boldsymbol{x}(t_0)$ or $\boldsymbol{x}_g$ overlap occupied space within $\mathcal{M}$), or when parameters of the JPS are poorly conditioned for generating a corridor $\mathcal{C}$ (e.g., $|\mathcal{C}|$ is high because the planning horizon distance is too large). The front-end of the motion planner typically runs in $<$ 1ms, thus for a given state $\boldsymbol{x}$ and map $\mathcal{M}$, front-end failures are easily determined by simply running the JPS and corridor generation at that state.

Much more difficult to predict, however, are failures at the back-end of the motion planner due to the fact that the environment is unknown a priori and the optimization is based only on current observations in $\mathcal{M}$. Since the back-end is based on a nonlinear optimizer, it can be difficult to characterize success or failure prior to actually running the back-end solver. Additionally, the time to run the back-end is typically $>$ 100ms, which is too large to directly test multiple future points for failure. Fig. 3(c) shows an example back-

end failure, in which the discovery of a previously unknown wall (shown as undiscovered space in Fig. 3(b)) requires a new avoidant trajectory to be generated. While the front-end is able to generate a corridor $\mathcal{C}$, the back-end is unable to find a feasible trajectory.

To concretely define these ideas, let $Z_{\boldsymbol{x},\mathcal{C}} \in \{0,1\}$ represent a success (0) or failure (1) of the motion planner pipeline, with $Z^f \in \{0,1\}$ representing a front-end failure and $Z^b \in \{0,1\}$ representing a back-end failure. Success of the back-end is dependent on success at the front-end, and failure of the front-end is interpreted as a failure of the back-end as well, so that $P\left(Z^b = 1 | Z^f = 1\right) = 1$. The probability of entire pipeline failure can be written as

$$P(Z_{\boldsymbol{x},\mathcal{C}}) = P\left(Z^b | Z^f\right) P\left(Z^f\right). \tag{3}$$

The probability of front-end failure is easily and rapidly checked by running the JPS for a given $\boldsymbol{x}$ and $\mathcal{C}$, so that effectively $P(Z^f) \in \{0,1\}$. Our contribution is in estimating the probability of back-end failure after a front-end success, $P(Z^b | Z^f = 0)$. For simplicity in notation, in the rest of the paper we will write this probability as $P(Z^b)$ and drop the dependence on the front-end outcome.

C. Gaussian Process for Predicting Back-End Failure

To accurately predict back-end failures, we propose a GP-based regression model trained on statistics inferred from simulated data. We choose GPs due to their non-parametric form and ability to quickly (< 10ms) and accurately infer from a small dataset. Each data point relates the robot and map state to the probability of back-end failure $P(Z^b_{\boldsymbol{x},\mathcal{C}})$. A GP model $\hat{P}(Z^b_{\boldsymbol{x},\mathcal{C}}|\cdot)$ can be trained to predict back-end failure probability at run time over future states $\{\hat{\boldsymbol{x}}_i\}$. These probabilities can then be used to assess the risk of future motion planning failure $\rho_{\boldsymbol{\tau}}$ over $\{\hat{\boldsymbol{x}}_i\}$.

A navigation stack comprising of both the planning policy Π and the MPC can be deployed in simulation to gather training examples for the GP model. To generate the training dataset, $\boldsymbol{D}$, we use the Poisson random forest dataset from [18], which contains 10 forest worlds, each with a collection of 90 start and goal positions for navigation. A Clearpath Jackal UGV was then tasked to navigate through the worlds in each of the start and goal configurations, collecting back-end success and failure data points at each planning iteration. With these data collected, features which correlate with back-end failure can be found. To promote generality, the chosen features should only depend on the corridor set $\mathcal{C}$, regardless of the sensing modality used to generate it (LiDAR, RGBd, etc.), along with the robot position and its time derivatives, which are common state features for most AMR.

1) *Feature Selection.* Each training tuple contains three pieces of information: (i) robot state $\boldsymbol{x}$, (ii) corridor $\mathcal{C}$, and (iii) binary variable $Z^b_{\boldsymbol{x},\mathcal{C}}$ which encodes a success or failure of the back-end. With these data, statistical inferences can be made that relate the robot state and corridor to the probability of back-end failure $P(Z^b_{\boldsymbol{x},\mathcal{C}})$. Through study of various possible features that could be used, we found two which were particularly well-suited to predicting the probability of back-end failure: the minimum time-to-intersect (TTI), t_C, from robot state $\boldsymbol{x}$ to corridor $\mathcal{C}$, and the number of polytopes that define the corridor $|\mathcal{C}|$.

The minimum TTI can be found by using the xy position $\boldsymbol{p} \in \mathbb{R}^2$ and velocity $\boldsymbol{v} \in \mathbb{R}^2$ of the robot state $\boldsymbol{x}$, then using kinematic equations to find the minimum TTI of the hyperplanes that define the polytope C containing $\boldsymbol{p}(t_0)$. Formally, if row $\boldsymbol{r}_i \in \boldsymbol{A}$ and $b_i \in \boldsymbol{b}$ form a hyperplane $\boldsymbol{r}_i \cdot \langle x, y \rangle = b_i$ of polytope C, then the time to intersect the hyperplane t_H can be calculated as:

$$t_H\left(\boldsymbol{r}_i, b_i, \boldsymbol{x}\right) = \begin{cases} \frac{b_i - \boldsymbol{r}_i \cdot \boldsymbol{p}}{\boldsymbol{r}_i \cdot \boldsymbol{v}} & \text{if } \boldsymbol{r}_i \cdot \boldsymbol{v} > 0 \\ \gamma_t & \text{otherwise} \end{cases} \tag{4}$$

where γ_t is a user-defined maximum value for t_H when the vehicle is stationary or moving away from the hyperplane. With t_H, t_C is calculated as the minimum TTI to the hyperplanes of C:

$$t_C = \min_i \{t_H\left(\boldsymbol{r}_i, b_i, \boldsymbol{x}\right)\}. \tag{5}$$

One of the biggest factors that affect the ability of the back-end solver to find a feasible solution is how close the current robot position $\boldsymbol{p}(t_0)$ is located to the boundary of the feasible set $\mathcal{C}$. Intuitively, TTI is an effective predictor of back-end failure because it captures several factors that determine success: (i) The physical distance between $\boldsymbol{p}(t_0)$ and the free space boundary, (ii) the velocity of the robot $\boldsymbol{v}(t_0)$, and (iii) the heading of the robot.

In addition to TTI, the cardinality $|\mathcal{C}|$ of the corridor also plays a role in failure of the back-end solver: if $\mathcal{C}$ is defined by many polytopes, then obstacles in the environment necessitate a very non-direct path to be planned for the robot, further complicating the search for a feasible path. Together, these two features were used inside a feature vector $\boldsymbol{d}(\mathcal{C}, \boldsymbol{x}) = [t_C, |\mathcal{C}|]$ to infer the probability of back-end failure. To find this probability, the back-end failure training data $\left\{Z^b_{\boldsymbol{x},\mathcal{C}}\right\}$ were binned based on feature vector value $\boldsymbol{d}$, and the ground-truth probability of failure $P(Z^b_{\boldsymbol{x},\mathcal{C}})$ was found within each bin. To validate the choice of input features for training, we plot the probability of back-end failure $P(Z^b_{\boldsymbol{x},\mathcal{C}})$ over t_C and $|\mathcal{C}|$, where the correlations are shown in Figs. 4(a) and (b). As t_C decreases, the probability of back-end failure increases. Furthermore, as the corridor length $|\mathcal{C}|$ increases, the probability of failure also increases.

2) *GP Regression.* The underlying GP model input is defined by a collection of M input training features, $\boldsymbol{D} = [\boldsymbol{d}_1, \ldots, \boldsymbol{d}_M]$, and values $\boldsymbol{P} = [P_1, \ldots, P_M]$, with an output defined by a joint Gaussian distribution [19]:

$$\begin{bmatrix} P \\ \hat{P} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{d}) \\ \mu(\boldsymbol{d}_*) \end{bmatrix}, \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}_* \\ \boldsymbol{K}_*^T & \boldsymbol{K}_{**} \end{bmatrix}\right), \tag{6}$$

where $\boldsymbol{K} = \kappa(\boldsymbol{D}, \boldsymbol{D})$, $\boldsymbol{K}_* = \kappa(\boldsymbol{D}, \boldsymbol{D}_*)$ and $\boldsymbol{K}_{**} = \kappa(\boldsymbol{D}_*, \boldsymbol{D}_*)$, μ is the mean function, $\boldsymbol{D}_*$ is the test input, and κ is a positive definite kernel function, which is the Radial Basis Function (RBF) in this work. From this, the predictive posterior distribution of $\hat{P}$ given $\boldsymbol{D}$ can be expressed as:

$$\hat{P} \sim \mathcal{N}(\mu_*, \sigma_*^2), \tag{7}$$

with μ_* and σ_*^2 defined as:

$$\mu_* = \mu(\boldsymbol{D}_*) + \boldsymbol{K}_*^T \boldsymbol{K}\left(P - \mu(\boldsymbol{D})\right) \tag{8}$$

$$\sigma_*^2 = \boldsymbol{K}_{**} - \boldsymbol{K}_*^T \boldsymbol{K}^{-1} \boldsymbol{K}_*. \tag{9}$$

With this, the estimated probability of back-end failure is taken as the mean values of this posterior:

$$\hat{P}\left(Z^b_{\boldsymbol{x},\mathcal{C}}|\boldsymbol{d}\right) = \mu_*. \tag{10}$$

To validate the quality of the trained GP models, the distribution of failures over t_C was collected from test worlds outside the forest dataset, and the resulting test set distribution was compared with the learned distribution $\hat{P}(Z^b_{\boldsymbol{x},\mathcal{C}}|\boldsymbol{d})$ for $|\mathcal{C}| = 2$ (Fig. 4(c)) and $|\mathcal{C}| = 3$ (Fig. 4(d)). These plots show the learned distributions closely match the test distribution, demonstrating that the GP models generalize well to new environments.

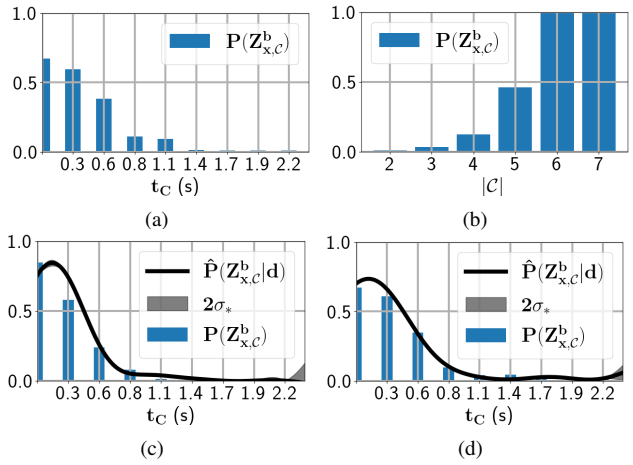


Fig. 4. Solver failure trends for (a) t_C and $|\mathcal{C}|$ along with learned distribution vs test distribution for (c) $|\mathcal{C}| = 2$ and (d) $|\mathcal{C}| = 3$.

3) *Defining Planning Risk.* With $\hat{P}(Z^b_{\boldsymbol{x},\mathcal{C}}|\boldsymbol{d})$ estimating back-end failure, the probability of failure for the entire motion planning pipeline $P(Z_{\boldsymbol{x},\mathcal{C}})$ can be calculated using (3). These probabilities can be used to infer the risk of motion planning failure along the future states $\{\hat{\boldsymbol{x}}_i\}$ predicted by the MPC. To formulate this risk, we consider the total number of future motion planning failures $Z_{\boldsymbol{\tau}}$, defined as

$$Z_{\boldsymbol{\tau}} = \sum_{\hat{\boldsymbol{x}} \in \{\hat{\boldsymbol{x}}_i\}} Z_{\hat{\boldsymbol{x}},\mathcal{C}}. \tag{11}$$

Because each $Z_{\hat{\boldsymbol{x}},\mathcal{C}}$ is a stochastic variable, $Z_{\boldsymbol{\tau}}$ is also a stochastic variable. The risk metric chosen in our approach is the expected number of collisions over the future horizon, $\rho_{\boldsymbol{\tau}} = \mathbb{E}(Z_{\boldsymbol{\tau}})$. The expected value is chosen here for its simplicity and speed to calculate, although other risk metrics may be used as well [20]. Since each $Z_{\hat{\boldsymbol{x}},\mathcal{C}}$ is a Bernoulli random variable with predicted probability $P(Z_{\hat{\boldsymbol{x}},\mathcal{C}})$ of failure, the expectation is calculated as:

$$\rho_{\boldsymbol{\tau}} = \sum_{\hat{\boldsymbol{x}} \in \{\hat{\boldsymbol{x}}_i\}} P(Z_{\hat{\boldsymbol{x}},\mathcal{C}}). \tag{12}$$

A risk threshold ψ_ρ may be set so that anytime the risk of planner failure over future states $\{\hat{\boldsymbol{x}}_i\}$ exceeds this value, the recovery behavior is triggered.

D. Recovering After Predicted Failures

When $\rho_{\boldsymbol{\tau}} > \psi_\rho$ is satisfied, it means that there is a collection of states in the vehicle's future horizon that the planner is likely unable to successfully operate. As such, the vehicle must stop or perform other recovery maneuvers in order to avoid collisions and navigate successfully through said regions. Unlike prior works where human operators intervene to recover the vehicle once failures are detected [4], [21], our framework includes a recovery planner Π_r which enables the vehicle to find and execute safe recovery maneuvers autonomously, as illustrated in Fig. 3(d).

Once the vehicle has stopped after switching to the recovery mode, the objective is to locate a nearby region where the planner will succeed, i.e., $Z_{\boldsymbol{x},\mathcal{C}} = 0$. It is not sufficient for the vehicle to retrace its steps along $\boldsymbol{\tau}(t)$, as the planner is likely to generate the same problematic trajectory again, causing oscillatory behavior. Thus, the first step in the recovery process is to sample points uniformly in free space around the current vehicle position $\boldsymbol{p}(t_0)$. To do so, an H-polytope C_r is generated around $\boldsymbol{p}(t_0)$, where hit-and-run Markov-chain Monte Carlo sampling [22] is used to find N_p candidate positions $\mathcal{P}_c = \{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_{N_p}\}$, where N_p is a user-defined parameter. $\mathcal{P}_c$ is then converted to states $\mathcal{X}_c$ by assuming the vehicle starts from rest. We make this choice because it significantly reduces the sample space and sampling only positions was enough to find recovery states in practice. With $\mathcal{X}_c$, we find the probability of planner failure, $P(Z_{\boldsymbol{x}_i,\mathcal{C}_i})$, at each $\boldsymbol{x}_i$, as well as neighboring states in close proximity for consistency. If all predictions have failure probability greater than η, the samples are thrown away and the sampling process is repeated. Here η is a user-defined parameter which controls how risk averse the recovery behavior should be. $\boldsymbol{x}_r$ is then chosen to be the state with lowest expected failure:

$$\boldsymbol{x}_r = \underset{\boldsymbol{x}_i \in \mathcal{X}_c}{\arg\min}\, \mathbb{E}\left[Z_{\boldsymbol{x}_i,\mathcal{C}_i}\right]. \tag{13}$$

After determining $\boldsymbol{x}_r$, the vehicle navigates to the recovery point using the GTG MPC with an added constraint, formulated as in (2), where $\boldsymbol{p}(t_0)$ must remain in C_r in order to avoid obstacles. Once the vehicle reaches $\boldsymbol{x}_r$, the planner switches back to the nominal safe corridor policy Π to generate trajectories $\boldsymbol{\tau}(t)$ and the entire process repeats.

V. Simulations

Simulations were performed to both train the GP classification model described in (6) and validate the proposed approach to detect and recover from motion planning failures. All simulations were performed in Gazebo using Ubuntu 20.04 and ROS Noetic. The robot used in simulation is a Clearpath Robotics Jackal UGV equipped with a 270° 2D Lidar depth sensor. Data were collected as described in Sec. IV-C and sent to the GP regressions for training.

With the models trained, we then validated our approach in four gazebo worlds of varying difficulty. The base world is a series of connected rooms with either sparse or dense obstacle density and 1m or 2m wide doorways. In each world we use the same start configuration $\boldsymbol{x}(0)$ and three goals $\boldsymbol{x}_g^0$, $\boldsymbol{x}_g^1$, and $\boldsymbol{x}_g^2$. Fig. 5(a) shows the world with 1m doorways and dense obstacle configuration, along with an example navigation failure without our approach (Fig. 5(b) and (c)) and success with (Fig. 5(d)). In Fig. 5(b), the vehicle plans a trajectory $\boldsymbol{\tau}(t)$ which intersects a part of the wall occluded by an obstacle. Since an avoiding trajectory cannot be computed in time, the vehicle collides with the wall at $\boldsymbol{x}_A$ in Fig. 5(c). If instead we use our approach, as shown

in Fig. 5(d), the robot detects the planner failure proactively and stops at $\boldsymbol{x}_B$. A reverse maneuver (green line) is then executed to reach the recovery state $\boldsymbol{x}_r$ found using (13). The vehicle then switches back to the nominal planner and continues towards $\boldsymbol{x}_g^0$.

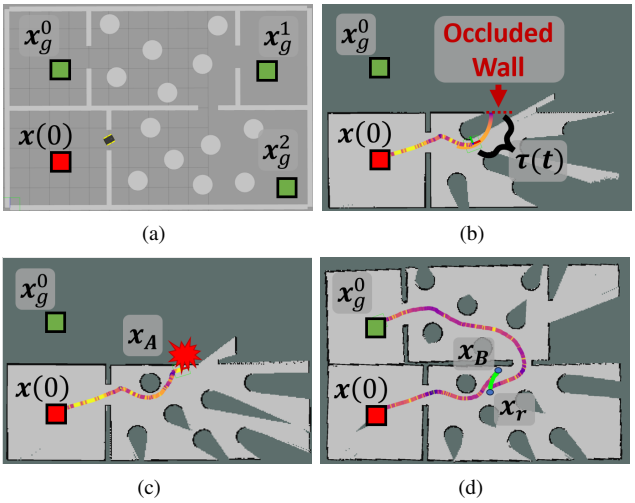


Fig. 5. (a) Gazebo world with 1m doorway and 3 different goals. In (b) an obstacle hides an occluded wall leading to a collision without our framework (c) vs a successful navigation toward x_g^0 in (d) with our approach.

The remaining 3 test worlds are generated by varying the doorway width between 1m and 2m, as well as varying the obstacle layout between a sparse and dense configuration. For each world tested, the robot is tasked to navigate 10 times to the goals $\boldsymbol{x}_g$, creating 30 test points per world, for 120 simulations total. The resulting success rates for the motion planner with and without our approach are shown in Fig. 6 for each goal and world combination, where it can be seen that using our failure detection and recovery framework improves the nominal planner's performance.

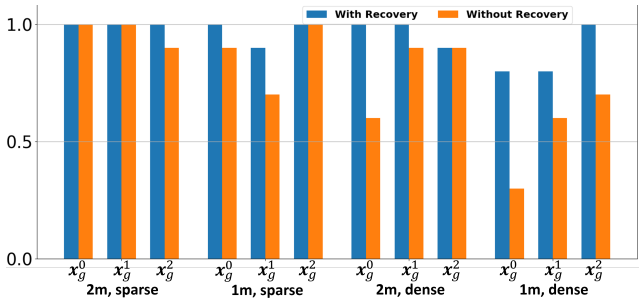


Fig. 6. Navigation success rates with (blue) and without (orange) failure detection and recovery for different simulation world and goal combinations.

VI. Physical Experiments

The proposed approach was validated with multiple robots across several experiments, all of which are shown in the supplementary material and website. Presented in this paper are two experiments with two real robotics platforms: a Boston Dynamics Spot quadruped, and the same Jackal differential drive UGV used in simulations. For each platform, the same motion planning pipeline was used to generate trajectories $\boldsymbol{\tau}$ to follow, using an MPC to generate the control signal $\boldsymbol{u}$ to track these trajectories. Lidar sensor readings were provided by Ouster for the Spot, and Velodyne for the Jackal. These were used by the SLAM package Gmapping in order to create a map $\mathcal{M}$ and estimate the state of the robot at runtime as each platform traveled through an environment unknown a priori. To emphasize the generality of the proposed approach, the GP model $\hat{P}(Z^b_{\boldsymbol{x},\mathcal{C}}|\boldsymbol{d})$ that was used to predict motion planning back-end failures was *trained entirely on data collected in simulation*, demonstrating how the approach is both sensor- and model-agnostic.

Two test cases were setup to test the approach. Fig. 7 shows the first case in which the Jackal is tasked to move towards a goal around an occluding corner, behind which are occluded obstacles previously unknown to the robot. Fig. 1 shows the second case in which the Spot is tasked with a similar mission, except it must negotiate an unexpected dead-end. Without the proposed approach, both cases lead to path-planning failures, which in turn lead to collisions. Both Fig. 7 and Fig. 1 show snapshots of the proposed approach being used to proactively detect risk of path planning failure $\rho_{\boldsymbol{\tau}}$, recovering at $\boldsymbol{x}_B$ when $\rho_{\boldsymbol{\tau}} > \psi_\rho$, moving to a recovery point $\boldsymbol{x}_r$, then continuing moving towards $\boldsymbol{x}_g$. For these experiments, the risk threshold was $\psi_\rho = 3$ *expected failures* over the predicted MPC future trajectory.

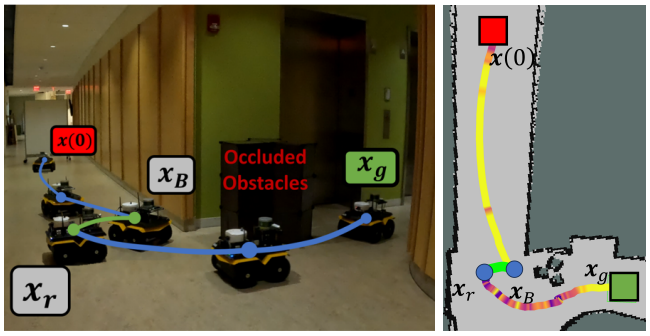


Fig. 7. Experiment case in which unexpected occluded obstacles would have caused a motion planner failure without our approach.

VII. Conclusions and Future Work

In this work, we have presented a novel GP-based, proactive failure detection and recovery scheme to prevent a mobile robot system from colliding with obstacles. Our approach is shown to improve the performance of a safe corridor motion planner in both simulation and experimental case studies. Furthermore, our approach is model- and sensor-agnostic and can be applied without prior real-world training data due to the careful selection of features.

Future work aims to enhance the system by incorporating distributional learning for failure detection, eliminating the need for multiple GP regressions. Additionally, we would like to utilize this approach for planner switching within a Simplex Architecture and incorporate dynamic obstacles.

VIII. Acknowledgements

Funding for this research are provided by an Amazon Research Award and by CoStar group.

REFERENCES

[1] X. Xiao, Z. Xu, Z. Wang, Y. Song, G. Warnell, P. Stone, T. Zhang, S. Ravi, G. Wang, H. Karnan, J. Biswas, N. Mohammad, L. Bramblett, R. Peddi, N. Bezzo, Z. Xie, and P. Dames, "Autonomous ground navigation in highly constrained spaces: Lessons learned from the benchmark autonomous robot navigation challenge at icra 2022 [competitions]," *IEEE Robotics & Automation Magazine*, vol. 29, no. 4, pp. 148–156, 2022.

[2] N. Mohammad and N. Bezzo, "A robust and fast occlusion-based frontier method for autonomous navigation in unknown cluttered environments," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 6324–6331.

[3] X. Zhang, Y. Shu, Y. Chen, G. Chen, J. Ye, X. Li, and X. Li, "Multimodal learning and relaxation of physical conflict for an exoskeleton robot with proprioceptive perception," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 10 490–10 496.

[4] T. Ji, A. N. Sivakumar, G. Chowdhary, and K. Driggs-Campbell, "Proactive anomaly detection for robot navigation with multi-sensor fusion," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4975–4982, 2022.

[5] G. Kahn, P. Abbeel, and S. Levine, "Badgr: An autonomous self-supervised learning-based navigation system," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1312–1319, 2021.

[6] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.

[7] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 344–351.

[8] L. Wang and Y. Guo, "Speed adaptive robot trajectory generation based on derivative property of b-spline curve," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 1905–1911, 2023.

[9] J. Tordesillas and J. P. How, "FASTER: Fast and safe trajectory planner for navigation in unknown environments," *IEEE Transactions on Robotics*, 2021.

[10] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically constrained trajectory optimization for multicopters," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 3259–3278, 2022.

[11] Y. Ren, F. Zhu, W. Liu, Z. Wang, Y. Lin, F. Gao, and F. Zhang, "Bubble planner: Planning high-speed smooth quadrotor trajectories using receding corridors," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 6332–6339.

[12] M. J. R. R. A. and S. A. Ning, *Unconstrained Gradient-Based Optimization*. Cambridge University Press, 2022.

[13] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-jacobi reachability: A brief overview and recent advances," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 2242–2253.

[14] A. Devonport and M. Arcak, "Data-driven reachable set computation using adaptive gaussian process classification and monte carlo methods," in *2020 American Control Conference (ACC)*, 2020, pp. 2629–2634.

[15] E. Yel, T. J. Carpenter, C. Di Franco, R. Ivanov, Y. Kantaros, I. Lee, J. Weimer, and N. Bezzo, "Assured runtime monitoring and planning: Toward verification of neural networks for safe autonomous operations," *IEEE Robotics & Automation Magazine*, vol. 27, no. 2, pp. 102–116, 2020.

[16] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, no. 1, 2011, pp. 1114–1119.

[17] J. Tordesillas and J. P. How, "Minvo basis: Finding simplexes with minimum volume enclosing polynomial curves," *Computer-Aided Design*, vol. 151, p. 103341, 2022.

[18] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online uav replanning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[19] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[20] A. Majumdar and M. Pavone, "How should a robot assess risk? towards an axiomatic theory of risk in robotics," in *Robotics Research: The 18th International Symposium ISRR*. Springer, 2020, pp. 75–84.

[21] G. Kahn, P. Abbeel, and S. Levine, "Land: Learning to navigate from disengagements," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1872–1879, 2021.

[22] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu