Online Predictive Monitoring and Proactive Planning for Safe Autonomous Robot Operations

---

A

Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Doctor of Philosophy

by

Esen Yel

August 2021

APPROVAL SHEET

This

Dissertation

is submitted in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

Author: Esen Yel

This Dissertation has been read and approved by the examing committee:

Advisor: Nicola Bezzo

Advisor:

Committee Member: Zongli Lin

Committee Member: Lu Feng

Committee Member: Sebastian Elbaum

Committee Member: Insup Lee

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

Jennifer L. West, School of Engineering and Applied Science

August 2021

To mom

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisor Nicola Bezzo for his mentorship and support throughout my stay at the University of Virginia. His continuous guidance, dedication to research, commitment to his students, passion for robotics and encouraging words helped me to become the researcher that I am today and will definitely impact my future.

I am very thankful to my committee members, Prof. Zongli Lin, Prof. Lu Feng, Prof. Sebastian Elbaum and Prof. Insup Lee for insightful and constructive feedback about my research as well as their support during my Ph.D. studies. I also thank Prof. Cody Fleming for serving as a committee member for my qualification exam and Prof. James Weimer for his collaboration.

I would like express my gratitude to current and former members of AMR Lab - Tony, Carmelo, Paul, Rahul, Shijie, Phil, and Jacob - for being excellent labmates and friends all these years. I would like to thank the members of PRECISE Lab - Radoslav, Taylor, Yiannis, and Matthew - for their collaboration. I thank all my friends in Charlottesville but especially, Başak, Volkan, Emily, Mahmoud, and Mark for all the joyful memories we shared.

I express my immense thanks to Richard with all my heart for his support and patience during the last two years of my Ph.D. Thanks to his continuous encouragement, I have been happier and healthier.

Last but not the least, I would like to thank my father Mustafa, my sister Başak, and my brother-in-law Özgür for their constant support, love and help. I would not be able to become the person that I am without them.

Funding: I acknowledge: the Defense Advanced Research Projects Agency (DARPA) for providing financial support for my Ph.D. research under the Assured Autonomy program, Contract FA8750-18-C-0090, the National Science Foundation (NSF) through grant # 1816591, the Office of Naval Research (ONR) under agreement number N000141712012, and Air Force Research Laboratory (AFRL).

Online Predictive Monitoring and Proactive Planning for Safe Autonomous Robot Operations

by

Esen Yel

B.S., Electrical and Electronics Engineering, Bogazici University, 2014

M.S., Electrical and Electronics Engineering, Bogazici University, 2016

Abstract

Autonomous mobile robotic systems are rapidly becoming part of our daily lives, and their use in transportation, delivery, surveillance, medical service, and household applications has been growing. With the increase in their popularity, assuring their and their surroundings' safety becomes more critical than ever. Unfortunately, these systems might be subject to various uncertainties such as unknown noises, external disturbances, intermittent sensing, model changes, and actuator faults when deployed in the real world. Such uncertainties bring challenges in motion planning as they can cause a system to deviate from its desired behavior and possibly lead to unsafe situations (e.g., an unmanned aerial vehicle (UAV) collision with an obstacle due to wind disturbance). In most cases, these uncertainties are unknown until the system starts its operation, making it difficult to train and develop techniques to increase resilience and safety during design time. Furthermore, autonomous mobile systems typically have limited computational resources on board, bringing an additional challenge at runtime. Thus, these uncertainties and their effects need to be considered in a computationally efficient way to improve the safety of autonomous systems.

This dissertation presents a set of motion planning, runtime monitoring, and validation techniques to provide safety for mobile autonomous systems operating under unforeseen uncertainties. The frameworks we develop predict the future states of the system under uncertainties and utilize these predictions to improve their safety by performing replanning accordingly while taking computational restrictions into consideration.

The techniques presented in this dissertation are extensively validated through realistic simulations and on real state-of-the-art vehicles.

First, we introduce a reachability-based self/event-triggered scheduling framework to relax the traditional periodic sensor monitoring to reduce computational power. This framework aims to minimize the computational burden related to the unnecessary sensor monitoring operations while still guaranteeing safety leveraging reachability analysis. To further reduce the computational requirements of reachability analysis and enable its runtime use, especially for systems with complex dynamics, we introduce a novel Gaussian Process regression-based fast reachability framework to guarantee the safety of aerial vehicles subject to intermittent sensing problems. Moreover, we develop an assured safety monitoring and motion planning framework that leverages verified neural networks to bypass the reachability analysis computation altogether at runtime while still guaranteeing safety under unforeseen disturbances.

As the runtime disturbances can lie outside of the training bounds, unsafe situations can occur due to a lack of training data. To deal with such issues, we present an online learning framework to predict the runtime disturbances and their effects on the system's behavior to prevent unsafe situations. With this novel technique, the system can adapt its behavior under unforeseen disturbances, particularly unknown payload variations, to remain safe while improving its decision-making over time. To further deal with unforeseen system failures that cause performance degradation, we introduce a meta-learning-based framework that allows the system to predict the future states under faults and adapt its trajectory to improve tracking performance at runtime.

Contents

List of Figures

List of Tables

List of Abbreviations

GP Gaussian Process.

MAML Model Agnostic Meta-Learning.

NN neural network.

ROS Robot Operating System.

UAV unmanned aerial vehicle.

Chapter 1

Introduction

Autonomous systems with minimal/no supervision have increasingly gained popularity in the last decade, and they are becoming a reality: autonomous cars and delivery robots are appearing around us, especially in major cities, while service and hobby robots are becoming as common as household appliances. Autonomous unmanned aerial vehicles (UAVs), in particular, are rapidly finding their way into our society for a myriad of operations from transportation/delivery to surveillance/inspection. Their technological advancements, including agile mobility, increasingly precise sensors, and actuators, have only augmented their popularity and deployment in our daily lives. Figure 1.1 depicts some examples of real-world applications of mobile aerial and ground robotics systems. As these vehicles become more commonplace, it becomes crucial to guarantee their and their surroundings' safety.

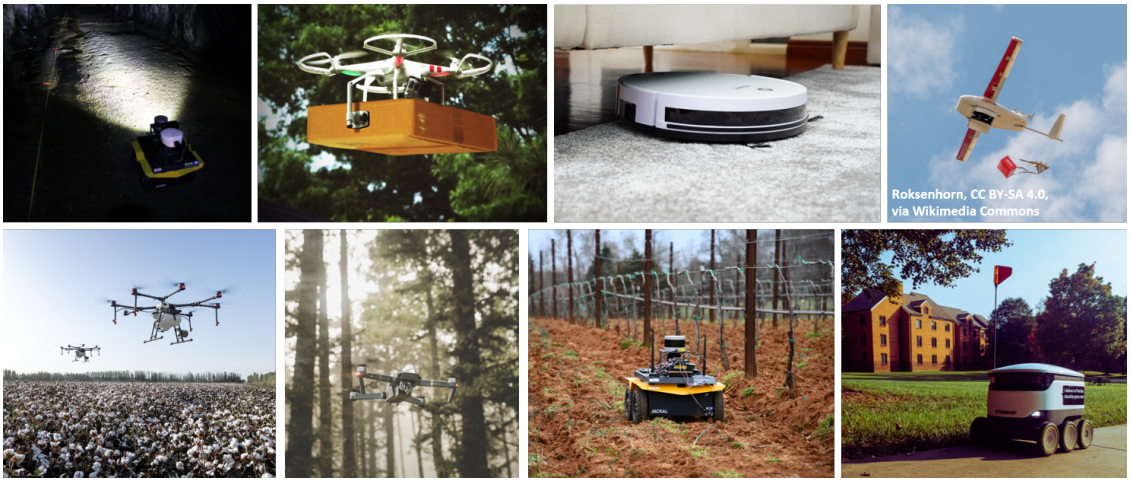


Figure 1.1: Autonomous mobile robotic systems are used for a variety of operations nowadays including inspection, delivery, household applications, agricultural purposes and photography.

Although autonomous robots have seen an incremental technological improvement, a big challenge remains on how to guarantee safety during autonomous operations, mainly due to the unknown disturbances and noises that can affect the behavior of an autonomous system at runtime. UAVs may be subject to external disturbances (e.g., wind, unknown payload), which can cause the system to deviate from its desired behavior and possibly lead to undesired consequences such as collisions.

In the real world, autonomous systems may also lose access to their state information due to various factors, such as signal occlusions, limited sensor capabilities, and sensor failures. These problems cause the system to operate under intermittent sensing. As typical autonomous operations rely on periodic state feedback to follow the planned trajectories closely and avoid collisions, intermittent sensing may cause the system to deviate from the desired behavior, especially under external disturbances.

Moreover, autonomous vehicles can also face problems such as component faults or system aging, which would cause model changes. Since these situations usually occur without any apriori knowledge, they are typically not considered during design time; thus, they cause the system to operate with degraded performance and may lead the system to fail its assigned task.

To this end, it is critical to design proactive planning frameworks to prevent unsafe consequences under the aforementioned uncertainties, disturbances, and degradations. To design such systems, the future states of the system under uncertainties need to be predicted accurately. However, these uncertainties, disturbances, and component faults that occur at runtime are usually unknown and unforeseen a priori, making it challenging to model the system's behavior at design time. Additionally, autonomous systems, UAVs in particular, are usually small vehicles with limited computation and payload capacities. Hence, it may not be possible to apply computationally expensive techniques at runtime to predict future states and guarantee safety.

Given these challenges, this dissertation presents both model-based and data-driven methods to make efficient and accurate future state predictions for autonomous systems under unknown disturbances, noises, and model changes. Our approaches focus on proactively using these predictions to schedule sensor monitoring operations, perform runtime monitoring and perform proactive replanning to guarantee safety. In the rest of this chapter, we review the related work and the state-of-the-art in safe motion planning. We also overview our approach, and lastly, we list the contributions of this dissertation.

1.1 Related Work

In this section, we provide an overview of related work in safe motion planning starting with traditional sampling-based and model-based motion planning techniques, followed by methods that leverage learning-

enabled components. Then we provide an overview for safe motion planning techniques under uncertainties and finally for fault tolerant motion planning approaches.

1.1.1 Traditional Motion Planning

The problem of motion planning for collision avoidance has been heavily studied in the literature of mobile robotics to enable safe and persistent autonomous operations. Sampling-based methods are commonly used in robotics operations to generate collision-free paths. Rapidly-exploring Random Tree (RRT) [41] is a sampling-based algorithm that constructs a random tree to find a collision-free path, and it is used for a broad range of path planning problems. Extensions of RRT approach such as RRT^X [61] and RRT* [32] have been leveraged for UAV navigation tasks [65, 7, 53]. Probabilistic Roadmap (PRM) [33] another sampling-based approach which connects random samples in a robot's configuration space by a graph search algorithm to find a path from the start to the goal location. PRM-based approaches have been also used to create collision-free paths for UAVs [66].

In addition to the sampling-based approaches, artificial potential fields have also been designed to avoid obstacles in the environment efficiently [36] and have been leveraged for UAV and mobile robot navigation [73, 22, 60, 83]. Model Predictive Control (MPC) and Nonlinear Model Predictive Control (NMPC) have also been used for collision-free UAV navigation in both dynamic and static environments [77, 8, 45].

1.1.2 Learning-Enabled Motion Planning

Recent developments in machine learning algorithms such as reinforcement learning and deep reinforcement learning enabled these techniques to be used for online obstacle avoidance in unknown environments [79, 28, 9, 87]. Neural network structures such as deep neural networks (DNNs), recurrent neural networks (RNNs) and convolutional neural networks (CNNs) have been widely used to solve complex path planning problems for mobile robots efficiently. For example, in [88], an end-to-end DNN architecture called online three-dimensional path planning network (OTDPP-Net) is used to learn 3D local path planner behavior in an environment modeled as a 3D grid map. [68] introduces a motion planning network MPNet which takes raw environmental data as an input to generate obstacle-free connectable paths and shows computational advantages against sampling-based planners.

To impose safety constraints on the designed trajectories for UAVs, [14] represent the obstacle-free area as convex regions and assign polynomial trajectories in those regions. Concepts of safe flight corridors [46, 64] and free-space flight corridor [10], representing the obstacle-free areas in the environment as convex regions, are used as constraints in the trajectory optimization problem to generate safe trajectories.

1.1.3 Safe Motion Planning under Uncertainties

Even though the designed path is obstacle-free, disturbances and uncertainties can cause the system to deviate from the planned path and potentially collide with obstacles. To guarantee safety under such uncertainties and disturbances, reachability analysis techniques have been utilized heavily. Reachability analysis consists of computing the set of states that a system under uncertainties could reach from a set of initial states under a set of admissible inputs over a certain time horizon. Hamilton-Jacobi reachability has been widely used to compute the reachable sets of hybrid systems and provide safety guarantees for optimal system trajectories [15]. Stochastic reachability analysis techniques [80, 52, 86], which include computing the forward stochastic reach probability measure and the forward stochastic reachable set, are also used to design collision-free trajectories for systems in uncertain and dynamic environments. Hybrid system reachability analysis tools have also been heavily used, mostly for verification purposes. Flow* [12] uses Taylor models to compute flowpipe over-approximations of the dynamics and dReach [37] encodes the reachability problem as first-order formulas over the real numbers, and solves it using a δ- decision procedures. Other types of reachability analysis techniques such as ellipsoid methods [40, 99], Robust Control Invariant (RCI) tubes [78] and funnels [51] which are analogous to reachable tubes have been also utilized for collision avoidance problem.

Reachability analysis techniques are generally very powerful for providing safety guaranties for systems under uncertainties. Hamilton-Jacobi reachability, in particular, performs well in terms of the generality of system dynamics, flexibility in representation of sets, and control policy computation; however, it suffers from computational scalability [72]. A significant effort has been given to overcome the scalability problem for high-dimensional systems, such as decomposing the system dynamics [11], using efficient initializations [26], using adaptive grids [25], by ellipsoidal approximations [76], and using neural networks to approximate the reachable sets [16], [72]. In the literature, there have been other efforts to make reachability analysis more usable for runtime applications. For example, in [51], the authors precomputed a library of trajectories and funnels (analogous to reachable sets) offline and combined these trajectories online to navigate in apriori unknown environments under disturbance effects. However, with this approach, the system is restricted to use a discrete set of motion primitives. In [24], using Hamilton-Jacobi reachability, a look-up table is computed offline to find the bounds on the planned trajectory which are used to augment the obstacles to guarantee a collision-free behavior under bounded disturbances in unknown environments. Similarly, in [38], forward reachable sets are computed offline for parameterized trajectories, and at runtime, safe trajectory parameters are picked to avoid the sensed obstacles in unknown environments with model uncertainties.

1.1.4 Fault-Tolerant Motion Planning

As mentioned earlier, in addition to the disturbances, in the real world, autonomous systems may also face problems like actuator faults or system aging that cause the model of the system to change at runtime. Such problems cause the system to deviate from its desired behavior and potentially lead to unsafe states. For consistent performance under such conditions, control techniques have been widely used to adapt the systems' control inputs according to the changes in the system dynamics and to alleviate the effects of faults. For example, for quadrotors with the complete loss of one or multiple propellers, specific controllers can be designed according to the failure to improve stability and performance [58, 81, 82, 27]. However, these techniques require explicit knowledge of specific failures and how these changes affect the system's dynamical model to design resilient controllers. When such knowledge is not available, fault identification or adaptive control techniques need to be leveraged. In [85], an Extended Kalman Filter (EKF)-based fault identification is used to decide if there is one or multiple rotor failures, and a control allocation is updated based on the failure using a nonlinear Model Predictive Control (MPC). In [21], a self-reconfiguration technique allows the system to decide on its configuration based on the actuator failure and its desired trajectory. Model Reference Adaptive Control (MRAC) is an adaptive control technique which adapts the control variables of the system based on the difference between the observations and reference model output to improve tracking for systems with uncertainties and it has been also used to compensate for failures [48, 47]. Recently, machine learning techniques such as Gaussian Processes (GP) [13] and deep neural networks (DNNs) [30] are utilized for the adaptive elements in MRAC frameworks.

In addition to the control approaches, machine learning techniques have also been widely used to improve the performance of UAVs under actuator faults or disturbances. In [74], the authors use MPC with active learning to learn the robot's new model under failure and provide necessary inputs. Reinforcement Learning (RL) techniques are also utilized to adjust the actuator control commands to compensate for component faults [19, 2]. Meta-learning approaches enable the systems to speed up their learning process for new tasks with a small number of training samples from new tasks. This property makes meta-learning suitable for learning the models of uncertain systems at runtime for safe planning [42]. Model Agnostic Meta-Learning (MAML) trains the model parameters explicitly to make them easy and fast to fine-tune for the new tasks [20]. MAML has been leveraged for fault-tolerant operations using MPC and RL [59, 3]. [75] introduces a concept of meta-active learning for in which Q-function is learned via meta-learning and used to find optimal actions to maximize the probability of staying in the safe region and promote information gain for systems with altered dynamics. In [71], meta-learning is utilized to model the system dynamics under external forces to be used with an adaptive control scheme improve the tracking performance. All of these approaches assume that

the user is given direct access to the controller or the actuator inputs. However, this assumption may not hold, especially when off-the-shelf robotics systems are used.

1.2 Overview of the Research

Our research presented in this dissertation consists of a sequence of techniques for safe motion planning under various uncertainties such as external disturbances, sensing uncertainties, payload disturbances, and model changes while considering the computational resources. Figure 1.2 gives an outline of our research. First, we minimize the sensor monitoring operations by reachability analysis-based self/event-triggered scheduling when systems are operating under bounded disturbances. Then we present our machine learning-based frameworks to decrease the reachability computation and to bypass it at runtime completely. Then we present our efforts on dealing with disturbances that are outside of the training bounds. Finally, we improve the trajectory tracking performance of an autonomous system with an unforeseen fault and provide proactive safe replanning. These frameworks are applied to state-of-the-art UAVs and enable them to have safer behavior when they operate between obstacles under various uncertainties.

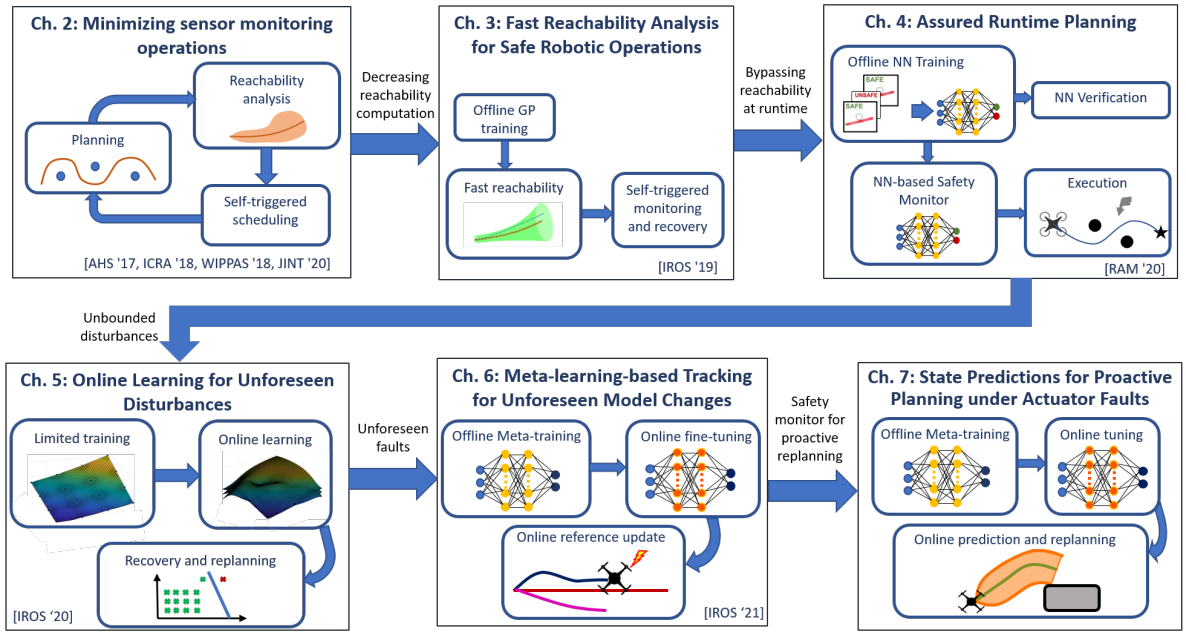


Figure 1.2: Overview of the presented research.

Self/Event-triggered Scheduling: The dissertation is opened with a discussion about our novel reachability-based self/event-triggered scheduling approach to minimize the unnecessary sensor monitoring and replanning operations while providing safety under uncertainties with reachability analysis in static and dynamic environments. We considered scheduling for both localization sensors and also perception sensors.

Figure 1.3 pictorially depicts the motivation for this research by presenting a scenario in which a UAV is subject to external wind disturbance and uses computed reachable tubes to monitor its sensors and replan its trajectory for safety under the presence of other objects in the environment.

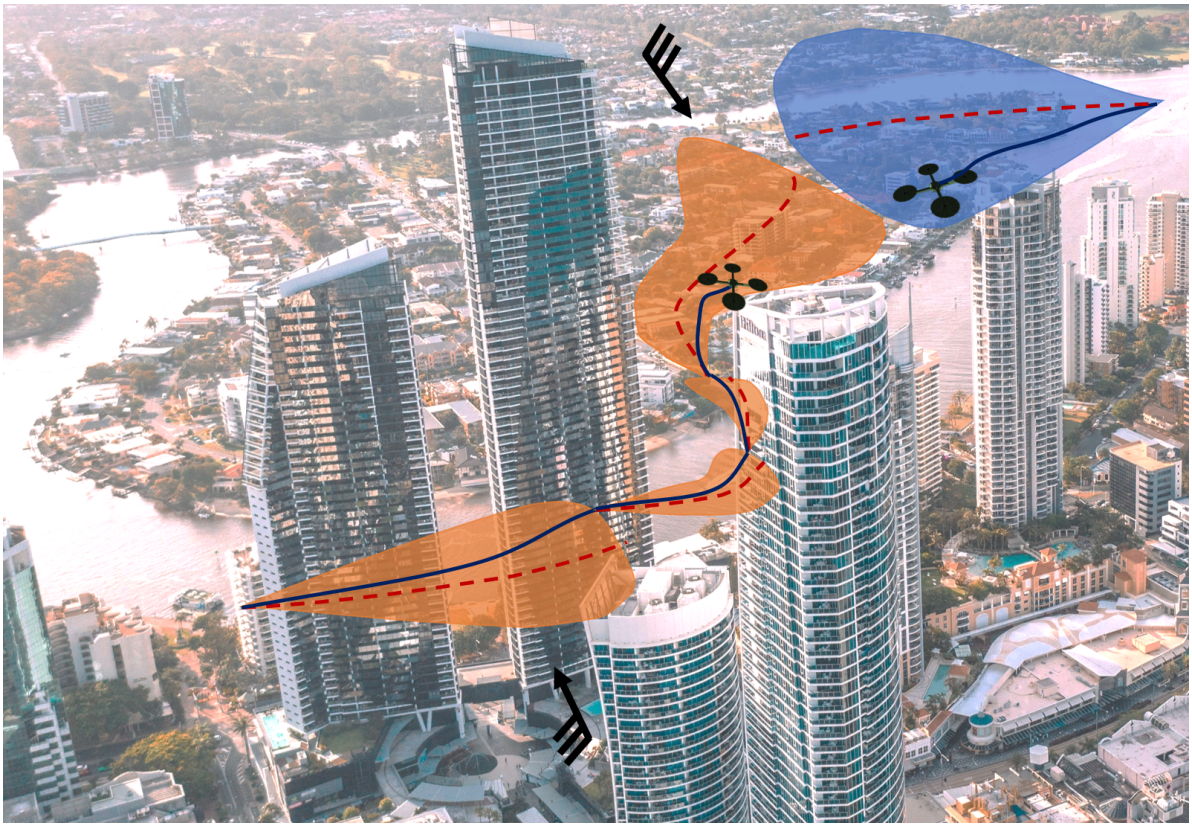


Figure 1.3: An environment with two UAVs under external wind disturbance. The environmental and system uncertainties should be taken into consideration for safety, performance and computation considerations.

Fast Reachability Analysis: We utilize various machine learning techniques to reduce the computational burden that reachability analysis can cause on the system. To predict the future deviations of the system from the desired behavior under wind disturbance and intermittent sensing problems, we use Gaussian Process (GP) regression theory and compute regions analogous to reachable sets. This technique allows to compute and refine these regions fast at runtime and decide when the system needs to replan its trajectory to recover from potentially unsafe situations without perfect state feedback.

In addition to speeding up the reachability process, we use machine learning techniques also to bypass the reachability computation at runtime completely. To achieve this, we use neural networks (NNs) to make safety decisions about a given trajectory under the assumption of bounded disturbances. First, a neural network is trained offline using the safety decisions from a traditional reachability analysis tool. Then, as a safety-critical system uses the decisions of the neural network, we use a verification tool to verify the

safety of the neural network before its deployment. This technique is tested on two different case studies to demonstrate its applicability for real-life scenarios: 1) a pickup/drop-off mission in a warehouse-like environment 2) UAV navigation mission in forest-like unknown environments. With this approach, we are able to plan safe trajectories without using computationally expensive reachability tools at runtime.

Online Learning for Unforeseen Disturbances and Model Changes: To incorporate the unforeseen and out-of-training bounds disturbances and faults, we develop online learning and planning frameworks. First, we propose to predict online disturbances (that are not assumed to be bounded by the training bounds) and their effects on the system behavior at runtime by extending our GP regression-based deviation estimation technique. These predictions are then used to replan a trajectory in conjunction with a recovery framework that prevents the system from going into unsafe states. The data collected at runtime are utilized to update the offline trained models to improve decision-making over time. By applying this approach to an unknown payload case study, we show that the system can replan its speed based on its future deviation predictions and make improved decisions over time.

Secondly, we leverage meta-learning to predict the future states under a component fault which causes deviations from the desired behavior. Meta-learning is chosen because of its ability to be fine-tuned with a few data at runtime, which is suitable for the problem we tackle in which the system is subject to a fault that has not been previously experienced. After fine-tuning the meta-learning model with a few data observed at runtime, the predictions of the fine-tuned model are used to update the reference input to the system in a robust control-based fashion to compensate for the fault. With this approach, we improve the trajectory tracking performance of the system even when it experiences out-of-training bounds faults. We also develop a safety monitor to assess the behavior of the system while using the reference input update method. We utilize meta-learning to predict the future states and state uncertainties of the system and perform proactive planning to prevent unsafe situations.

1.3 Contributions of the Dissertation

This dissertation contributes to the state-of-the-art in safe planning by creating a unison set of efficient runtime monitoring and proactive planning frameworks for safe autonomous robot operations under uncertainties.

Specifically, the contributions of this dissertation can be listed as follows:

- We present a novel approach to minimize the sensor monitoring operations by introducing the concept of aperiodic sensor monitoring via a novel reachability analysis-based self/event-triggered monitoring approach. This technique guarantees safety under disturbances and noises while saving computational resources related to sensor monitoring operations.

- Since model-based reachability analysis can get computationally expensive, we propose an efficient reachable set estimation technique for systems operating under disturbances and intermittent sensing.
- To further reduce the computation at runtime, we introduce a novel framework that eliminates the need for reachability analysis at runtime while still providing assurance guarantees.
- To deal with out-of-training disturbances and improve monitoring and decision-making over time at runtime, we propose a novel online learning framework that uses runtime observations. This framework results in decision-making improvement over time for the systems under unknown, unforeseen, and out-of-the-training bounds disturbances.
- To further consider unforeseen failures that can occur at runtime, we introduce a novel trajectory tracking and proactive replanning framework that does not require accessing the controller to improve tracking performance and improve safety.
- The final contribution of this thesis is in the extensive and rigorous implementations with realistic simulations and in particular with real-world experiments with state-of-the-art quadrotor UAVs under different uncertainties to demonstrate the applicability of the techniques presented in this dissertation.

1.4 Organization of the Dissertation

In Chapter 2, we present a reachability analysis-based scheduling approach to minimize the computation related to sensor monitoring operations while keeping the system safe under disturbances. In Chapter 3, we introduce a data-driven approach to speed up the reachability computation for autonomous operations under intermittent sensing uncertainty. In Chapter 4, we limit the reachability computation at design time by utilizing verified neural networks for assured planning at runtime. We introduce Gaussian Process-based online learning framework for safe planning and decision making improvements at runtime in Chapter 5. In Chapter 6, we present a meta-learning-based online learning and replanning framework to improve trajectory tracking performance under actuator degradations. Finally, in Chapter 8, we provide some concluding remarks and potential directions for future work.

Chapter 2

Self/Event-triggered Scheduling and Planning

In this chapter, we introduce our self-triggered framework that minimizes sensor checking and replanning operations while guaranteeing safety. This framework leverages reachability analysis to predict the future states of the system under the effect of noises and disturbances and schedule next sensor monitoring and replanning operations while guaranteeing safety. The replanning operation is further relaxed by performing an online reachable tube shrinking. This approach is supplemented with an online speed adaptation policy based on the planned trajectory curvature to minimize drift from the desired path due to complex system dynamics and controller limitations. We validate this approach with both simulations and experiments focusing on a UAV quadrotor motion planning problem in environments consisting of both static and dynamic obstacles. This work has been published in 2017 NASA/ESA Conference on Adaptive Hardware and Systems [95], 2018 IEEE International Conference on Robotics and Automation (ICRA) [96] and the Journal of Intelligent & Robotic Systems [94] in 2020.

2.1 Introduction

Typically, unmanned aerial vehicles (UAVs) monitor their 1) pose and configuration using pose sensors like IMU, GPS, speedometers, motion capture systems and 2) distance to the obstacles using range sensors like lidar, radar, sonar, and IR sensors, in order to avoid collisions under external disturbances and noises. This information is then used to plan and replan the vehicle motion accordingly to guarantee the desired objective, in a robust, optimal, and safe fashion. However, periodic sensor checking and planning brings computational burden to the system, and can be relaxed if the future states of the system under different uncertainties can be predicted reliably. For example, let's consider the pictorial representation in Figure 2.1: when a UAV moves in an obstacle-free environment, it may act without checking its sensors for a longer time and avoid

collisions, whereas when the vehicle gets close to the obstacles or to the other aerial vehicles, it needs to monitor its sensors more often. Furthermore, whereas the vehicle can move faster in obstacle-free areas, it would need to slow down to closely follow winding trajectories between obstacles.

Figure 2.1: Pictorial representation of the problem presented in this paper: a quadrotor in navigating in a cluttered environment in which it needs to avoid collision with both static and dynamic obstacles while minimizing periodic sensor checking under uncertainties.

To deal with these issues, we introduce an online adaptive framework which allows a UAV to limit its sensor monitoring times to the instances in which it is necessary, thus minimizing computation while guaranteeing safety (e.g., no collisions) and liveness (i.e., following the desired trajectory closely). This approach addresses the following challenges:

1. how to minimize sensor monitoring and replanning operations while satisfying safety and liveness conditions in cluttered and dynamic environments;

2. how to plan and replan a UAV operation, including adapting its speed, while solving the previous challenge.

Our self-triggered scheduling and planning approach leverages reachability analysis to predict the future states of the system, utilizes self-triggered scheduling methods to compute the next sensor monitoring and replanning time, and introduces a replanning approach to adapt the speed of the UAV that is navigating in cluttered and unknown environments with static and dynamic obstacles under the effect of noises and external disturbances.

2.2 Preliminaries

This section introduces the notation, details of the quadrotor, and noise and disturbance models that will be used throughout this chapter.

2.2.1 Notation

We use bold lower case italic letters (e.g., $\boldsymbol{q}$) to denote vectors and bold upper case italic letters (e.g, $\boldsymbol{A}$) to denote matrices. $\|.\|$ represents the Euclidean norm.

We define the state vector of the UAV as:

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{p}_q^\mathsf{T} & \phi & \theta & \psi & v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^\mathsf{T}$$

where $\boldsymbol{p}_q = [x\ y\ z]^\mathsf{T}$ is the world frame position, v_x, v_y and v_z are the world frame velocities, ϕ, θ and ψ are the roll, pitch and yaw Euler angles and ω_x, ω_y and ω_z are the body frame angular velocities. $\boldsymbol{x} \in \mathbb{R}^4$ refers to the position and velocity part of the state in x-y direction and $\boldsymbol{p} \in \mathbb{R}^2$ denotes only the position in x-y direction [56].

2.2.2 UAV Quadrotor Dynamics and Capabilities

The UAV is assumed to be equipped with sensors capable of observing its angular position and velocity. It is also assumed that the UAV can observe its position and velocity in the $x-y$ plane (i.e. $\boldsymbol{x}$) via pose sensor at scheduled times. The position of an obstacle in the $x-y$ plane is denoted by $\boldsymbol{o} \in \mathbb{R}^2$ with a subscript indicating the index of the obstacle and it is assumed to be observed via an on-board range sensor at scheduled times. We neglect the third dimension because we assume that the robot moves on a plane at desired z level, however, the approach is still valid when the z position of the robot varies.

A quadrotor has four rotors with two rotating clockwise and two rotating counter-clockwise. The angular speed of each rotor is denoted by ω_i. As also descried in [56], [6], the thrust (F_i) and moment (M_i) produced by each rotor is proportional to their angular speed:

$$F_i = \kappa_f \omega_i^2, \quad M_i = \kappa_m \omega_i^2, \quad i = 1, \cdots, 4$$

where κ_f and κ_m are proportionality constant for thrust and moment respectively. The net thrust and moments generated on the quadrotor is calculated by:

$$\begin{bmatrix} F \\ M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} \kappa_f & \kappa_f & \kappa_f & \kappa_f \\ 0 & d\kappa_f & 0 & -d\kappa_f \\ -d\kappa_f & 0 & d\kappa_f & 0 \\ \kappa_m & -\kappa_m & \kappa_m & -\kappa_m \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}$$

where d is the arm length of the quadrotor.

The dynamics of the quadrotor are then described as follows:

$$\begin{aligned} \dot{\boldsymbol{p}}_q^{\mathsf{T}} &= \begin{bmatrix} v_x & v_y & v_z \end{bmatrix} \\ \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{1}{m} \begin{bmatrix} \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi \\ \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} u_1 \\ \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} &= \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \\ \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} &= \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}}\omega_y\omega_z \\ \frac{I_{zz}-I_{xx}}{I_{yy}}\omega_x\omega_z \\ \frac{I_{xx}-I_{yy}}{I_{zz}}\omega_x\omega_y \end{bmatrix} + \begin{bmatrix} \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} \end{aligned} \tag{2.1}$$

During the simulations, we linearized the dynamics of the quadrotor [6].

2.2.3 High-Level Motion Model

To capture the evolution of the UAV's position and velocity on the $x-y$ plane, we use the following high-level model. This simplified model will be used to compute the reachable tubes in Section 2.5.1.

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}(\boldsymbol{u}(t) + \boldsymbol{\eta}_u + \boldsymbol{\eta}_d) \\ \boldsymbol{y}(t) &= \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{\eta}_y \end{aligned} \tag{2.2}$$

where $\boldsymbol{x} = \begin{bmatrix} x & y & v_x & v_y \end{bmatrix}^{\mathsf{T}}$ is the state, $\boldsymbol{y}$ is the position and velocity measurement with sensor noise $\boldsymbol{\eta}_y$. $\boldsymbol{u}$ is the input acceleration with noise $\boldsymbol{\eta}_u$ and the effect of disturbance $\boldsymbol{\eta}_d$.

$\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ matrices are given by:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \boldsymbol{C} = \boldsymbol{I}$$

where $\boldsymbol{I}$ is a 4×4 identity matrix.

This model assumes that the position of the quadrotor along the z axis is constant and the yaw angle is equal to zero.

2.2.4 Position, Low Level, and Attitude Controls

By adopting the framework in Figure 2.2, to follow the desired trajectory $\boldsymbol{x}_\tau = [\ x_\tau \quad y_\tau \quad v_{x,\tau} \quad v_{y,\tau}\ ]^\mathsf{T}$, the position controller which is implemented as a series of PD loops generates the desired acceleration inputs:

$$\ddot{x}_{des}(t) = K_p(x_\tau(t) - x(t)) + K_d(v_{x,\tau}(t) - v_x(t))$$
$$\ddot{y}_{des}(t) = K_p(y_\tau(t) - y(t)) + K_d(v_{y,\tau}(t) - v_y(t))$$

where K_p and K_d are proportional and derivative coefficients of the controller respectively. To provide these desired acceleration inputs, low-level controllers generate the necessary angle inputs to the attitude control. The necessary angular speeds are calculated by the attitude controller and the thrust and moment values that each rotor should provide are calculated through motor dynamics. Finally, the response of the quadrotor to these thrust and moment values in terms of its state is generated by rigid body dynamics.

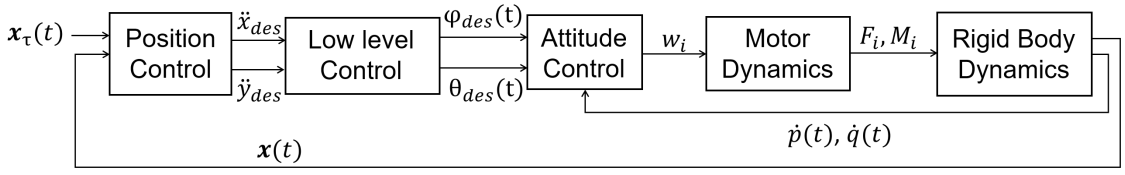


Figure 2.2: The control diagram for a quadrotor trajectory following operation [56].

2.2.5 Noise Models

The behavior of a quadrotor can be negatively affected by various factors such as sensor and process noises, and external disturbances which may cause the vehicle to drift from its planned trajectory and possibly collide with obstacles. Thus, to guarantee safety, these factors should be taken into account during planning.

In this chapter, we assume that the effect of disturbances, noises and uncertainties are uniformly distributed and bounded by ellipsoids ϵ. It should be noted that beside simplifying the calculation of reachable sets, this is a valid assumption because high uncertainties with very low probability can be neglected and thus the value can be bounded. For ease of discussion and computation, the noise values are drawn from a uniform distribution. Note that this assumption is valid because we want to treat all uncertainty values in the same way from a safety point of view. It should be also noted that the presented framework is independent from the distribution of the noise values as long as they can be bounded by ellipsoids. The state uncertainty caused by the sensor measurement noise is represented by $\boldsymbol{\eta}_y \in \epsilon(\mathbf{0}, \boldsymbol{Y})$ which represents an ellipsoid with center around the origin and shape matrix $\boldsymbol{Y}$. The combination of the mechanical uncertainties from the rotors, gears and propellers, and the low-level controllers $\boldsymbol{\eta}_u$, as well as the effect of the external wind disturbance $\boldsymbol{\eta}_d$ are considered as noise on the applied input. The total uncertainty on the input is assumed to be bounded by an ellipsoid $(\boldsymbol{\eta}_u + \boldsymbol{\eta}_d) \in \epsilon(\mathbf{0}, \boldsymbol{U})$ centered around the origin with shape matrix $\boldsymbol{U}$.

2.2.6 Assumptions

We assume that the given motion and noise models capture the behavior of the system's dynamics and the noises and disturbances. The system is assumed to be equipped with necessary omnidirectional sensors to detect and track all the obstacles within its rage accurately. The dynamic obstacles are assumed to move in a fixed direction.

2.3 Problem Formulation

This section formulates the problems we address in this chapter. The first problem is formally defined as follows:

Problem 1: ***Self-triggered Replanning*:** A UAV has an objective to visit one or more goal locations in a cluttered environment. The positions of the obstacles are detected using an on-board range sensor and a trajectory to the desired goals is generated online considering obstacle locations.

We consider two cases within the scope of this problem. In the first case the UAV does not monitor its pose and range sensors and moves with open loop controller between planning times. Since monitoring the pose sensors is not computationally as expensive as range sensors, we introduce also a second case in which the vehicle monitors its pose sensor periodically and moves with a closed loop controller between replanning times.

Case 1: *Open Loop Scheduling and Replanning*: Given the UAV dynamics introduced in Section 2.2.2, find a policy to schedule the next pose and range sensor monitoring and replanning time t_{p+1}, while the

UAV is operating with a precomputed sequence of inputs in open loop guaranteeing both safety and liveness constraints between replanning operations as mathematically represented in (2.3) and (2.4).

Case 2: *Closed Loop Scheduling and Replanning*: Find a policy to schedule next range sensor monitoring and replanning time t_{p+1}, while the UAV is checking periodically its pose sensor and operating in closed loop guaranteeing both safety and liveness constraints between replanning operations as mathematically represented in (2.3) and (2.4).

In both cases, the next replanning time is determined such that the following safety and liveness requirements between replanning operations are met:

1. *Safety Constraint:* Collisions with obstacles should be avoided between two consecutive replanning times, or mathematically:

$$\|\boldsymbol{p}(t) - \boldsymbol{o}_i(t)\| > r_o^i, \forall t \in [t_p, t_{p+1}], \forall i \in \{1, \cdots, n_o\} \tag{2.3}$$

   in which $\boldsymbol{p}(t) = [\ x(t) \quad y(t)\ ]^\mathsf{T}$ is the position of the quadrotor and $\boldsymbol{o}_i(t) = [\ x_o^i(t) \quad y_o^i(t)\ ]^\mathsf{T}$ is the position of the i^{th} obstacle in the $x-y$ plane at time t with n_o the number of obstacles in the environment and r_o^i is the radius of the i^{th} obstacle. Note that in static environments, the obstacle positions do not change over time.

2. *Liveness:* The UAV should stay within a certain proximity of the planned trajectory:

$$\|\boldsymbol{p}(t) - \boldsymbol{p}_\tau(t)\| \leq \lambda_d, \forall t \in [t_p, t_{p+1}] \tag{2.4}$$

   where $\boldsymbol{p}_\tau(t)$ is the desired position of the the quadrotor on the trajectory at time t, and λ_d is the allowed deviation threshold.

If the UAV is following its trajectory without deviating too much, replanning operations can be relaxed while still operating safely. To minimize unnecessary replanning operations we introduce the following problem:

Problem 2: ***Replanning Relaxation*:** Given the assumptions in Problem 1, find a policy to decide next time t_{s+1} to monitor the state of the system and postpone next replanning time, obtained in Problem 1, to a later $t_{p+1}^* \geq t_{p+1}$ such that $t_{p+1} \leq t_{s+1} \leq t_{p+1}^*$ and the same safety and liveness constraints hold:

1. *Safety Constraint:* $\|\boldsymbol{p}(t) - \boldsymbol{o}_i(t)\| > r_o^i, \forall t \in [t_p, t_{s+1}], \forall i \in \{1, \cdots, n_o\}$

2. *Liveness Constraint:* $\|\boldsymbol{p}(t) - \boldsymbol{p}_\tau(t)\| < \lambda_d, \forall t \in [t_p, t_{s+1}]$

In cluttered environments, avoiding obstacles may require a UAV to follow winding trajectories which may lead to a significant drift from the planned trajectory, especially when the speed of the vehicle is high. We formally cast this problem as follows:

Problem 3. ***Speed Adaptation*:** Given (2.1), and assumptions listed in the previous problems, at replanning time t_p, after defining a trajectory τ, find a policy to determine the maximum speed v^* such that the following conditions are satisfied:

$$d_{avg}(\kappa_m, v^*) \leq \xi_t, \text{with } v^* \in [v_{min}, v_{max}]$$

where d_{avg} is the average deviation from the planned trajectory with maximum curvature κ_m, ξ_t is a deviation threshold defined by the user, and v_{min} and v_{max} are the minimum and maximum allowed UAV speeds, respectively.

2.3.1 Sample Scenario

As a baseline reference to our proposed approach, we use a sample case study: a UAV is tasked to travel 40m along the x direction with constant speed $v = 1.0$m/s in a cluttered environment under wind disturbance $\boldsymbol{d} = [-0.1, 0.1]^\mathsf{T}$m/s. The UAV is equipped with an omnidirectional range sensor with 10m range which is used to detect the obstacles in the environment. It is assumed that the vehicle is able to detect all the obstacles within the sensor range.

In traditional motion planning methods, the state of the system and the obstacle positions are monitored at high frequency and replanning occurs whenever the UAV detects a new obstacle along its path [50] [31]. Monitoring the range or vision sensors to detect the obstacles at high frequency brings unnecessary computational burden to the system. In Figure 2.3, the UAV travels to its goal position using this traditional motion planning approach under the conditions described in the sample scenario. The desired trajectory of the UAV is shown by a red curve and its actual path is shown by a blue dotted line. The maximum deviation from its desired trajectory is recorded as 11.42cm. Replanning (depicted by black cross marks $\times$) happens 13 times through the execution of the operation. In order to detect and avoid obstacles, pose and range sensor are monitored periodically at 40Hz rate to resemble the rate of real lidars and are depicted by magenta diamonds 1721 times in Figure 2.3.

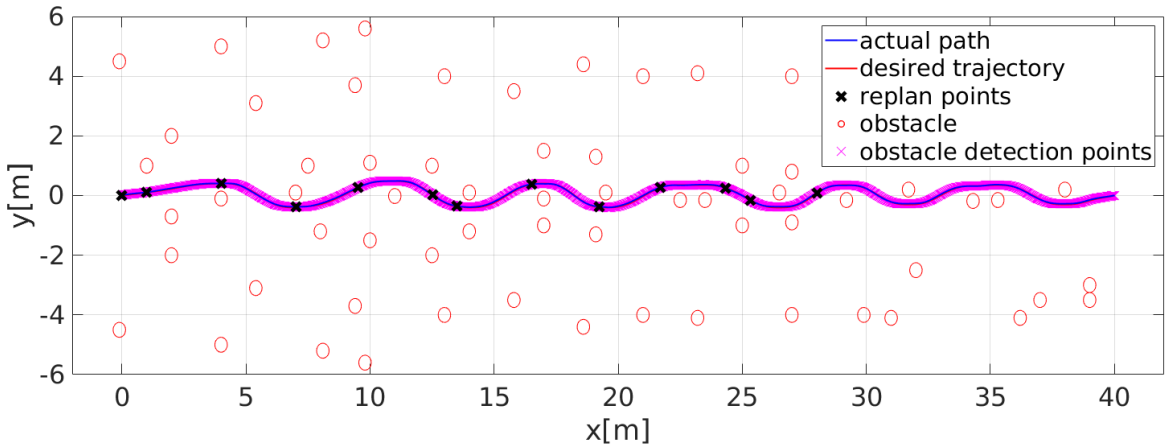


Figure 2.3: Periodic sensor monitoring and obstacle detection.

2.4 Framework

To solve the problems listed in Section 2.3, we propose a reachability-based self-triggered scheduling and replanning approach which can be broken down in to several components, as depicted in the framework shown in Figure 2.4. The framework consists of reachability analysis both for UAVs and dynamic obstacles, self-triggered scheduling, replanning relaxation, curvature-based speed adaptation and event-triggered obstacle avoidance. These individual components are analyzed in detail in Sections 2.5 and 2.6.

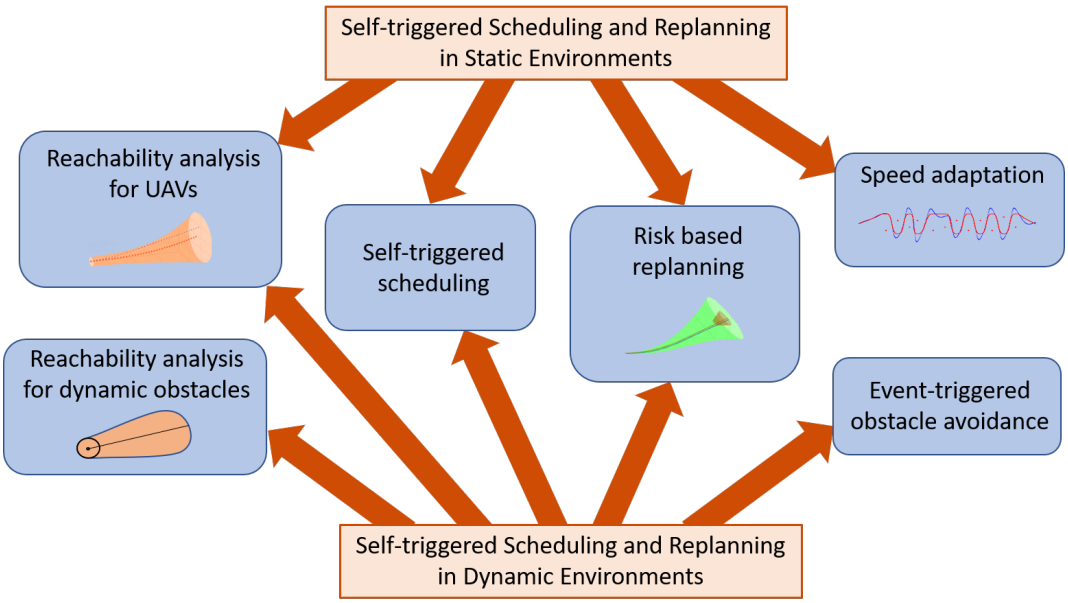


Figure 2.4: Overall reachability-based self-triggered scheduling and replanning framework for dynamic and static environments.

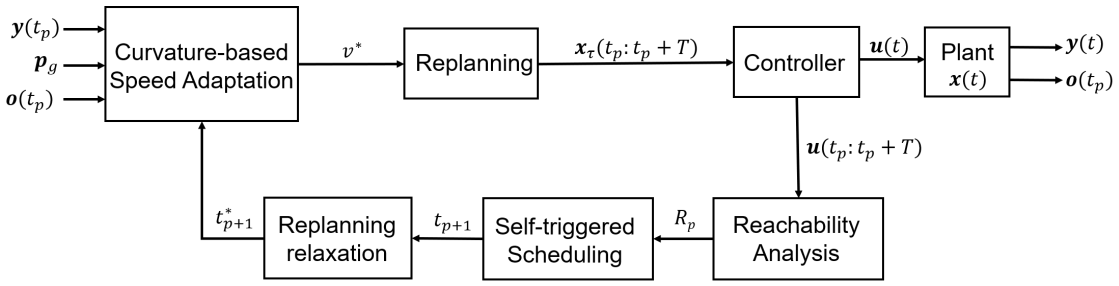


Figure 2.5: Self-triggered scheduling and replanning framework with speed adaptation.

2.5 Self-Triggered Scheduling and Replanning for Static Environments

To use the available computational resources in an efficient way, we introduce scheduling policies to make decisions about next sensor monitoring and replanning times. Our approach leverages reachability analysis to predict the future states of the system and utilizes self-triggered control for scheduling. In this section, we also introduce i) an approach to update reachable sets and relax replanning operations to further minimize computation and ii) a curvature-based speed adaptation method for better tracking performance. The overall framework is shown in Figure 2.5.

2.5.1 Reachablity Analysis for Trajectory Tracking on Quadrotors

A *reachable set* or *reach set* of a system is defined as the set of states that can be reached from a given initial state with an admissible input over a certain time horizon [39]. A reachable set computed at time t_0 for a future time t_f and represented by $R(\boldsymbol{x}_0, \boldsymbol{u}(t), t_f)$ is an ellipsoid ϵ that contains all the possible future states $\boldsymbol{x}(t)$ for $t_0 \leq t \leq t_f$ where the initial set $\epsilon(\boldsymbol{x}_0, \boldsymbol{X}_0)$ is an ellipsoid with center $\boldsymbol{x}_0$ and shape matrix $\boldsymbol{X}_0$ and the input $\boldsymbol{u}(t) \in \epsilon(\boldsymbol{u}(t), \boldsymbol{U})$ is bounded by an ellipsoid with center $\boldsymbol{u}(t)$ and shape matrix $\boldsymbol{U}$. $R^+(\boldsymbol{x}_0, \boldsymbol{u}(t), t_f)$ is the external bound of the reachable set and $R^+_p(\boldsymbol{x}_0, \boldsymbol{u}(t), t_f)$ is the projection of the external bound of the reachable set on the position space.

The external bound for the reach set at time t_f starting from an initial time t_0 is calculated based on the initial state ellipsoid, the plant model, and the input ellipsoid as follows:

$$R^+(\boldsymbol{x}_0, \boldsymbol{u}(t), t_f) = \boldsymbol{\Phi}(t_f, t_0)\epsilon(\boldsymbol{x}_0, \boldsymbol{X}_0) \oplus \int_{t_0}^{t_f} \boldsymbol{\Phi}(t_f, \zeta)\boldsymbol{B}\epsilon(\boldsymbol{u}(\zeta), \boldsymbol{U})d\zeta$$

where $\boldsymbol{\Phi}(t, t_0) = e^{\boldsymbol{A}(t-t_0)}$, and the symbol $\oplus$ represents geometric sum. A reachable tube $R(\boldsymbol{x}_0, \boldsymbol{u}(t), [t_0, t_0+T])$ is the set of all reachable sets over the time interval $\Delta T = [t_0, t_0 + T]$ and its external bound is described as

follows:

$$R^+(\boldsymbol{x}_0, \boldsymbol{u}(t))|_{t_0}^{t_0+T} = R^+(\boldsymbol{x}_0, \boldsymbol{u}(t), [t_0, t_0+T]) = \bigcup_{t_0}^{t_0+T} R^+(\boldsymbol{x}_0, \boldsymbol{u}(\zeta), \zeta) d\zeta$$

which is the union of all reachable sets from time t_0 to $t_0 + T$. $R_p^+(\boldsymbol{x}_0, \boldsymbol{u}(t), [t_0, t_0+T])$ is the projection of the external bound of the reachable tube on to the position space.

For a UAV following a desired trajectory $\boldsymbol{x}_\tau(t_p : t_{p+1})$, the reachable sets are generated over the time interval $\Delta t_p = [t_p, t_{p+1}]$ where t_p is the current replanning time and t_{p+1} is the next replanning time, initially set to $t_p + T$ before the rescheduling operation. The desired trajectories are computed minimizing jerk [55], and they contain the desired positions and velocities that the UAV has to track along the path. Using a PD controller, the acceleration input required to track the trajectory is calculated as follows:

$$\boldsymbol{u}(t) = K_P(\boldsymbol{p}_\tau(t) - \boldsymbol{p}(t)) + K_D(\dot{\boldsymbol{p}}_\tau(t) - \dot{\boldsymbol{p}}(t))$$

where $t \in [t_p, t_{p+1}]$. In order to calculate the set of inputs that would be applied to the UAV during its motion, an online simulation is run as if there is no disturbance and noise in the environment. The control input $\boldsymbol{u}(t)$ is calculated using this PD controller and applied to the simulated system for $t \in [t_p : t_{p+1}]$. As a result, a set of control inputs $\boldsymbol{u}(t_p : t_{p+1})$ is generated.

The set of inputs calculated using the online simulator are used to construct the reachable tubes considering the disturbances and uncertainties. A position reachable tube constructed from $t_p = 0$ to $t_{p+1} = 1.0$s for a quadrotor following a straight 1m long trajectory in the $+x$ direction in open loop is shown in Figure 2.6. The actual path of the quadrotor (blue dotted curve) deviates from the desired trajectory (red start curve) due to the presence of wind disturbance $\boldsymbol{d} = [0, 0.45]$m/s. Nevertheless, the path of the quadrotor is contained inside the reachable tube since the system uncertainties and disturbances are taken into consideration during the reachable set computation. To perform such reachability analysis, we leveraged the Ellipsoidal Toolbox [39] for ease of integration with our Matlab simulations and physical experiments. However, any other reachability tool could be used within our framework.

2.5.2 Self-triggered Scheduling

To schedule next sensor monitoring time while guaranteeing safety and liveness between replanning operations, we leverage the reachable tubes calculated in the previous section. In this section we introduce our self-triggered scheduling approach to guarantee safety and liveness between aperiodic replanning operations.

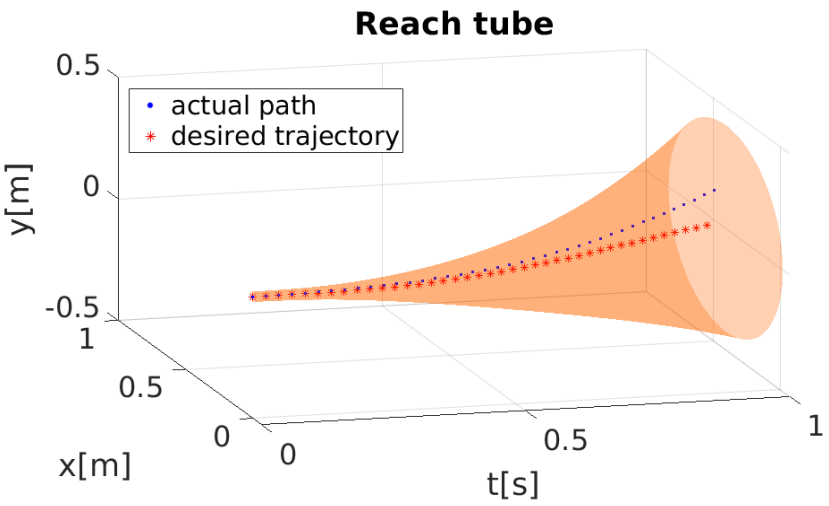


Figure 2.6: The reachable tube associated with a planned trajectory for a quadrotor moving in the $+x$ direction over a time interval [0.0,1.0]s.

Case 1: Open Loop Scheduling and Replanning

Our novel self-triggered scheduling policy consists of deciding the next sensor monitoring and motion replanning time t_{p+1} such that safety and liveness of the UAV are guaranteed between replanning times t_p and t_{p+1} without monitoring its pose and range sensors between replanning times, for example in case of GPS signal loss. The safety requirement is violated whenever a collision with an obstacle becomes possible. We calculate the first time in which the UAV can collide with an obstacle, t_c, using a reachable tube as follows:

$$t_c = \min(t | R_p^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t \in [t_p, t_p + T]) \cap \boldsymbol{O} \neq \emptyset) \tag{2.5}$$

where $\boldsymbol{O}$ is the set of obstacles detected at time t_p.

As the UAV does not constantly check its range sensor while moving towards its goal, it may leave the region that is detected by the range sensor. This situation raises a potential danger of obstacle collisions because the UAV does not have information about the environment beyond its sensory range. The earliest time that the UAV can leave the region sensed by the range sensor field of view, t_l is calculated as follows:

$$t_l = \min(t | R_p^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t \in [t_p, t_p + T]) \not\subset \boldsymbol{r}(t_p)) \tag{2.6}$$

where $\boldsymbol{r}(t_p)$ is the region covered by the field of view of the range sensor of the UAV at the planning time t_p.

Liveness constraint might be violated if it is possible for the UAV to deviate more than a threshold λ_d from its desired trajectory. The first time in which the liveness condition might be violated, t_d, is calculated

as follows:

$$t_d = \min(t | \|R_p^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t \in [t_p, t_p + T]) - \boldsymbol{p}_\tau(t)\| > \lambda_d) \tag{2.7}$$

where $\boldsymbol{p}_\tau(t)$ is the desired position of the UAV along the computed trajectory at time $t \in [t_p, t_p + T]$.

Before the end of the planning horizon $t_p + T$, if one or more of the following conditions occur:

- a collision with an obstacle becomes possible at $t_c < t_p + T$
- the deviation could be larger than the permitted threshold t time $t_d < t_p + T$
- the vehicle may leave the region covered by the range sensor at time $t_l < t_p + T$

then the UAV needs to check its state at one of these three times, whichever is the earliest. Otherwise, the next replanning time is scheduled at the end of the time horizon.

$$t_{p+1} = \begin{cases} \min(t_c, t_d, t_l) - t_r, & \text{if } t_c < t_p + T \text{ or } t_d < t_p + T \text{ or } t_l < t_p + T \\ t_p + T - t_r, & \text{otherwise} \end{cases} \tag{2.8}$$

where t_r is the amount of time necessary for the replanning calculation.

Lemma 1 *Given t_{p+1} as defined in* (2.8)*, the UAV is guaranteed to stay within λ_d proximity of its planned trajectory, not to collide with any obstacle and to stay within the region covered by the range sensor field of view.*

Proof: By definition, $R(\boldsymbol{x}_0, \boldsymbol{u}(t), t \in [t_p, t_p + T])$ is the set of all states $\boldsymbol{x}$, such that there exists an input $\boldsymbol{u} \in \epsilon(\boldsymbol{u}(t), \boldsymbol{U})$ and an initial state $\boldsymbol{x}_0 \in \epsilon(\boldsymbol{x}_0, \boldsymbol{X}_0)$ which steers the UAV from $\boldsymbol{x}_0$ to $\boldsymbol{x}$ in time t [39]. $R_p^+(\boldsymbol{x}_0, \boldsymbol{u}(t), t \in [t_p, t_p + T])$ is the projection of the external bound of $R(\boldsymbol{x}_0, \boldsymbol{u}(t), t \in [t_p, t_p + T])$ onto the position space. Considering a time value t^* between t_p and t_{p+1}, $t_p < t^* < t_c$ and $t_p < t^* < t_d$, by the definitions of t_c in (2.5), t_l in (2.6) and t_d in (2.7),

$$\begin{aligned} & R_p^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t_k), t \in [t_p, t^*]) \cap \boldsymbol{O} = \emptyset \\ & \|R_p^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t \in [t_p, t^*]) - \boldsymbol{p}_\tau(t)\| < \lambda_d \\ & R_p^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t \in [t_p, t_p + T]) \subset \boldsymbol{r}(t_p) \end{aligned}$$

which proves that for a $t_p < t^* < t_{p+1}$, there doesn't exist an input $\boldsymbol{u}(t^*) \in \epsilon(\boldsymbol{u}(t^*), \boldsymbol{U})$ and an initial state $\boldsymbol{x}(t_p) \in \epsilon(\boldsymbol{x}(t_p), \boldsymbol{X}_0)$ which makes the system reach a state $\boldsymbol{x}$ from $\boldsymbol{x}(t_p)$ such that the position of the system intersects with an obstacle, or leaves the region sensed by the range sensor, or deviates from the desired trajectory more than λ_d.

■

In Figure 2.7, we demonstrate the results of this approach for our sample case. The desired trajectory of the UAV is shown by red curve and the actual trajectory followed by the UAV is shown by blue dots. The deviation threshold is picked as $\lambda_d = 50$cm and the wind disturbance is constant everywhere $\boldsymbol{d} = [-0.1, 0.1]^\mathsf{T}$m/s. The UAV checks its states and replans its motion only at the points shown by black crosses 46 times and travels with an open loop controller in between replanning points. The maximum deviation along the path is recorded as 16.69cm. As can be noticed, the UAV never collides with an obstacle or deviates from its desired trajectory more than the permitted threshold thanks to our self-triggered approach.

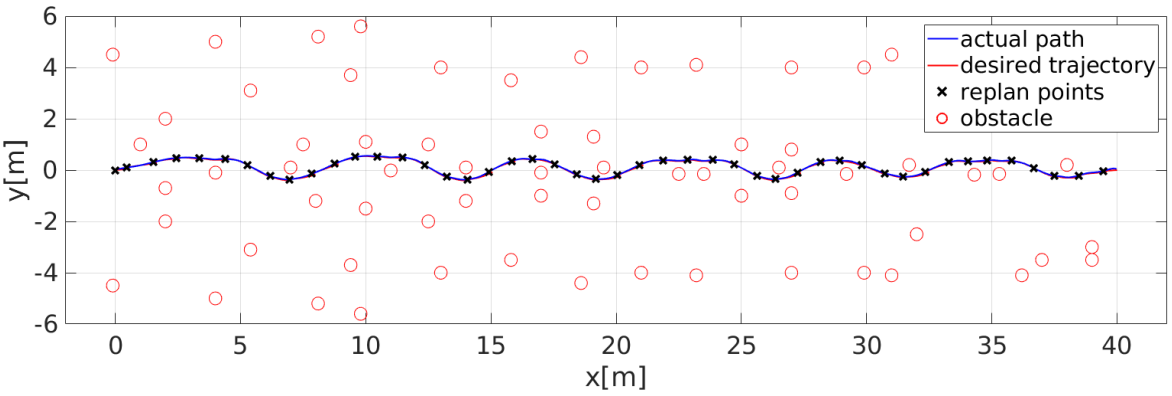


Figure 2.7: Desired trajectory (red curve) and actual path (blue dots) of the quadrotor with self-triggered for position, velocity and obstacle monitoring.

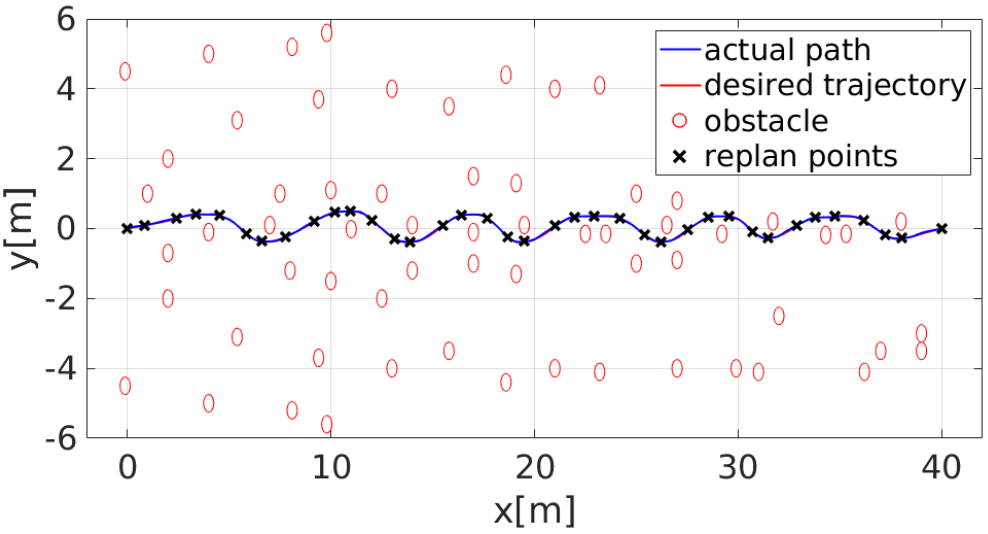
Case 2: Closed Loop Scheduling and Replanning:

Compared to monitoring range sensors, we note that monitoring pose sensors is usually computationally negligible and is necessary for closed loop trajectory-tracking operations. Here, we consider the case in which the UAV can monitor its position and velocity periodically while checking for obstacles is scheduled aperiodically at replanning times. Since the UAV checks its position sensor constantly, the liveness constraint is not taken into consideration in this case. If it becomes possible for the UAV to collide with an obstacle or to leave the region sensed by the range sensor before the end of its time horizon, the UAV monitors its range sensor and replans its trajectory at either t_c or t_l, whichever is earlier.

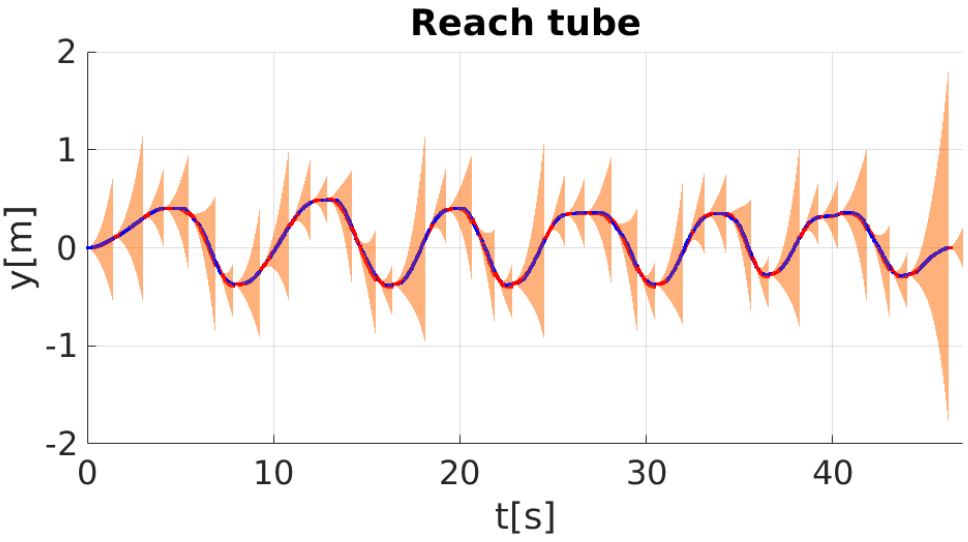
$$t_{p+1} = \begin{cases} \min(t_c, t_l) - t_r, & \text{if } t_c < t_p + T \text{ or } t_l < t_p + T \\ t_p + T - t_r, & \text{otherwise} \end{cases} \tag{2.9}$$

Using this approach, the UAV is guaranteed to stay within its sensor field of view and avoid collisions with an obstacle, following the same reasoning in Lemma 1.

In Figure 2.8, we demonstrate the sample case, where the UAV monitors its position and velocity periodically and schedules next obstacle monitoring time based on the proposed self-triggered scheduling approach. The UAV checks its range sensor to detect the obstacles and replans its motion at points shown by black crosses only 37 times. The desired trajectory is shown by a red curve and the actual path of the UAV is shown by blue dots. The maximum deviation from the desired trajectory is recorded as 11.61cm. To further decrease the number of replanning operations, we update the reachable tubes based on the observed state of the system which is discussed in the next section.



(a) Desired trajectory (red curve) and actual path (blue dots) of the quadrotor with self-triggered scheduling for obstacle monitoring.



(b) Corresponding reachable tubes.

Figure 2.8: Self-triggered scheduling for obstacle detection approach applied to the sample scenario.

2.5.3 Reachable Set Shrinking and Replanning Relaxation

Reachable tubes are constructed to capture the worst case scenario in terms of external disturbances and noises. In ideal conditions, without disturbance, the system can follow its desired behavior very closely, and therefore replanning (i.e., computing a new reachable tube) at each time that the reachable tube collides with an obstacle might be over-conservative and computationally expensive. To overcome this limitation and postpone next monitoring and replanning operations, we leverage the deviation from the center of the reachable set at the replanning time t_{p+1} to update the reachable tube without recalculating them. The deviation from the center of the position reachable tube can be calculated as follows:

$$d(t_{p+1}) = \|\boldsymbol{p}(t_{p+1}) - \boldsymbol{p}_r(t_{p+1})\|$$

where $\boldsymbol{p}(t_{p+1})$ is the position of the UAV at time t_{p+1} and $\boldsymbol{p}_r(t_{p+1})$ is the center of the position reachable tube at t_{p+1}. In the ideal conditions (perfect system model and controller), the position reachable tube would be centered around the desired trajectory. The radius of the external bound of the reachable tube $R_p^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t))|_{t_p}^{t_p+T}$ can be reduced by $r_t(t_{p+1}) - d(t_{p+1}) - \eta_p$ where $r_t(t_{p+1})$ is the radius of the position reachable tube at time t_{p+1} and η_p is the position measurement uncertainty bound. The reasoning behind this shrinking is that if the system has not deviated from the center of the reachable set by a large amount, it becomes impossible for the system to reach the previously computed reachable set border.

By shrinking, we obtain an updated position reachable set $\bar{R}_p^+(\boldsymbol{x}(t_{p+1}), \boldsymbol{u}(t), t)$ with a smaller external bound than the original reachable set. Figure 2.9(a) is a pictorial representation of the reachable tube update procedure where the original position reachable tube $R_p^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t))|_{t_p}^{t_p+T}$ is shown by green region and the updated position reachable tube $\bar{R}_p^+(\boldsymbol{x}(t_{p+1}), \boldsymbol{u}(t))|_{t_{p+1}}^{t_p+T}$ is shown by dark orange region.

Using this approach, the UAV is guaranteed to stay within the updated reachable tube, as formally described in Lemma 2:

Lemma 2 *Given that $\boldsymbol{x}(t_{p+1}) \in R^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t))|_{t_p}^{t_p+T}$, the state of the UAV is guaranteed to stay within the updated reachable tube at any future time $t_{p+1} \leq t \leq t_p + T$:*

$$\boldsymbol{x}(t) \in \bar{R}^+(\boldsymbol{x}(t_{p+1}), \boldsymbol{u}(t))|_{t_{p+1}}^{t_p+T},\ \forall t \in [t_{p+1}, t_p + T]$$

Proof: To prove this lemma, we leverage the geometric properties of reachable sets following the representation in Figure 2.9(b). The external bound for the reachable tube calculated at planning time t_p,

$R_1^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t)$ is given as follows:

$$R_1^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t) = e^{\boldsymbol{A}(t-t_p)}\epsilon(\boldsymbol{x}(t_p), X_0) \oplus \int_{t_p}^{t} e^{\boldsymbol{A}(t-\zeta)}\boldsymbol{B}\epsilon(\boldsymbol{u}(\zeta), \boldsymbol{U})d\zeta$$

for $t_p \leq t \leq t_p + T$. At the time t_{p+1}, a reachable set from the measured state $\boldsymbol{y}(t_{p+1})$ with the same input sequence can be calculated as follows:

$$R_2^+(\boldsymbol{y}(t_{p+1}), \boldsymbol{u}(t), t) = e^{\boldsymbol{A}(t-t_{p+1})}\epsilon(\boldsymbol{y}(t_{p+1}), X_0) \oplus \int_{t_{p+1}}^{t} e^{\boldsymbol{A}(t-\zeta)}\boldsymbol{B}\epsilon(\boldsymbol{u}(\zeta), \boldsymbol{U})d\zeta$$

for $t_{p+1} \leq t \leq t_{p+1} + T$. The reachable set R_1^+ after t_{p+1} can also be written as follows:

$$R_1^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t) = e^{\boldsymbol{A}(t-t_{p+1})}R_1^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t_{p+1}) \oplus \int_{t_{p+1}}^{t} e^{\boldsymbol{A}(t-\zeta)}\boldsymbol{B}\epsilon(\boldsymbol{u}(\zeta), \boldsymbol{U})d\zeta$$

for $t_{p+1} \leq t \leq t_p + T$. By definition, the reachable set at R_1^+ at time t_{p+1} contains the measured state:

$$R_1^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t_{p+1}) \supset \epsilon(\boldsymbol{y}(t_{p+1}), X_0)$$

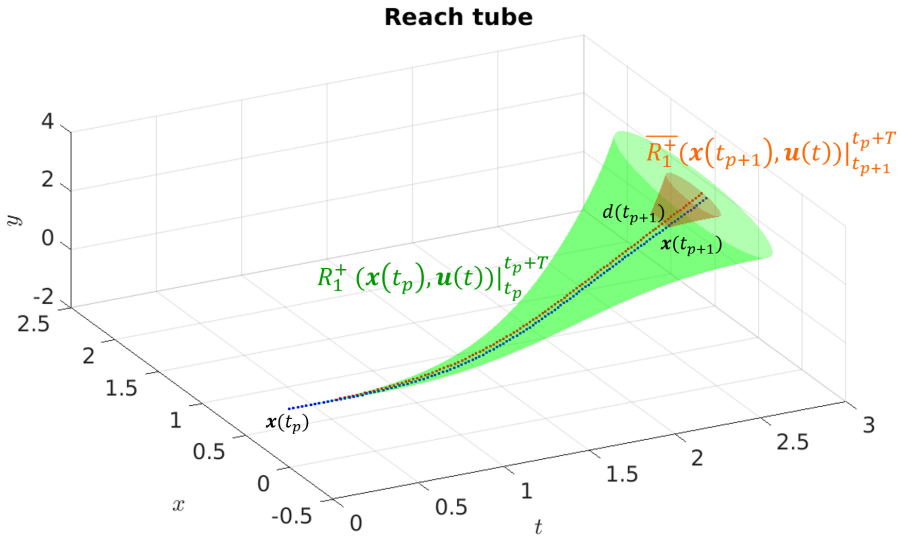
Therefore, at time t_{p+1}, the reachable set R_1^+ contains R_2^+. By the definition of the external bounds of these reachable sets, it can be seen that the radii of both reachable sets grow exponentially over time and at time t_{p+1} the radius of R_1^+ is larger. Therefore, the difference between the reachable set areas increases quadratically with the radius of the reachable sets. Our shrinking procedure consists in reducing the reachable set radius by a constant value $r_t(t_{p+1}) - d(t_{p+1}) - \eta_p$, therefore the initial shrunk set also contains the measured state $\boldsymbol{y}(t_{p+1})$:

$$\bar{R}_1^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t) \supset \epsilon(\boldsymbol{y}(t_{p+1}), X_0)$$

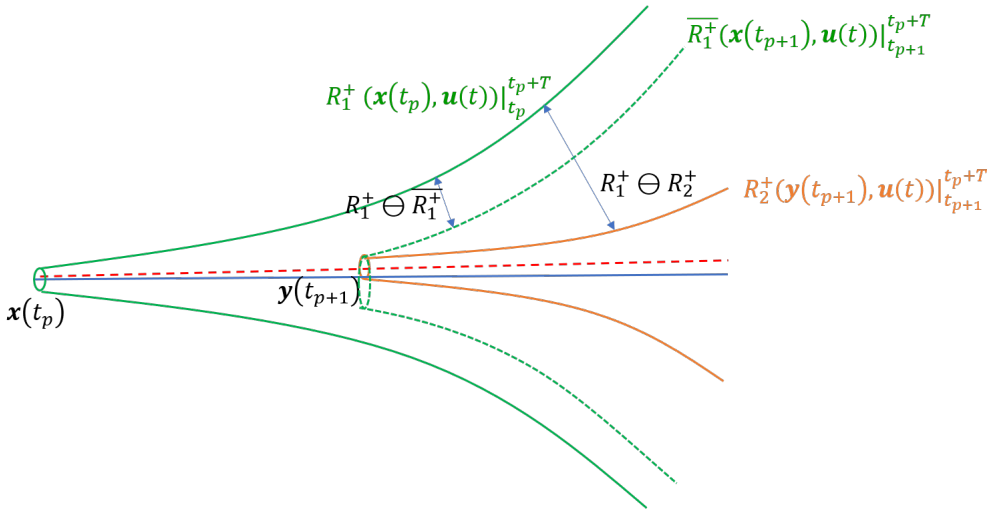
As the shrinking is done with a constant reduction, the difference between the reachable set R_1^+ area and the shrunk reachable set $\bar{R}_1^+$ area grows linearly with the reachable set radius. This concludes that the shrunk reachable set always contains the reachable set if a new set is calculated from the observed state:

$$\bar{R}_1^+(\boldsymbol{x}(t_p), \boldsymbol{u}(t), t) \supset R_2^+(\boldsymbol{y}(t_{p+1}), \boldsymbol{u}(t), t) \supset \boldsymbol{x}(t), \forall t \in [t_{p+1}, t_p + T]$$

■



(a) Updating the reachable tubes based on the deviation from the reachable tube center.



(b) Pictorial explanation of the procedure to update the reachable tubes online.

Figure 2.9: Reachable tube update procedure.

The next scheduling time to update the reachable tube is calculated as defined in Section 2.5.2. When the difference between consecutive scheduling times gets smaller, a *Zeno phenomenon* [18] may occur and lead to an infinite number of reachable tube updates before t_{p+1}^*. In order to prevent such behavior, we consider time threshold δt and include the following constraint:

$$t_{p+1}^* = t_{s+1} \quad \text{if } t_{s+1} - t_s \leq \delta_t \tag{2.10}$$

Equation (2.10) shows that when the scheduled times to update the reachable tube become very close to

each other, a new reachable tube is calculated as described in Section 2.5.1, which prevents infinite number of reachable tube updates caused by the Zeno phenomenon.

In Figure 2.10, using the same sample case described in Section 2.3.1, we demonstrate the results of the UAV motion where the next obstacle monitoring time is scheduled using the shrunk tubes similar to the one shown in Figure 2.9(a). The UAV checks its range sensor to detect the obstacles and replans its motion at points shown by black crosses only 17 times whereas without replanning relaxation, replanning occurs 37 times as shown in Section 2.5.2. The desired trajectory is shown by red curve and the actual path of the UAV is shown by blue dots. The maximum deviation from the desired trajectory is recorded as 11.80cm.

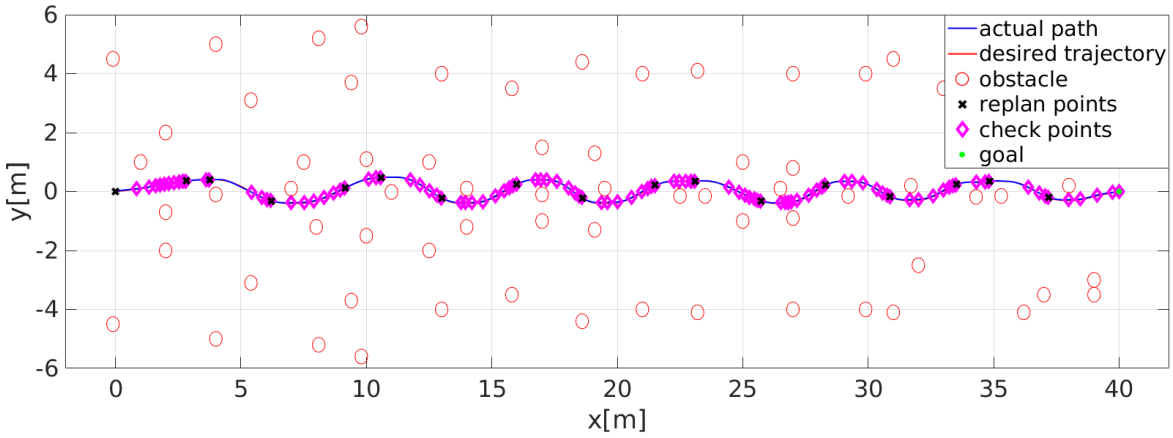


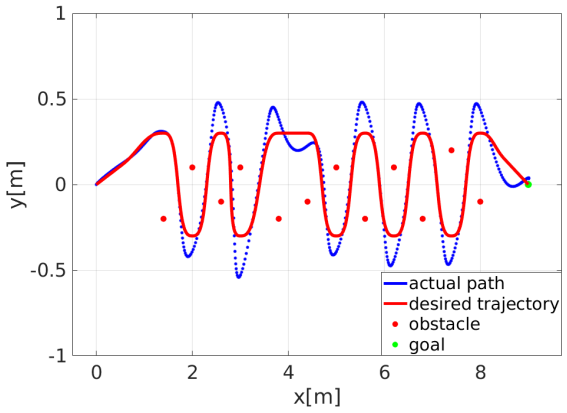
Figure 2.10: Replanning relaxation approach applied to sample scenario.

2.5.4 Curvature Based Speed Adaptation

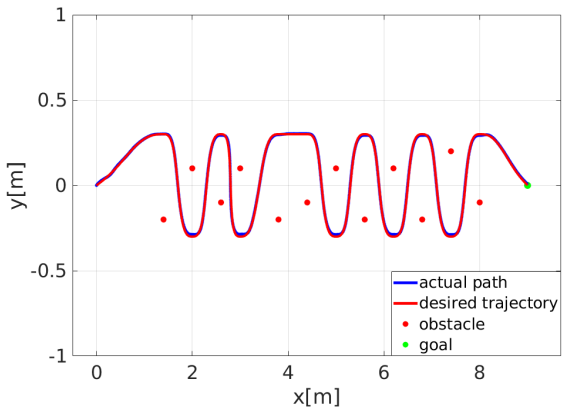
To guarantee liveness constraints, the UAV needs to follow its desired trajectory closely. In cluttered environments, trajectories may become very curvy in order to avoid the obstacles. Even though a UAV can perform better tracking performance with low speeds, it becomes very hard to achieve tracking with high speeds without drifting from the planned trajectory. In contrast, trajectories with low curvatures can be closely followed even with high speeds. For example, in Figure 2.11, the actual path of a UAV is compared when it is following the same desired trajectory (red curve) with different speeds: $v = 1$m/s in Figure 2.11(a) and $v = 0.25$m/s in Figure 2.11(b). As expected and can be noticed by comparing the two figures, the UAV is able to follow its trajectory with less deviation with a lower speed.

To decide the optimal speed to use for a given path, we create a policy which adapts the desired speed to follow an obstacle avoidance path based on the maximum curvature along the path. For curvature computation, we leverage the analysis presented in [1], as follows:

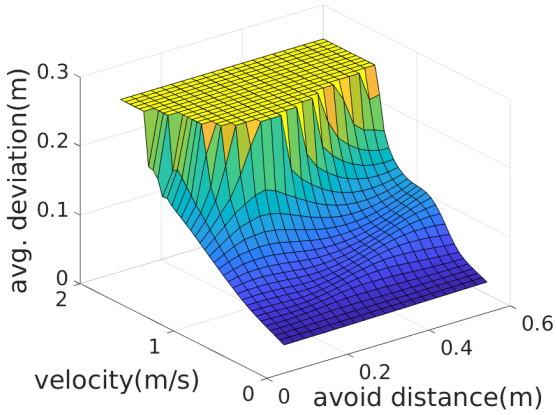
$$\kappa_i = \frac{4A}{d_{(i-1)i} d_{i(i+1)} d_{(i-1)(i+1)}}, \quad i \in \{1, \cdots, n-1\} \tag{2.11}$$



(a) The path of the UAV moving with $v = 1$m/s on a path with high curvature, drifting on average 9.05cm.



(b) The path of the UAV moving with $v = 0.25$m/s on the same path, drifting on average 1.50cm.



(c) The experimental relationship between velocity, avoid distance from obstacle and the average deviation from the trajectory.

Figure 2.11: Effects of the velocity on the deviation. 2018 ©IEEE

where d_{ij} is the distance between two waypoints i and j, A is the area of the triangle formed by three consecutive waypoints, and n is the total number of waypoints. Here, waypoints are the points that the UAV needs to visit in order to avoid the obstacle. In Figure 2.11(c), the experimental results demonstrating the relationship between the velocity, desired avoiding distance from the obstacle (which closely affects the curvature of the path) and average deviation from desired trajectory are shown. As can be noticed, the average deviation from the desired trajectory increases with the velocity. When the avoid distance from the obstacles are set around 0.3m, the curvature becomes the maximum, resulting in more average deviation from the trajectory for the same speed. These results suggest an exponential relationship between the estimation of average deviation from the desired trajectory and the maximum curvature of the path and the speed of

the UAV as follows:

$$d_{avg}(\kappa_m, v) = \frac{1}{\Omega} e^{\kappa_m \cdot v} \tag{2.12}$$

where d_{avg} is the average deviation from the trajectory during its motion. The parameter Ω is determined using the experimental data so that the estimation is an over-approximation, and it is different for different types of vehicles. κ_m is the maximum curvature along the trajectory:

$$\kappa_m = \max_{i \in [1, n-1]} \kappa_i$$

The objective of the UAV is to finish its task as fast as possible without deviating too much from the planned trajectory. Therefore, the optimal velocity for the UAV v^* is calculated as the largest possible velocity which keeps the average deviation under a given threshold ξ_t:

$$\mathcal{V} = \{v : d_{avg}(\kappa_m, v) < \xi_t,\ v_{min} < v < v_{max}\}$$

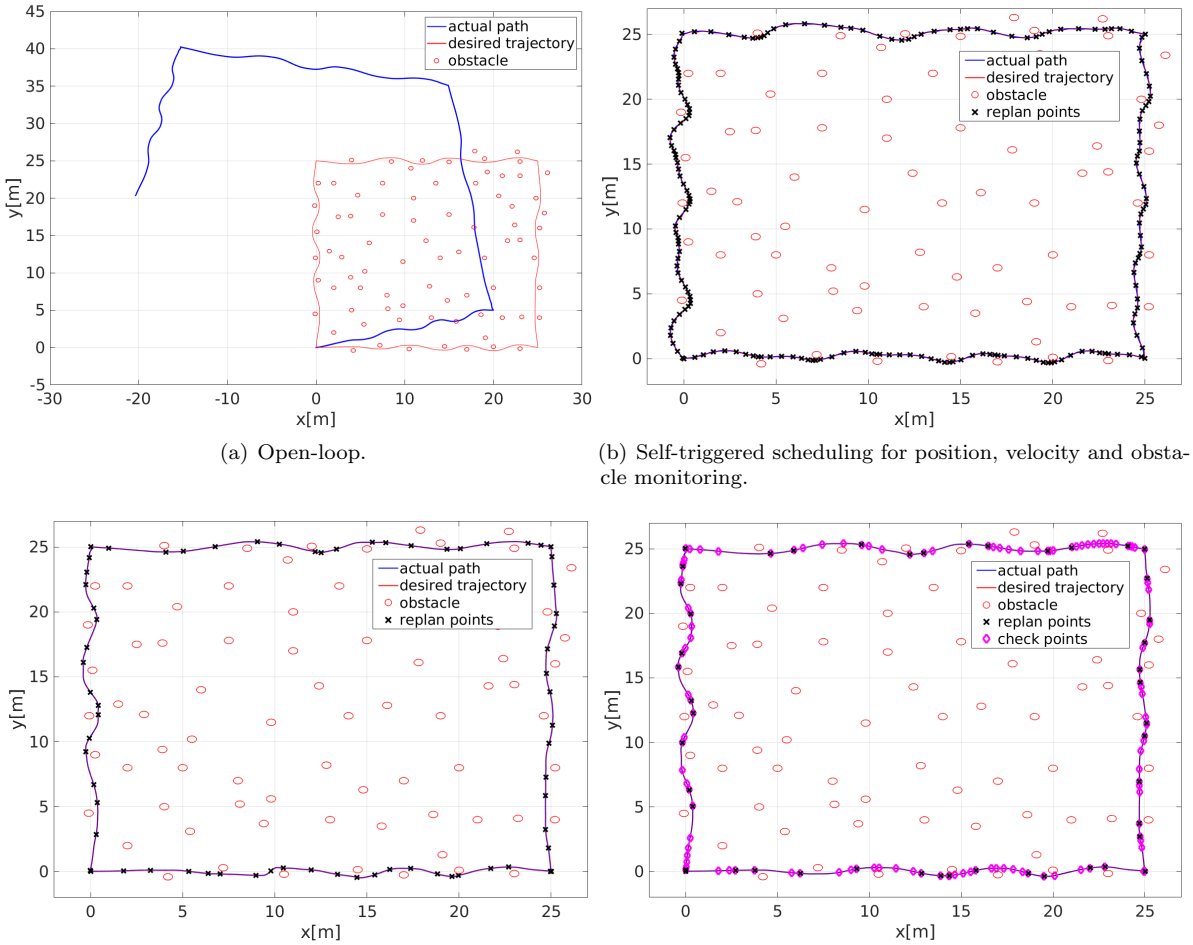
$$v^* = \max(\mathcal{V}) \tag{2.13}$$

where $\mathcal{V}$ is the set of allowable velocities between v_{min} and v_{max}, that keeps the average deviation d_{avg} below ξ_t.

2.5.5 Simulation Results

In this section, we show and compare the simulation results of each approach presented above for a UAV waypoint navigation case study under sensor and process noises, and wind disturbance in an environment cluttered with circular obstacles. The quadrotor UAV that we considered during these simulations uses a localization sensor (e.g., GPS) to monitor its position in the environment and a range sensor (e.g., a lidar) with a limited 10m range and a 360° field of view to detect the obstacles. The UAV is initialized at the origin with zero velocity. Its mission consists in navigating a square trajectory with 25m sides. The wind disturbance $\boldsymbol{d} = [-0.1, 0.1]^\mathsf{T}$ is present everywhere in the environment and its value is unknown to the vehicle.

In Figure 2.12(a), the trajectory of a quadrotor is planned in the beginning of its motion assuming that the exact positions of all the obstacles are known a priori. The trajectory with constant desired speed of 0.5m/s is controlled in open-loop without getting feedback from neither the localization nor the range sensors. Since the quadrotor does not observe its position while moving under the effect of disturbance, it deviates from its planned trajectory significantly and collides with multiple obstacles.



(a) Open-loop.

(b) Self-triggered scheduling for position, velocity and obstacle monitoring.

(c) Self-triggered scheduling for obstacle monitoring.

(d) Replanning relaxation.

Figure 2.12: Comparison between the simulation results of different techniques.

In the simulations demonstrated in Figure 2.12 (b-d) the UAV does not have any prior knowledge about the obstacle locations in the environment. In Figure 2.7, the UAV computes its reachable sets considering the disturbance and noise bounds and uses these reachable sets to schedule the times that it needs to monitor its range and pose sensor as described in Section 2.5.2. The quadrotor moves in open loop between replanning times and follows a trajectory with constant desired speed of 0.5m/s. In this case, the replanning occurs 194 times at points shown by black '$\times$' symbols in Figure 2.12(b). Even though the UAV does not monitor its sensors between replanning times, it is able to complete its task without colliding with any obstacles or deviating too much from the desired trajectory. In Figure 2.12(c), the pose sensor is monitored periodically (usually not computationally demanding) and the range sensor is monitored aperiodically at times scheduled according to proposed self-triggered approach described in Section 2.5.2. Additionally, depending on the curvature of the trajectory, the speed of the vehicle is also adapted between $v_{min} = 0.25$m/s and $v_{max} = 1.25$m/s as explained in Section 2.5.4 The replanning and range sensor detection happens only 70

times at the points shown by black '×' symbols and the vehicle completes its task without colliding with any obstacles. To further minimize the sensor checking and replanning operations, we apply the proposed reachable set update method presented in Section 2.5.3 in the case shown in Figure 2.12(d). At the points shown by magenta '◇' symbols, the UAV updates the reachable sets based on its position and schedules the next raplanning time. With the proposed approach, the UAV completed its task by planning its trajectory only 47 times (black '×') and by tracking its desired trajectory closely with an average deviation of 6.18cm. If instead, the UAV was using periodic sensor monitoring, it would have needed over 4000 range sensor checks, which is avoided thanks to our proposed self-triggered approach.

The number of replanning operations, number of range sensor checks, and the maximum and average deviation values for different cases are compared in Table 2.1. The best results are achieved using the self-triggered replanning approach with relaxation (Figure 2.12(d)). The UAV performs a reasonably good tracking performance with the minimum number of sensor monitoring and replanning operations.

Table 2.1: Comparison between the different simulation cases.

	Number of replanning and sensor monitoring	Max. deviation	Avg. deviation
Figure 2.12(a)	1	2878.84cm	1432.67cm
Figure 2.12(b)	194	16.71cm	4.27cm
Figure 2.12(c)	70	13.66cm	5.63cm
Figure 2.12(d)	47	14.61cm	6.18cm

2.5.6 Experimental Results

In order to validate our proposed self-triggered replanning approach, we implemented a series of experiments: i) self-triggered scheduling for both pose and range sensor monitoring (Case 1 presented in Section 2.5.2) and ii) self-triggered scheduling for only range sensor monitoring (Case 2 presented in Section 2.5.2).

Case 1: Open Loop Scheduling and Replanning

The proposed self-triggered approach to schedule the pose sensor monitoring times was validated using an AscTec Hummingbird quadrotor UAV. The range sensor scheduling is not addressed during this experiment. As shown by the framework in Figure 2.13, the Matlab ellipsoidal toolbox [39] is used to calculate the reachable sets and the control commands to the quadrotor are sent through Robot Operating System (ROS). The communication between Matlab and ROS was bridged using the Robotics System Toolbox. The self-triggered scheduling and replanning framework was implemented in Matlab and the position and the velocity of the quadrotor in $x-y$ plane at each scheduled monitoring time were monitored using a Vicon motion capture system. During this experiment, the obstacle is assumed to be static with a known position.

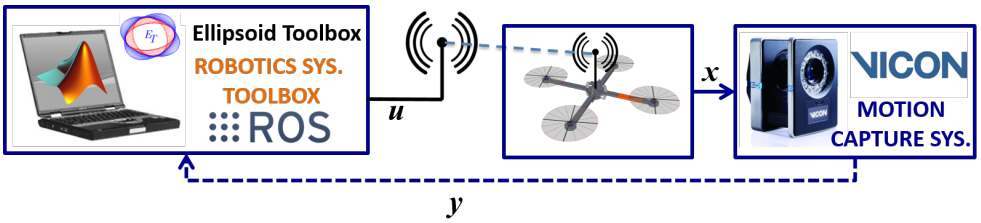


Figure 2.13: Framework of the experimental setup. 2017 ©IEEE

The starting position of the robot is at $(-1.75, 0.4, 1)$m and the goal is located at $(1.75, 0.4, 1)$m. The quadrotor aims to track the planned obstacle-free trajectory without performing periodic sensor monitoring for its position and velocity in $x-y$ plane. The next sensor monitoring time is scheduled considering only the safety condition during this experiment. In Figure 2.14(a), a sequence of snapshots of the quadrotor is shown in which the quadrotor monitor its position with the rate of 40Hz. In Figure 2.15(a), the corresponding obstacle-free desired trajectory (red dashed curve) and the actual path of the UAV (blue curve) are shown. Thanks to periodic sensor checking, the actual and the desired trajectory are very close to each other. In Figure 2.14(b), a sequence of snapshots of the quadrotor while it is performing self-triggered scheduling and replanning is presented. In Figure 2.15(b), the actual and the desired trajectory of the quadrotor is shown. The quadrotor monitors its state and replans its trajectory only 13 times at points shown by black '$\times$' symbols. Lack of periodic sensor monitoring caused the quadrotor to deviate from its planned trajectory by nearly 0.5m, however our proposed approach prevented it from colliding with any obstacles. These experiments show that even though periodic sensor monitoring provides better trajectory tracking, it is not necessary as the safety can be guaranteed using the proposed approach with aperiodic monitoring.

(a) (b)

Figure 2.14: Obstacle avoidance experiment results with (a) periodic position and velocity monitoring and (b) self-triggered scheduling for sensor monitoring and replanning considering only safety constraint. 2017 ©IEEE

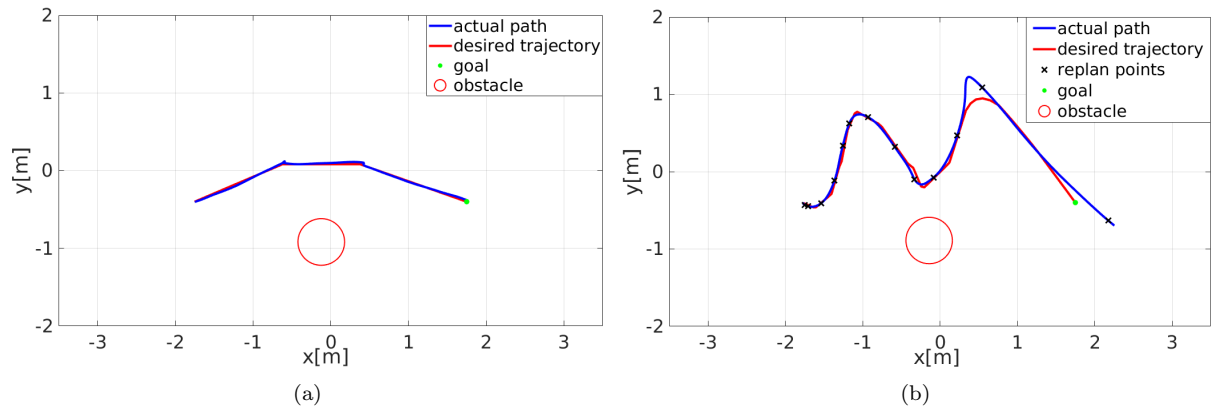


Figure 2.15: The desired and actual path of the quadrotor for the experiments shown in Fig. 2.14. (a) periodic position and velocity monitoring and (b) self-triggered scheduling and replanning with only safety constraint. 2017 ©IEEE

Case 2: Closed Loop Scheduling and Replanning:

The proposed self-triggered scheduling for range sensor monitoring and replanning approach with speed adaptation was validated using an AscTec Pelican quadrotor UAV. The UAV has an i7 CPU on board for computation and a Hokuyo Lidar range sensor for obstacle detection. The framework presented in Figure 2.16 was followed during the experiments in which our Vicon motion capture system was used to monitor the state of the UAV. The quadrotor used its on-board lidar to estimate the positions of the obstacle. Our self-triggered approach was implemented in Matlab similar to the previous case study.

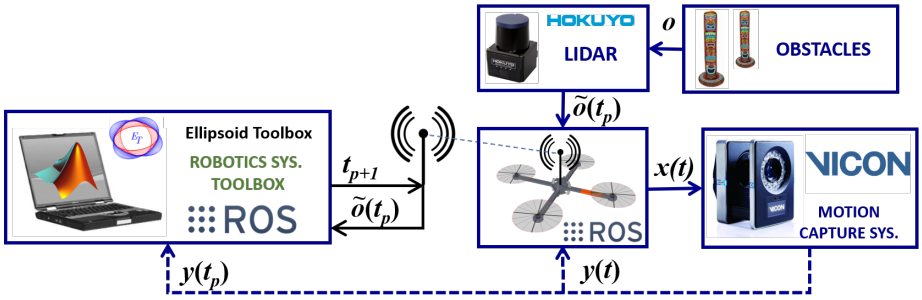
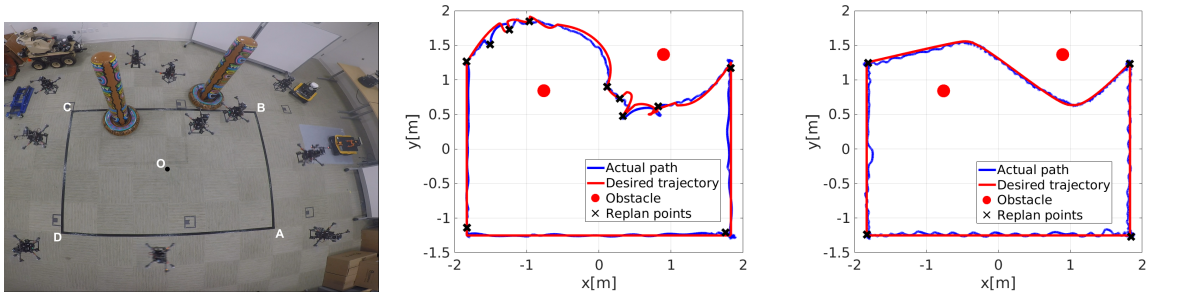


Figure 2.16: Framework of the experiment setup. 2018 ©IEEE

The UAV was tasked to complete a rectangular trajectory by visiting the corners in order: $\{O \to A \to B \to C \to D \to A \to O\}$. Two obstacles (inflated poles) were positioned on the top edge of the rectangle. At run-time, the quadrotor built a trajectory to move to the desired waypoints and to avoid the obstacles and computed its reachable sets. Using the proposed approach, the next sensor monitoring and replanning time was scheduled to the instance in which the reachable sets collide with an obstacle for the first time. Since the range of the lidar sensor is long enough to cover the workspace, the time that the system can leave the field

of view of the lidar is not considered during this experiment. The speed of the UAV was also adapted when it was moving around the obstacles.

Figure 2.17(a) shows an overlapped sequence of snapshots of this experiment and Figure 2.17(b) shows the actual path (blue curve) and the desired trajectory (red curve) of the UAV. The average deviation is recorded as 5.35cm and its speed is adapted in the range of $0.125 - 0.5$ m/s. Sensor checking and replanning occurred only 11 times and the UAV was able to complete its trajectory without colliding with any obstacles. We compared the results of our approach with results of traditional replanning approach with periodic sensor monitoring shown in Figure 2.17(c). The UAV moved with constant speed $v = 0.125$m/s and it monitored for the obstacles periodically with 40Hz frequency. In this case the average deviation decreased to 3.10cm. However, with the proposed approach, the CPU utilization decreased to 2.7% from 9% with periodic 40Hz lidar monitoring.



(a) Overlapped sequence of snapshots. (b) Desired vs. actual trajectories of the UAV during the experiment. (c) Desired vs. actual trajectories of the UAV with periodic obstacle detection.

Figure 2.17: Waypoint navigation experimental results. 2018 ©IEEE

2.6 Self/Event-Triggered Scheduling and Replanning in Dynamic Environments

So far, we have considered scheduling and replanning of UAV operations in static environments, however the UAV may operate in dynamic environments, for example in the presence of other aerial vehicles. For safe operations in dynamic environments, we extend the framework discussed in the previous sections and propose a self/event-triggered scheduling and replanning approach which follows the framework depicted in Figure 2.18 [90].

Based on the initial observations about the obstacle positions, a trajectory to the desired goal position $\boldsymbol{p}_g$ is generated. The future states of the UAV are predicted using the reachable tubes calculated as outlined in

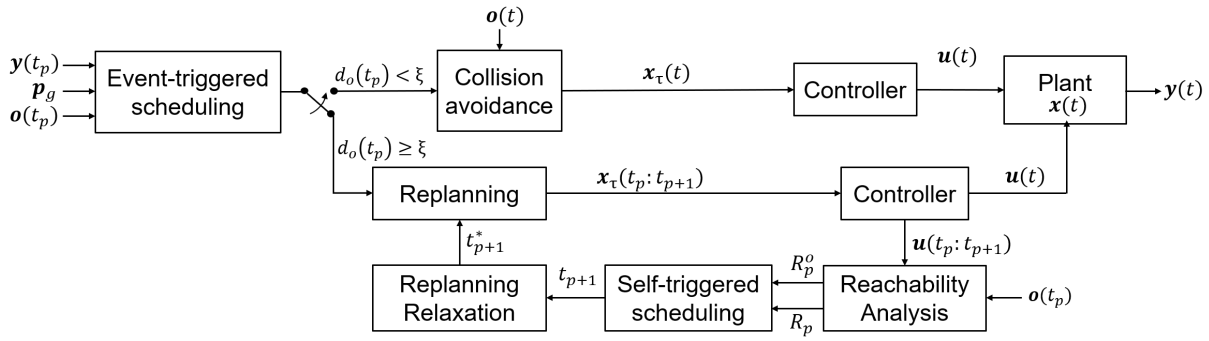


Figure 2.18: Overall self/event-triggered scheduling and replanning framework in dynamic environments.

Section 2.5.1. If the distance $d_o(t_p)$ between the UAV and the closest mobile obstacle is smaller than a user defined threshold ξ, a collision avoidance behavior is triggered in which sensor checking switches to periodic. Collision avoidance is then performed using repulsive potential fields, [5].

2.6.1 Dynamic Obstacle Reachability Analysis

To guarantee safety of the UAV operations in dynamic environments without monitoring the sensors periodically, it is required to predict the future states of the dynamic obstacles. To capture the possible future states of a dynamic obstacle, its reachable sets are calculated based on the available information about its state, maximum velocity and direction of movement. For the sake of simplicity, we assume that the obstacle has a known maximum speed $v_{o,max}$, and a heading which is not known a priori. Therefore, at the initial planning time t_0, the position reachable set of the obstacle, $R^o_p(\boldsymbol{o}_i(t_0), [t_0 + T])$, is a circle centered around the obstacle position as shown in Figure 2.19(b), which contains all the points that can be reached within the planning horizon T. The direction and velocity of the mobile obstacle at time t_p, $\vec{\boldsymbol{v}}_{oi}(t_p)$, can be estimated based on previous observations about its position, as follows:

$$\vec{\boldsymbol{v}}_{oi}(t_p) = \frac{\boldsymbol{o}_i(t_p) - \boldsymbol{o}_i(t_{p-1})}{\|\boldsymbol{o}_i(t_p) - \boldsymbol{o}_i(t_{p-1})\|}, \ \forall i \in [0, n_o] \tag{2.14}$$

Its reachable set can be constructed along its trajectory while taking the estimation errors and noise into account as depicted in Figure 2.19(c). Due to the measurement noise and uncertainties, the actual direction of the mobile obstacle might be different from the estimated one. The minimum and maximum limits of its

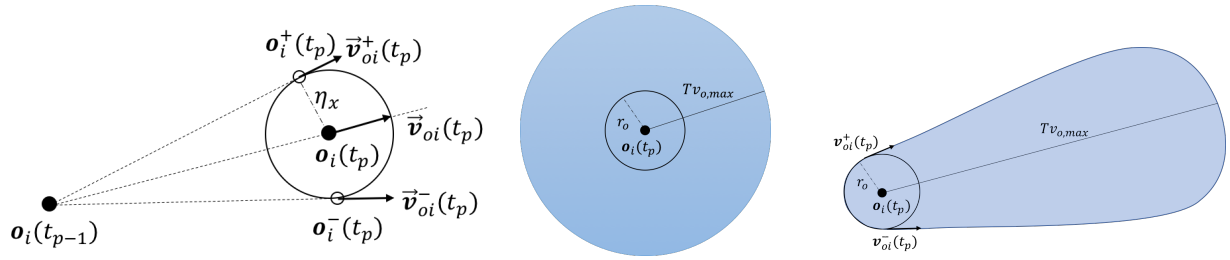
direction of motion can be calculated as follows:

$$\vec{\boldsymbol{v}}_{oi}^{+}(t_p) = \frac{\boldsymbol{o}_i^{+}(t_p) - \boldsymbol{o}_i(t_{p-1})}{\|\boldsymbol{o}_i^{+}(t_p) - \boldsymbol{o}_i(t_{p-1})\|}, \ \forall i \in [0, n_o] \tag{2.15}$$

$$\vec{\boldsymbol{v}}_{oi}^{-}(t_p) = \frac{\boldsymbol{o}_i^{-}(t_p) - \boldsymbol{o}_i(t_{p-1})}{\|\boldsymbol{o}_i^{-}(t_p) - \boldsymbol{o}_i(t_{p-1})\|}, \ \forall i \in [0, n_o] \tag{2.16}$$

where $\vec{\boldsymbol{v}}_{oi}^{+}(t_p)$ and $\vec{\boldsymbol{v}}_{oi}^{-}(t_p)$ are upper and lower limits of the direction of the obstacle where $\boldsymbol{o}_i^{+}(t_p)$ and $\boldsymbol{o}_i^{-}(t_p)$ are two extreme points where the obstacle can be at t_p due to uncertainties. $\boldsymbol{o}_i^{+}(t_p)$ and $\boldsymbol{o}_i^{-}(t_p)$ are shown in Figure 2.19(a), where η_x is the maximum measurement noise.

The updated reach set of the obstacle for the time horizon T is shown in Figure 2.19(c) with a blue shaded region. As can be noticed, the reachable set of the obstacle grows along its direction with a rate proportional to the maximum velocity of the vehicle.



(a) Calculation of the obstacle direction based on the previous position of the obstacle and the magnitude of uncertainties.

(b) Initial reach set of the obstacle.

(c) Reach set of the obstacle updated based on the observed direction.

Figure 2.19: Reachable tube calculation for dynamic obstacles.

2.6.2 Self/Event-triggered Scheduling and Replanning

In a dynamic environment, the first time a collision may occur, $t_{c,o}$, is calculated as follows:

$$t_{c,o} = \min(t_k | R_p^{+}(\boldsymbol{x}(t_p), \boldsymbol{u}(t_k), t_k \in [t_p, t_p + T]) \cap R_p^{o}(\boldsymbol{o}_i(t_p), t_k \in [t_p, t_p + T]) \neq \emptyset)$$

where $R_p^{o}(\boldsymbol{o}_i(t_p), t_k \in [t_p, t_p + T])$ is the geometric sum of the position reachable sets of the obstacle between t_p and $t_p + T$. In Figure 2.20, reachable sets of the UAV (blue circles) and reachable sets of the obstacle (magenta region) are shown at $t_{c,o}$, the time that they collide for the first time.

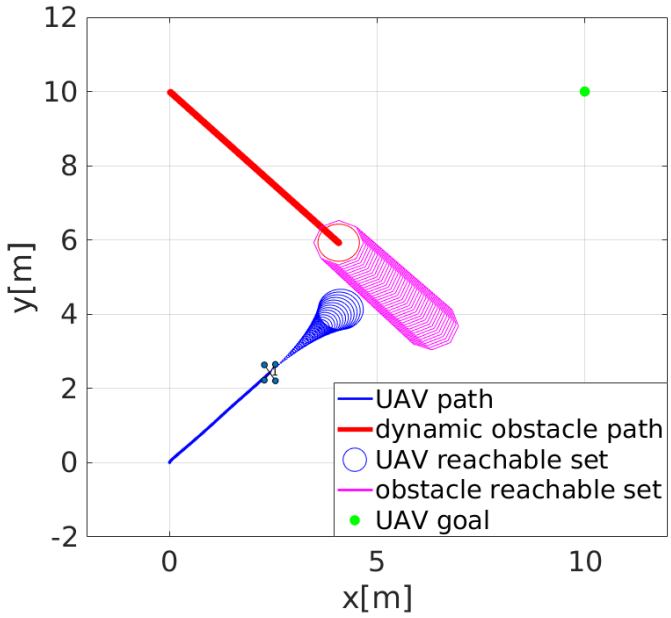


Figure 2.20: Collision between the reachable tube of the UAV and reachable set of the obstacle at time $t_{c,o}$.

Similar to the case with static obstacles, the next sensor checking time is scheduled to t_l (which is calculated in (2.6)) or to $t_{c,o}$ whichever is minimum:

$$t_{p+1} = \begin{cases} \min(t_{c,o}, t_l) - t_r, & \text{if } t_{c,o} < t_p + T \text{ or } t_l < t_p + T \\ t_p + T - t_r, & \text{otherwise} \end{cases} \tag{2.17}$$

where t_r is the amount of time necessary for the replanning calculation. In case that the UAV doesn't detect any obstacle in the environment, next sensor checking time can be computed considering the worst case scenario, that is when the obstacle is right on the boundary of the sensor field of view.

It should be noted that the same safety guarantees presented in the static case apply while minimizing sensor checking operations also in the dynamic environments.

2.6.3 Dynamic Obstacle Repulsive Potential Field Collision Avoidance

If $d_o(t_p) < \xi$ a collision avoidance is initiated following a repulsive potential filed approach. The repulsive potential field around the reachable set of the obstacle $W_O(\boldsymbol{p}(t))$ can be computed as follows:

$$W_O(\boldsymbol{p}(t)) = \begin{cases} \frac{1}{2}\alpha_i \left(\frac{1}{\rho(\boldsymbol{p}(t))} - \frac{1}{\rho_0}\right)^2, & \text{if } \rho(\boldsymbol{p}(t)) \leq \rho_0 \\ 0, & \text{if } \rho(\boldsymbol{p}(t)) > \rho_0 \end{cases} \tag{2.18}$$

where $\rho(\boldsymbol{p}(t))$ is the shortest distance to the obstacle reachable tube from the UAV position $\boldsymbol{p}(t)$, ρ_0 is the distance threshold for the repulsive field and α_i is a positive constant. Then the repulsive force $F_O(\boldsymbol{p}(t))$ is equal to the negative gradient of $W_O(\boldsymbol{p}(t))$:

$$F_O(\boldsymbol{p}(t)) = \alpha_i \left(\frac{1}{\rho(\boldsymbol{p}(t))} - \frac{1}{\rho_0} \right) \frac{1}{\rho(\boldsymbol{p}(t))^2} \nabla \rho(\boldsymbol{p}(t)) \tag{2.19}$$

The attractive potential field $W_G(\boldsymbol{p}(t))$ to go to the goal position is calculated as follows:

$$W_G(\boldsymbol{p}(t)) = \frac{1}{2} \zeta_i (\|\boldsymbol{p}(t) - \boldsymbol{p}_g\|^2) \tag{2.20}$$

where $\boldsymbol{p}_g$ is the position of the goal and ζ_i is a positive constant. The attractive force is the gradient of the attractive field at $\boldsymbol{p}(t)$:

$$F_G(\boldsymbol{p}(t)) = -\zeta_i (\boldsymbol{p}(t) - \boldsymbol{p}_g) \tag{2.21}$$

Finally, the UAV moves with the combination of repulsive and attractive forces:

$$F(\boldsymbol{p}(t)) = F_G(\boldsymbol{p}(t)) + F_O(\boldsymbol{p}(t)) \tag{2.22}$$

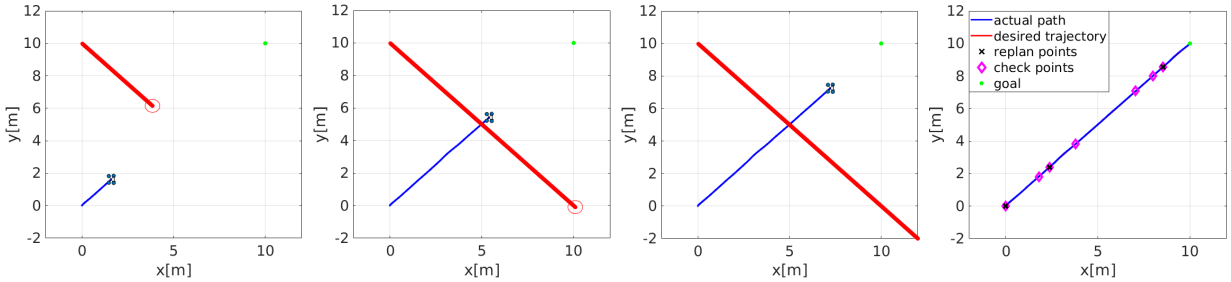
As soon as $d_o(t_p) \geq \xi$ the UAV switches back to the self/event-triggered scheduling policy presented above.

2.6.4 Simulation Results

The case study investigated in this section is a UAV waypoint navigation through a simple environment with one mobile obstacle. We consider a similar UAV described in Section 2.5.5, with sensor range of 20m. The same wind disturbance described in Section 2.5.5 is used during these simulations.

In the simulations shown in Figure 2.21, the UAV is tasked to go to a goal point $\boldsymbol{p}_g = [10, 10]$m shown by a green circle. The red circle represents a dynamic obstacle which and both the vehicle and the obstacle move towards the center of the environment to reach their goal. As the obstacle moves faster than the UAV, their trajectories do not collide with each other. The obstacle and UAV positions before, near and after the intersection point of their paths are displayed in the first three figures in Figure 2.21. As can be seen, since the obstacle passes the intersection point before the UAV, the UAV doesn't perform any avoidance action and it keeps following its originally planned trajectory. The replanning occurs only 3 times at the points shown by black '$\times$' symbols in Figure 2.21(d). At the points shown by magenta 'diamond' symbols, the UAV

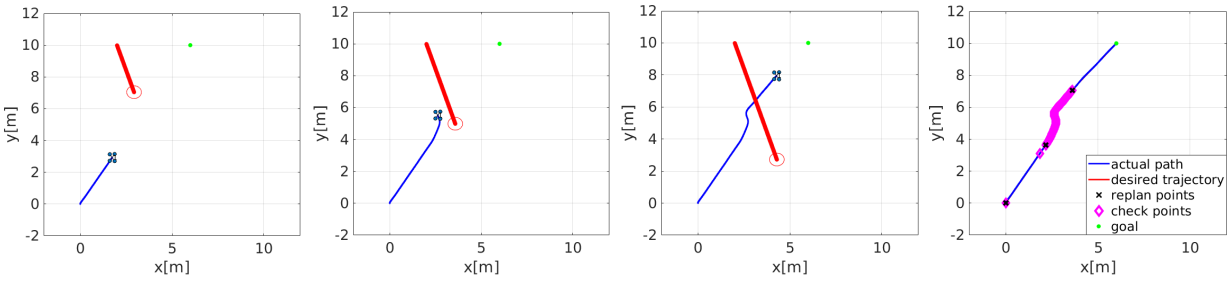
checks its sensors, updates its reachable sets but it does not replan its trajectory. In this simulation, the average and maximum deviation from the desired trajectory is recorded as 6.36cm and 9.53cm respectively.



(a) Before the intersection point. (b) Near the intersection point. (c) After the intersection point. (d) Complete path of the UAV.

Figure 2.21: Simulation Results in which the paths of the UAV and the obstacle intersects but collision doesn't happen because the obstacle passes the intersecting point earlier than the UAV.

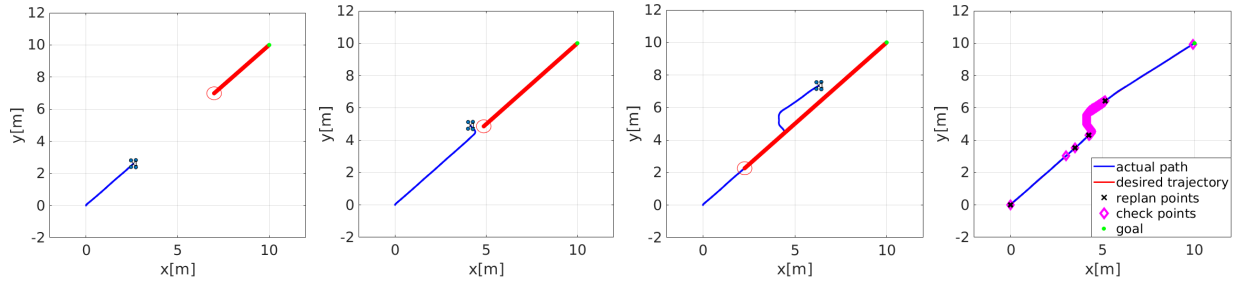
In the case presented in Figure 2.22, the UAV is tasked to the same goal location $\boldsymbol{p}_g = [6, 10]$. A dynamic obstacle starting from a different position move towards the path of the UAV and this case a collision would occur at the intersection point if the trajectory of UAV is not replanned. The first three figures in Figure 2.22 shows the movement of UAV and the obstacle. When the distance between the UAV and the obstacle gets smaller than the threshold, the obstacle avoidance behavior is triggered by our event-triggered replanning approach and the UAV performs an obstacle avoidance maneuver. Once the obstacle is passed and far away, the UAV original self-triggered planning approach moving towards the goal. In this case, the UAV replans its trajectory only 3 times and checks its sensors at the points shown by magenta 'diamond' symbols in Figure 2.22(d).As can be noticed, while avoiding the obstacle, the UAV periodically checks its sensor. In this simulation, the average and maximum deviation from the desired trajectory is recorded as 4.02cm and 9.01cm respectively.



(a) Before the intersection point. (b) Near the intersection point. (c) After the intersection point. (d) Complete path of the UAV.

Figure 2.22: Simulation results of collision avoidance. The obstacle and the UAV could collide with each other without replanning.

In Figure 2.23, the UAV is tasked to go to a goal point $\boldsymbol{p}_g = [10, 10]$m. A dynamic obstacle approaches the UAV from the opposite direction. The first figure in Figure 2.23 shows the paths of the obstacle and the UAV before reaching the intersection point. Similar to the previous case, as they get closer to each other, our event-triggered replanning approach triggers a collision avoidance behavior. After the UAV passes the obstacle, as shown in Figure 2.22(c), it goes back to the original self-triggered behavior towards its goal. In this case, the UAV replans its trajectory only 4 times at the black '×' marks in Figure 2.23(d). In this simulation, the average and maximum deviation from the desired trajectory is recorded as 5.25cm and 9.76cm respectively.



(a) Before the intersection point. (b) Near the intersection point. (c) After the intersection point. (d) Complete path of the UAV.

Figure 2.23: Simulation results of collision avoidance in which the obstacle and the UAV move towards each others.

2.6.5 Experimental Results

The self/event-triggered scheduling and replanning approach in dynamic environments was validated using two AscTec Hummingbird quadtrotor UAVs. The second UAV has a predefined trajectory to follow and the first UAV needs to update its trajectory according to our self-triggered scheduling and replanning approach to avoid the second UAV when necessary. The framework followed in this experiment is shown in Figure 2.24. Similar to the previous sections, the reachable tubes for the first UAV are calculated using Matlab ellipsoidal toolbox [39] and both UAVs are controlled using ROS framework. A Vicon motion capture system was used to track the position and velocity of both UAVs at the scheduled times.

In Figure 2.25(a), a sequence of overlapped snapshots of the two quadrotors is shown. The first UAV starts its motion at $(-2.0, 0.0, 1.0)$m and it is tasked to navigate to its goal position at $(2.0, 0.0, 1.0)$m and the second UAV starts its motion at $(0.0, 1.0, 1.0)$m and it is tasked to go to its goal position at $(0.0, -1.0, 1.0)$m. During the experiments, it is assumed that the direction of the movement of the second UAV is known and it does not change over time. Both UAVs travel with average velocity 0.3m/s in this case. Since the trajectory of the second UAV is shorter, it passes the intersection point before the first UAV. Therefore, the first UAV

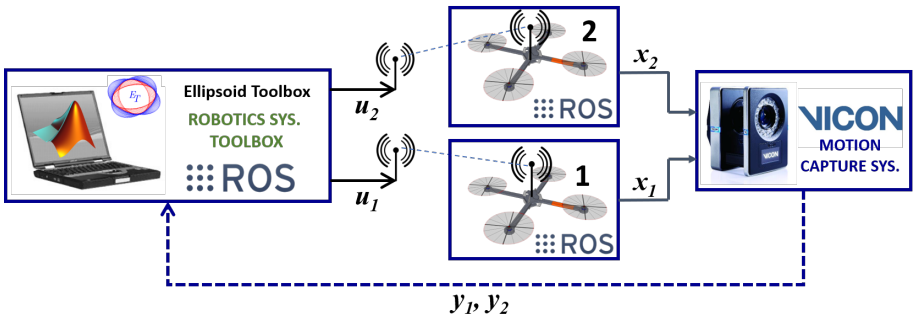
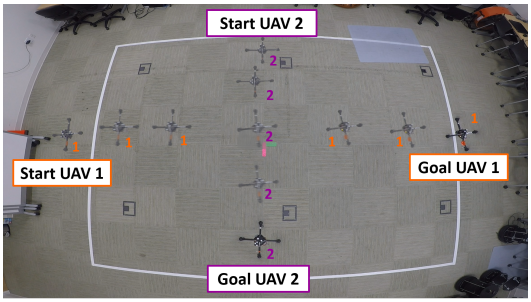
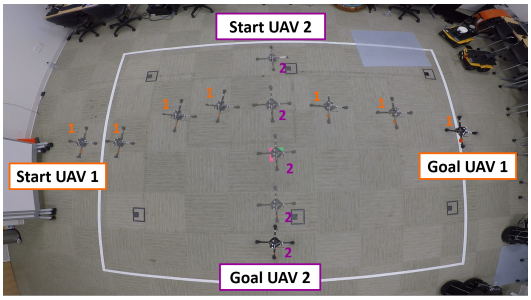


Figure 2.24: Framework of the experiment setup.

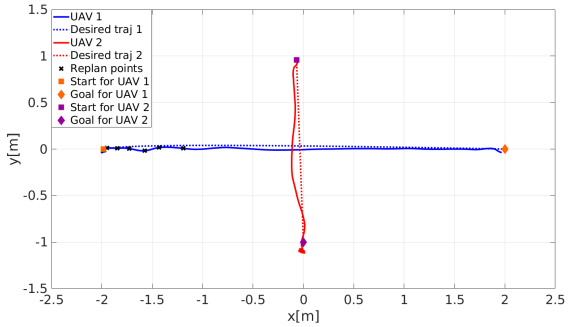
doesn't need to update its trajectory to avoid the second one. In Figure 2.25(d), we compare the desired and actual trajectories of the two UAVs during the mission. The first UAV checks for the position of the second UAV only 7 times at the points shown by black '×' symbols. At these points, since the first UAV is following its trajectory closely, it keeps following the existing trajectory and it recalculates the reachable tubes of both vehicles. The average and maximum deviation from its desired trajectory are recorded as 5.28cm and 12.41cm.



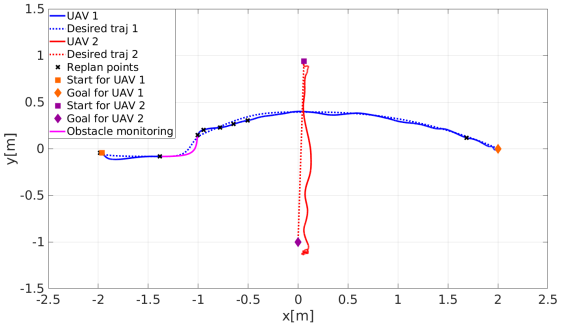
(a) Overlapped sequence of screenshots for non-colliding trajectories.



(b) Overlapped sequence of screenshots for colliding trajectories.



(c) Trajectories of the both UAVs for non-colliding trajectories.



(d) Trajectories of the both UAVs for colliding trajectories.

Figure 2.25: Comparison of experimental results with self/event-triggered approach in dynamic environments.

In Figure 2.25(b), both UAVs have the same initial and goal positions as in the previous case. The first UAV moves with the average velocity 0.3m/s and the second UAV moves with 0.05m/s. In this case, a collision between two UAVs would have occurred but the first UAV adapts its trajectory to avoid the second UAV to prevent collision when the distance between the two vehicles is less than $\rho_0 = 1.5$m and when they are approaching each others. In Figure 2.25(d), we compare the desired and actual trajectories of the two UAVs during the mission. When the reachable sets of the two UAVs collides and when they are close to each other, the first UAV monitors the position of the second UAV at points shown by magenta color, which happens 122 times, while following the adapted trajectory. Similar to the previous case, at the points shown with black $\times$ symbols, the UAV monitors the position of the second UAV only 8 times and it recalculates the reachable tubes of both vehicles. The average and maximum deviation of the first UAV from its desired trajectory are recorded as 2.88cm and 8.75cm in this case. If instead, traditional periodic monitoring was used, the deviation would have been less, however it would have required to monitor the position of the second UAV 2225 times as opposed to 130 times with our approach.

2.7 Discussions

In this chapter, we have presented an adaptive scheduling and replanning framework for UAV operations in cluttered and dynamic environments. The future states of the system under bounded external disturbances and system noises are computed by leveraging reachability analysis. Reachable sets are utilized to schedule next sensor monitoring and replanning times while guaranteeing safety and liveness. We also presented a computationally effective way of updating the reachable tubes based on the monitored system state to further minimize the replanning and sensor checking operations. The speed of the vehicle is also adapted using a curvature based approach to limit the deviation while the system is moving in cluttered environments.

Limitations: Our reachability-based self/event-triggered scheduling and planning approach considers knowledge about the system dynamics and bounded disturbances. We also use a simplified dynamics for reachability analysis. The actuator noise and the effect of disturbance are both modeled as an additive to the control input, and the measurement noise is also modeled as additive noise. If the system's behavior does not satisfy these modeling assumptions, a different model needs to be used for safety guarantees. Additionally, this framework is limited to work for dynamic obstacles with simple dynamics with bounded noise and the potential field controller needs to be well-tuned to avoid the obstacles when the system switches to periodic monitoring.

Future Directions: This framework shows how reachability analysis techniques can be used to schedule sensor monitoring times while guaranteeing safety. The main drawback of reachability analysis is that it gets

computationally expensive for systems with complex nonlinear dynamics and when the planning horizon is long. To deal with this problem, we leverage machine learning techniques and introduced fast reachability techniques as introduced in the following chapters. In the future, we also plan to consider more realistic dynamics for the mobile obstacles and utilize this framework for safe motion planning in multi-robot systems and in highly dynamic environments (e.g., autonomous driving).

Chapter 3

Fast Reachability Analysis for Safe Autonomous Operations with Intermittent Sensing

In this chapter, we present our framework for fast estimation of reachable sets to provide safety for autonomous operations under intermittent sensing uncertainty. This novel fast reachability analysis approach leverages Gaussian process regression theory to predict future states of the system at runtime. This scheme is used in conjunction with our self/event-triggered monitoring and replanning approach to recover the system to guarantee safety constraints when needed. This approach is validated both with simulations and experiments on unmanned aerial vehicles case studies in cluttered environments under the effect of unknown wind disturbance at runtime. The work presented in this chapter is published at 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) [92].

3.1 Introduction

Typically, autonomous vehicle operations require the system to follow a trajectory and reach a goal position. However, an autonomous vehicle may not always be able to obtain its state information consistently due to various reasons, including signal occlusions, limited sensor capabilities, and sensor failures. For instance, an aerial vehicle may lose its GPS signal while flying in between tall buildings or under trees and may not be able to obtain its position information unless it is above a certain altitude. In such cases, since measurements are not available, the vehicle is not capable of adapting its behavior according to its current state, which may lead to unsafe states (e.g., collision with obstacles) due to uncertainties and disturbances. Thus, it is necessary to create proactive systems capable of predicting and assessing future states and replan accordingly

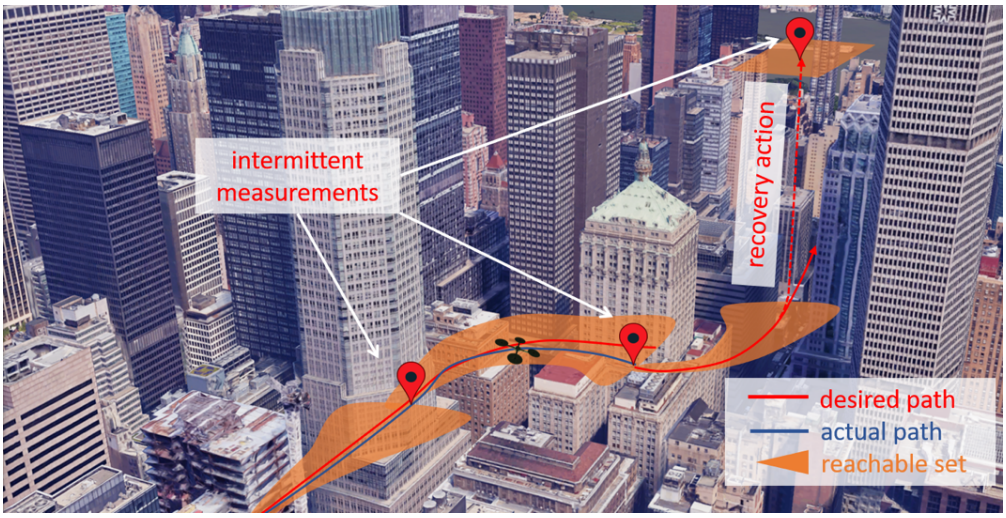


Figure 3.1: Pictorial representation of the envisioned fast monitoring, replanning, and recovery approach in which a UAV computes fast reachable sets during runtime and predict recovery and replanning actions when necessary. 2019 ©IEEE

to avoid possible unsafe conditions in the future. Reachability analysis is a well-known approach to deal with such a problem and estimate future states of a system under uncertainties; however, this process can get computationally complex which makes them challenging to be used for runtime applications.

To estimate the future states of a system fast at runtime, we leverage Gaussian process (GP) theory to obtain a regression that is used online to estimate the maximum deviation for new trajectories. Guarantees for this reachable set estimation approach are presented by showing that the actual deviation is less than its GP-based estimation. By using the proposed method, the system becomes capable of predicting and assessing its future positions and taking actions in a timely manner to guarantee safety (i.e., avoid collision with any obstacle) and liveness (i.e., follow the desired trajectory closely) when measurements are intermittent. The proposed self/event-triggered monitoring, recovery, and replanning framework schedules safe recovery maneuvers to obtain state information and guarantee safety, and replans the trajectory from the observed states whenever deemed necessary. Figure 3.1 shows a pictorial representation of the proposed framework in which a quadrotor builds reachable sets at runtime fast to estimate future states that it could cover in GPS denied environments with sporadic pose observations, also predicting recovery maneuvers whenever needed if the reachable set intersects with any obstacle and no observations are available before entering an unsafe state.

To summarize, we aim to solve the following two challenges:

- how to perform fast reachable set estimation online and assure that the system will be inside these sets;

- how to guarantee safety and liveness properties when observations are intermittent in the presence of noise and unknown disturbance effects in the environment.

Assumptions: Throughout this chapter, we consider a navigation case study with a UAV aiming to reach a goal position in a cluttered environment with static obstacles. The system is subject to intermittent sensing problems caused by environmental conditions (e.g., because of trees or tall buildings), and it can obtain its position information at random times. We also assume that the system can perform recovery actions (e.g., changing altitude to go above a certain height) to obtain sensory measurements when necessary.

3.2 Problem Formulation

In this chapter, we are interested in finding a technique for fast reachability to monitor the state of a system and recover and replan accordingly to guarantee both safety and liveness properties. These problems can be formally defined as follows:

Problem 1: *Fast Reachability*: A UAV has the objective to follow an obstacle-free trajectory in a cluttered environment with intermittent state measurement under disturbances. Given the UAV dynamics $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{w}(t))$ as a function of its state $\boldsymbol{x}$, input $\boldsymbol{u}$, and disturbance $\boldsymbol{w}$, find a policy to quickly estimate the reachable sets $\boldsymbol{R}(\boldsymbol{x}_\tau, t)$ of the system at time t while tracking a desired trajectory $\boldsymbol{x}_\tau$ under the effect of unknown disturbance, measurement and input noises during runtime. Both the disturbance and noise values are assumed to be bounded in magnitude but unknown at runtime.

Problem 2: *Self/Event-triggered Monitoring, Recovery and Replanning*: Once reachable sets are obtained by solving Problem 1, find an online policy to schedule the time in which the system needs to switch to a safe recovery mode to observe its state, and to replan its trajectory in order to satisfy the following safety and liveness conditions:

- *Safety Constraint:* The UAV should avoid collisions with obstacles:

$$\|\boldsymbol{p}(t) - \boldsymbol{p}_{oi}\| > r_{oi}, \forall t \in [t_p, t_p + T], \forall i \in [0, N_o] \tag{3.1}$$

in which $\boldsymbol{p}(t) = \begin{bmatrix} x, y \end{bmatrix}^\mathsf{T}$ is the position of the vehicle, $\boldsymbol{p}_{oi} = \begin{bmatrix} x_{oi}, y_{oi} \end{bmatrix}^\mathsf{T}$ is the position of the i^{th} obstacle in the $x-y$ plane and r_{oi} is its radius, N_o is the number of obstacles in the environment, t_p is the planning time of the operation, and T is the duration of the trajectory. Obstacles are assumed to have circular shapes.

- *Liveness Constraint:* The UAV should stay within a certain proximity of the planned trajectory:

$$\|\boldsymbol{p}(t) - \boldsymbol{p}_\tau(t)\| \leq \lambda_d, \forall t \in [t_p, t_p + T] \tag{3.2}$$

where $\boldsymbol{p}(t)$ and $\boldsymbol{p}_\tau(t)$ are the actual and desired positions of the vehicle along the trajectory at time t respectively, and λ_d is the allowed deviation threshold.

3.3 Fast Runtime Monitoring, Recovery and Replanning

In this section, we describe our framework for fast reachability analysis and online monitoring, recovery, and replanning. The proposed architecture consists of offline and online stages, as displayed in Figure 3.2.

We estimate reachable sets online by using a Gaussian Process (GP)-based regression technique. Training for GP regression model is performed on a library of trajectory primitives with different durations. These primitives are run on the UAV offline under various disturbances, and their corresponding maximum deviation values are also saved in the library. At runtime, an obstacle-free trajectory is planned from a given initial state to a goal state. The trained GP regression model is then used to estimate the maximum deviation of the vehicle for this new trajectory.

Because the state measurement is assumed to be intermittently available as discussed earlier, the UAV may not be capable of observing its position information constantly. Instead, when measurements are missing, the system will use its model to calculate its position considering ideal conditions (i.e., no disturbance and noise).

The task here is to estimate the maximum deviation over time (i.e., the position reachable tube) due to noise and disturbances during the interval of times in which the system is running without receiving updated measurements. Based on the estimated position reachable tube, a self-triggered monitor is deployed to compute the first time that a system may violate the safety constraint under the worst-case assumption that it will never be able to obtain its position information. At that time, the UAV will need to recover its position information by switching into a predefined safe maneuver, which is defined as moving to an altitude above a certain level in this work. In general, recovery may not always be feasible, and it is subject to disturbance and uncertainties as well. Therefore, during planning, the time required to perform the recovery operation needs to be considered too. Once a recovery operation is completed, the trajectory is replanned using the obtained state information. If the position information becomes available before the predicted unsafe time (i.e., GPS information might be unavailable or available depending on the position of trees, buildings, etc.), an event-triggered procedure is deployed to assess the current state of the vehicle. If the vehicle is within the

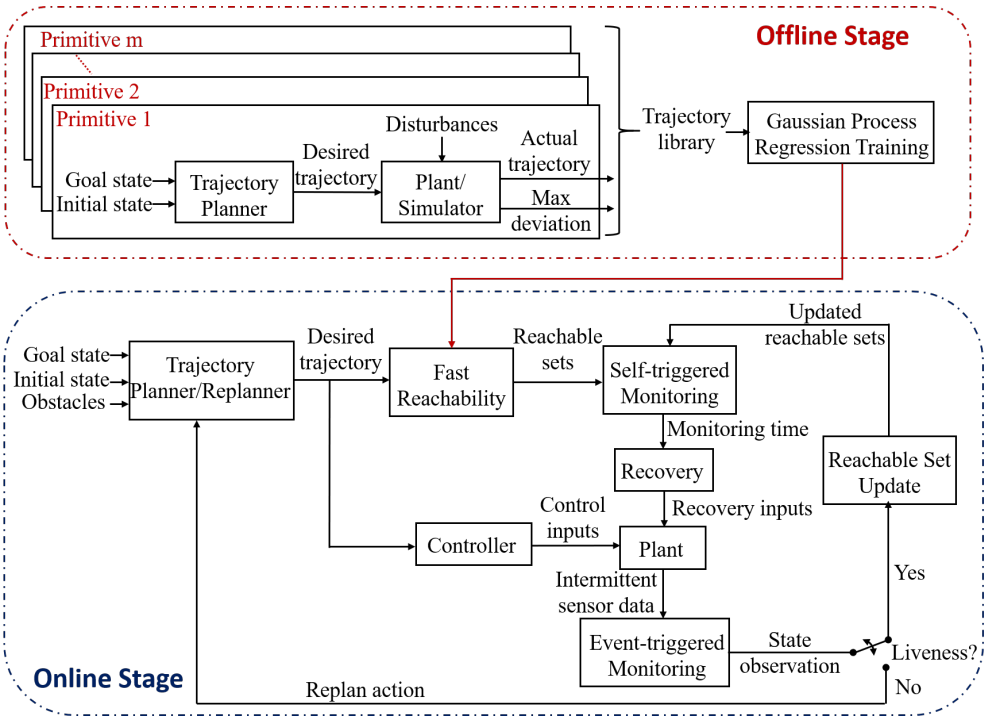


Figure 3.2: Architecture of the fast runtime monitoring, recovery and replanning approach. 2019 ©IEEE

close proximity of the desired trajectory, the reachable sets are shrunk depending on the deviation. If the vehicle deviates too much from the desired trajectory, the trajectory is replanned for liveness.

The first step in our approach consists in using Gaussian processes theory to estimate reachable sets, which will be explained in detail in the following section.

3.3.1 Gaussian Process-based Fast Reachability

Given the premises in the previous section, the position reachable set at time t when the UAV is running without state measurement is defined as:

$$\boldsymbol{R}(\boldsymbol{p}_\tau, t) = \{\boldsymbol{p}(t) : \|\boldsymbol{p}(t) - \boldsymbol{p}_\tau(t)\| \leq d_m(t)\} \tag{3.3}$$

where $d_m(t)$ is the maximum deviation at time t from the desired trajectory position $\boldsymbol{p}_\tau$.

In order to estimate the position reachable sets fast and avoid computationally expensive traditional reachability analysis tools [39], we leverage GP regression to estimate $d_m(t)$ based on a library of previously collected trajectory primitives data.

Training Data Collection

GP regression training is obtained by calculating and running primitive trajectories offline and by generating a library of trajectories of different duration. The trajectory library should be rich enough in terms of the duration, length, initial, and final velocities to model the maximum deviation behavior. For this reason, we generated trajectories starting from the origin with various initial velocities to different goal locations using minimum jerk trajectory generation [55]. These trajectories were executed using a PD controller without position measurements under a rich wind disturbance set during training. The wind disturbances inside the disturbance set $\mathcal{W}$ are in four main directions with constant or sinusoidal magnitudes with different frequencies, and they are assumed not to change direction during the operation. During the online stage, the disturbance has unknown direction and magnitude; however, its magnitude is assumed to be bounded by the maximum magnitude value in $\mathcal{W}$. The controller is assumed to remain the same during the offline and online stages.

A set of sample trajectories from the UAV trajectory primitive library can be seen in Figure 3.3. The red curves correspond to the desired trajectory, and the blue curves correspond to the actual trajectories. In total, we collected 1250 different trajectory primitives, which are assumed to be representative enough to model the maximum deviation behavior at runtime as they capture a wide range of durations, initial and final velocities and lengths.

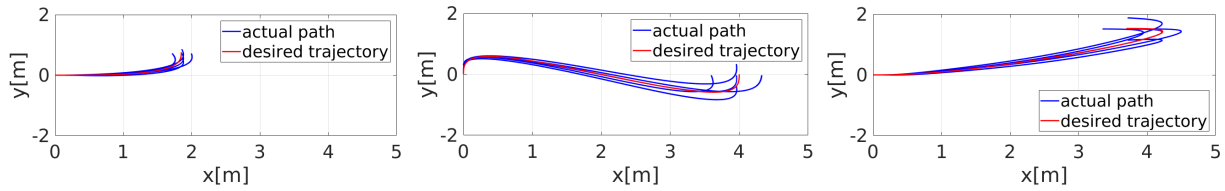
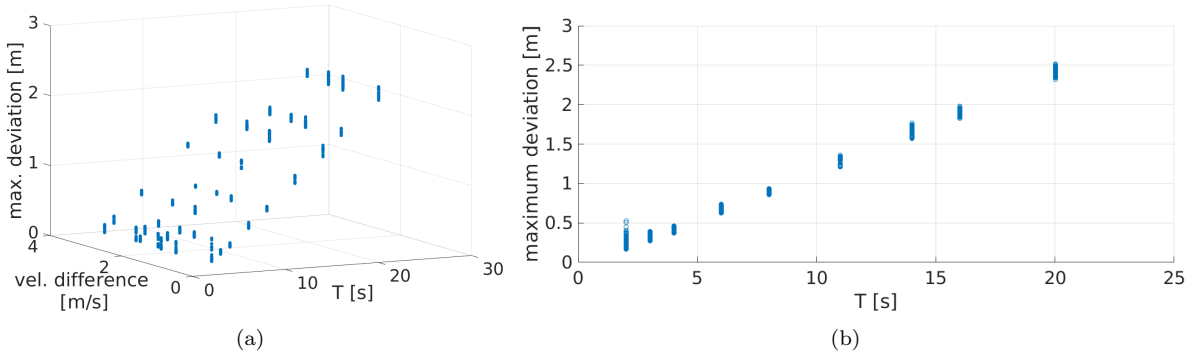


Figure 3.3: Example trajectories from the trajectory primitive library. 2019 ©IEEE

To understand which factors affect the maximum deviation from the desired trajectory, we have analyzed the relationship between the maximum deviation and various trajectory specifications such as the difference between initial and final states, average velocity, trajectory length, and duration. For example, consider the plot in Figure 3.4(a). As can be noticed, there is no correlation between the magnitude of the maximum deviation from the desired trajectory and the difference between initial and final velocities. On the contrary, the time duration of the trajectories has a definite impact on the maximum deviation (Figure 3.4(b)), and therefore, it is chosen as the regression variable. The results of this analysis were expected since deviation increases when a system is exposed to a disturbance for a longer time, especially in open loop.



(a) (b)

Figure 3.4: (a) Maximum deviation as a function of time (i.e.,duration of the trajectory) and difference between initial and final velocities. (b) Maximum deviation as a function of time. 2019 ©IEEE

For a given trajectory of duration T, the maximum deviation from the desired trajectory is calculated as follows:

$$d_M(T) = \max_{\boldsymbol{w}\in\mathcal{W}} \max_{t\in[0,T]} \|\boldsymbol{p}_{\boldsymbol{w}}(t) - \boldsymbol{p}_\tau(t)\| \tag{3.4}$$

where $\boldsymbol{p}_{\boldsymbol{w}}(t)$ is the position of the vehicle at time t which is following the trajectory $\boldsymbol{p}_\tau(t)$ under the disturbance $\boldsymbol{w} \in \mathcal{W}$.

For each primitive trajectory, the corresponding duration and maximum deviation values are stored in a trajectory primitive library. For a library consisting on m different trajectories, the trajectory durations are saved in a vector $\boldsymbol{t} = [T_1, T_2, \cdots, T_m]$ and the corresponding maximum deviation values are saved in a vector $\mathbf{d}_M = [d_M(T_1), d_M(T_2), \cdots, d_M(T_m)]$. Given this trajectory primitive library, Gaussian process regression is utilized in the following subsection to estimate the maximum deviation of a new trajectory.

Gaussian Process Regression

Gaussian process (GP) regression is a nonparametric regression technique which is used in this work to find a mapping between trajectory duration T and maximum deviation $d_M(T)$. Given a trajectory library containing the set of collected observations $\mathcal{D} = \{\mathbf{t}, \mathbf{d}_M\}$, our goal is to predict the maximum deviation for a new input $\boldsymbol{t}^*$ by drawing $\boldsymbol{d}_M^*$ from the posterior distribution $p(\boldsymbol{d}_M^*|\mathcal{D})$. By definition of GP [69], previous observations $\boldsymbol{d}_M$ and function values $\boldsymbol{d}_M^*$ follow a joint (multivariate) normal distribution:

$$\begin{bmatrix} \boldsymbol{d}_M \\ \boldsymbol{d}_M^* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{t}) \\ \mu(\boldsymbol{t}_*) \end{bmatrix}, \begin{bmatrix} \boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I} & \boldsymbol{K}_* \\ \boldsymbol{K}_*^T & \boldsymbol{K}_{**} \end{bmatrix} \right) \tag{3.5}$$

where $\boldsymbol{K} \in \mathbb{R}^{m\times m}$ has entries $\boldsymbol{K}_{(i,j)} = k(t_i, t_j)$ for $i, j \in \{1, \cdots, m\}$, $\boldsymbol{K}_* = \begin{bmatrix} k(t_1, \boldsymbol{t}^*) & \cdots & k(t_m, \boldsymbol{t}^*) \end{bmatrix} \in \mathbb{R}^{m\times m_*}$ and $\boldsymbol{K}_{**} = k(\mathbf{t}_*, \mathbf{t}_*) \in \mathbb{R}^{m_*\times m_*}$. σ_ϵ^2 is the noise level associated with the observations, m is the size

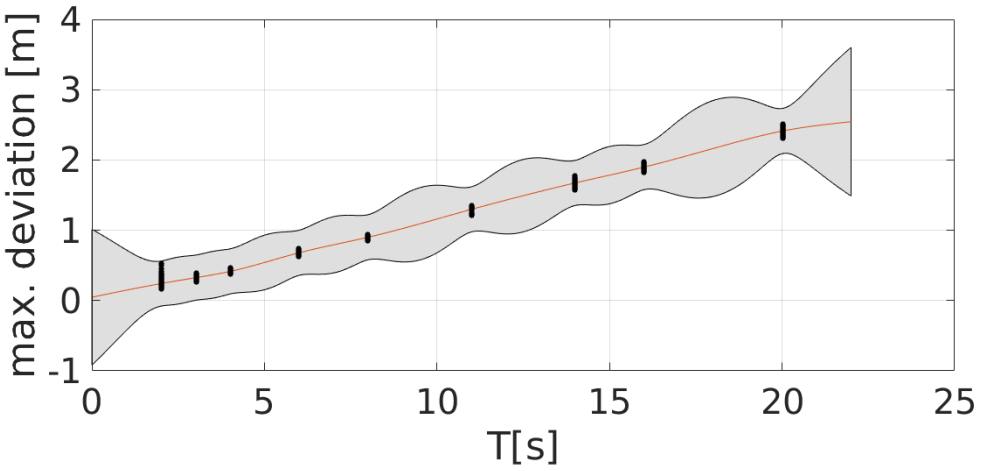


Figure 3.5: GP regression of maximum deviation as a function of the trajectory duration implemented on Matlab using GPML toolbox [70]. 2019 ©IEEE

of the observation set, m_* is the size of the test set, μ is the mean function and k is the covariance function. We chose to use the widely known Matern kernel [69] as a covariance function.

The estimation of $\boldsymbol{d}_M^*$ conditioned on the observations $\mathcal{D}$ is calculated using the properties of joint Gaussian distributions. Namely, the posterior probability is also a Gaussian distribution:

$$p(\boldsymbol{d}_M^*|\boldsymbol{t}_*, \boldsymbol{t}, \boldsymbol{d}_M) \sim N(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}_*) \tag{3.6}$$

with the following mean and covariance:

$$\boldsymbol{\mu}^* = \mu(\boldsymbol{t}_*) + \boldsymbol{K}_*^T(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1}(\boldsymbol{d}_M - \mu(\boldsymbol{t})) \tag{3.7}$$

$$\boldsymbol{\Sigma}_* = \boldsymbol{K}_{**} - \boldsymbol{K}_*^T(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1}\boldsymbol{K}_* \tag{3.8}$$

In Figure 3.5, the GP regression model learned using our trajectory primitive library is shown. Learning is performed using 1250 data points shown by black dots in Figure 3.5, and regression is performed on 1000 time duration data points with values between 0 and 22 seconds. The red curve is the mean for the data points and the shaded gray region is the 95% confidence interval. In order to verify this regression, we performed a test on 110 untrained trajectories and we observed that all of them had smaller maximum deviation values than the upper bound of the confidence interval.

At runtime, given a new trajectory of duration T^*, the maximum deviation estimation is finally calculated as:

$$\tilde{d}_M(T^*) = \mu^* + 2\sigma_* \tag{3.9}$$

which corresponds to the upper bound of the 95% confidence interval where μ^* and σ_*^2 are calculated according to (3.7) and (3.8) respectively for $\boldsymbol{t}_* = T^*$.

From the collected training data, we observed that the maximum deviation from the trajectory grows linearly over time, as can be noted in Figure 3.6. We have also observed that the initial overshoot in the maximum deviation is due to the aggressiveness of some trajectories. For ease of discussion in this work, we are neglecting this behavior, but it can be easily included in our model by setting an initial deviation offset. Based on this linear dependency, the maximum deviation at time t is calculated as follows:

$$d_m(t) = \tilde{d}_M(T^*)\frac{t}{T^*} \tag{3.10}$$

The reachable set $\boldsymbol{R}(\boldsymbol{p}_\tau, t)$ for a trajectory of duration T^* is finally obtained according to (3.3) with $d_m(t)$ in (3.10).

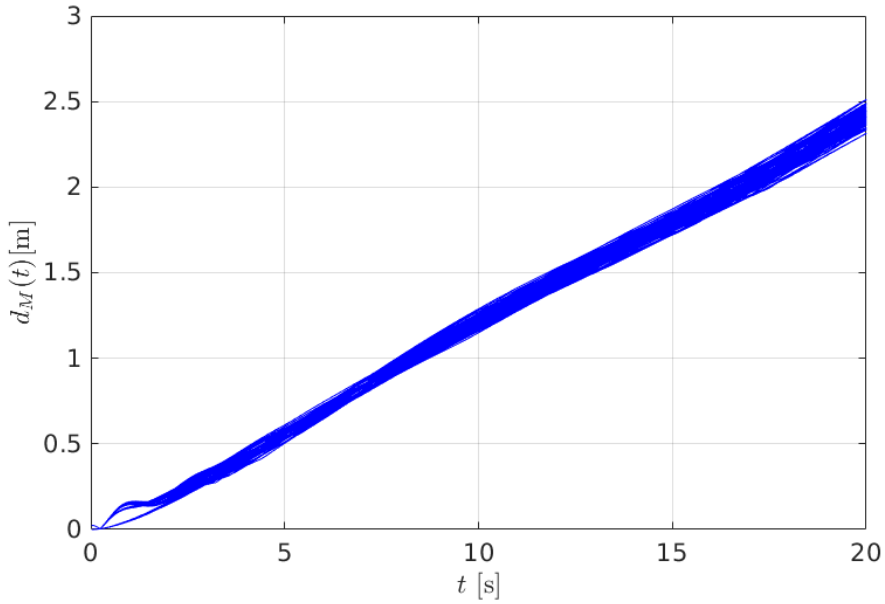


Figure 3.6: Maximum deviation values over time for trajectories with 20 seconds duration under the effect of a range of disturbances. 2019 ©IEEE

GP regression can be used to calculate reachable sets because the maximum deviation from the desired trajectory is bounded by its estimated value $\tilde{d}_M(T^*)$ if the training set includes the maximum possible deviations as will be shown in Lemma 3 and 4:

Lemma 3 *For two trajectories in the training set of consecutive durations T_1 and T_2 respectively, the difference between their maximum possible deviation values $|d_M(T_1) - d_M(T_2)|$ is bounded if the difference between their time duration is also bounded: $|T_1 - T_2| \leq \epsilon$.*

Proof: The system state without the effect of disturbance $\hat{\boldsymbol{x}}(t)$ and under disturbance $\boldsymbol{x}(t)$ evolves over time as follows:

$$\dot{\hat{\boldsymbol{x}}}(t) = g(\hat{\boldsymbol{x}}(t), \boldsymbol{u}(t)) \tag{3.11}$$

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{w}(t)) = g(\boldsymbol{x}(t), \boldsymbol{u}(t)) + \boldsymbol{G}\boldsymbol{w}(t) \tag{3.12}$$

where $\boldsymbol{u}(t)$ is the controller input, $\boldsymbol{w}(t)$ is the bounded external disturbance: $\|\boldsymbol{w}(t)\| \leq W$ and $\boldsymbol{G}$ is the disturbance matrix [6]. The initial conditions for both systems are the same: $\boldsymbol{x}(0) = \hat{\boldsymbol{x}}(0) = \boldsymbol{x}_0$. From (3.12) the disturbance effect is additive. When there is no state measurement, the system generates the control inputs with the assumption of ideal conditions (i.e., no disturbance and noise), hence, the same input $\boldsymbol{u}(t)$ is applied to both systems in (3.11) and (3.12).

The difference between the systems states with and without the effect of disturbance at the end of a trajectory is calculated based on the difference between the dynamics of the two systems in (3.11) and (3.12):

$$\hat{\boldsymbol{x}}(T_1) - \boldsymbol{x}(T_1) = \int_0^{T_1} \boldsymbol{G}\boldsymbol{w}(\tau) d\tau \tag{3.13}$$

Since the disturbance is bounded, the norm of the difference between the actual and the nominal state is also bounded at the end of the trajectory.

$$\|\hat{\boldsymbol{x}}(T_1) - \boldsymbol{x}(T_1)\| \leq T_1 W_G \tag{3.14}$$

where W_G is the upper bound of the effect of the disturbance: $\|\boldsymbol{G}\boldsymbol{w}(t)\| \leq W_G, \forall t \in [0, T]$.

Thanks to the stability of the system [55], the deviation between the desired state (desired trajectory) and the system state without the effect of disturbance is bounded at the end of the trajectory when there is no disturbance in the environment:

$$\|\hat{\boldsymbol{x}}(T_1) - \boldsymbol{x}_\tau(T_1)\| \leq \xi \tag{3.15}$$

Using (3.15) and (3.14), it can be shown that the distance between the actual state and the desired state is bounded at the end of the trajectory:

$$\|\boldsymbol{x}(T_1) - \boldsymbol{x}_\tau(T_1)\| = d(T_1) \leq d_M(T_1) = \xi + T_1 W_G \tag{3.16}$$

For a trajectory of duration $T_2 = T_1 + \epsilon$ where ϵ is a positive real number, the deviation is bounded as follows:

$$\|\boldsymbol{x}(T_2) - \boldsymbol{x}_\tau(T_2)\| = d(T_2) \leq d_M(T_2) = \xi + (T_1 + \epsilon)W_G \tag{3.17}$$

Therefore, if the difference in the durations of two different trajectories is bounded, the difference between their maximum possible deviation values is also bounded:

$$d_M(T_2) - d_M(T_1) = \epsilon W_G \tag{3.18}$$

■

Lemma 4 *Given a trajectory, the GP regression estimation for the maximum deviation value $\tilde{d}_M$ calculated as in* (3.9) *is an upper bound to the actual maximum deviation d_M from that trajectory.*

Proof: Let's consider the data points in the training set with maximum deviations $d_M(T_1) = \xi + T_1 W_G$ and $d_M(T_2) = \xi + T_2 W_G$. Based on the system dynamics and (3.16), the deviation at time $T^* = T_1 + \varepsilon < T_2$ is bounded: $d(T^*) \leq d_M(T^*) = \xi + (T_1 + \varepsilon)W_G$ where $\varepsilon > 0$.

The slope of the virtual line which connects these two data points is: $\frac{d_M(T_2) - d_M(T_1)}{T_2 - T_1}$ and the deviation value on this line for T^* can be calculated as follows:

$$\begin{aligned}\bar{d}_M(T^*) &= d_M(T_1) + \frac{T^* - T_1}{T_2 - T_1}(d_M(T_2) - d_M(T_1)) \\ &= d_M(T_1) + \frac{\varepsilon}{T_2 - T_1}(T_2 - T_1)W_G = d_M(T_1) + \varepsilon W_G\end{aligned}$$

For a GP regression with the upper bound of the confidence interval above this virtual line, the maximum deviation estimation is always larger than the actual maximum deviation:

$$\tilde{d}_M(T^*) \geq \bar{d}_M(T^*) = d_M(T_1) + \varepsilon W_G = d_M(T^*) \tag{3.19}$$

■

As shown in the previous lemma, if the training set contains the maximum possible deviations for the given trajectory durations, GP-based maximum deviation estimation can provide upper bounds to the actual deviation for any type of stable system. With rich enough training sets, these upper bounds can be tight whereas lack of enough training data would cause over-conservative estimations. In any case, the estimated reachable sets contain all the states that the system can actually reach under bounded disturbance. Therefore, these estimated reachable sets can be used to guarantee safety as will be explained in the following section.

3.3.2 Self/Event-triggered Monitoring, Recovery, and Replanning

With the ability to estimate reachable sets at runtime, we design a policy for online monitoring to assess how long the system could continue its motion without a position measurement and for scheduling the recovery which allows the vehicle to obtain its position information. The system needs to monitor its position before a collision may occur. The earliest time that a collision may occur, here referred to as *monitoring time*, is computed as:

$$t_{s+1} = \min(t_k | \boldsymbol{R}(\boldsymbol{p}_\tau, t_k \in [t_p, t_p + T]) \cap \boldsymbol{O} \neq \emptyset) - t_r \tag{3.20}$$

where t_r is the amount of time necessary for a safe recovery maneuver. At t_{s+1}, the system switches to a recovery operation which allows it to observe its position information (e.g., by flying above a building). After recovering the position information, the system replans its trajectory accordingly as represented in Figure 3.2.

After the start of the operation until the recovery maneuver, the position information may become available at random times $t_m \in [t_p, t_{s+1}]$ which are unknown a priori. According to the obtained position information, two procedures (explained next) are implemented on the vehicle: either 1) reachable set shrinking if the deviation from the desired trajectory is less than a predefined liveness threshold λ_d, or 2) trajectory replanning if the deviation is above λ_d.

As these reachable sets are estimated based on the worst-case scenario assumptions in terms of disturbance and noise, when the disturbance in the environment is not as strong as its maximum value, the reachable set estimation might become over-conservative. To have more accurate estimations about where the system could reach and farther reduce computation, we leverage the deviation from the desired trajectory at the time in which the position information becomes available and shrink the reachable sets as explained in Section 2.5.3.

Event-triggered Replanning

When the system state information is obtained, the UAV may decide to replan its trajectory if the observed state is too far from the desired trajectory for the sake of liveness as defined in (3.2). Specifically, a new trajectory is planned to the goal from the observed position if the observed deviation is larger than the liveness threshold: $d(t_m) > \lambda_d$.

After replanning the trajectory: 1) the reachable sets are regenerated using the GP-based approach in Section 3.3.1 and 2) self-triggered monitoring introduced in Section 3.3.2 is performed until the operation is completed.

When the self-triggered monitor decides that the system needs to check its state by performing a recovery action, event-triggered replanning is again invoked to replan a trajectory for recovery. From the recovered

state, an obstacle-free trajectory to the goal position is replanned again and self-triggered monitoring, recovery and replanning continue to be performed until reaching the desired goal position.

3.4 Simulations

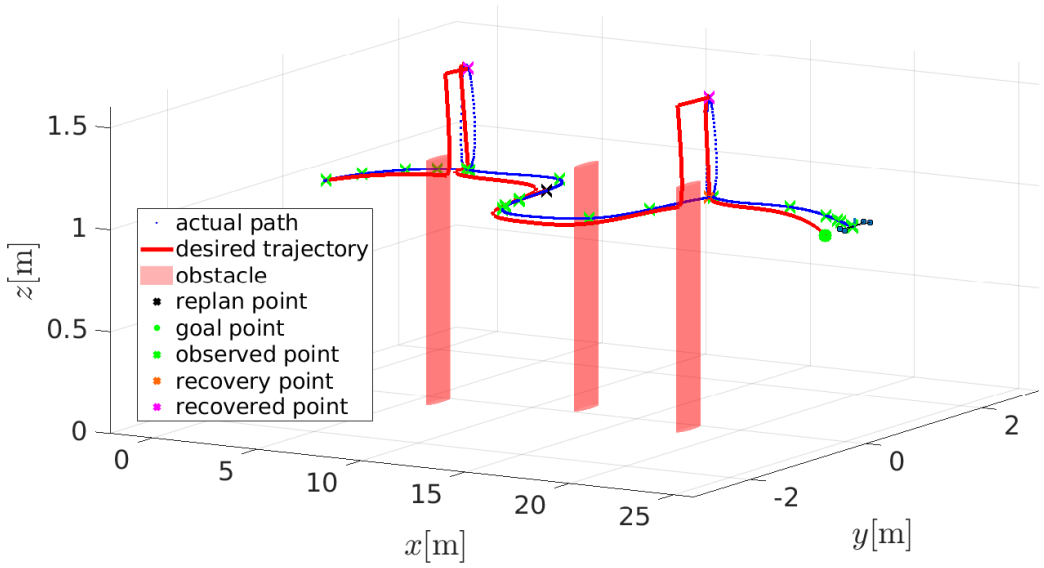
We validate the runtime monitoring, recovery and replanning approach with quadrotor UAV simulations for an autonomous navigation case study in a cluttered environment. In this environment, the GPS signal is available intermittently at the altitude that the system is required to fly at, and it is always available at higher altitudes. During the simulations, we use a quadrotor UAV modeled with a 12^{th} order system state and with linearized dynamics. The details of this quadrotor UAV model can be found in [6].

As a representative navigation case study, we present an environment with three cylindrical shaped obstacles as shown in Figure 3.7(a). The UAV is tasked to reach a goal position $\boldsymbol{p}_g = [24, 0, 1]$ with zero velocity. The obstacles are located at $\boldsymbol{p}_{o1} = [6, -0.2], \boldsymbol{p}_{o2} = [12, 0.2], \boldsymbol{p}_{o3} = [18, -0.2]$, all of them are 1.2m tall with the radius of 0.2m. The velocity of the wind disturbance is varying over time in the y direction: $\boldsymbol{w} = [0, 0.04 + 0.01\sin(t), 0]$m/s.

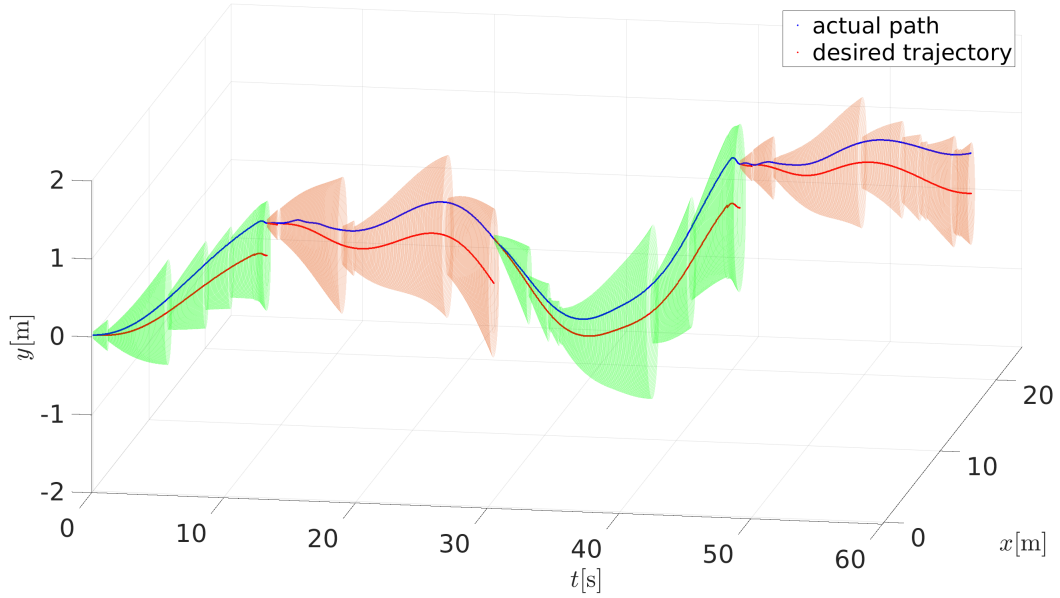
In the beginning of its operation, the UAV creates an obstacle free trajectory to the goal position and computes the reachable sets based on the time to complete this trajectory using the proposed GP approach in Section 3.3.1. The reachable sets of the UAV are shown by the green and orange tubes in Figure 3.7(b). Whenever a new reachable tube is generated (either for recovery, or because the deviation at monitoring time is too large) the color of the reachable tube in Figure 3.7(b) swaps. At random points which are shown by green $\times$ symbols in Figure 3.7(a), the sensor measurement for position becomes available and the reachable tubes are updated based on the current observed position. At these points, the reachable tubes shrink, but replanning doesn't occur (the color of the reachable tube in the figure remains the same). If the observed position is deviated from the desired trajectory more than the user defined threshold $\lambda_d = 0.5$m, the UAV replans its trajectory and recomputes the corresponding reachable sets. This replanning point is shown by a black $\times$ symbol in Figure 3.7(a). At the point shown by an orange $\times$ symbol, the reachable set collides with an obstacle and the UAV triggers a recovery action to observe its state information at $z = 1.5$m level. At the point shown by a magenta $\times$ symbol, it obtains its state information and replans a trajectory accordingly.

Through the course of the entire operation, the state information of the UAV becomes available only 25 times, and the vehicle performs recovery maneuver twice to obtain its state information. A trajectory replanning due to deviation is performed once at the black $\times$ point. Even though the UAV is not able to observe its state continuously, it is able to perform its planned operation safely (i.e. no collision with an obstacle occurred) with a maximum deviation of 0.60m from its desired trajectory. Similarly, in all the other

simulations, the UAV was able to finish its operation safely and the mean and standard deviation of the maximum deviation were recorded as 0.67m and 0.14m respectively.



(a) Desired vs. actual trajectory of the UAV.



(b) Estimated reachable sets of the vehicle during the entire operation.

Figure 3.7: UAV simulation results in an environment with three obstacles. 2019 ©IEEE

We have used an Intel Core i7-6700HQ CPU at 2.60GHz to run these simulations and it took 0.248s on average to estimate the maximum deviation reachable set for a given trajectory, which is independent from the length of the trajectory. The simulations in order to generate the reachable sets offline for training takes 3.50s for 10 second long trajectory on an average, and this time increases/decreases linearly with the

trajectory duration. Similarly, the reachable tube calculation with Ellipsoidal Toolbox increases linearly with the number of time steps used [39], and it took 2.618 seconds on an average to generate the reachable sets for a 10 second long trajectory at 40Hz. These results demonstrate that this approach is able to reduce the time for reachability computation significantly toward an online implementation.

3.5 Experiments

The GP-based fast reachability and replanning approach was validated experimentally using an AscTec Hummingbird quadrotor UAV where the control commands are communicated using ROS. The ground truth state information is obtained using a Vicon motion capture system.

The training data for GP regression was collected by running a set of trajectories with different durations. During training, the quadrotor did not observe its position and the control inputs were generated as if it was following the trajectory perfectly (i.e., closing the loop with the desired states along the trajectory). Wind disturbance was created using a 24" industrial heavy duty drum fan placed on top of a mobile ground vehicle as seen in Figure 3.9(a) and moved to follow the motion of the quadrotor from both sides of the room. The duration of the trajectories and the corresponding maximum deviation values were recorded and the GP regression model was built using these data following the same procedure outlined in Section 3.3. The resultant GP regression is displayed in Figure 3.8.

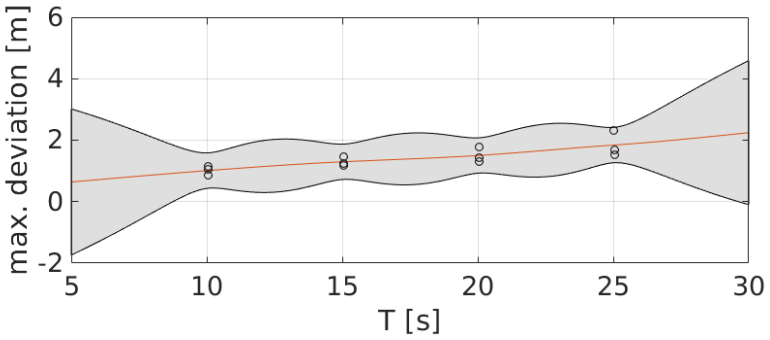
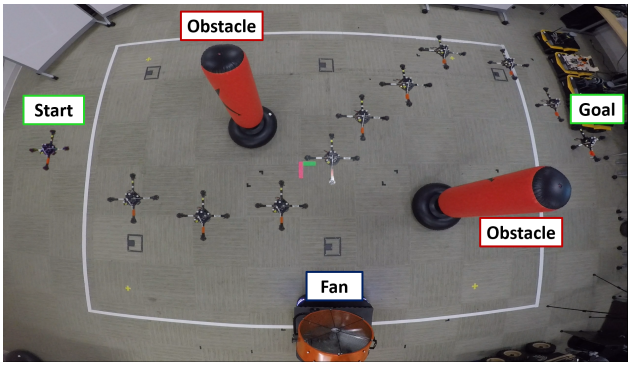
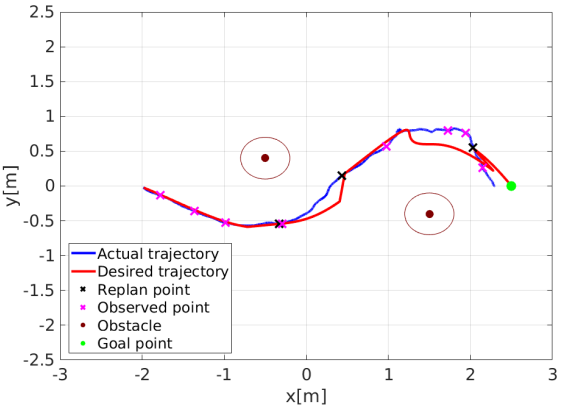


Figure 3.8: GP regression of maximum deviation based on the trajectory duration. 2019 ©IEEE

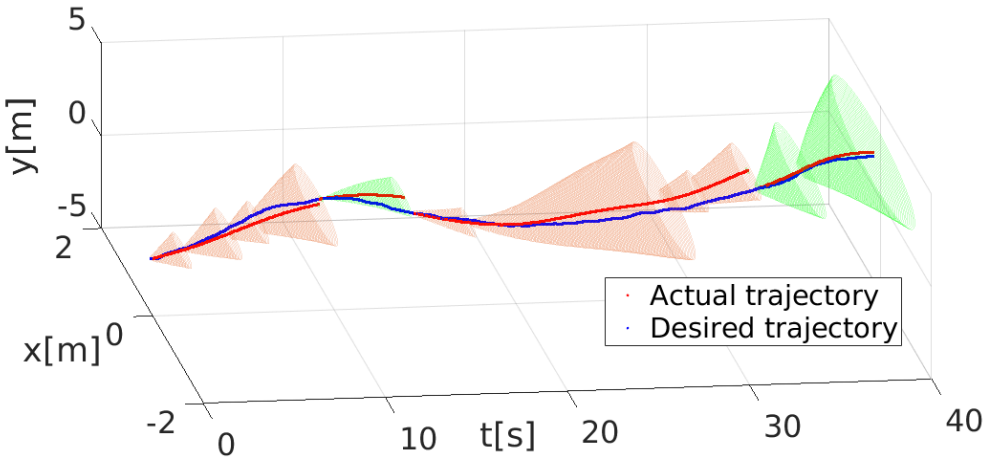
Similar to the simulations, the quadrotor is tasked to visit a goal following a new previously unseen trajectory with unknown intermittent measurements under unknown disturbances. Figure 3.9(a) shows an overlapped sequence of snapshots for a waypoint navigation experiment in which the UAV is tasked to go to a goal position at $[2.5, 0, 1]$m under the wind disturbance generated by the fan at a fixed position. It should be noted that fixing the fan position generates a disturbance whose magnitude is different from, but bounded by the magnitude of the disturbance used during training. Two obstacles (inflated poles) are present



(a) Overlapped sequence of snapshots during the experiment.



(b) Desired vs. actual trajectories of the UAV.



(c) The reachable sets associated to the experiment in (a).

Figure 3.9: Waypoint navigation experimental results. 2019 ©IEEE

along the path at the following positions: $\boldsymbol{p}_{o1} = [-0.5, 0.4]$, $\boldsymbol{p}_{o2} = [1.5, -0.4]$. In Figure 3.9(b), the actual path of the quadrotor and its desired trajectory are shown by blue and red curves respectively. Similar to the simulations, at the points marked by magenta "×" symbols in Figure 3.9(b), the quadrotor obtains its position information and the reachable sets are shrunk. At the points shown by black "×" symbols, the UAV replans its trajectory since the observed position is off more than the liveness threshold $\lambda_d = 0.4$m, thus new reachable sets are computed. Note that the actual path of the quadrotor stays inside the associated reachable sets at all the times during the experiment.

By using this approach, the quadrotor was able to complete its task without colliding with any obstacle and with a maximum deviation of 0.4109m. The position information became available only 11 times during this experiment and the quadrotor replanned its trajectory 3 times.

3.6 Discussions

In this chapter, we have presented a fast monitoring and replanning framework for autonomous systems under intermittent sensing and under the effect of disturbances and noises. Our approach leverages Gaussian processes theory for fast reachability analysis and utilizes reachable sets within our self/event-triggered monitoring and replanning framework to guarantee safety when the sensory measurements are not always available. Using this approach, we show that the autonomous system is able to estimate its reachable sets fast at runtime and safely perform its task in cluttered environments even when observations about the state of the system are not always available.

Limitations: When designing a fast reachability analysis model, the richness of the training set becomes important for making accurate worst-case predictions. For this particular application, we observe that the duration of the trajectory is a crucial factor for deviation estimation. Still, for other systems, different factors might significantly affect the deviation from the desired trajectory and need to be considered while building the GP regression model. If this framework is applied in a densely cluttered environment, and if the system does not get any random sensory measurements, the system might need to perform recovery actions too often, which may be inefficient for task completion. A runtime monitor to detect such conditions would be necessary to change the task planning in such cases.

Future Work: In addition to autonomous aerial vehicles, this approach could also be applied to other types of vehicles, especially on unmanned underwater vehicles as intermittent sensing problem under external disturbances is particularly evident in underwater environments. Additionally, this framework can also be extended in multi-vehicle planning problems when there are intermittent communication problem between the agents and in planning problems in extreme environments where the communication signals can be interrupted easily.

This fast monitoring and replanning framework introduces a way to approximate reachable sets fast at runtime using Gaussian process regression. Another way of making fast safety decisions at runtime is to utilize neural networks as they are very powerful to learn complex models, as explained in the next chapter.

Chapter 4

Assured Runtime Monitoring and Planning

In this chapter, we present our assured runtime monitoring and planning approach which utilizes verified neural networks. The framework contains offline neural network training based on precise reachability analysis tools and online safety monitoring and motion planning using the trained and verified network. At runtime, safety checker determines if a planned trajectory is safe or unsafe: when unsafe, a replanning procedure is triggered until a safe trajectory is obtained. To validate the proposed framework both in simulations and experiments, two illustrative case studies on a quadrotor aerial vehicle - a pick-up drop-off operation and a navigation in a cluttered environment - are presented. This work has been published in Robotics and Automation Magazine (Special Issue on Deep Learning and Machine Learning in Robotics) in 2020 [97].

4.1 Introduction

Traditional reachability analysis tools like Hamilton-Jacobi reachability [4] or hybrid system reachability analysis techniques [12, 37] have demonstrated to be very effective in predicting the future states of the system and to provide safety guarantees by leveraging the knowledge about the model of the system. However their computational complexity makes them challenging to be used at runtime.

In the previous chapter, we leverage regression techniques to compute reachable sets fast at runtime. When the reachable sets are only utilized to make safety decisions of the planned trajectories, the reachability computation can be completely bypassed using neural networks thanks to their ability to learn complex models. Unfortunately, since the performance of such learning enabled components highly depends on the properties of the training data, it is challenging to provide guarantees in safety-critical operations. To deal with these challenges, we present a framework to verify learning-enabled components for fast and safe monitoring and planning of autonomous operations.

A neural network (NN) which is trained to predict if an operation will be safe or unsafe is *verified* if its output decision (safe or unsafe) always concurs with the results obtained by running the operation under the worst case conditions in which it was trained. Since it is unlikely that a NN with this strong property exists, in this work we define a NN *conservatively verified*, in short verified, if at least the safe decision output of the NN always concurs with the result obtained by running the actual operation. In other words, if the NN output is *Safe* then the system can never reach an unsafe state; however, it is allowed for the NN to output *Unsafe* when the system will be safe since this is a conservative safety decision. This definition holds also in the extreme case that a NN outputs always only *Unsafe* decisions in which a vehicle will be always safe however it will not be able to move anywhere.

In this framework, a NN is trained to recognize safe and unsafe trajectories for an autonomous vehicle in an obstacle populated environment. Specifically, training is performed creating a library of trajectories and computing their reachable sets to make safety decisions, hence the computational burden is limited to the offline stage. A trajectory is labeled safe if its reachable sets do not overlap with any obstacle in the environment and it is otherwise marked unsafe.

Verification of the obtained NN is achieved using a verification tool Verisig [29] in which the NN is transformed into a hybrid system and the verification problem is cast as a hybrid system reachability problem. If the NN is not verified, meaning that it is not safe to be used, the output of the verification is used as a feedback to retrain the NN more conservatively until it is verified. Once the NN is verified, it is used at runtime as a safety checker for the planned trajectories from an untrained initial state under the effect of unknown online disturbances. If unsafe, a new trajectory is replanned and tested again for safety until a safety-guaranteed trajectory is obtained, if possible.

With this framework: *1)* we develop a fast safety checking and replanning approach for autonomous vehicles operations in cluttered environments under unknown runtime disturbances and *2)* we leverage a novel NN verification tool, Verisig, to verify and retrain the NN used as a fast safety monitor, before its deployment. As a result, we eliminate the need for computationally expensive reachability analysis tools for planning safe trajectories at runtime, and our verification method Verisig is capable of assessing the validity of the trained neural network.

To better illustrate our framework, two case studies on a quadrotor aerial vehicle are presented both with simulations and experiments under the presence of unknown disturbance during runtime: 1) a pickup/drop-off mission and 2) a safe navigation in a cluttered environment.

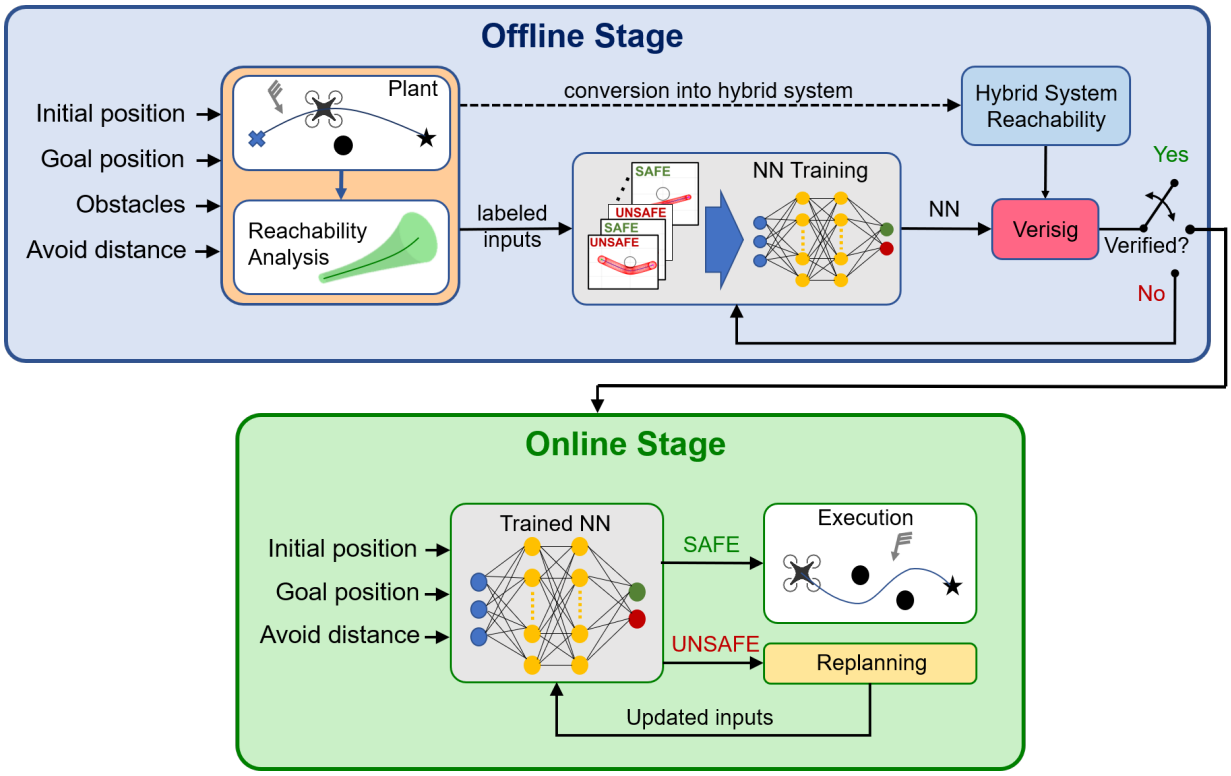


Figure 4.1: Overall architecture of the framework for verification of NNs for runtime monitoring and planning of autonomous operations. During the offline stage a NN is trained and verified, followed by its deployment at runtime for both monitoring and replanning purposes. 2020 ©IEEE

4.2 Verified Safe Motion Planning

Our verified safe motion planning framework consists of an offline and an online stage, as depicted in Figure 4.1. During the offline phase, a library of trajectories is generated. We parametrize the trajectory generation based on the initial state of the UAV, the desired goal position, the position of the obstacles on the way to the goal position, and the distance to keep from these obstacles. These trajectory parameters are labeled as safe or unsafe using reachability analysis, and a neural network is trained using this set of labeled parameters. To be able to use the neural network in autonomous operations, it should be guaranteed that the trained network never outputs safe when the trajectory is actually unsafe. To provide this guarantee, we verify the trained neural network using our verification tool Verisig which will be outlined in Section 4.2.3. In case the neural network is not verified, the Verisig output is utilized to retrain the neural network more conservatively (i.e., it outputs more unsafe decisions) until it is verified. Once the neural network is verified, it is used to make decisions about the safety of a new set of trajectory parameters at runtime. If the neural network decides that the trajectory is safe, the UAV executes the trajectory. If the decision is unsafe, the trajectory is replanned by changing the trajectory parameters until a safe decision is obtained.

4.2.1 Reachability Analysis

As also shown in previous chapters, reachability analysis is a very powerful method for computing the sets that a system could reach starting from an initial set. To train a neural network to predict the safety of a planned trajectory, we chose to use a hybrid system-reachability analysis tool Flow* [12] which uses Taylor models to compute flowpipe over-approximations of the dynamics, however our approach is independent from the choice of reachability analysis tool.

Specifically, during the offline stage, reachable sets for a planned trajectory are generated and if these reachable sets are not intersecting with the obstacles, the corresponding trajectory parameters are labeled as safe:

$$\begin{cases} s_\tau = 1 & \text{if } \boldsymbol{R}(\boldsymbol{p}_\tau, t) \cap \boldsymbol{p}_{o,j} = \emptyset \\ & \forall t \in [0, T], \forall j \in \{1, \cdots, N_o\}, \\ s_\tau = 0 & \text{otherwise} \end{cases} \tag{4.1}$$

where $\boldsymbol{p}_\tau$ is the desired positions along the trajectory τ, $\boldsymbol{R}(\boldsymbol{p}_\tau, t)$ is the corresponding reachable set, s_τ is the safety label, $\boldsymbol{p}_{o,j}$ is the j^{th} obstacle position, and N_o is the number of obstacles in the environment. A NN is trained using these safety labeled parameters to make safety decisions for trajectories with different parameters.

4.2.2 Neural Network Training for Safety Decisions

Following the diagram in Figure 4.1, after collecting a rich library of labeled trajectory parameters, we train a neural network to provide safety decisions without having to run computationally expensive reachability analysis operations at runtime. The inputs to the NN are the trajectory parameters, which are the initial and final position of the system, and the obstacle avoidance distance. The output of the NN is a binary *Safe/Unsafe* decision. Since these decisions are used by a safety-critical system, it is required that the NN never outputs a decision of *Safe* if the trajectory is *Unsafe*, as it can lead to dangerous outcomes (e.g., colliding with an obstacle). Therefore we are interested in training a NN with zero False Positive (FP). The drawback of reducing the number of FPs is that the number of False Negatives (FN) (safe trajectory marked as unsafe) may increase. Even a conservatively trained NN can output a safe decision for a set of untrained unsafe trajectory parameters. Therefore, absence of such FPs needs to be verified prior to its deployment.

4.2.3 Verification

To verify that the trained NN does not output *Safe* if the robot plans an unsafe trajectory, we use a verification tool called Verisig. As described in [29], Verisig was developed to verify safety properties of closed-loop systems with neural network components. Verisig focuses specifically on sigmoid-based neural networks and works by transforming the neural network into an equivalent hybrid system. The neural network's hybrid system is then composed with the plant, resulting in a new hybrid system that describes the entire closed-loop system as depicted in Figure 4.1. This allows us to cast the verification problem as a hybrid system reachability problem, which is solved by an optimized hybrid system verification tool such as Flow* [12]. Specifically, Verisig transforms the NN S into a hybrid system H_S by noting that the sigmoid derivative can be expressed in terms of the sigmoid itself, i.e.,

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)),$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid. With this observation in mind, we introduce a proxy function $\sigma_p(t, x) = \sigma(xt)$ such that $\sigma_p(1, x) = \sigma(x)$ and

$$\dot{\sigma_p}(x) = x\sigma_p(t, x)(1 - \sigma_p(t, x))$$

using the chain rule. In other words, σ_p can be considered as a state in a dynamical system that starts at $\sigma_p(0, x) = 0.5$ and is equal to $\sigma(x)$ at "time" 1. Thus, each neuron in the NN can be mapped to a state in a hybrid system, and each layer can be mapped to a mode. Transitions between modes occur when $t = 1$. Note that this time is local to the NN; global time does not progress during the NN's execution.

Although Verisig was originally applied to closed-loop systems with neural network controllers, it applies to the setup considered in this framework as well. In particular, in the verification problem, the NN's hybrid system does not interact with the plant directly; rather, for a given set of initial conditions, we compute the reachable set for the NN's output and check whether it is possible for the robot to crash (when started from that initial set) while the NN outputs *Safe*.

The composed hybrid system $H = H_q \| H_S$ is shown in Figure 4.2 where H_q is the hybrid system describing the robot's dynamics. The plant dynamics evolve as a function of the state, input and disturbance $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{d})$. H contains the union of modes and states of H_q and H_S. H starts in the initial mode of H_S and adds a transition from the last mode of H_S to the initial mode of H_q, at which point the H_q execution begins. The goal is to verify that H does not enter the $Plant(Unsafe)$ mode when $S(\boldsymbol{x}_0) = SAFE$.

Given a hybrid system description of the closed-loop system, one could use a tool such as Flow* to verify

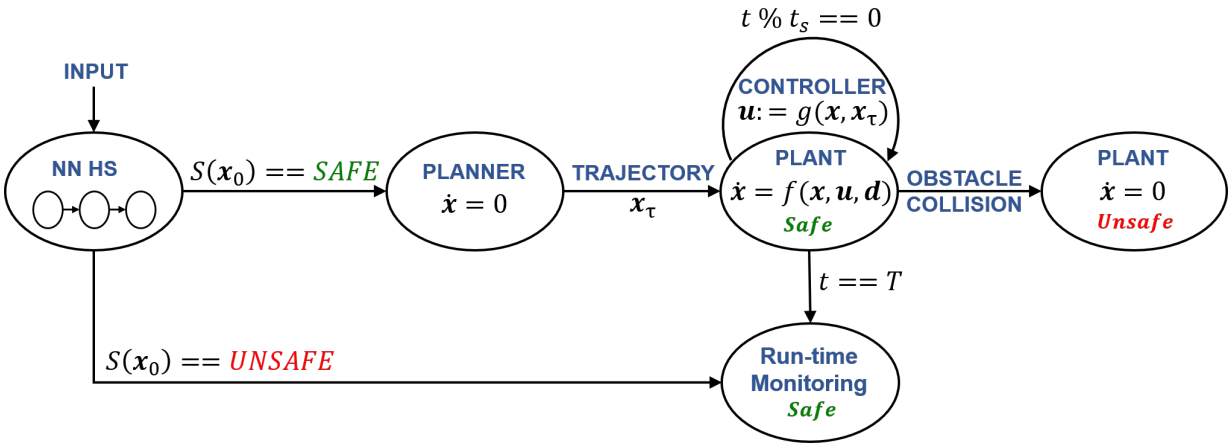


Figure 4.2: The composed hybrid system considered for verification of NNs for runtime monitoring.

the system's safety. Note that since hybrid system verification is undecidable in general, the typical approach used in these tools is to overapproximate the reachable sets. If the overapproximation does not contain any unsafe states, then the system is safe. If the overapproximation contains both safe and unsafe states, then the outcome is *Unknown* since the unsafe states could be spurious, i.e., they do not exist in the true reachable set but only in the overapproximated one. Finally, if all states are unsafe, then the system is unsafe. Various shapes have been explored in order to overapproximate the reachable sets, including polytopes, ellipsoids and hyperrectangles. Flow* uses a Taylor model approximation, which is a Taylor series approximation together with worst-case error bounds. Taylor models scale well when used with interval analysis and have been shown to have low approximation error for a large class of nonlinear systems [12].

4.2.4 Neural Network Retraining

At the end of the verification, Verisig specifies the regions in which: 1) the NN is safe to be used (i.e., the plant is not unsafe when the NN output is *Safe*), 2) the NN is not safe to be used (i.e., the plant is unsafe when the NN output is *Safe*), and 3) it is not able to make a decision due to the approximation errors introduced in the hybrid system reachability analysis during verification (i.e., the NN output is *Unknown*). In the case that there are regions in which the NN is not safe to be used, or Verisig cannot decide, the NN needs to be retrained. The output of Verisig can be leveraged to retrain the NN in several ways. One way is to collect more data around those regions where Verisig is not able to make a decision followed by a NN re-training. An increased density of data around previously untrained regions may help with the verification. However, how much data needs to be collected in those regions is not known a priori and hard to predict, so it could require multiple iterations of data collection and retraining. In addition, collecting new data to improve the training

set may not always be possible. Instead, we to add points from the unsafe/unknown regions obtained from the Verisig output into the existing training set, mark them with unsafe labels and finally retrain the NN. By retraining the NN with more unsafe points, a more conservative NN is obtained in which unsafe regions are inflated, helping with the verification process. This retraining process is repeated until the NN is verified.

4.3 Case Studies

As a proof of concept, our verified safe monitoring and planning approach is applied to two case studies on quadrotor motion planning: i) a pickup/drop-off mission and ii) a navigation operation in a cluttered environment. Both case studies use similar neural networks to predict if the vehicle trajectory will be safe or unsafe and thus require offline training and verification. At runtime, the verified neural network is used for different purposes. The pickup/drop-off task requires the UAV to go from one side of a static environment to the other side resembling operations that could happen inside a warehouse or a factory. The UAV makes decisions about the safety of the planned trajectory based on the neural network results and replans by adjusting the obstacle avoid distance until it finds a longer but safe trajectory to its goal position. In the latter case study, the UAV is tasked to navigate through a previously unknown cluttered environment. Training is executed in a smaller environment with only one obstacle acting as a primitive scenario that can appear and be composed multiple times at runtime. Training in a primitive environment allows the neural network and verification to be generalized to different environments with the same type of obstacles located in previously unknown positions. Replanning here is executed by querying different waypoints along the path to the goal until the neural network outputs a *Safe* decision. In both case studies we use the same vehicle, controller, planner, and disturbances, whose models are briefly summarized in the following subsection:

4.3.1 System Models

Quadrotor UAV and Controller Model

We use a simplified quadrotor model with the state vector $\boldsymbol{x} = [\ x \quad y \quad z \quad v_x \quad v_y \quad v_z\ ]^\mathsf{T}$ where x, y, z are the world frame positions and v_x, v_y and v_z are the world frame velocities. The quadrotor dynamics can be defined as: $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{d})$ where $\boldsymbol{u} = [\ F \quad \phi \quad \theta\ ]^\mathsf{T}$ is the input vector with thrust, roll and pitch commands,

and $\boldsymbol{d} = [\ d_x \quad d_y \quad d_z\ ]^\mathsf{T}$ is the external disturbance vector. The dynamics can be described as:

$$\begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^\mathsf{T} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^\mathsf{T}$$

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} g\theta \\ -g\phi \\ \frac{F}{m} - g \end{bmatrix} + k_d \begin{bmatrix} d_x - v_x \\ d_y - v_y \\ d_z - v_z \end{bmatrix}$$

where m is the mass of the quadrotor, g is gravity, k_d is the drag coefficient. It should be noted that in this paper a simplified quadrotor UAV model is used in order to alleviate the verification problem. Verifying a high-fidelity model is left for future work.

In order to generate the necessary roll, pitch, and thrust inputs to follow the desired trajectory, a cascaded set of PD controllers are used [56].

Disturbance Model

We considered an external disturbance which is bounded in magnitude: : $\|\boldsymbol{d}\| \leq D_{max}, \forall \boldsymbol{d} \in \mathcal{D}$ where D_{max} is the upper bound to the disturbance magnitude, and $\mathcal{D}$ is the set of all possible disturbances. Here we assume that the online disturbance is unknown but constant over time. This is a reasonable assumption as wind disturbance generally follows a Brownian motion and does not change erratically over short periods of time [17, 62].

Trajectory Planning

Obstacle avoidance trajectories are computed using a simple geometric approach. Specifically, if an obstacle is present along the way to the goal position, a waypoint away from the obstacle center by a specified avoid distance r_a is added to the path. A trajectory is finally generated to visit all the waypoints on the path by using a minimum jerk trajectory generation [55]. It should be noted that we use this path planning method due to its simplicity in implementation in simulations and experiments, however, the overall proposed framework is independent from the choice of path planning method.

4.3.2 Pickup/Drop-off Task

The first case study that we present is a pickup and a drop-off task, an operation which is commonly used in factory applications where the vehicle moves back and forth between a warehouse and a workstation. The environment has a designated pickup area (warehouse) and a drop-off position (workstation), with obstacles

at known locations in between. The vehicle is tasked to move from a point inside the pickup area to the drop-off location. Once it reaches the drop-off location, it can move back to a new point in the pickup area. In order to complete its mission safely, the UAV needs to decide if the planned trajectory – parametrized by the initial position $\boldsymbol{p}_0$, final goal $\boldsymbol{p}_g$ and the avoid distance r_a – is safe or not, and if not safe, replanning is required. In this case replanning is executed by adapting r_a.

To train a NN to make safety decisions in this scenario, two sets of trajectories from a rich set of initial positions in the pickup area to the drop-off position and from the drop-off position to a rich set of final positions in the pickup area with different avoid distances are generated and labeled using reachability analysis.

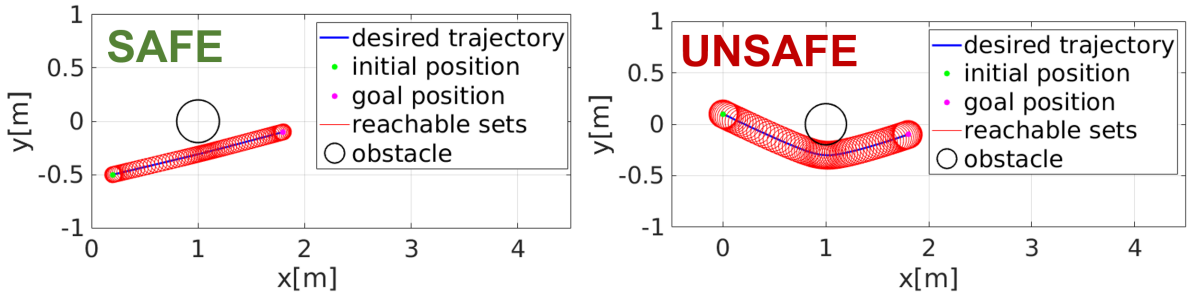
The NN queries the initial, final positions and the avoid distances and if unsafe, it checks a larger avoid distance until it outputs a safe decision.

Simulation-based Reachability

In this specific case study, we use a simulation-based reachability analysis. During the offline stage, we run each training trajectory under the worst case scenario which in our case is the largest possible disturbance attainable in the environment. Under this condition, for a given trajectory $\boldsymbol{p}_\tau$, the maximum deviation d_m is calculated as follows:

$$d_m = \max_{\boldsymbol{d}\in\mathcal{D}} \max_{t\in[0,T]} \min_{\xi\in[0,T]} \|\boldsymbol{p}_{\boldsymbol{d}}(t) - \boldsymbol{p}_\tau(\xi)\| \tag{4.2}$$

where $\boldsymbol{p}_{\boldsymbol{d}}$ is the position of the vehicle under disturbance $\boldsymbol{d}$. d_m is used as an upper bound for the actual deviation from the trajectory and it is conservative since it is the maximum deviation measured through the entire trajectory.



(a) A safe trajectory.

(b) An unsafe trajectory.

Figure 4.3: Reachable sets for two sample trajectories. 2020 ©IEEE

The position reachable sets are then generated as follows:

$$\boldsymbol{R}(\boldsymbol{p}_\tau, t) = \{\boldsymbol{p}(t) : ||\boldsymbol{p}(t) - \boldsymbol{p}_\tau(t)|| \leq d_m\} \tag{4.3}$$

After generating the reachable sets, the trajectory is labeled as safe or unsafe according to (4.1). In Figure 4.3, we show the reachable sets of two sample trajectories.

Offline Training

The environment has a designated rectangular pickup area which is limited between $[0.0, 1.3]$m in the x axis and $[-1.0, 1.0]$m in the y axis. The drop-off point is located at $\boldsymbol{p}_g = [4.0, 0.0]$m. There are two obstacles between the pickup area and drop-off location positioned at $\boldsymbol{p}_{o1} = [2.0, 0.1]$m and $\boldsymbol{p}_{o2} = [3.0, -0.1]$m. For training, 294 points are uniformly distributed in the pickup area and used both as initial and final positions. For trajectory generation, seven different avoid distances are considered: $r_a \in \{0.3, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7\}$ m.

Two NNs were trained: one for the drop-off operation and the other for the pickup task. To implement NNs, we chose Keras (`https://keras.io`), a deep learning library capable of running on top of Tensorflow (`https://tensorflow.org`) through a set of APIs written in Python. For all the layers (input, hidden, and output), we use a sigmoid activation function. The NN is composed of three input nodes (x-y initial position and avoid distance pair), one hidden layer of 40 nodes, and one output which determines if the label is *Safe* or *Unsafe*. We trained two different NNs, one for each subtask (drop-off and pickup). The training results showed 0% FP and about 5.2% FN for the first NN and 0% FP and about 1.2% FN for the second one.

In Figure 4.4, the initial positions in the training set for the drop-off (left) and pickup (right) operations are presented together with their respective labels and the NN results inside the proposed workspace. Each subfigure denotes a different avoid distance marked on top of the figure. Due to space constraints, we show examples of data for only three avoid distance values: $r_a = 0.3$m, $r_a = 0.45$m and $r_a = 0.7$m. Arrows inside the workspace indicates the direction of motion of the vehicle. Inside each subfigure, the green (red) dots represent initial positions from which the trajectories to the goal are labeled as *Safe* (*Unsafe*) using reachability analysis. The green (red) circles around the dots denote the decisions of the NN on the same training points. As can be noticed and as expected, when the avoid distance increases, the number of safe initial positions also increases in both missions because the distance between the desired trajectories and obstacles becomes larger. Therefore, increasing the avoid distance improves safety, however the trajectories become longer, which generally is not desirable due to energy concerns.

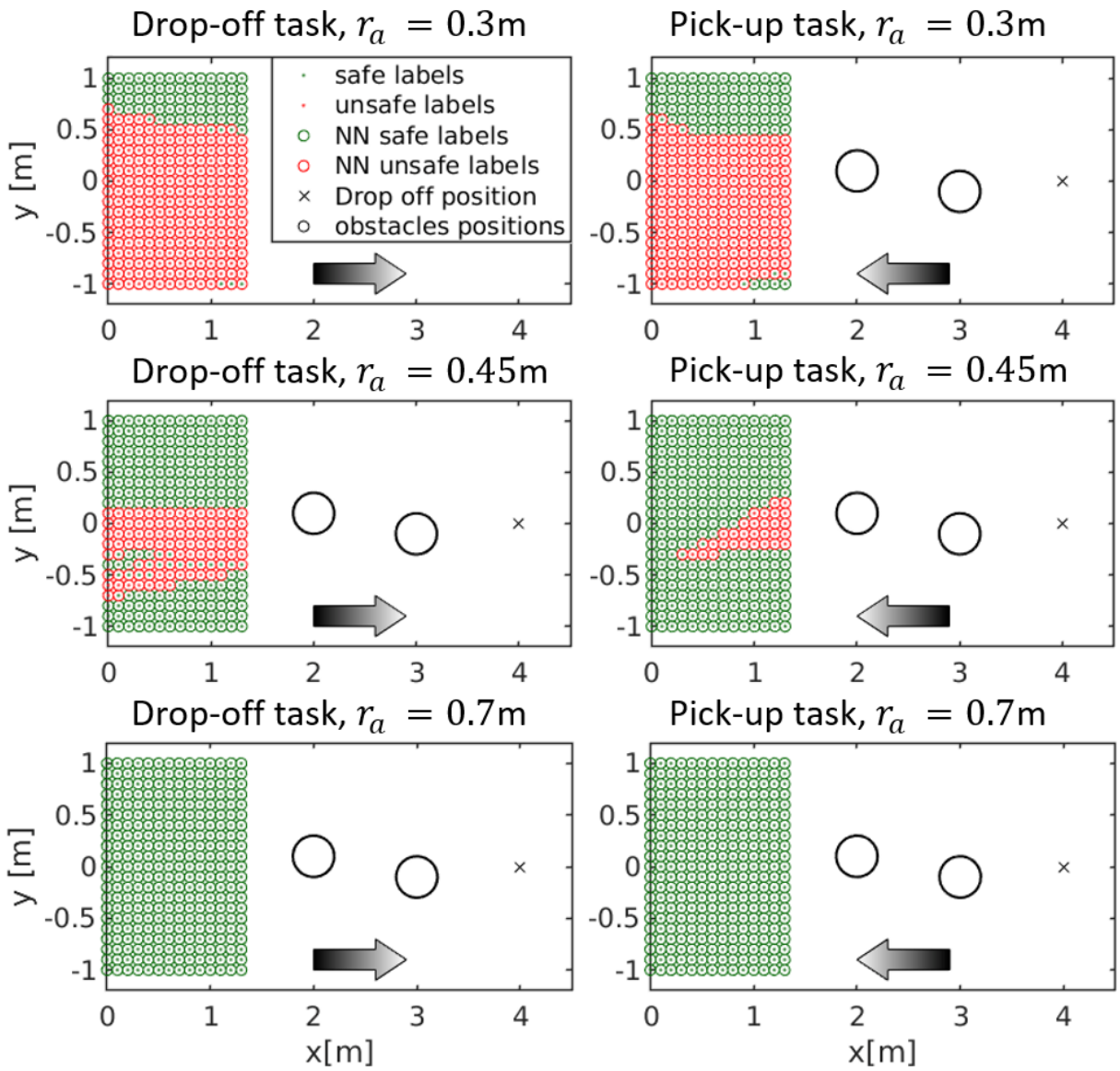


Figure 4.4: Safety maps for initial and final positions in training sets with different avoid distances for drop-off (left) and pickup (right) missions. 2020 ©IEEE

Verification Results

Verification was performed with the methods in Section 4.2.3. Here we present the results for the second drop-off task shown in Figure 4.4, namely the case in which the UAV starts in the set $x_0 \in [0, 1], y_0 \in [-0.5, 0.5]$ and aims to reach the goal at $[4, 0]$m with $r_a = 0.45$m and disturbances $d_x, d_y \in \{-0.1, 0.1\}$m/s.

The verification results are shown in Figure 4.5. We divided the initial set into smaller subsets and verified for each one separately in order to keep the approximation error in Flow* small enough. The size of these subsets (i.e., the 5cm boxes inside the right subfigure in Figure 4.5) was chosen after some preliminary testing

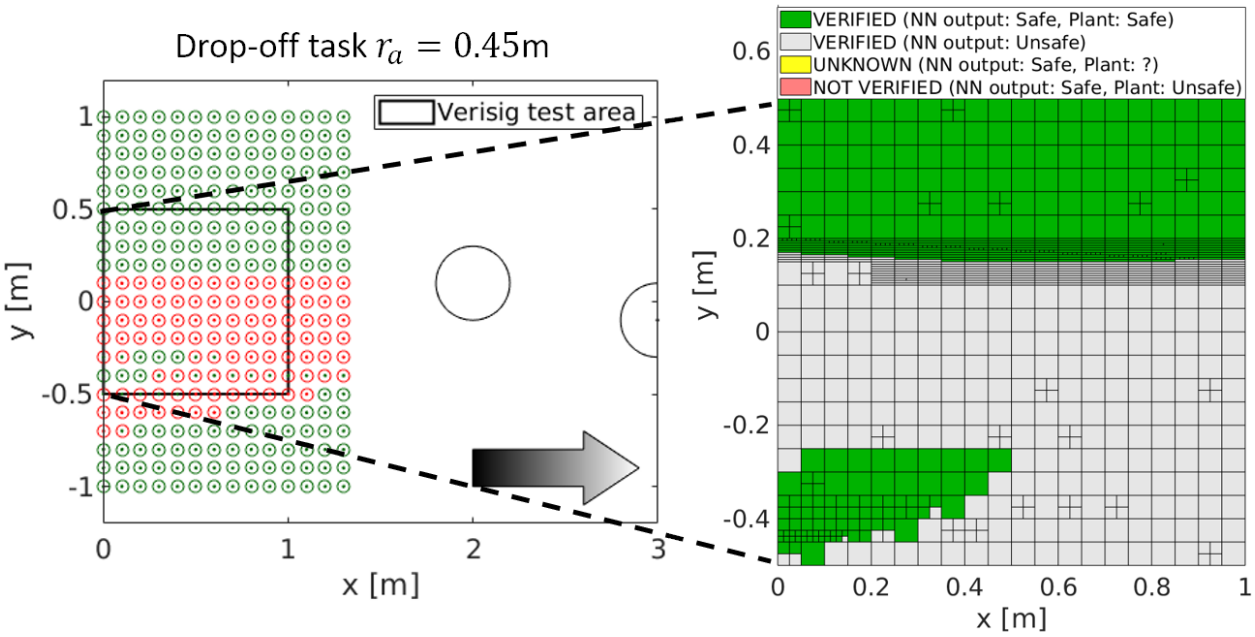


Figure 4.5: Verification results for the pickup task with avoid distance $r_a = 0.45$m in Figure 4.4. The initial set was divided into small subsets and verified. No unsafe sets were obtained. 2020 ©IEEE

– these subsets were large enough so as to verify the majority of the initial set with small approximation error. Some subsets were further refined when an instance resulted in a too large error. Refinements were necessary at the NN's decision boundary as well as for sets that triggered multiple if-cases in the planner (e.g., when planning around multiple obstacles). The verification was performed using Amazon Web Services (`https://aws.amazon.com`). Each subset took roughly an hour to verify, although some took longer due to branching introduced by if-cases in the planner. Figure 4.5 matches closely the corresponding graph in Figure 4.4. We verified that the NN is indeed conservative so that no unsafe events occur when the NN's output is *Safe*.

The same procedure can be used for all other cases. Note that we only verified safety for the first NN as a proof of concept; the verification procedure for the second NN would be exactly the same.

Experimental Results

We applied our framework on a real quadrotor experiments in which we designed a similar pickup/drop-off scenario (see Figure 4.6). The quadrotor was tasked to visit different points in the pickup area and return back to the drop-off location every round, while avoiding two obstacles (Figure 4.7).

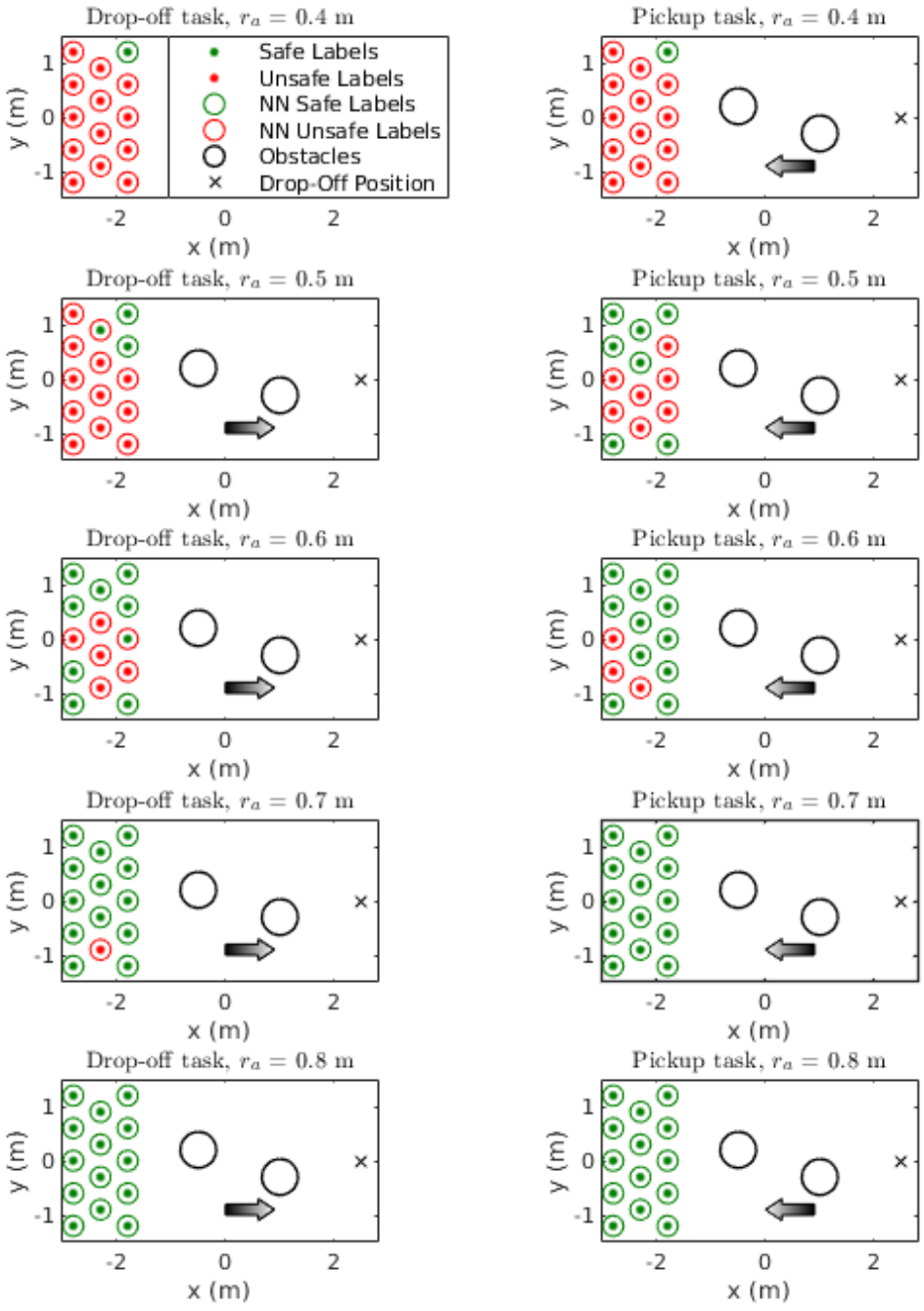


Figure 4.6: *Safe* and *Unsafe* training and NN results for the pickup and drop-off tasks in the area where experiments were performed. 2020 ©IEEE

Real flights were performed with an Asctec Hummingbird quadrotor controlled through ROS. A Vicon Motion Capture system was used to track the position of the quadrotor and to provide ground truth position information. Two industrial fans blew wind in the middle of the area creating a disturbance towards the obstacles. In order to generate *Safe* and *Unsafe* labels for the real scenario, 14 positions equidistant from each

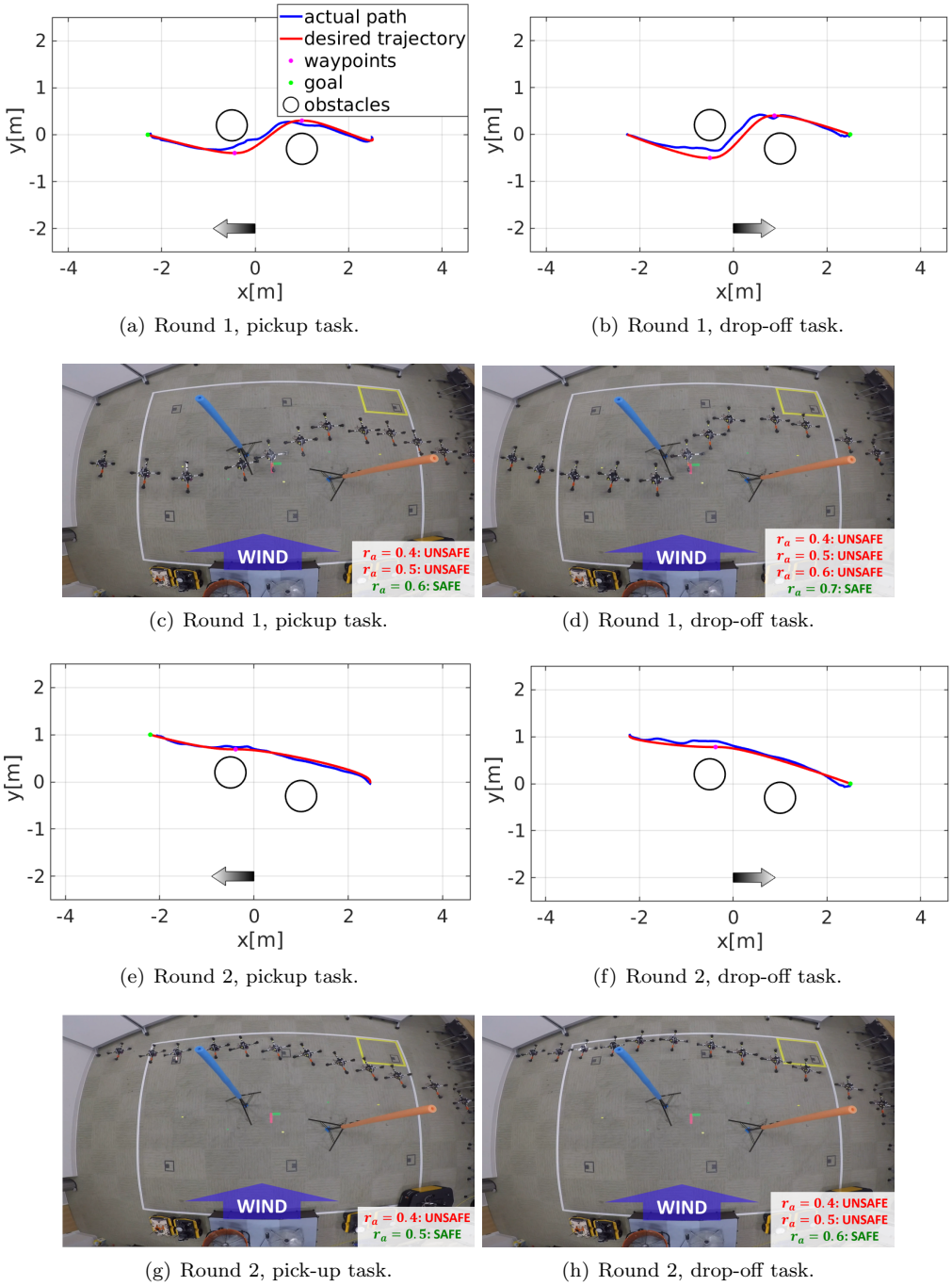
other in the pick-up area were considered. For each avoid distance $r_a \in \{0.4, 0.5, 0.6, 0.7, 0.8\}$m, trajectories were generated from each starting position to the drop-off goal, and vice-versa, and safety decisions about these trajectories were made using a similar approach explained in Section 4.2.1. A trajectory here was labeled as *Unsafe* if the maximum deviation from the desired trajectory became larger than the distance between the obstacle and the desired trajectory at any point along the trajectory.

We trained two NNs, one for each sub-task. In Figure 4.6, the safety decisions are shown when the training set is provided as input to the NN. A FP (red dot inside a green circle) corresponds to an *Unsafe* label (red dot) with a *Safe* NN decision (green circle) while a FN (green dot inside a red circle) is a *Safe* label (green dot) marked as *Unsafe* (red circle). The number of FPs is zero for both NNs, and there are two and zero FNs for the first and second NN, respectively.

During testing, we exploited the trained NN by executing multiple passes back and forth from-and-to different points in the pick-up area under an unknown disturbance generated in the middle of the area that could push the quadrotor towards the obstacles.

Through the experiment, the vehicle was tasked to navigate using the smallest avoid distance $r_a = 0.4$m if possible. As expected, the NN generated *Unsafe* outputs for some of the points. Consequently, r_a was increased by 0.1m increments until a *Safe* decision was returned by the NN before sending the vehicle to the goal. The computation time for the NN to make a safety decision is in the order of milliseconds and it is constant for all trajectories, making it suitable for online replanning operations. On the contrary, the simulation based reachability used during training is not suitable at runtime as its computation time increases linearly with the duration of the trajectory. Figure 4.7 shows the sequence of snapshots, the decisions of the NN, and the comparison between the desired and the actual trajectories (top figures) for two rounds of the operation.

Finally, we also run an experiment in which we decided to generate trajectories with the minimum avoid distance $r_a = 0.4$m under the same wind disturbance in the previous experiment disregarding the decision from the NN. During the first round, which was predicted unsafe with $r_a = 0.4$m, the quadrotor collided with an obstacle (Figure 4.8) confirming the *Unsafe* decision predicted by the NN.



(a) Round 1, pickup task.

(b) Round 1, drop-off task.

(c) Round 1, pickup task.

(d) Round 1, drop-off task.

(e) Round 2, pickup task.

(f) Round 2, drop-off task.

(g) Round 2, pick-up task.

(h) Round 2, drop-off task.

Figure 4.7: Experimental results in which the trained NN was used to make safety decisions and replan accordingly. 2020 ©IEEE

4.3.3 Navigation in Cluttered Environments

The other case study that we demonstrate is a navigation operation in a cluttered environment like a heavily forested area. A UAV is tasked to reach a goal position in an environment in which obstacles are scattered

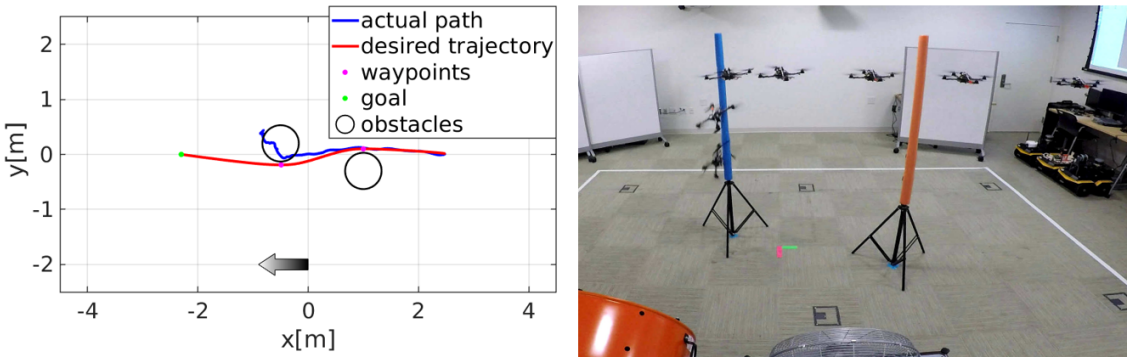


Figure 4.8: The trajectory followed by the quadrotor for the first pick-up task in Figure 4.7(a), with $r_a = 0.4$ m and marked unsafe, resulting in a crash. 2020 ©IEEE

in apriori unknown locations discoverable at runtime as the system moves to the final goal. In order to plan safety-guaranteed trajectories, here we considered a smaller environment with only one obstacle in the center and trained and verified an neural network to predict the safety of the trajectories within this smaller environment. The trained and verified *primitive* space can then be fitted and composed multiple times at runtime to assess safety in larger spaces with more obstacles.

Through a mission, every time that the UAV encounters an obstacle along its path, it selects an intermediate goal point around the obstacle and queries the trained NN about the safety of the trajectory to the selected goal. If *Unsafe*, a new intermediate goal is queried until the output of the NN is *Safe*. This procedure is repeated multiple times, for each obstacle along the path, until the vehicle reaches the final goal position.

Offline Training

To train the NN for this operation, we used a small box-shaped workspace with one obstacle in the center as shown in Figure 4.9. We picked 44 initial and 44 final positions uniformly distributed across the start (left of the obstacle) and goal (right of the obstacle) regions.

The safety of the trajectories generated from these 1936 start-goal position pairs is decided offline using Flow* reachability analysis [12]. On average it took about 5 minutes for Flow* to make a safety decision for one start-goal position pair under bounded disturbance conditions ($d_x, d_y \in [-0.4, 0.4]$m/s), reinforcing the fact that reachability analysis is expensive to be performed at runtime.

Using this set of initial-final position pairs, a NN was trained to predict the safety of an untrained pair of initial-final goals. The NN is composed of four input nodes (x-y initial position x-y goal position pair), one hidden layer of 40 nodes, and one output which determines if the label is *Safe* or *Unsafe*. The NN showed 0% FP and about 0.2% FN performance. In Figure 4.9, we show examples of labels and NN decisions for trajectories starting from three different initial positions to all the final goals in the training set. Similar to

the previous case, a green (red) dot represents a final position in which the trajectory was labeled as *Safe* (*Unsafe*) from a given starting point and a green (red) circle represents the neural network decision for the same point.

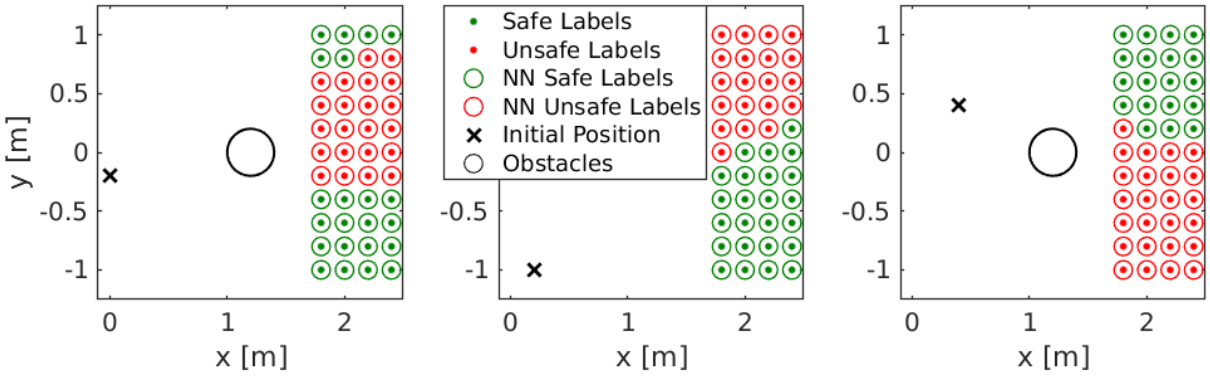


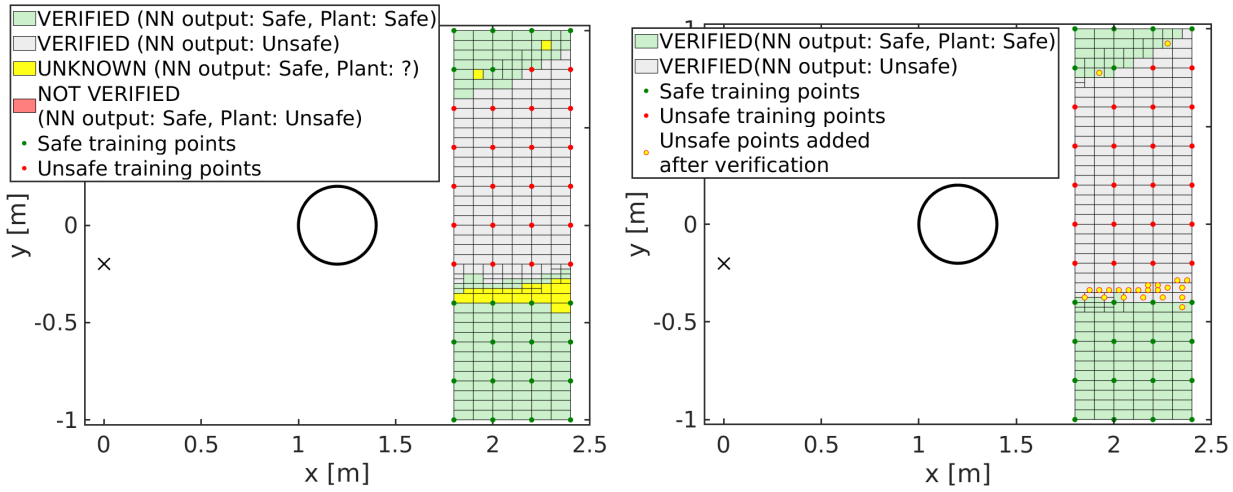
Figure 4.9: *Safe* and *Unsafe* trained final goals and NN decisions from various initial positions. 2020 ©IEEE

Neural Network Verification

Similar to the previous case study, since the results of a NN could be erroneous, we use Verisig to verify the safety predictions obtained by the trained NN. As a proof of concept, to demonstrate the procedures explained in Sections 4.2.3, and 4.2.4, the results of the neural network from a single initial position to all goal positions in the primitive environment are verified. However, the same NN verification and retraining procedure can be done for all possible initial and final regions. The training data for this case is presented in the first subfigure in Figure 4.9 while Figure 4.10(a) shows the results of the verification. The gray shaded regions in Figure 4.10(a) represent areas where the NN outputs *Unsafe* and Verisig concurs without performing the whole verification since the NN output is already unsafe and thus it is in the worst case scenario conservative. Green regions represent areas where the NN outputs *Safe* and Verisig verifies that the plant is safe, too. In the yellow regions, the NN outputs *Safe*, but Verisig cannot decide if the plant is safe or not.

Since the NN is not completely verified due to the undecided regions, points from these regions are added into the existing training set and the NN is retrained as explained in Section 4.2.4.

These points are shown by yellow dots in Figure 4.10(b). After retraining the NN with the addition of these points, the NN showed 0% FP and about 1.9% FN performance which is expected since it is trained to be more conservative. Figure 4.10(b) shows the verification results with this retrained NN and as it can be noted, the entire goal region is verified by Verisig.



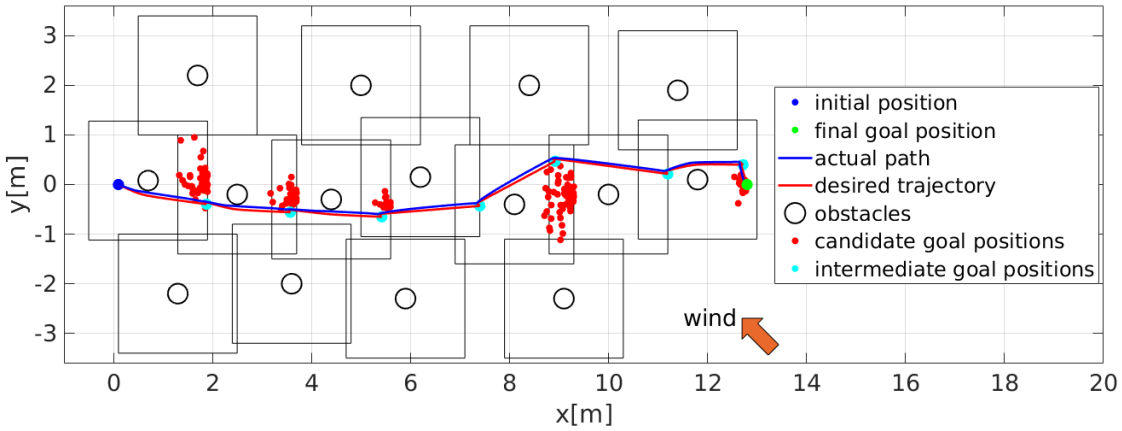
(a) Verification of the NN trained with the original dataset. (b) Verification of the retrained NN with the conservative dataset.

Figure 4.10: Neural network verification results. 2020 ©IEEE

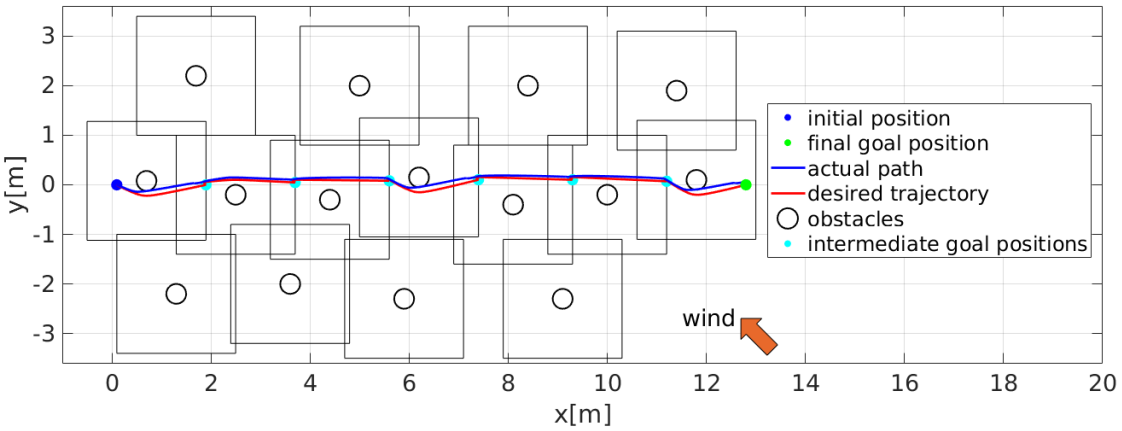
Simulation Results

In this simulation, the trained NN is used to make decisions about the safety of a trajectory in the cluttered environment presented in Figure 4.11. First, the primitive box space used for training is superimposed around each obstacle (square areas inside Figure 4.11) in such a way that there is only one obstacle in each primitive space, otherwise the results of the NN may not be reliable due to the difference from the training conditions.

Once the mission is started, the quadrotor picks the closest point to the final goal inside the goal region of the first primitive as an intermediate goal. The NN makes a decision about the safety of this intermediate goal from the current position of the UAV. If the decision is deemed *Safe*, the UAV moves to this intermediate position. If the NN decision is *Unsafe*, it searches for a safe goal position in the goal region of the current primitive. This search is performed by randomly querying points in the goal area of the primitive starting from a closer proximity of the initially selected goal and radially enlarging the search area if no safe goals are obtained immediately. Note that in order to deploy such approach, the NN must contain at least one safe point in the goal area of the primitive. This process continues until the UAV reaches its final destination. Figure 4.11(a) shows the trajectory followed by the quadrotor in this environment. The queried intermediate goal positions that were found by the NN unsafe are shown by red dots in the goal regions in the primitives areas, while the safe intermediate goals traveled by the UAV are shown by cyan dots. Wind disturbance is present through the entire mission blowing in the northeast direction as shown by the orange arrow inside the figures. The UAV is able to complete the mission without any collision. In Figure 4.11(b), we repeated the same case without using the NN decisions in which the UAV moves to the intermediate goal positions



(a) Safe replanning using NN decisions.



(b) Planning without using NN decisions.

Figure 4.11: Navigation simulation in a cluttered environment. 2020 ©IEEE

even if the NN decision is *Unsafe*. As expected, there are instances where the UAV crashes or gets very close to the obstacles. These results confirm that NNs can be used to monitor safety properties of motion planning operations using composition of smaller verified regions into larger, more complex and untrained environments.

Experimental Results

The same case study was also performed in experiments following a similar setup as the one presented in the previous experiment. NN training was performed on a smaller primitive environment with one obstacle, by performing 100 flights with our aerial testbed under wind disturbance blowing in the $+y$ direction. An initial-final position pair was labeled as unsafe if the reachable sets generated using the approach explained in Section 4.3.2 collided with the obstacle.

Using these safe/unsafe labeled initial-final position pairs, a conservative NN was trained to make safety decisions about untrained initial-final position pairs at runtime. Figure 4.12, shows the safe and unsafe initial-final position pairs and corresponding NN decisions, using the same color coding as the previous cases.

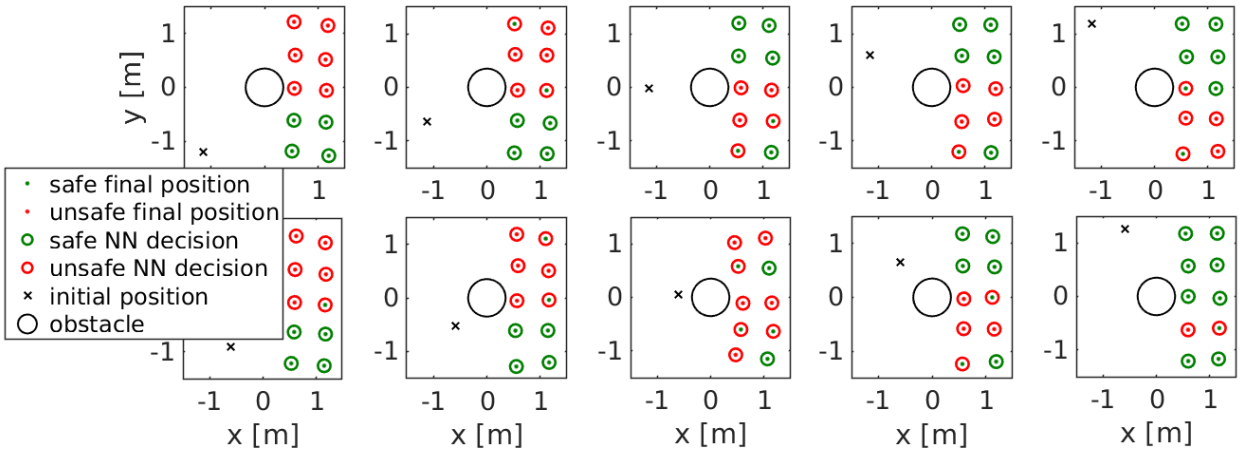


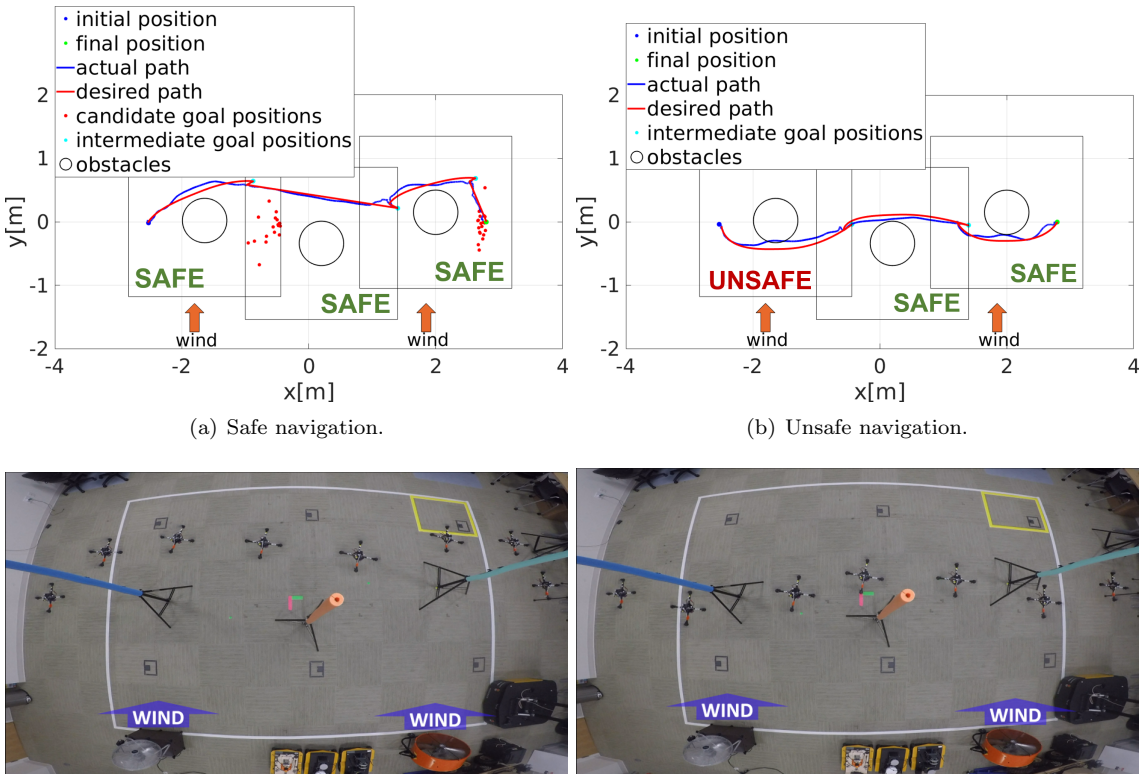
Figure 4.12: Safe and unsafe final positions and NN results from different initial positions in the experimental setting. The NN here is composed of four input nodes ($x-y$ initial position $x-y$ final position pair), one hidden layer of 40 nodes, and one output for the safety decision. The NN showed 0% FP and about 16% FN performance. 2020 ©IEEE

The safe navigation approach was validated in an environment with three obstacles and two fans blowing air in the $+y$ direction as seen in Figure 4.13. Obstacles in Figure 4.13 are represented as circles having radius equal to the actual obstacle size plus the size of the UAV.

In Figure 4.13(a), the intermediate goal positions queried by the NN are shown using the same color code as the simulation results. The UAV queried 20 points to avoid the first obstacle until it found a safe intermediate goal and repeated this operation until it reached its final destination. Using this NN-based framework, it takes a few milliseconds to search for a safe goal. The experiment results of the proposed approach are compared with the ones in which the NN is not utilized to make safety decisions about intermediate goal positions. As expected, in Figure 4.13(b) the UAV fails to complete its mission safely as it crashes and gets very close to the other obstacles. These experiments were executed without the obstacles. The obstacles are overlaid in the figure for reference.

4.4 Discussions

In this chapter, we presented an assured planning framework which enables fast and assured predictive and proactive monitoring of autonomous systems operations in cluttered and uncertain environments at runtime. Neural networks trained based on reachability analysis safety decisions are leveraged to make safety decisions



(a) Safe navigation.

(b) Unsafe navigation.

Figure 4.13: (a) Experimental results of the navigation of the quadrotor using the trained NN to make safety decisions and replan and (b) unsafe navigation in which NN decisions were disregarded. The bottom row of subfigures show snapshots of the experiments relative to the top row. 2020 ©IEEE

about the safety of planned trajectories at runtime bypassing the reachability computation. In this way, most of the computation burden is limited to the offline operations leaving the fast decision making and replanning tasks for the online application. Using the verification tool Verisig, the safety of the neural networks to be used in safety critical systems is assured. The applicability of the framework is demonstrated in two case studies.

Limitations: This assured planning framework requires accurate knowledge of the system dynamics, controller, and path planner model for verification, which prevents the system from adapting its path planner or controller at runtime. It is also assumed that the testing disturbances are bounded by the training conditions. For the pickup/drop-off case study, the environment that the system is operating at runtime needs to be known during the design time. For the navigation task, the approach is limited to work in environments where the obstacles are sparse enough to compose the primitive environments so that there will be one obstacle in each primitive environment. Additionally, we assume that there is always a safe goal location to go in the primitive environment. If the system is known to be deployed in a dense environment, a smaller primitive environment needs to be used during training, which might decrease the smoothness and efficiency of the resultant path.

Future Work: The framework for verification discussed in this work is a step towards assurance driven design of learning enabled components. However, many challenges are still present ahead of us that need to be addressed to fully integrate these technologies in safety critical operations.

A first challenge typical of LECs lays on how to select the appropriate training set. The accuracy of any machine learning technique depends largely on the type, amount, and heterogeneity of data used during the training phase. A poorly trained neural network results in poor performance leading to unsafe or over-conservative behavior of autonomous systems. Similarly, overfitting can introduce overhead and poor prediction. To deal with these issues, it is possible to perform sensitivity analysis [57] or nonconformity analysis [63] on the system prior to perform training to better interpret and select data, or leverage knowledge about the system dynamics to perform verification before the deployment of the LEC, as suggested in this work.

Secondly, verification provides safety guarantees on the outputs of such LECs, however, it does not provide any robustness guarantees against changes between training and testing conditions. Currently, state of the art verification techniques, beyond machine learning applications, [89] require knowledge about system model and bounded conditions. However no guarantees can be provided if for example the disturbance acting on a system is above or below the bounds for which training was performed. In the next chapters, we address this challenges by introducing online learning approaches to update the offline trained models when the testing conditions differ from the training conditions.

Additionally, verification techniques typically consider a specific model that may and will change from the actual model of a real system. Even precise system identification techniques are not able to provide fully accurate models and often rely on specific operating conditions. One way to provide verification for systems with model uncertainties is to consider a conservative abstracted model and verify that the real system is equivalent or safer than this abstraction. Building on the intuition in [23], if the system performance and behavior is verified to be closer to the desired behavior than the abstraction, then verifying the abstraction also verifies the real system. Note that, since these conservative abstractions capture the worst-case behavior of the system, failing to verify the abstraction may lead to believe that the actual system is not safe which may not necessarily be true.

Lastly, safety has been the main focus in this framework, therefore the planned trajectories can result in sub-optimal performance in terms of the efficiency. Improving the efficiency and optimizing the planned trajectories while still assuring safety is an open and interesting problem that could be addressed in the future.

Chapter 5

Runtime Planning and Learning for Unforeseen Uncertainties

In this chapter we introduce our online planning, learning, and recovery framework for autonomous operations under unforeseen and outside of training bounds uncertainties. Our approach estimates the behavior of the system with an unknown model and provides safe plans at runtime under previously unseen disturbances by leveraging Gaussian Process regression theory in which a model is continuously trained and adapted using data collected during the autonomous operation. A recovery procedure is event-triggered any time a safety constraint is violated to guarantee safety and enable learning and replanning. The proposed framework is applied and validated both in simulation and experiment on an unmanned aerial vehicle (UAV) delivery case study in which the UAV is tasked to carry an unknown payload to a goal location in a cluttered/constrained environment. This approach has been published at 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) [93].

5.1 Introduction

In the previous sections, we introduce model-based and learning-enabled approaches to improve safety of autonomous systems operating under uncertainties and disturbances. The performance of learning-based techniques highly depends on the training data, and it becomes challenging to provide safety guarantees when the training data are limited or when the test data are outside of the training bounds. It is thus essential to adapt and update the learning enabled components of the system with new data acquired at runtime all while guaranteeing safety.

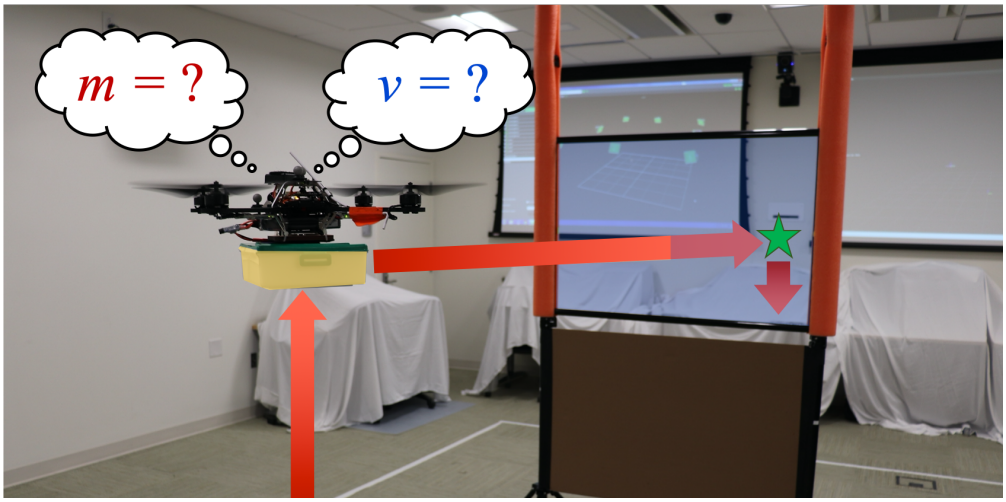


Figure 5.1: Pictorial representation of the problem of planning with unknown payload. 2020 ©IEEE

For example, let's consider an autonomous delivery case study in which a UAV has to carry unknown loads between two points while flying through narrow corridors and windows and around and over obstacles of varying heights and dimensions, as depicted in Figure 5.1. The load will affect the motion of the vehicle creating deviations from the desired planned trajectory and could lead to unstable and unsafe conditions (e.g. a collision), if not properly considered during planning.

In this chapter, we introduce a recursive dynamic Gaussian Process (GP) regression-based framework to detect disturbance changes and learn and adapt the system performance at runtime to guarantee safety (i.e., no collision) and liveness (i.e., completion of the task) conditions. To better drive the explanation of our technique, we tackle a UAV pickup and delivery problem (e.g.,package delivery) in which the payload is treated as a disturbance since the system might be required to carry different unknown payloads during real-world applications. As the user may not know the system dynamics and may not have access to the controller especially when off-the-shelf commercial vehicles are used, changes in the payload will affect the system behavior and cause the system to deviate from its desired behavior, and potentially lead to unsafe states. For this reason, the system needs to estimate the disturbance and its effects and adapt its plan accordingly and quickly at runtime. We propose to use GP regression built on offline training data to estimate the disturbance and deviations at runtime. As the training could be limited (not complete or shifting), the system may not behave as expected at runtime when the conditions are outside of the training boundaries. To deal with these situations, we consider an event-triggered runtime recovery approach and propose a method to update the trained GP model with the new information gained at runtime to improve system performance and safety. To summarize, we aim to solve the following challenges:

- *how to estimate, recover, update, and adapt the decision of a learning enabled component online, using*

data obtained at runtime, while guaranteeing the autonomous system safety and improving its future performance.

5.2 Problem Formulation

We are interested in finding an online adaptive policy for safe UAV navigation under unforeseen disturbances. A general UAV with state $\boldsymbol{q}$ can be modeled as a non-linear system of the form $\dot{\boldsymbol{q}} = f(\boldsymbol{q}, \boldsymbol{u})$. Typical control techniques like PID controllers are designed to control $\boldsymbol{u}$ to minimize tracking error $\boldsymbol{e} = \boldsymbol{q}_{des} - \boldsymbol{q}$ for the given system dynamics. If a disturbance $\boldsymbol{d}$ is present and unknown, the system dynamics change: $\dot{\boldsymbol{q}} = \tilde{f}(\boldsymbol{q}, \boldsymbol{u}, \boldsymbol{d})$. If the controller is designed under no disturbance assumption, or to work under different disturbance boundaries, the tracking performance of the system will be deteriorated leading to possible unsafe states (e.g., a collision). By changing the motion planning of the system, however, it could be possible to perform safe operation under limited performance (e.g., reduced speed).

This problem is formally defined as follows:

Problem 1: ***Runtime Fast Adaptive Safe Planning:*** Given a UAV tasked to navigate in a cluttered environment, under the effect of an unknown disturbance $\boldsymbol{d}$, find an online policy to quickly adapt its motion plan, and constantly update the prediction to satisfy the following safety constraint:

- *Safety Constraint:* The UAV must avoid collision with the obstacles in the environment during its motion:

$$\|\boldsymbol{p}(t) - \boldsymbol{o}_i\| \geq \boldsymbol{\delta} \quad \forall t \in [t_0, T], \forall i \in \{1, \ldots, n_o\} \tag{5.1}$$

  where $\boldsymbol{p}(t) = [x, y, z] \subset \boldsymbol{q}$ is the 3D position of the vehicle at time t. Similarly, $\boldsymbol{o}_i$ is the 3D position of the i^{th} obstacle, with n_o the number of obstacles in the environment and $\boldsymbol{\delta}$ is the minimum safe distance to the obstacle considering the dimensions of the quadrotor.

In order to solve Problem 1, we leverage GP-based theory to predict the future states of the system, which requires to identify the current configuration of the system, as formally defined in the following problem:

Problem 2: ***Model Disturbance Estimation:*** Given a system with a pre-trained learning enabled component to estimate future states of the system, find a policy to quickly identify the disturbance in the model at runtime which causes the system to deviate from its nominal behavior.

Payload Disturbance Estimation Case Study and Assumptions: While our proposed framework is general for different types of systems and runtime disturbances, for the sake of clarity in this paper we

will focus primarily on payload pick-up/drop-off UAV operations in which the disturbance is the unknown payload lifted by the vehicle at runtime: a UAV is tasked to lift and carry an a priori unknown load to a goal while avoiding obstacles in a cluttered environment. The low-level controller of the UAV is considered as a black box while within the high level planner different waypoints as well as the velocity of the UAV can be changed depending on the mission. We assume that the vehicle will minimize its mission time and will try to travel to the goal with the highest speed possible. However, moving at high speed may make the system deviate from its desired behavior in particular in the z level due to the system dynamics and limited thrust as shown in Figure 5.2, whereas reducing the speed can help the system to perform better tracking. Hence, the primary goal is to find a policy to obtain the highest average speed for a UAV to avoid vertical obstacles while carrying an unknown payload. From Problem 1, the safety constraint to respect is thus:

$$|z(t) - z_i^o| \geq \delta \quad \forall t \in [t_0, T], \forall i \in \{1, \ldots, n_o\} \tag{5.2}$$

We consider only z direction for the safety in this case study.

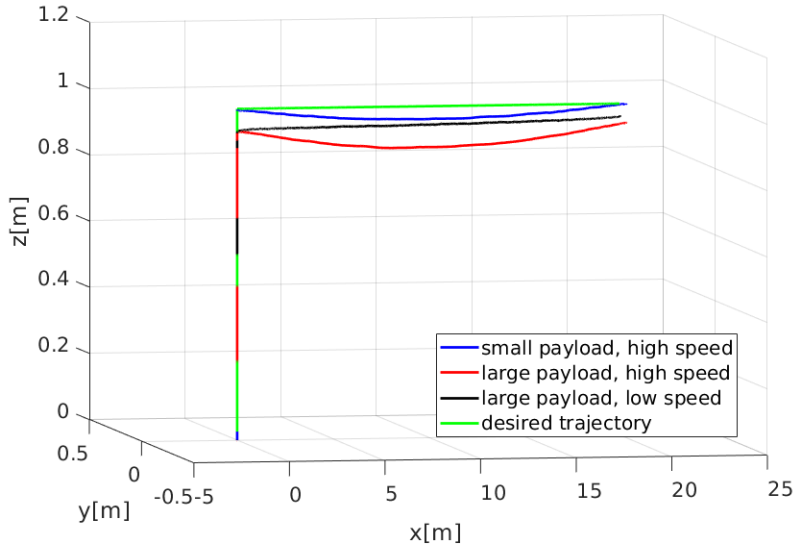


Figure 5.2: System behavior comparison with different payload disturbances and speeds. To minimize deviations while carrying a large load it is necessary to reduce speed. 2020 ©IEEE

5.3 Gaussian Process-based Safe Planning, Recovery and Adaptation

In this section, we describe our framework for safe planning, recovery, and adaptation in UAV navigation operations under unknown payload disturbances. Our approach consists of offline and online stages, as shown in Figure 5.3. During the offline stage, a set of trajectories consisting of different lengths, goals, and average speeds are run on the UAV with a bounded payload disturbance. For each trajectory, the maximum deviation from the desired z level, and the time taken by the UAV to take off from the ground are recorded.

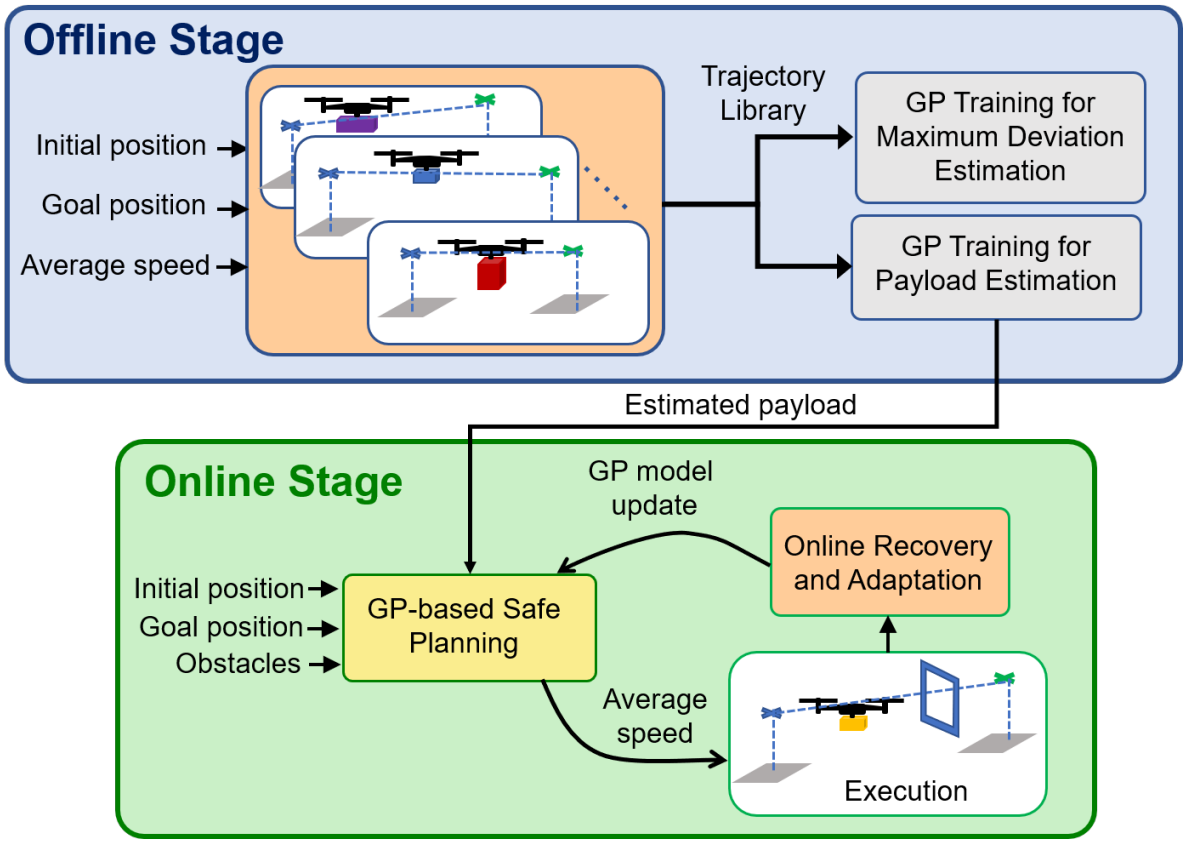


Figure 5.3: Architecture of the proposed approach. 2020 ©IEEE

A GP-based regression is then trained and considered at runtime to estimate the maximum deviation from the desired trajectory and in turns compute the maximum average speed to maintain during the operation and avoid obstacles along the way. We choose to use GP regression to solve these problems because of its ability to provide confidence intervals which makes it easier to handle noise and uncertainties as well as adapt its model at runtime. However, the GP regression may result in inaccurate estimations when the test conditions at runtime are outside of the training boundaries, or due to errors in payload estimation. In

such cases, a recovery procedure is triggered at runtime when an unsafe situation is detected by monitoring the system's position and velocity in z direction. The GP regression model is then updated with the data acquired at runtime to improve future decisions.

In the following section, we explain the proposed GP-based z deviation estimation approach.

5.3.1 Gaussian Process Regression for Deviation Estimation

In order to enable online prediction of deviations during an operation, a GP regression model is trained from a rich library of trajectories under different average velocities and payloads. We will refer to such GP model as GP^d to differentiate from the one for payload GP^p introduced in Section 5.3.3. Since the controller is treated as a black box and cannot be accessed and adapted according to the payload, the deviation from the desired height increases with the payload and average speed, as demonstrated in Figure 5.2. The maximum deviation from the desired height for an average speed $\bar{v}$ and for a payload mass m_p is extracted from the recorded trajectory as follows:

$$d_z^m(\bar{v}, m_p) = \max_{t\in[T_h, T_h+T_g]} \max_{\boldsymbol{p}_g \in \mathcal{P}_g} |z_{\tau(g,\bar{v})}(t) - z(t)| \tag{5.3}$$

where $z_{\tau(g,\bar{v})}(t)$ is the desired height of the trajectory with average speed $\bar{v}$, T_g is the duration of the trajectory to the goal location, $\boldsymbol{p}_g$ is the position of the goal g and $\mathcal{P}_g$ is the set of all goal positions used in training.

To estimate the maximum deviation from the desired height based on the estimated payload mass and the average speed of the planned trajectory, we leverage Gaussian Process regression theory. Training inputs for the GP^d regression consist of the payload mass and average speed values: $\boldsymbol{x} = [\boldsymbol{x}_{1,1} \cdots \boldsymbol{x}_{i,j} \cdots \boldsymbol{x}_{n_v,n_m}] \in \mathbb{R}^{(n_v n_m)\times 2}$ where $\boldsymbol{x}_{i,j} = [\bar{v}^i, m_p^j]^T, \forall i \in [1, \cdots, n_v], \forall j \in [1, \cdots, n_m]$, where n_v and n_m are the number of average speed and payload mass values in the training set. The output of the training is the maximum deviation value: $\boldsymbol{d_z} = [d_z^m(\boldsymbol{x}_{1,1}), \cdots, d_z^m(\boldsymbol{x}_{i,j}), \cdots, d_z^m(\boldsymbol{x}_{n_v,n_m})] \in \mathbb{R}^{(n_v n_m)}$. Given the training set $\mathcal{D} = [\boldsymbol{x}, \boldsymbol{d_z}]$, by the definition of GP, training and test outputs follow a joint Gaussian distribution:

$$\begin{bmatrix} \boldsymbol{d_z} \\ d_z^* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{x}) \\ \mu(\boldsymbol{x}^*) \end{bmatrix}, \begin{bmatrix} \boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I} & \boldsymbol{K}_* \\ \boldsymbol{K}_*^T & \boldsymbol{K}_{**} \end{bmatrix} \right) \tag{5.4}$$

where $\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{x}_{1,1}, \boldsymbol{x}_{1,1}) & \cdots & k(\boldsymbol{x}_{1,1}, \boldsymbol{x}_{n_v,n_m}) \\ \vdots & \vdots & \vdots \\ k(\boldsymbol{x}_{n_v,n_m}, \boldsymbol{x}_{1,1}) & \cdots & k(\boldsymbol{x}_{n_v,n_m}, \boldsymbol{x}_{n_v,n_m}) \end{bmatrix}$, $\boldsymbol{K}_* = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}^*) & \cdots & k(\boldsymbol{x}_{(n_v n_m)}, \boldsymbol{x}^*) \end{bmatrix} \in \mathbb{R}^{(n_v n_m)}$ and $\boldsymbol{K}_{**} = k(\boldsymbol{x}^*, \boldsymbol{x}^*) \in \mathbb{R}$. σ_ε^2 is the noise level associated with the observations, μ is the mean

function and k is the covariance function and $\boldsymbol{x}^* = [\bar{v}^*, m_p^*] \in \mathbb{R}^2$ is the test input. The widely used Matern kernel is considered here as a covariance function and the hyperparameters are set by optimizing the marginal likelihood [69]. The predictive posterior distribution of d_z^* is also a Gaussian distribution:

$$p(d_z^*|\boldsymbol{x}^*, \boldsymbol{x}, \boldsymbol{d_z}) \sim N(\mu_*, \sigma_*) \tag{5.5}$$

where μ_* and σ_* are calculated as follows:

$$\mu_* = \mu(\boldsymbol{x}^*) + \boldsymbol{K}_*^T(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I})^{-1}(\boldsymbol{d_z} - \mu(\boldsymbol{x})) \tag{5.6}$$

$$\sigma_*^2 = \boldsymbol{K}_{**} - \boldsymbol{K}_*^T(\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I})^{-1}\boldsymbol{K}_* \tag{5.7}$$

Finally, the maximum deviation for a test input $\boldsymbol{x}^*$ is estimated by using the upper limit of the 95% confidence interval which is computed by:

$$\bar{d_z}^*(\bar{v}^*, m_p^*) = \mu^* + 2\sigma^* \tag{5.8}$$

with μ_* and σ_* obtained from (5.6) and (5.7) respectively.

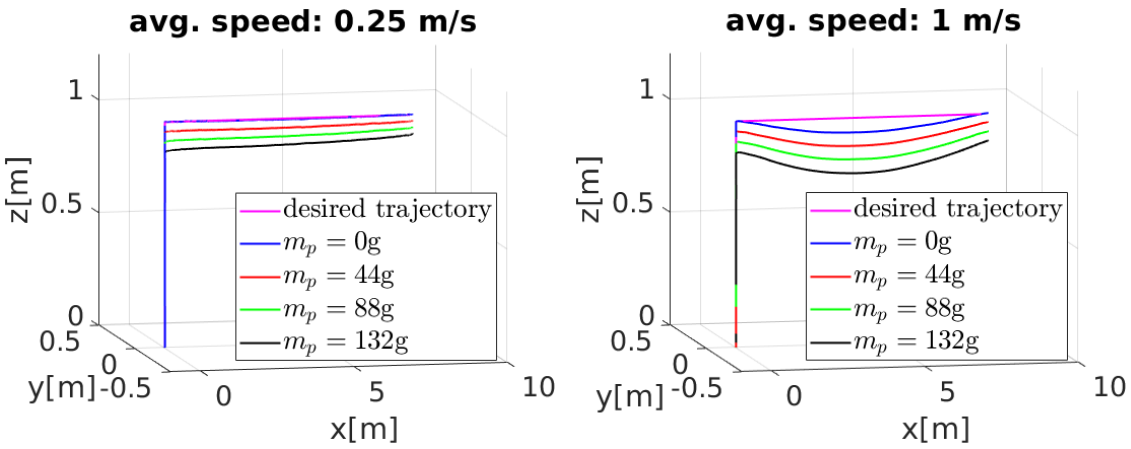
To create a training set, given $m_q = 176$g the mass of the UAV used in the simulations, the trajectories are executed with four different payload masses $m_p = 0 \times m_q = 0$g (blue curve), $m_p = 0.25 \times m_q = 44$g (red curve), $m_p = 0.5 \times m_q = 88$g (green curve), $m_p = 0.75 \times m_q = 132$g (black curve) using a quadrotor model defined in [6]. The trajectory library includes trajectories with four different average speed values: $\bar{v} \in \{0.25, 0.50, 0.75, 1.00\}$m/s. Figure 5.4 shows two examples of such training trajectories with a quadrotor tasked to reach first a 1m height and then move in the $+x$ direction for 8m under the four loads and two of the speeds specified above: $\bar{v} = 0.25$m/s in Figure 5.4(a) and $\bar{v} = 1$m/s in Figure 5.4(b).

The trajectory library contains 200 different trajectories executed with the four different payload masses and Figure 5.5(a) shows the recorded maximum deviation values for each average speed and payload mass value in the training set. The results of the GP regression training is shown in Figure 5.5(b).

The estimated maximum deviation is then used at runtime to plan and adapt the average speed of the UAV trajectory to reach its desired goal fast without violating the safety constraints, as explained in the following section.

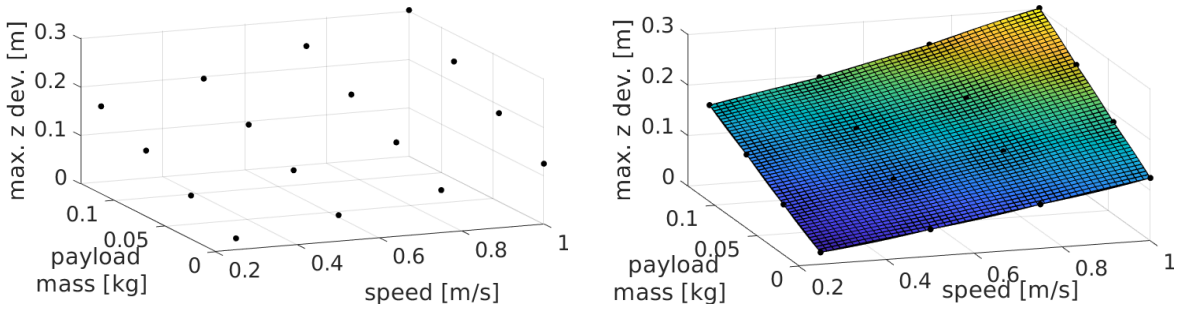
5.3.2 Fast, Runtime Speed Adaptation, Online Recovery and Learning

As noted in the previous section, a payload has two effects on the UAV: 1) it slows down its take-off procedure bringing the vehicle to a lower height from the desired height and 2) with larger speeds, it creates more



(a) (b)

Figure 5.4: Sample trajectories from the training library running with average speed of a) 0.25m/s, and b) 1.0m/s under four payload disturbances. 2020 ©IEEE



(a) Maximum z deviation as a function of payload and average speed.

(b) GP regression results for maximum deviation estimation obtained using GPML toolbox [70].

Figure 5.5: Offline training for maximum deviation estimation at runtime. 2020 ©IEEE

deviation from the desired trajectory while moving toward its goal. Thus, as soon as the UAV has reached a certain height before it starts moving toward the goal, it needs to quickly decide the average speed to maintain during the trajectory in order to reach the desired goal as fast as possible while maintaining safety at all times. To this end, the trained GP^d obtained in the previous section is used to determine the maximum speed of the UAV to safely navigate in the environment and avoid collisions, as follows:

$$\bar{v}^* = \max_{\bar{v} \in \mathcal{V}_s(\bar{m}_p)} \{\bar{v} | \bar{d}_z(\bar{v}, \bar{m}_p) < \zeta\} \tag{5.9}$$

where $\bar{m}_p$ is the estimation of the payload mass (discussed in the next section), $\mathcal{V}_s(\bar{m}_p)$ is the set of stable speeds for the given payload, and ζ is the maximum allowed deviation:

$$\zeta = \min_{i \in [0, n_o], t \in [T_h, T_h + T_g]} |z_\tau(t) - z^{o,i}| \tag{5.10}$$

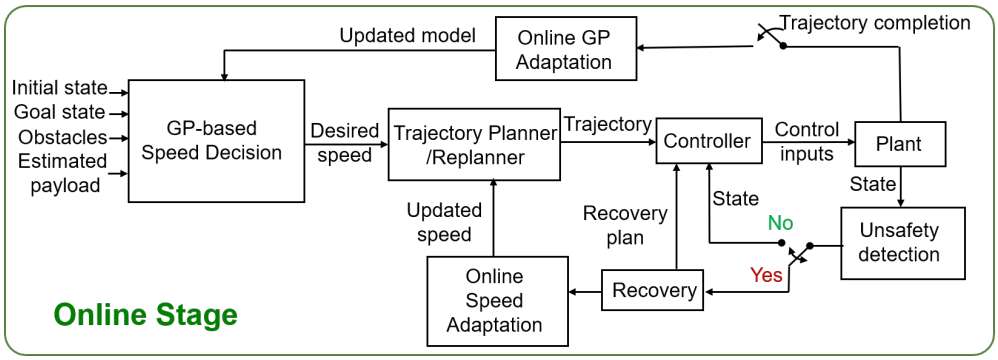


Figure 5.6: Architecture of the proposed runtime speed adaptation, online recovery and learning approach. 2020 ©IEEE

where z_τ is the desired height at time t and $z^{o,i}$ is the z position of the i^{th} obstacle located along the desired trajectory. Once the desired average speed is picked, a trajectory is planned to the desired goal location, and the system starts to move following the desired trajectory. However, because the training is limited, it does not cover all the possible runtime conditions. When the test input at runtime is outside of the training boundaries, the system may behave different than expected. For example, when the system carries a large payload and choose to travel with high speed, it may lose height and become unsafe or if the deviation estimation is inaccurate the vehicle may collide with the obstacles. To prevent such situations, we follow an online recovery and adaptation procedure which is summarized in Figure 5.6.

To detect inaccuracies in prediction, changes in the z velocity and position of the system are monitored at runtime: A high negative $\dot{z}$ is considered unsafe because it causes the system to lose height and become unsafe and a low z close to the clearance level is also considered unsafe because it can cause the system to collide with the obstacles. These situations trigger a recovery mode in which the vehicle stops its current go-to-goal task and switches to a hovering mode to bring the system to a certain height and decide a new speed. To guarantee that recovery is possible, the maximum payload and speed limits were tested offline.

If the recovery is performed because the z level of the UAV becomes lower than the allowed level to pass the obstacle, a new speed is chosen by considering a lower deviation threshold. If the recovery is performed due to high $-\dot{z}$, the detected unsafe speed is added to the training set and labeled unsafe.

The set of stable speeds is decided using Support Vector Machine (SVM) classification trained with the training input $\boldsymbol{x} \in \mathbb{R}^{(n_v n_m)\times 2}$ and training output $\boldsymbol{s} \in \mathbb{R}^{(n_v n_m)}$ which is initially set to $\boldsymbol{s} = \vec{\mathbf{1}}$ (i.e.,safe). Therefore the initial set of stable speeds consists of all admissible speed values.

$$\boldsymbol{x}_{svm} = [\boldsymbol{x}, [\bar{v}, \bar{m}_p]^T], \boldsymbol{s}_{svm} = [\boldsymbol{s}, 0] \tag{5.11}$$

After retraining the SVM with the updated dataset that includes both safe and unsafe trajectories, a new

speed value is picked using (5.9). If the selected speed is still detected to cause the system to deviate more then the permitted threshold, a recovery is triggered and a lower speed is considered. Once the system completes its trajectory, the estimated payload, final average speed and the maximum recorded deviation are added to the training set and used to retrain the GP online for more accurate decisions in future iterations.

$$\boldsymbol{x}^{gp} = [\boldsymbol{x}, [\bar{v}_f, \bar{m}_p]^T]], \boldsymbol{d}_z^{gp} = [\boldsymbol{d}_z, d_z(\bar{v}_f, \bar{m}_p)] \tag{5.12}$$

By following this online recovery procedure, the system is able to use the pre-trained learning enabled components to make decisions at runtime, even if the test conditions are different from the training conditions. Furthermore, the system is able to improve the decisions over time thanks to the online learning.

The drawback of such approach is that the computational complexity of the GP regression and SVM classification increases with the data size. However, this drawback can be overcome by keeping the training data size bounded. This can be achieved by removing the redundant data collected at runtime. In this work, the size of the data set used by the learning enabled components was within the range of fast training and estimation, therefore we left data pruning as a future work.

In order to be able to use the GP-based deviation estimation model explained in Section 5.3.1 and to update this model at runtime as explained in Section 5.3.2, the payload disturbance should be estimated because input to GP^d. We again leverage GP regression theory to estimate the payload and explain the details in the next section.

5.3.3 Gaussian Process Regression for Payload Estimation

The UAV is tasked to reach a desired height in a certain amount of time after picking up an object of unknown mass. In order to estimate the amount of payload m_p that the UAV is carrying, we leverage again GP regression based on a library of previously collected data. To train such GP, GP^p, the UAV is tasked to reach a desired height following a vertical trajectory for a fixed time duration under different payloads and measurement and process noises.

Since the internal controller is treated as a black box, due to the thrust-to-weight ratio properties inherent of UAVs, the increased mass of the vehicle will affect the time it takes to reach the desired height and thus there is a direct correlation between take-off time and weight.

During training, for each m_p value, the time which takes the system to pass a given z level is obtained from the collected data as follows:

$$t_h = \min_{t \in [0, T_h]} \{t | z(t) \geq z_h\} \tag{5.13}$$

where $z_h \geq 0$ is the user defined detection height to estimate the weight of the load and T_h is the vertical trajectory duration. These observations are recorded in the training set: $\boldsymbol{t_h} = [t_{h,1}, \cdots, t_{h,n}]$ and $\boldsymbol{m_p} = [m_{p,1}, \cdots, m_{p,n}]$ where n is the size of the training set. Given the training set of the collected observations of $\mathcal{M} = \{\boldsymbol{t_h}, \boldsymbol{m_p}\}$, the goal is to predict the payload mass for a new input t_h^* by drawing m_p^* from the posterior distribution $p(m_p^*|\mathcal{M})$. By definition of GP [69], the training set output $\boldsymbol{m_p}$ and the test output m_p^* are joint multivariate Gaussian distributed:

$$\begin{bmatrix} \boldsymbol{m_p} \\ m_p^* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(\boldsymbol{t_h}) \\ \mu(t_h^*) \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}^m + \sigma_\epsilon^2 \boldsymbol{I} & \boldsymbol{K}_*^m \\ (\boldsymbol{K}_*^m)T & \boldsymbol{K}_{**}^m \end{bmatrix} \right) \tag{5.14}$$

where σ_ϵ^2 is the noise level associated with the observations, μ is the mean function and k is the covariance function. $\boldsymbol{K}^m \in \mathbb{R}^{n \times n}$ has entries $\boldsymbol{K}_{(i,j)}^m = k(t_{h,i}, t_{h,j})$ for $i, j \in \{1, \cdots, n\}$, $\boldsymbol{K}_*^m = \begin{bmatrix} k(t_{h,1}, t_h^*) & \cdots & k(t_{h,n}, t_h^*) \end{bmatrix} \in \mathbb{R}^{n \times 1}$ and $\boldsymbol{K}_{**}^m = k(t_h^*, t_h^*) \in \mathbb{R}$ with a Matern kernel [69] as a covariance function. The predictive posterior distribution of m_p^* is also a Gaussian distribution:

$$p(m_p^*|t_h^*, \boldsymbol{t_h}, \boldsymbol{m_p}) \sim N(\mu_*, \sigma_*) \tag{5.15}$$

with the following mean and covariance:

$$\mu_*^m = \mu(t_h^*) + (\boldsymbol{K}_*^m)^T (\boldsymbol{K}^m + \sigma_\epsilon^2 \boldsymbol{I})^{-1} (\boldsymbol{m_p} - \mu(\boldsymbol{t_h})) \tag{5.16}$$

$$(\sigma_*^m)^2 = \boldsymbol{K}_{**}^m - \boldsymbol{K}_*^T (\boldsymbol{K}^m + \sigma_\epsilon^2 \boldsymbol{I})^{-1} \boldsymbol{K}_*^m \tag{5.17}$$

Once the vehicle lift an object from the ground at runtime, the payload mass is estimated based on the time it takes for it to reach z_h level using the upper limit of the confidence interval calculated as follows:

$$\bar{m}_p^* = \mu_*^m + 2\sigma_*^m \tag{5.18}$$

where μ_*^m and σ_*^m are calculated according to (5.16) and (5.17) respectively. This mass estimation is then used as one of the inputs for the GP^d presented in Section 5.3.1 to estimate the deviations from the desired height and make decisions about the optimal speed to maintain along a trajectory.

The GP^p regression model trained for payload estimation is shown in Figure 5.7. Training was executed under the same conditions used for training GP^d and with a detection height threshold set to $z_h = 0.05$m.

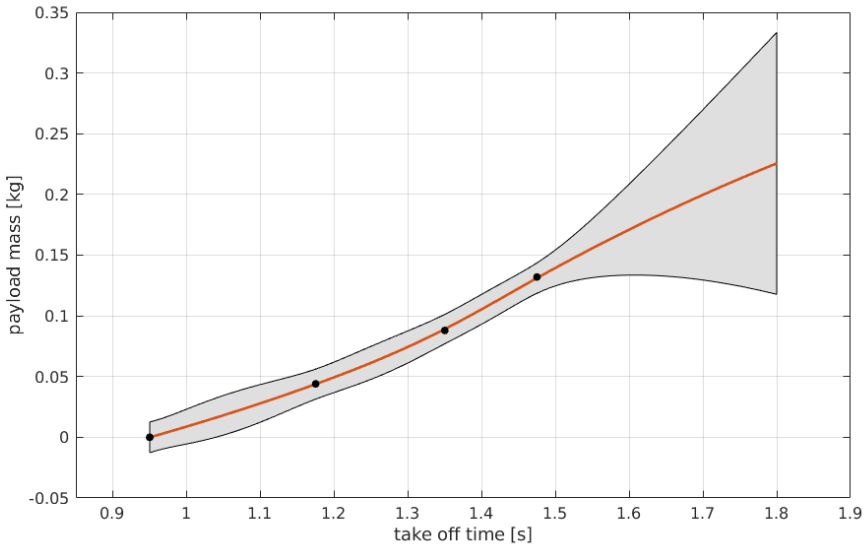
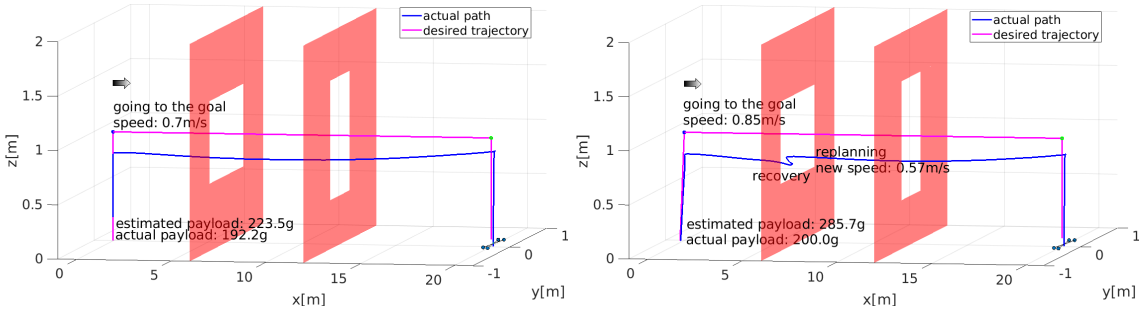


Figure 5.7: GP regression results for payload estimation based on the take-off time using GPML toolbox [70]. 2020 ©IEEE

5.4 Simulations

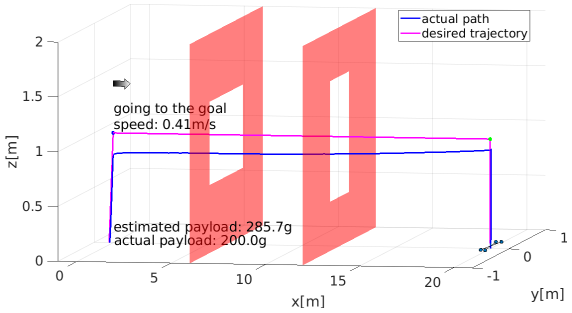
We validate the proposed online adaptation, recovery and learning approach with a quadrotor UAV pick-up and a drop-off case study: specifically, the vehicle is tasked to lift and carry objects with a priori unknown weights between two designated locations ($\boldsymbol{p}_0 = [0, 0, 0], \boldsymbol{p}_1 = [20, 0, 0]$) flying at $z = 1$m level through windows in the middle. We performed 100 simulations with random window positions and observed safe behavior of the system. In this section, we show one case with different mass values where there are two windows in the environment located at $(x_w = 6, z_w = 0.62)$m and $(x_w = 12, z_w = 0.45)$m shown in Figure 5.8 where z_w is the height of the obstacle to avoid and for the window shaped obstacle it is the height of the frame base. The quadrotor is required to pass through these windows to move between the designated locations.

When the vehicle picks up an object from the ground, the payload is estimated based on the time it takes to take off from the ground (i.e., to reach a height $z_h = 0.05$m) according to (5.18) using the pre-trained GP^p. After estimating the payload mass, the average speed to go to the goal is picked using (5.9), based on the GP^d. As the vehicle completes its trajectory, GP^d is updated with the data obtained at runtime. In Figure 5.8(a), we show the actual path (blue curve) and the desired trajectory (magenta curve) of the quadrotor which is tasked to carry a 192.2g payload. The vehicle estimates the payload as 223.5g and decides to move to its goal with $\bar{v} = 0.7$m/s based on the pre-trained GP^d shown in Figure 5.5(b). After successfully reaching the goal location, the quadrotor updates GP^d with the estimated payload mass and observed maximum deviation. The updated GP^d result is shown in Figure 5.9(a). The estimated payload mass and speed pair is also labeled



(a) 1st task with 192.2 g payload. The estimated payload is 223.5g.

(b) 3rd task with 200.0 g payload. The estimated payload is 285.7g.

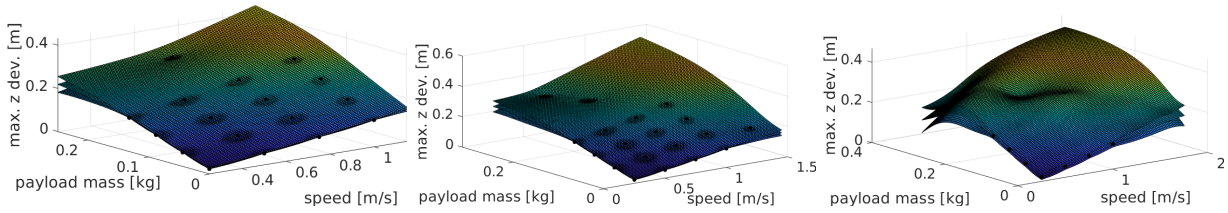


(c) 21st task with 200.0 g payload. The estimated payload is 285.7g.

Figure 5.8: Desired and actual trajectories of the quadrotor while it is performing a pick up/drop off task in an obstacle cluttered environment. The arrow shows the direction of the trajectory. 2020 ©IEEE

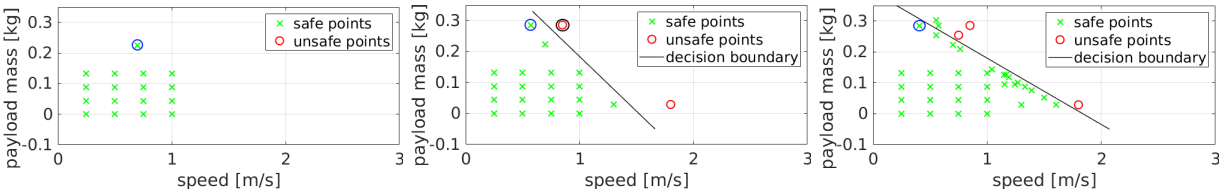
as safe and added to the safety data set shown in Figure 5.10(a) where the added point is marked with a blue circle. In Figure 5.8(b), we show the 3rd task where the quadrotor carries a 200g payload which is estimated as 285.7g and it decides to go to its goal with a speed $\bar{v} = 0.85$m/s. After the vehicle starts moving towards its goal, its velocity in the $-z$ direction increases and the inconsistency detector discussed in Section 5.3.2 triggers a recovery action. After completing the recovery action, the initial velocity and payload estimation pair is labeled as unsafe and added to the data set (marked with a black circle in Figure 5.10(b)) and the vehicle decides to move with average velocity $\bar{v} = 0.57$m/s which is picked according to the updated SVM safety decision model. After the UAV is able to complete its trajectory safely with the new average speed value, GP^d is updated with this speed value and estimated payload mass pair as shown in Figure 5.9(b).

In Figure 5.8(c), we show the 21st task where the quadrotor UAV carries the same load as in the third task. Thanks to the data acquired online and updated GP regression and SVM decision models, the vehicle decides to travel with average velocity 0.41m/s and is able to complete its trajectory without performing a recovery action. The safety decision model with all the data collected offline and online is shown in Figure 5.10(c). As the system obtains more data, it becomes able to make better decisions about the speed avoiding recovery operations. The GP^d after 21 tasks is presented in Figure 5.9(c) and because of the uncertainties in the data



(a) Updated GP estimation after the 1st task. (b) Updated GP estimation after the 3rd task. (c) Updated GP estimation after the 21st task.

Figure 5.9: Updated GP regression estimation models after each task with the data acquired at runtime. 2020 ©IEEE



(a) Updated safety decisions after the 1st task. (b) Updated safety decisions after the 3rd task. (c) Updated safety decisions after the 21st task.

Figure 5.10: Updated SVM safety decision models after each task with the data acquired at runtime. 2020 ©IEEE

collected online, the confidence interval of the GP regression becomes larger. Because this work considers the safety of the vehicle as a priority, both the deviation estimation and safety decisions are performed in a conservative way as demonstrated in this set of simulations. The vehicle is able to complete its tasks without going into unsafe states, and improve its decision making over time thanks to our online recovery and learning approach.

These demonstrated simulations were run on an Intel Core i9-9900K CPU at 3.60GHz taking 62.83ms on average to update GP^d and 0.97 ms on average to estimate the maximum deviation estimations using the trained GP regression model. SVM decision model updates took 4.82ms on average. These results demonstrate that the proposed GP based approach is suitable for online applications.

5.5 Experiments

Experiments were performed on an AscTec Hummingbird quadrotor UAV tasked to fly with different payload disturbances and through a window located at varying distances from the robot's initial position. A Vicon motion capture system was used to monitor the state of the quadrotor. The control commands to the quadrotor are communicated through the Robot Operating System (ROS). We used Matlab GPML Toolbox [70] to perform GP regression and the interface between Matlab and ROS was established through the Matlab ROS Toolbox.

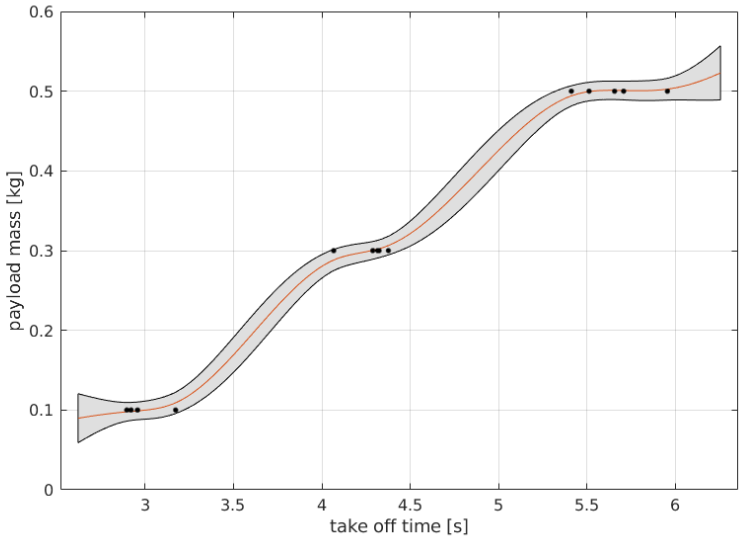
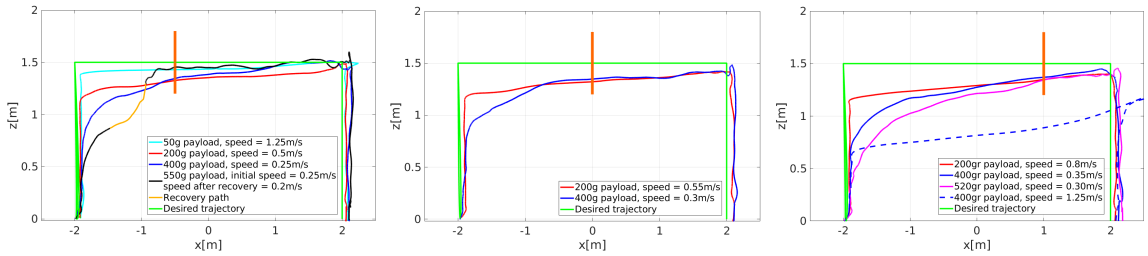


Figure 5.11: GP regression results for payload estimation based on the take-off time for the experiment test bed. 2020 ©IEEE

The training data for payload and deviation estimation for GP regression were collected by running a trajectory with five different average speed values: $\mathcal{V} = \{0.25, 0.50, 0.75, 1.00, 1.25\}$m/s, and with three different payload disturbance: $\mathcal{M}_p = \{100, 300, 500\}$g. In order to perform payload estimation, the take-off time of the trajectories in the training set is measured using (5.13) for $z_h = 0.2$m. GP^p results for payload estimation are shown in Figure 5.11. As the purpose of the quadrotor is to reach its goal location as quickly as possible, it does not wait until it reaches a certain height to create a trajectory to the goal. At the online stage, the system trains a GP^d using the data collected at the offline stage to estimate the deviation at a position where the system is required to pass through a narrow window. After the system starts moving vertically, the payload disturbance is estimated using (5.18) based on the observed take-off time and the system plans a trajectory with an average speed decided using (5.9).

In Figure 5.12, we compare the actual path and speeds followed by the UAV with different payloads and positions of the window. In Figure 5.13(a), we demonstrate the experiment setup for a quadrotor starting its mission at $\boldsymbol{p}_0 = [-2, 0, 0]$m and tasked to reach the goal located at $\boldsymbol{p}_1 = [2, 0, 0]$m by flying at $z = 1.5$m height in an environment with a window shaped obstacle with 0.6m clearance. The overlapped sequence of snapshots for a quadrotor carrying 400g payload though a window located in three different locations: a) $x_w = -0.5$m b) $x_w = 0$m and in c) $x_w = 1$m at $z_w = 1.2$m are shown in Figure 5.13. The payloads tested in these experiments were not included in the training set.

In Figure 5.12(a), the window is positioned at $x_w = -0.5$m and the quadrotor is tasked to move with 50g, 200g, 400g and 550g payloads. When the payload is 50g (cyan curve), the quadrotor is able to reach



(a) Results for 50g, 200g, 400g, and 550g payloads and window located at $x_w = -0.5$m.

(b) Results with payloads 200g and 400g when the window is located at $x_w = 0.0$m.

(c) Results with payloads 200g, 400g and 520g when the window is located at $x_w = 1.0$m.

Figure 5.12: Experiment results with three different window positions and varying payloads. 2020 ©IEEE

(a) Results with average speed 0.25m/s to pass the window at $x_w = -0.5$m.

(b) Results with average speed 0.30m/s to pass the window at $x_w = 0.0$m.

(c) Results with average speed 0.35m/s to pass the window at $x_w = 1.0$m.

Figure 5.13: Overlapped sequence of snapshots for the quadrotor carrying a 400g payload. 2020 ©IEEE

Table 5.1: Experiment results.

	(a) x_w = -0.5m		(b) x_w = 0.0m		(c) x_w = 1.0m	
payload (g)	**payload est. (g)**	**speed (m/s)**	**payload est. (g)**	**speed (m/s)**	**payload est. (g)**	**speed (m/s)**
50	136.74	1.25	-	-	-	-
200	293.89	0.50	276.95	0.55	276.78	0.80
400	443.87	0.25	421.29	0.3	479.85	0.35
520	-	-	-	-	540.62	0.30

its goal position with $\bar{v}^* = 1.25$m/s which is the maximum permitted average speed value. For payload disturbance 200g (red curve) and 400g (blue curve), the system decides to go with average speed values $\bar{v}^* = 0.5$m/s and $\bar{v}^* = 0.25$m/s respectively successfully passing through the window without collision. With high payload values, decreasing the speed allows the system to follow its trajectory more closely than lower payload values over time. With 550g (black curve), the system cannot find an average speed value which keeps the deviation lower than the desired threshold and decides to go with minimum average speed of 0.25m/s in the beginning and performs a recovery (yellow curve) due to its low height before the obstacle. After recovery, a trajectory with average speed of 0.2m/s is generated and the quadrotor is able to pass the window without a collision. In Figure 5.12(b), we show the experiment results for the same window located at $x_w = 0$m. As

the deviation in z gets smaller over time, the quadrotor was able to move with higher speeds in this case. For payload disturbance 200g and 400g, the quadrotor picks average speed values $\bar{v}^* = 0.55$m/s and $\bar{v}^* = 0.30$m/s respectively. We did not repeat this experiment for payload disturbance 50g as the quadrotor was able to move with the maximum average speed in the previous case and hence moving the window closer to the goal can only improve the situation. Lastly, we moved the window to the $x_w = 1.0$m position and repeated first the tests with payload disturbances 200g, and 400g. The quadrotor was able to move with $\bar{v}^* = 0.80$m/s and $\bar{v}^* = 0.35$m/s respectively, which are higher than previous (a) and (b) cases, as expected. We also tested with a payload disturbance of 520g (magenta curve), which is higher than the largest payload in training set, and the quadrotor was able to move with 0.30m/s average speed and safely pass the window. With the dashed blue curve in Figure 5.12(c), we show the trajectory of the quadrotor moving with the maximum average speed value without using our approach and we demonstrate that the system would have crashed with the wall below the window as expected. These results are also summarized in Table 5.1.

5.6 Discussions

In this chapter, we have presented a fast runtime planing, recovery and learning framework for safe navigation of autonomous systems with unknown payload disturbance. We have leveraged Gaussian Process regression theory to estimate the payload disturbance and the deviation of the system from the desired behavior and to adapt the speed of the planned trajectory. The model used for prediction is also constantly updated at runtime with the data acquired online to improve the future decision making.

Limitations: The presented runtime planning, learning, and recovery technique requires a prediction mechanism to accurately detect when the designed plan leads the system to an unsafe situation. This prediction also needs to be early enough to allow the recovery planner to take over safely. Runtime disturbances can be outside of the training bounds; however, the system needs to be physically capable of operating under the runtime disturbance (e.g., runtime payload disturbance needs to be under the payload capacity of the UAV), and remain controllable. For safety, the GP regression model for payload disturbance estimation needs to be over-conservative.

Future Work: This framework introduced a way of incorporating the feedback at runtime into offline trained models to adapt the plans for the system in a safe way. In the future, to formally guarantee safety within this method, assurance needs to be provided during the recovery process. Additionally, different types of disturbances (such as wind disturbance) can be considered within this framework to adapt the plans accordingly.

The presented runtime planning and learning approach mainly focuses on the disturbances outside of the training set. Vehicles can also be subject to faults at runtime that can cause them to operate under degraded conditions and they need to be considered to improve performance and safety at runtime. In the next chapter, we tackle this problem and introduce our meta-learning-based planning technique.

Chapter 6

Meta-Learning-based Trajectory Tracking under Degradations

In this chapter, we present our meta-learning-based approach to improve the trajectory tracking performance of an unmanned aerial vehicle (UAV) under actuator faults and disturbances which have not been previously experienced. Our approach leverages meta-learning to train a model that is easily adaptable runtime to make accurate predictions about the system's future state. A runtime monitoring and validation technique is introduced to decide when the system needs to adapt its model by considering a data pruning procedure for efficient learning. Finally the reference trajectory is adapted based on future predictions by borrowing feedback control logic to make the system track the original and desired path without needing to access the system's controller. The proposed framework is applied and validated in both simulations and experiments on a faulty UAV navigation case study demonstrating a drastic increase in tracking performance. This work has been accepted for publication at 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) [91].

6.1 Introduction

When operating in real world, robotic systems can face component faults that usually occur without apriori knowledge, making them challenging to be considered during the design time. For this reason, such conditions can lead the system to operate under degraded conditions at runtime.

For example, let's consider a UAV depicted in Figure 6.1, tasked to track a pipeline for inspection purposes. An unexpected fault or disturbance can compromise the stability of the system, making it deviate from the

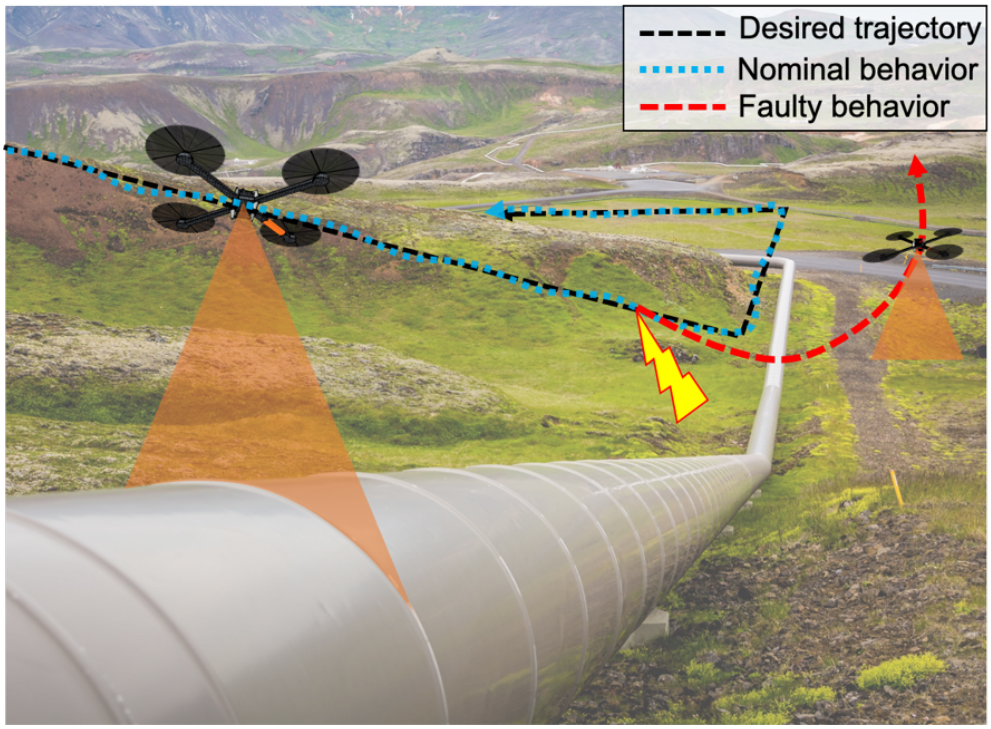


Figure 6.1: Pictorial representation of a UAV experiencing a failure at runtime causing task degradation like losing track of the pipe. 2021 ©IEEE

desired path possibly losing sight of the pipeline. Similarly, the same vehicle in a cluttered environment (e.g., heavy forested area) may end up deviating from a planned trajectory leading to possible unsafe situations like collisions.

To deal with such unforeseen and undesired situations, the system needs to quickly learn its model under new conditions at runtime and adapt its behavior accordingly. To adapt the system's behavior, one can act on the system's controller, however, it is not always possible to reconfigure the system's controller, especially when off-the-shelf commercial vehicles with black-box control architectures are used. Instead, we observe that the high-level planner which is typically in charge of providing references for the existing controller can be easily adapted.

In order to update the reference trajectory, the system's model under new condition needs to be learned at runtime to make predictions about its future states. However, learning complex autonomous system models usually requires leveraging either 1) computationally demanding system identification and adaptive control techniques which may not be fast enough to characterize the new model online or 2) advanced machine learning techniques which may be limited by the amount and quality of training data. Among learning enabled components (LECs), deep neural networks (DNNs) have been demonstrated to be very effective to model complex system dynamics [67]. However, they typically require a large amount of data and long

training time for accurate and reliable predictions. Meta-learning [20], on the other hand, is a recent method developed to "learn to learn" by leveraging optimization techniques to design machine learning models that can be adapted to the new tasks easily by using a few training examples. In this chapter, we introduce a meta-learning based framework to predict the future states of an autonomous system under an unknown degraded condition and we use these predictions within our reference trajectory planning framework. Inspired by classical feedback control theory we then propose to adapt the reference trajectory provided to the system to improve the trajectory tracking performance without accessing the controller.

The performance of the meta-learning system's model adaptation and prediction depends on the data collected at runtime. The initial data collected at runtime may not always be enough to represent the model as a whole. For this reason, we present a runtime monitoring and validation framework to decide when the model needs to be re-learned due to inaccurate predictions. Additionally, to keep the runtime learning data size limited, we introduce a data pruning approach for selecting the most representative data for the model.

In the literature, there are also works utilizing machine learning techniques such as deep neural networks to design trajectories that a UAV is expected to demonstrate good tracking performance [43]. [98] builds on this technique by improving the network training over time using active training trajectory generation based on the network prediction uncertainty. Even though these methods also propose trajectory update for improved tracking behavior, they are not designed for systems experiencing model changes between the design time and runtime; therefore, they are not suitable for systems experiencing component failures as presented in this chapter. In this work, we also adapt the reference trajectory to improve trajectory tracking, but we consider systems under disturbances and faults. [54] also uses meta-learning to improve tracking performance of systems with uncertain models. Meta-learning is utilized to adapt to model changes and estimate model uncertainty at runtime, and this online model is used within stochastic MPC for better control of the system. Different from our proposed approach, this work does not specifically consider system with faults, and assume to have an access to the control inputs.

Assumptions: Our framework assumes that the system does not have access to the controller or the control inputs, but it has access to the planner and updates the reference input. We assume the access to different faulty models for training purposes during the design time, and the runtime faults are also similar to the training faults.

6.2 Problem Formulation

In this chapter, we are interested in finding a technique to predict the future states of a system under unforeseen actuator faults, to validate and relearn the prediction model at runtime, and to adapt system's

reference trajectory to improve its trajectory tracking performance according to the predictions. These problems are formally defined as follows:

Problem 1: ***Future State Prediction under Failure*****:** A UAV with a nominal dynamical model $f(\boldsymbol{x}, \boldsymbol{u})$ as a function of its states $\boldsymbol{x}$ and controller inputs $\boldsymbol{u}$ has the objective of following a predefined desired trajectory $\boldsymbol{x}_\tau$. Under actuator failures and disturbances the system's model changes to $\boldsymbol{x}(k+1) = f'(\boldsymbol{x}(k), \boldsymbol{u}(k))$. In either case, control inputs are generated by a fixed controller $\boldsymbol{u}(k) = g(\boldsymbol{x}(k), \boldsymbol{x}_\tau(k+1))$. With these premises, find a policy to predict the future state of the system as a function of its current state and reference trajectory: $\tilde{\boldsymbol{x}}(k+1) = \tilde{f'}(\boldsymbol{x}(k), \boldsymbol{x}_\tau(k+1))$.

Problem 2: ***Runtime Monitoring:*** As the prediction model can vary or change over time, the data which are used to update the model in the beginning of the operation may become unable to represent the model. To overcome this problem, design a runtime monitor to decide when the learned model becomes inaccurate, and to select the online learning data to effectively relearn the model at runtime.

Problem 3: ***Reference Trajectory Update*****:** Once the faulty system model is learned and the future state of the system is predicted by solving Problem 1, find an online policy to dynamically update the reference trajectory $\tilde{\boldsymbol{x}}_\tau(k+1)$ input to the system such that the deviation from the original desired trajectory ($d = \|\boldsymbol{x} - \boldsymbol{x}_\tau\|$) is minimized.

6.3 Trajectory Tracking Improvement Using Meta-Learning

Our framework to improve the tracking performance of a system with actuator faults consists of offline and online stages, as shown in Figure 6.2. During the offline stage, UAVs facing various actuator faults are tasked to follow a set of trajectories at different speeds. For each actuator fault, the state of the system and the desired trajectory are recorded in the offline dataset, and a model for future position prediction is trained with this dataset using meta-learning. We use MAML [20] as the meta-learning approach due to its ability to train neural networks easy and fast to fine-tune with a small amount of data.

At runtime, a UAV with a different failure than the ones used during the offline stage is tasked to follow a trajectory towards a goal. Once the vehicle starts its operation and collects a certain amount of data, the meta-trained model is fine-tuned according to the new observed data. The fine-tuned neural network is then used to predict the system's future state according to its reference trajectory. Our trajectory update approach adaptively updates the reference trajectory according to the predicted deviation from the desired path. This approach allows us to trick the system into trying to follow a reference trajectory different than the actual desired trajectory; however, it, in fact, makes the UAV follow its original desired trajectory with a smaller deviation. With the proposed runtime monitoring approach, the meta-learned model is constantly

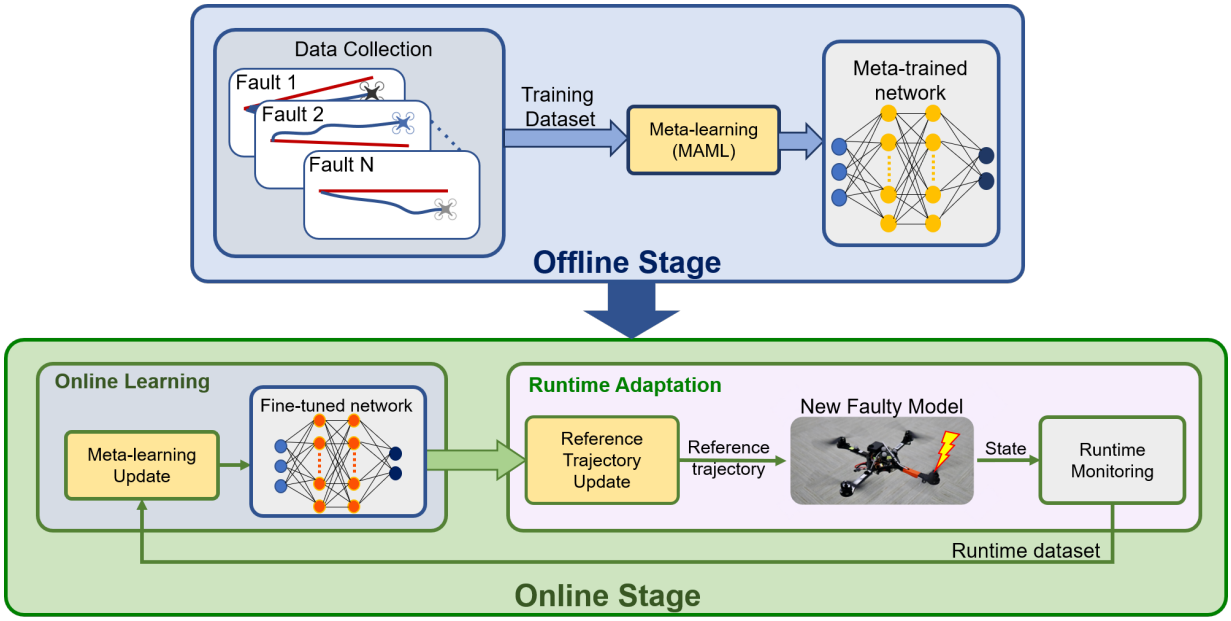


Figure 6.2: Meta-learning-based framework for trajectory tracking recovery under unknown faults/disturbances. 2021 ©IEEE

validated and updated when necessary. In the next sections, we will explain each component of our approach more in details.

6.3.1 MAML for State Prediction under Degraded Conditions

During the offline stage, we create a dataset by collecting data from a UAV with an actuator fault/disturbance from a discrete fault set $\mathcal{F}$ following different trajectories. This fault set also contains the case in which a UAV without a fault follows the same trajectories. At runtime, we aim to learn a model, denoted as $\bar{f}'$ which takes as input the desired position and velocity with respect to the current position and velocity, and outputs the system's next position with respect to its current position. A training input $\boldsymbol{x}^i_{\text{DNN}} \in \mathbb{R}^6$ and a training output $\boldsymbol{y}^i_{\text{DNN}} \in \mathbb{R}^3$ for a UAV with the fault $\mathcal{F}_i \subset \mathcal{F}$ are calculated as follows:

$$\begin{aligned} \boldsymbol{x}^i_{\text{DNN}} &= \begin{bmatrix} \boldsymbol{p}_{\tau_j}(k+1) \\ \boldsymbol{v}_{\tau_j}(k+1) \end{bmatrix} - \begin{bmatrix} \boldsymbol{p}^i(k) \\ \boldsymbol{v}^i(k) \end{bmatrix} \\ \boldsymbol{y}^i_{\text{DNN}} &= \boldsymbol{p}^i(k+1) - \boldsymbol{p}^i(k) \\ &\forall j \in 1, \ldots, N, \forall k \in 1, \ldots, T(\tau_j) \end{aligned} \tag{6.1}$$

where N is the number of training trajectories, $\boldsymbol{p}^i$ and $\boldsymbol{v}^i$ are the position and velocity of the UAV under fault i respectively and $T(\tau_j)$ is the duration of the trajectory τ_j in unit time steps. The dataset $\mathcal{D}_i$ for the

fault $\mathcal{F}_i$ contains the training input matrix $\boldsymbol{X}^i_{\text{DNN}} \in \mathbb{R}^{6\times M_i}$ and $\boldsymbol{Y}^i_{\text{DNN}} \in \mathbb{R}^{3\times M_i}$ with the columns $\boldsymbol{x}^i_{\text{DNN}}$ and $\boldsymbol{y}^i_{\text{DNN}}$ respectively. M_i is the number of data samples for fault $\mathcal{F}_i$. The training dataset for meta-learning contains the data from each failure: $\mathcal{D}_i \subset \mathcal{D}$. It should be noted that we used position and velocity as training inputs, however, our approach is independent of this choice and higher dimensional inputs can also be used depending on the complexity of the behavior and the prediction.

The purpose of meta-learning is to train a model which is adaptable for different tasks at runtime. As a review, MAML considers a model represented by a parametrized function $\bar{f}'_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$. For ease of notation, for the rest of the paper, we will use $f_{\boldsymbol{\theta}}$ to indicate the meta-learning model. During the offline meta-training, the model parameters $\boldsymbol{\theta}$ are meta-optimized according to Algorithm 1 in [20]. In summary, $\boldsymbol{\theta}$ is initialized randomly and updated to $\boldsymbol{\theta}^{'}_i$ while adapting to fault $\mathcal{F}_i$:

$$\boldsymbol{\theta}^{'}_i = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{F}_i}(f_{\boldsymbol{\theta}}) \tag{6.2}$$

Meta-optimization across the tasks then updates the model parameters according to Equation 1 in [20]:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \sum_{\mathcal{F}_i \subset \mathcal{F}} \mathcal{L}_{\mathcal{F}_i}(f_{\boldsymbol{\theta}^{'}_i}) \tag{6.3}$$

where α and β are hyperparameters for optimization step size and $\mathcal{L}$ is the loss function. As our problem is a supervised learning problem, the loss function is given as follows:

$$\mathcal{L}_{\mathcal{F}_i}(f_{\boldsymbol{\phi}}) = \sum_{\boldsymbol{x}^i_{\text{DNN}}, \boldsymbol{y}^i_{\text{DNN}} \in \mathcal{D}_i} \| f_{\boldsymbol{\phi}}(\boldsymbol{x}^i_{\text{DNN}}) - \boldsymbol{y}^i_{\text{DNN}} \|_2^2 \tag{6.4}$$

where $\boldsymbol{x}^i_{\text{DNN}}$ and $\boldsymbol{y}^i_{\text{DNN}}$ are given in (6.1).

6.3.2 Online Meta-Network Update

At runtime, as the UAV experiences a fault outside of the training fault set, it collects data from its position and velocity sensors upon starting its operation and adapts its offline meta-learned model. The system is able to quickly collect enough data to adapt its model as the model parameters are optimized for easy adaptation with meta-learning. Initially, the vehicle collects K consecutive data and constructs an online learning dataset with input $\boldsymbol{X}^*_{\text{DNN}} \in \mathbb{R}^{6\times K}$ and output $\boldsymbol{Y}^*_{\text{DNN}} \in \mathbb{R}^{3\times K}$. The input and output sample data from the dataset

are calculated as follows:

$$\boldsymbol{x}^*_{\text{DNN}}(k) = \begin{bmatrix} \boldsymbol{p}^*_\tau(k+1) \\ \boldsymbol{v}^*_\tau(k+1) \end{bmatrix} - \begin{bmatrix} \boldsymbol{p}^*(k) \\ \boldsymbol{v}^*(k) \end{bmatrix}$$
$$\boldsymbol{y}^*_{\text{DNN}}(k) = \boldsymbol{p}^*(k+1) - \boldsymbol{p}^*(k) \tag{6.5}$$

where $k \in 1, \ldots, K$, $\boldsymbol{p}^*$ and $\boldsymbol{v}^*$ are the position and velocity of the UAV under unknown fault $\mathcal{F}^*$ respectively and $\boldsymbol{p}^*_\tau$ and $\boldsymbol{v}^*_\tau$ are desired trajectory positions and velocities respectively. With this runtime dataset, the meta-trained model is adapted using gradient descent update and a fine-tuned model $\boldsymbol{f}_{\theta^*}$ with updated parameters $\boldsymbol{\theta}^*$ is obtained. The fine-tuned model is then used and re-adapted to update the reference trajectory as explained in the following section.

6.3.3 Runtime Reference Update, Monitoring, and Re-learning

After the initial adaptation of the meta-trained model, the system constantly updates its reference trajectory to improve trajectory tracking and performs runtime monitoring and relearning for model validation by following the architecture demonstrated in Figure 6.3. The reference trajectory is updated using the state predictions as explained next.

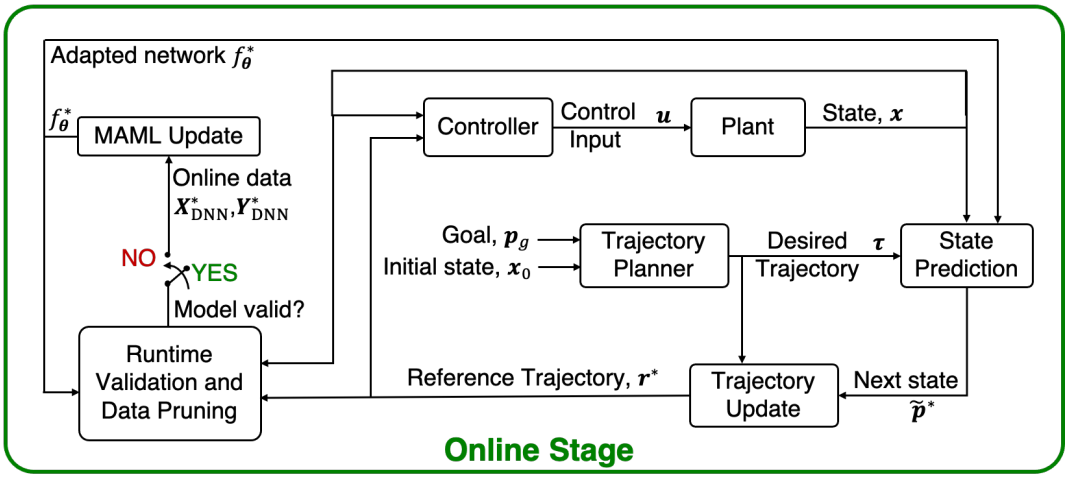


Figure 6.3: Architecture of the proposed runtime trajectory updating, runtime monitoring, and online learning approach. 2021 ©IEEE

Reference Trajectory Update using Meta-Trained Network

To explain the proposed reference trajectory update procedure, we use the pictorial representation in Figure 6.4(a) as a guideline. By using the online training data shown by black crosses for $K = 5$, the system

fine-tunes its meta-trained model as explained in the previous section and predicts its next position:

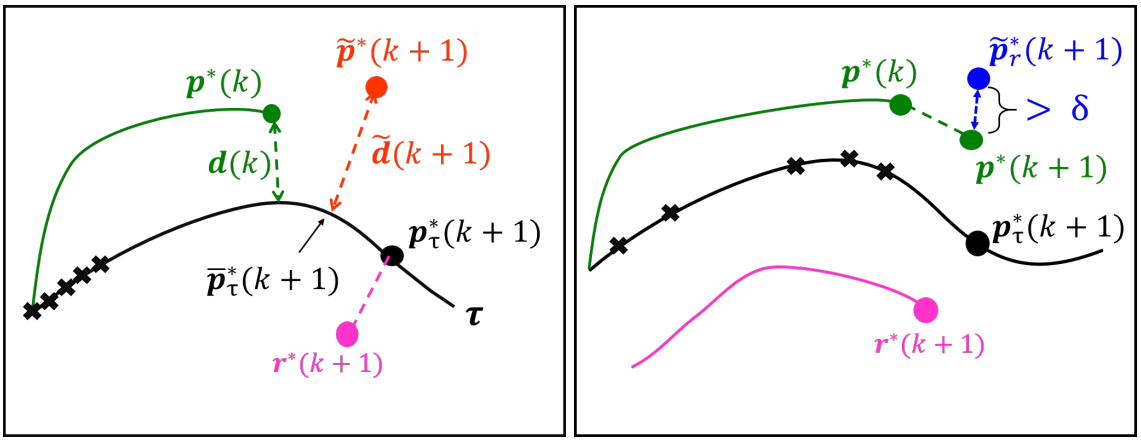
$$\tilde{\boldsymbol{p}}^*(k+1) = \boldsymbol{f}_{\theta^*}\left(\begin{bmatrix} \boldsymbol{p}^*_\tau(k+1) \\ \boldsymbol{v}^*_\tau(k+1) \end{bmatrix} - \begin{bmatrix} \boldsymbol{p}^*(k) \\ \boldsymbol{v}^*(k) \end{bmatrix}\right) + \boldsymbol{p}^*(k) \tag{6.6}$$

The next position prediction $\tilde{\boldsymbol{p}}^*(k+1)$ (shown by the orange dot in Figure 6.4(a)) is used to update the reference trajectory to compensate for the fault that the system is experiencing. By correcting the reference trajectory in the opposite side of the predicted deviation, the system is tricked to apply the necessary inputs to follow its desired path without a need for accessing the controller. The predicted deviation from the desired path is shown by orange dashed line in Figure 6.4(a) and is calculated as follows:

$$\tilde{\boldsymbol{d}}(k+1) = \bar{\boldsymbol{p}}^*_\tau(k+1) - \tilde{\boldsymbol{p}}^*(k+1) \tag{6.7}$$

where $\tilde{\boldsymbol{p}}^*(k+1)$ is the position prediction calculated in (6.6) and $\bar{\boldsymbol{p}}^*_\tau(k+1)$ is the closest point on the trajectory τ^* to the predicted position:

$$\bar{\boldsymbol{p}}^*_\tau(k+1) = \underset{\boldsymbol{p}^*_\tau(t), t\in\{0,\dots,T(\tau_j)\}}{\arg\min} \|\tilde{\boldsymbol{p}}^*(k+1) - \boldsymbol{p}^*_\tau(t)\| \tag{6.8}$$



(a) Trajectory update procedure. (b) Runtime validation and relearning procedure.

Figure 6.4: Pictorial representations of the proposed trajectory update and online learning methods. 2021 ©IEEE

To compensate for the fault, the reference trajectory position (shown by magenta dot in Figure 6.4(a)) is updated based on the original desired trajectory:

$$\boldsymbol{r}^*(k+1) = \boldsymbol{p}^*_\tau(k+1) + \boldsymbol{c}(k+1) \tag{6.9}$$

where $\boldsymbol{c}(k+1)$ is the trajectory update vector calculated based on the history of observed deviations and the predicted deviation using a PID-based rule:

$$\boldsymbol{c}(k+1) = \kappa_p \tilde{\boldsymbol{d}}(k+1) + \kappa_d(\tilde{\boldsymbol{d}}(k+1) - \boldsymbol{d}(k)) + \kappa_i \left(\sum_{t=0}^{t=k} \boldsymbol{d}(t) + \tilde{\boldsymbol{d}}(k+1) \right) \tag{6.10}$$

where $\boldsymbol{d}(k)$ (green dashed line in Figure 6.4(a)) is the system's deviation from the desired path at time k and κ_p, κ_d and κ_i are positive proportional, derivative and integral gains respectively. The reference velocity to the system is also updated accordingly:

$$\boldsymbol{r}_v^*(k+1) = \frac{\boldsymbol{r}^*(k+1) - \boldsymbol{r}^*(k)}{\Delta k} \tag{6.11}$$

With this reference updating strategy, the faulty system converges to its desired path over time. After the updated reference is applied to the controller as an input, the learned model is validated by comparing the model predictions with the actual position of the system as described in the next section.

Runtime Model Validation

When the system is operating under a previously unseen fault, an initially adapted meta-trained model may not be able to make accurate predictions if the system's behavior has a lot of variations over time. To prevent inaccurate predictions and their potential negative effects at runtime, the model is constantly validated by comparing its prediction with the actual state. The proposed runtime model validation scheme is pictorially presented in Figure 6.4(b). After updating the reference trajectory, the future position of the system $\tilde{\boldsymbol{p}}_r^*(k+1)$ (shown by a blue point), while it's following the new reference trajectory $\boldsymbol{r}^*$, is predicted using the fine-tuned model:

$$\tilde{\boldsymbol{p}}_r^*(k+1) = \boldsymbol{f}_{\theta^*} \left(\begin{bmatrix} \boldsymbol{r}^*(k+1) \\ \boldsymbol{r}_v^*(k+1) \end{bmatrix} - \begin{bmatrix} \boldsymbol{p}^*(k) \\ \boldsymbol{v}^*(k) \end{bmatrix} \right) + \boldsymbol{p}^*(k) \tag{6.12}$$

If the predicted future position differs from the actual position one step later more than a given threshold (i.e., if the deviation shown by blue dashed line in Figure 6.4(b) exceeds a certain threshold), the learned model is invalidated:

$$s(k+1) = \begin{cases} 0 & \text{if } \|\tilde{\boldsymbol{p}}_r^*(k+1) - \boldsymbol{p}^*(k+1)\| > \delta \\ 1 & \text{otherwise} \end{cases} \tag{6.13}$$

where $s(k+1)$ is a binary variable that denotes the validity of the learned model $f_{\boldsymbol{\theta}}^*$ and δ is a user-defined threshold for prediction deviation. If the model is invalidated ($s = 0$), online re-learning is triggered and the model is re-adapted using new online data selected as explained next.

Data Pruning and Online Relearning

To quickly update the meta-learned model with the runtime data, the number of online training samples needs to be kept bounded. However, selecting data which are not representative of the system may cause poor learning performance, and lead to unnecessary relearning operations. To prevent this and relearn the model with the most representative data, we use k-means clustering [49].

At time $k+1$ (for $k > K$), the system collects data from the history of the relative reference positions and velocities:

$$\boldsymbol{X}_* = \begin{bmatrix} \boldsymbol{r}^*(2:k+1) \\ \boldsymbol{r}_v^*(2:k+1) \end{bmatrix} - \begin{bmatrix} \boldsymbol{p}^*(1:k) \\ \boldsymbol{v}^*(1:k) \end{bmatrix}$$
$$\boldsymbol{Y}_* = \boldsymbol{p}^*(2:k+1) - \boldsymbol{p}^*(1:k) \tag{6.14}$$

where $\boldsymbol{X}_* \in \mathbb{R}^{6\times k}$ is the history of potential inputs to the model learning and $\boldsymbol{Y}_* \in \mathbb{R}^{3\times k}$ is the history of potential outputs. The history of inputs are clustered into K clusters and the centroid of each cluster j is obtained: $\boldsymbol{C}_j \in \mathbb{R}^6, \forall j \in \{1,\dots,K\}$. For online training, the closest point to each centroid in the input data history is selected as a training input:

$$\boldsymbol{X}^*_{\text{DNN}}(j) = \underset{\boldsymbol{X}_*(n), n\in\{1,\dots,k\}}{\arg\min} \|\boldsymbol{X}_*(n) - \boldsymbol{C}_j\| \ \forall j \in \{1,\dots,K\} \tag{6.15}$$

The model is re-learned as explained in Section 6.3.2 by using the pruned input data $\boldsymbol{X}^*_{\text{DNN}} \in \mathbb{R}^{6\times K}$ and the corresponding output data $\boldsymbol{Y}^*_{\text{DNN}} \in \mathbb{R}^{3\times K}$. As pictorially shown in Figure 6.4(b), with the proposed re-learning procedure the system picks new learning data from its history which are more representative and sparse than the initial data used. By using this data pruning and online learning approach, more diverse training data are chosen online, and redundant data are removed, leading to more efficient and effective online learning at runtime.

It should be noted that the complexity of the k-means clustering increases with the number of data points to be clustered, which can potentially cause runtime problems for longer trajectories. This issue can easily be solved by limiting the data size by removing older data – with the intuition that the significance of the previous data on the system's current behavior decreases over time.

6.4 Simulations

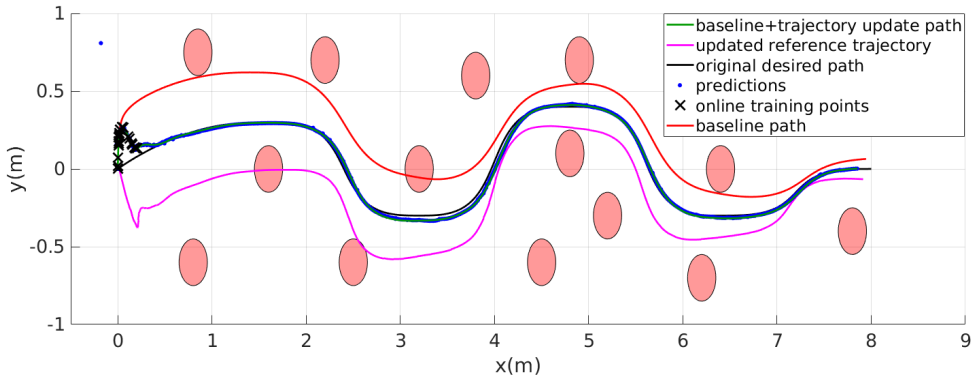
We validate the proposed meta-learning based reference trajectory adaptation, runtime monitoring, and re-learning approach with a quadrotor trajectory tracking case study. In this case study, we use a 12-dimensional quadrotor UAV model with a baseline PID controller for position and attitude control which is designed for the nominal model (i.e., with no faults) [56]. The actuator fault is simulated by reduced thrust on various propellers. During the offline stage, meta-training data are collected with a nominal UAV model and with UAV models under four different actuator faults given in Table 6.1. To obtain enough training data, minimum-jerk trajectories [55], to four different goal position with different initial and final speed values are generated, and a faulty UAV is tasked to follow these trajectories. We utilized a Tensorflow [84] Keras [35] implementation of MAML [20] for training and adapting the meta-learning. We train a neural network with two hidden layers with 40 nodes using the offline training data.

At runtime, the UAV is tasked to follow a path moving in an obstacle cluttered environment. In the case shown in Figure 6.5, the system is tasked to move to a goal at $\boldsymbol{p}_g = [8, 0, 1]$m from its initial position $\boldsymbol{p}_0 = [0, 0, 1]$m following an obstacle-free minimum-jerk trajectory. We consider an actuator fault between $\mathcal{F}_3$ and $\mathcal{F}_4$ which is not used during the training $\mathcal{F}_1^*$: 70% of the commanded thrust on propeller 2.

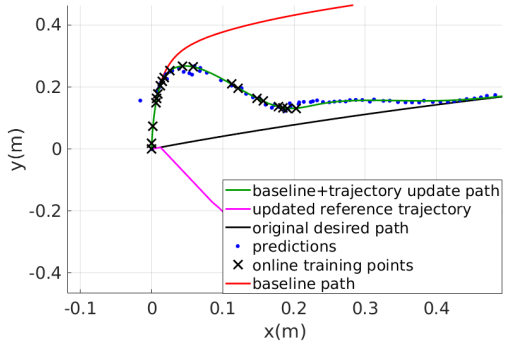
In Figure 6.5(a), the UAV follows the desired trajectory (black curve) moving between the obstacles (red semi-transparent circles). At the beginning of the operation, the UAV collects $K = 20$ training samples (black crosses) and updates its meta-trained neural network accordingly. Using the updated model, the UAV's next position is predicted and the reference trajectory inputted to the system is updated. The magenta curve shows the updated reference trajectory. After the trajectory update, the next position of the system tracking the updated reference is predicted (blue dots) and compared with the system's actual position (green curve) for validation and re-learning. In Figure 6.5(b), we show a zoomed-in version of the area marked with the box in Figure 6.5(a) to show the data more clearly. Using our approach, the system deviates much less than the case where it follows the original trajectory and collides with obstacles (red curve). In Figure 6.5(c), the deviation from the desired path using our reference update approach on top of the baseline controller

Table 6.1: Fault types used during simulations.

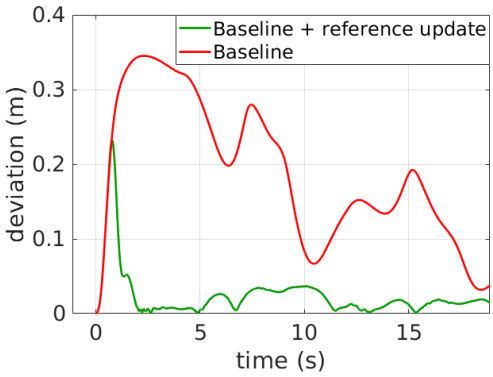
Training fault name	Fault type
$\mathcal{F}_1$	60% of the commanded thrust on propeller 1
$\mathcal{F}_2$	80% of the commanded thrust on propeller 1
$\mathcal{F}_3$	60% of the commanded thrust on propeller 2
$\mathcal{F}_4$	80% of the commanded thrust on propeller 2
Test fault name	**Fault type**
$\mathcal{F}_1^*$	70% of the commanded thrust on propeller 2
$\mathcal{F}_2^*$	60% of the commanded thrust on propeller 4



(a) Path of a UAV with a fault in propeller 2 following a slalom path in between obstacles.



(b) Zoomed in version of the area marked with the box in Figure 6.5(a).

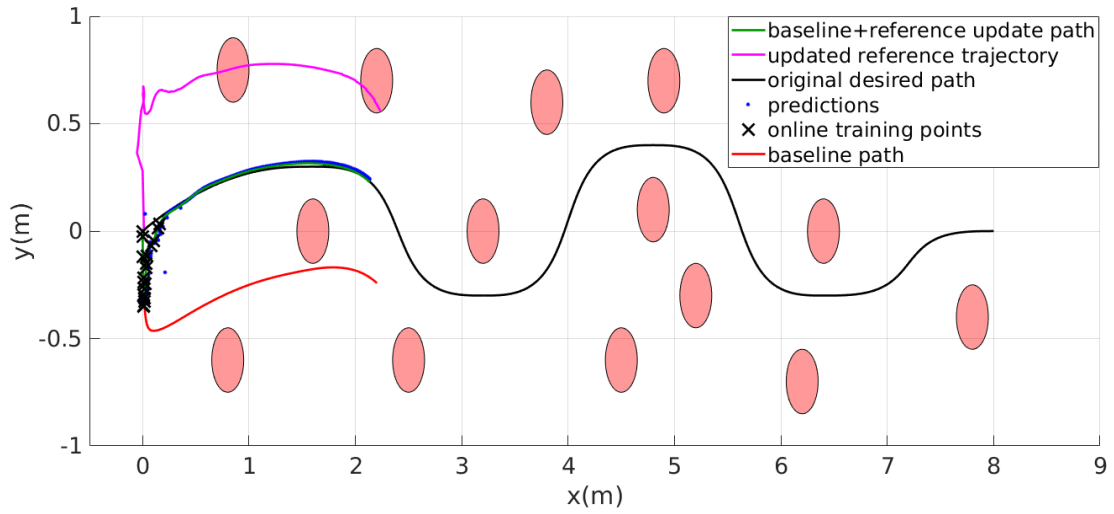


(c) Deviation over time with and without our approach for the case shown in Figure 6.5(a).

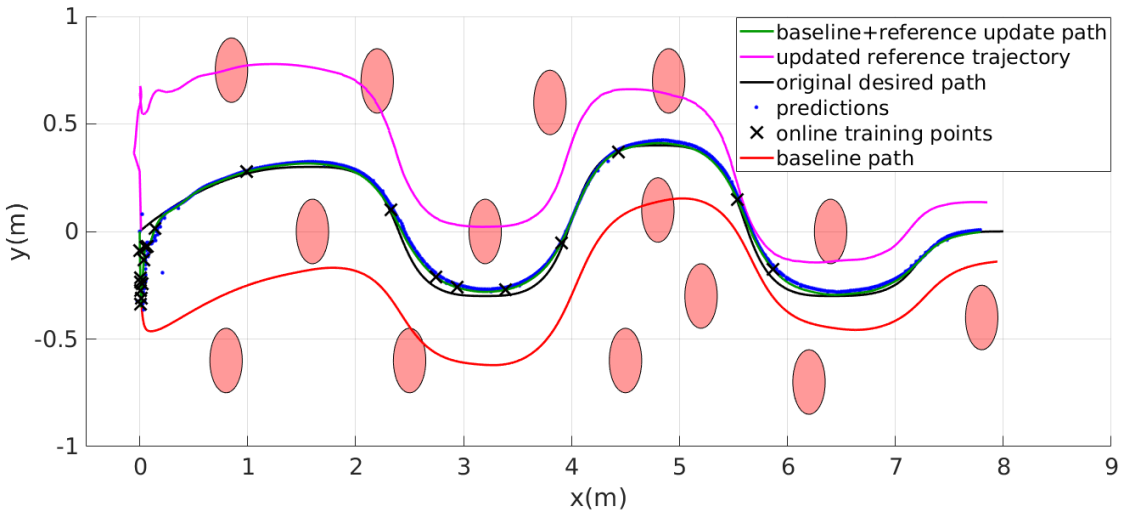
Figure 6.5: Simulation results for UAV with $\mathcal{F}_1^*$. 2021 ©IEEE

is compared with the case with the baseline controller. The average deviation from the desired trajectory is recorded as 18.17 cm with the baseline controller and with our approach, it is reduced to 2.24 cm. The system performs relearning operations to adapt its model nine times at the beginning of the operation.

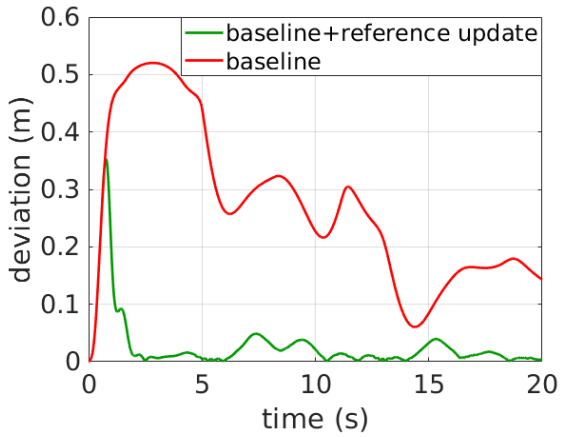
In the case shown in Figure 6.6, we consider a UAV with an actuator fault, which is outside of the training bounds $\mathcal{F}_2^*$: 60% of the commanded thrust on propeller 4. Similar to the previous case, the vehicle collects training data at the beginning of its operation and adapts its meta-trained model accordingly. Using the adapted model, it makes predictions about its future states and updates the reference trajectory, as shown in Figure 6.6(a). When the difference between the model predictions and the actual state exceeds the desired threshold $\delta = 0.02$m, the system prunes its history of observations, as explained in Section 6.3.3. In Figure 6.6(b), the vehicle's complete path is shown by the green curve, and the black crosses represent the position data of the vehicle used for the last model re-learning. The system adapts its meta-trained model 122 times. The deviation from the desired path with proposed approach is compared to the baseline controller in Figure 6.6(c). The average deviation from the desired path is recorded as 2.61 cm with our approach as opposed to 27.05 cm with baseline controller without using any reference update.



(a) UAV with fault 2 in the middle of its operation.



(b) The remaining part of the UAV operation highlighting different training points than in (a) for meta-learning.

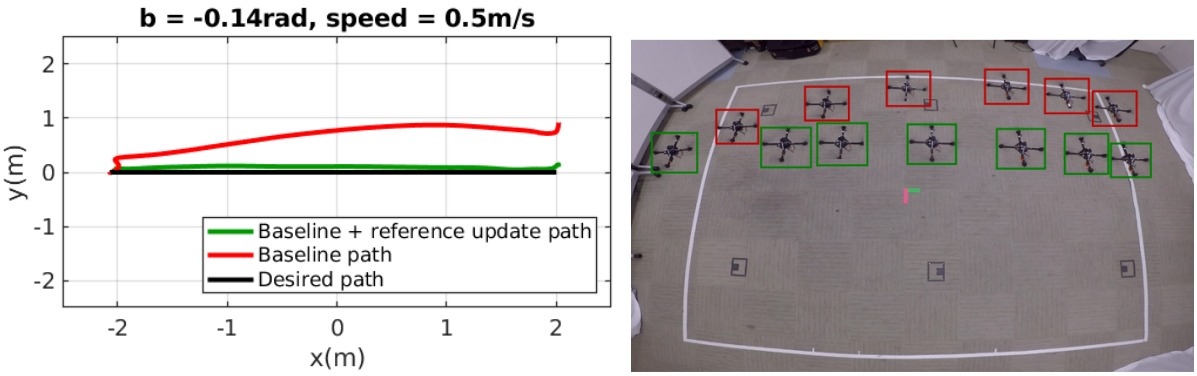


(c) Deviation over time.

Figure 6.6: Simulation results for UAV with fault $\mathcal{F}_2^*$. 2021 ©IEEE

6.5 Experiments

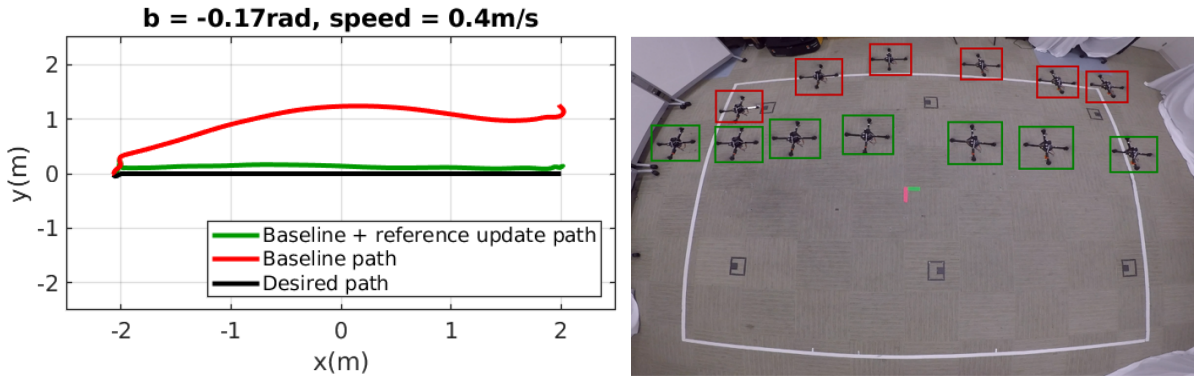
We validated the proposed meta-learning based reference trajectory update approach with experiments on an Asctec Hummingbird quadrotor UAV implemented in ROS. We used a Vicon motion capture system to monitor the state of the quadrotor. The UAV was tasked to follow a minimum-jerk trajectory while experiencing an unknown fault. In these experiments, a fault was implemented as a bias on the commanded roll angle to the vehicle's attitude controller.



(a) UAV paths.

(b) Overlapped sequence of screenshots.

Figure 6.7: Experiments results for the fault $b = -0.14$rad with $\bar{v} = 0.5$m/s. 2021 ©IEEE

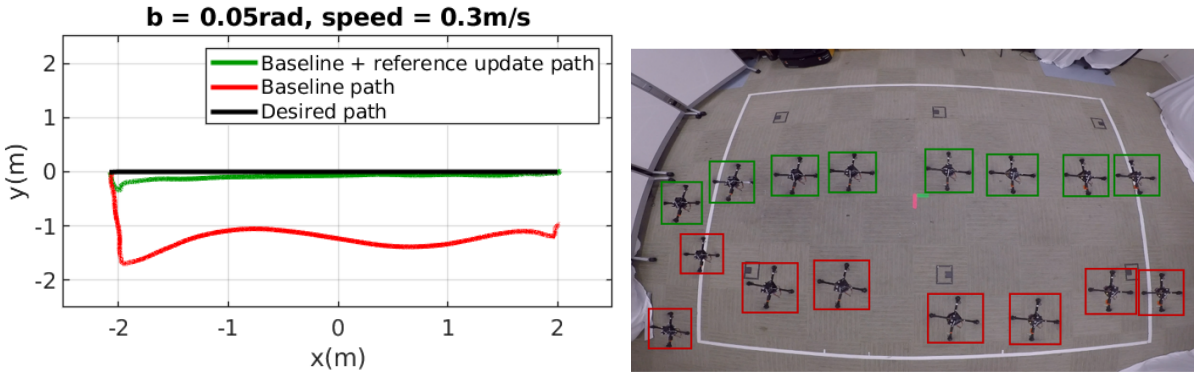


(a) UAV paths

(b) Overlapped sequence of screenshots.

Figure 6.8: Experiments results for the fault $b = -0.17$rad with $\bar{v} = 0.4$m/s. 2021 ©IEEE

During the offline training, we generated a trajectory with five different faults with bias values: $b \in \{-0.18, -0.12, -0.06, 0.06, 0.12\}$ rad and three different average speed values: $\mathcal{V} = \{0.25, 0.35, 0.45\}$ m/s. Using the data from these flights, a neural network was meta-trained. During the online stage, the UAV with an untrained fault was tasked to follow a trajectory with a speed value not included in $\mathcal{V}$. At the beginning of its operation, the UAV adapted the meta-trained network using $K = 50$ initial data points. The adapted



(a) UAV paths.

(b) Overlapped sequence of screenshots.

Figure 6.9: Experiments results for the fault $b = 0.05$rad with $\bar{v} = 0.3$m/s. 2021 ©IEEE

network was used to make predictions and to update the reference trajectory according to (6.9). During the experiments, we did not apply the trajectory update on the z axis as the faults considered did not cause z deviations.

In Figure 6.7, the results for a quadrotor under fault $b = -0.14$ rad tasked to follow a desired trajectory (black curve) with an average speed of 0.5 m/s which is outside of training bounds is shown. As can be noted, with our approach, the quadrotor flew in close proximity of the original desired trajectory (green curve). In contrast, with only the baseline controller, it deviated by a large amount (red curve), which is undesirable for both safety and liveness concerns. With the proposed approach, the average deviation was reduced to 8.09 cm from 55.78 cm. The comparison of the two behaviors is displayed also in Figure6.7(b) by showing sequences of snapshots of the quadrotors moving inside our lab with the reference update approach on top of the baseline controller (green boxes) compared to the baseline controller (red boxes). Similarly, in Figure 6.8(a), the quadrotor with a bias $b = -0.17$ rad followed the desired trajectory much more closely with an average deviation of 12.39 cm as opposed to 79.73 m without using our proposed meta-learning based reference trajectory update. The corresponding sequence of snapshots of the quadrotors is given in Figure 6.8(b). In the case demonstrated in Figure 6.9(a), the UAV experienced a bias $b = -0.05$ rad and was tasked to move with the average speed of 0.3m/s. The average deviation was recorded as 10.23 cm, while the average deviation with the baseline controller was 1.16 m. The related overlapped sequence of snapshots of the quadrotors is presented in Figure 6.9(b).

6.6 Discussions

In this chapter, we have presented a trajectory tracking adaptive approach for UAVs moving under degraded conditions like actuator faults. We have leveraged meta-learning, which can easily adapt the system's model at

runtime using a small number of online data for future state prediction. The robust control inspired reference trajectory update method improves trajectory tracking performance of the system with actuator fault by using the future state predictions without needing access to the control inputs. With the runtime validation and data pruning scheme, the updated meta-learned model is continuously monitored and re-updated with a small number of representative data at runtime when necessary.

Limitations: This meta-learning-based trajectory tracking framework updates a meta-trained network on the runtime observed data. The performance of the predictions of the meta-trained network depends on the similarity of the runtime task to the training tasks. The fault is assumed to be always present at runtime and not lead the system to uncontrollable situations. When a fault is not present at the beginning of the operation and occurs later or when it is dynamically changing, fault detection techniques [34] can be combined with this approach to detect the fault before applying our reference update method.

Future Work: This work opens an exciting path towards using meta-learning approaches for UAV fault rejection. This framework can be extended with a safety monitor to detect and avoid unsafe situations over a given future time horizon as explained in the next chapter. As part of future work, the reference trajectory update method can be improved by leveraging gain scheduling or learning techniques. Additionally, learning approaches can also be used to monitor the validity of the runtime updated network.

Chapter 7

Predictions and Proactive Replanning for Systems under Actuator Faults

In this chapter, we present our current efforts to predict the future states and state uncertainties of a faulty system that uses the reference update method introduced in Chapter 6. The proposed approach leverages meta-learning to train a network for making predictions about the future states and their uncertainties, which are used to detect and prevent unsafe situations with proactive replanning. We present UAV navigation simulation results to validate the proposed technique.

7.1 Introduction

As mentioned in Chapter 6, robotic systems may need to operate under degraded conditions due to unforeseen component faults occurring at runtime. To alleviate the effects of such component faults, we presented a meta-learning-based reference trajectory update approach in Chapter 6. This technique improved the system's trajectory tracking performance under an unforeseen component fault; however, depending on the fault and tuning of the reference trajectory update method, the system may still deviate from its desired behavior, possibly leading to unsafe situations. In this chapter, we extend our meta-learning-based reference trajectory update method by introducing a safety monitoring framework. With this framework, future states and state uncertainties of the faulty system with our reference update method are predicted. These predictions are then utilized to monitor if the reference update method is insufficient to make the system follow its desired trajectory and stay safe. If an unsafe situation is detected, a safe trajectory is replanned.

This monitoring and replanning framework consists of offline and online stages. At the offline stage, a system with various faults follows a set of trajectories with our proposed reference trajectory update method under a set of actuator uncertainties. A meta-learning network is trained using data from these trajectories to predict the states and state uncertainties of the system for a given horizon. The system with an unforeseen fault starts applying reference trajectory update at runtime, as explained in the previous chapter, and runtime data are collected. With only a limited amount of data at the beginning of the operation, offline trained meta-network is fine-tuned. Using the predictions based on the fine-tuned network, the system's desired trajectory is replanned as deemed necessary.

7.2 Preliminaries

7.2.1 Assumptions

Similar to the previous chapter, our framework assumes that the system does not have access to the controller or the control inputs. We assume that the system is already applying our reference trajectory update method to alleviate the effects of faults and follow its trajectory closely. We assume access to various faulty systems that apply our reference trajectory update method during the design time. At runtime, the same reference trajectory update method is assumed to be applied for trajectory tracking.

7.2.2 Notations

In this chapter we use $\boldsymbol{x}(k)$ to present the state of the system at time k. $\boldsymbol{p}(k)$ and $\boldsymbol{v}(k)$ represent the position and velocity of the system at time k. The symbol $\tilde{\boldsymbol{x}}$ is used to represent the predicted state, and the symbol $\bar{\boldsymbol{x}}$ is used to represent the mean of sampled states at time k. The notation $x(k_1 : k_N)$ represents an array of values from time k_1 to k_N: $x(k_1 : k_N) = [x(k_1), x(k_1 + 1), \cdots, x(k_N)]$ where $k_N > k_1$. The notation $x(k_1 : \delta_k : k_N)$ represents and array of values from time k_1 to k_N with δ_k increments: $x(k_1 : \delta_k : k_2) = [x(k_1), x(k_1 + \delta_k), x(k_1 + 2\delta_k), \cdots, x(k_N)]$ where $k_N > k_1$ and $\delta_k > 1$.

7.3 Problem Definition

In this chapter, we are interested in finding a technique to predict the future states of a faulty system that is applying meta-learning-based reference trajectory update framework and to use these predictions for safe replanning. These problems are formally defined as follows:

Problem 1: ***Future State Prediction under Failure*****:** An autonomous system with a nominal dynamical model $f(\boldsymbol{x}, \boldsymbol{u})$ as a function of its states $\boldsymbol{x}$ and controller inputs $\boldsymbol{u}$ has the objective of following a predefined desired trajectory $\boldsymbol{x}_\tau$. Under actuator faults and disturbances, the system's model changes to $\boldsymbol{x}(k+1) = f'(\boldsymbol{x}(k), \boldsymbol{u}(k))$. Based on the reference trajectory update method presented in Section 6.3, the system's reference trajectory is updated to $\tilde{\boldsymbol{x}}_\tau$ and its control inputs are generated by a fixed controller: $\boldsymbol{u}(k) = g(\boldsymbol{x}, \tilde{\boldsymbol{x}}_\tau(k+1))$. With these premises, design a predictor to predict the future states and state uncertainties ($\boldsymbol{\zeta}$) of the system as a function of history of states and reference trajectory: $[\tilde{\boldsymbol{x}}(k:k+H), \tilde{\boldsymbol{\zeta}}(k:k+H)] = \tilde{h}(\boldsymbol{x}(k-T:k), \boldsymbol{x}_\tau)$. H is the state prediction horizon and T is the size of the data history used to make predictions.

Problem 2: ***Safe Replanning*****:** Find an online policy to monitor the safety of the future states of the system and to replan the trajectory when an unsafe situation is detected to ensure that the following safety condition will be satisfied by the future state predictions:

$$R_{\boldsymbol{p}}|_k^{k+H} \cap \boldsymbol{\mathcal{O}} = \emptyset \tag{7.1}$$

where $\boldsymbol{\mathcal{O}}$ is the set of obstacle positions and $R_{\boldsymbol{p}}|_k^{k+H}$ is the union of future position sets that the system is predicted to reach for the time horizon H with time increments of δ_H:

$$R_{\boldsymbol{p}}|_k^{k+H} = R_{\boldsymbol{p}}(k) \cup R_{\boldsymbol{p}}(k+\delta_H) \cdots \cup R_{\boldsymbol{p}}(k+H) \tag{7.2}$$

The set of positions that the system is predicted to reach at time k is computed as follows:

$$R_{\boldsymbol{p}}(k) = \{\boldsymbol{p}|\ |\boldsymbol{p} - \tilde{\boldsymbol{p}}(k)| \leq \tilde{\boldsymbol{\zeta}}_{\boldsymbol{p}}(k)\} \tag{7.3}$$

where $\tilde{\boldsymbol{p}}$ is the predicted position and $\tilde{\boldsymbol{\zeta}}_{\boldsymbol{p}}$ is the predicted position uncertainty.

7.4 Meta-Learning-based Predictions and Replanning for Faulty Systems with Reference Trajectory Update

Our framework is designed to predict the future states and state uncertainties of the system using meta-learning, and it consists of offline and online stages as depicted in Figure 7.1. During the offline stage, a system with various faults follows a set of trajectories while using the reference trajectory update method explained in Section 6.3. As the model of the system with a fault is known during training, we use the

system's model to compute the next state of the system, instead of using a meta-trained network. Based on the training data collected, a meta-network is trained offline to make predictions about the future states and state uncertainties. At the online stage, the system with a new unforeseen fault starts its operation and applies the reference update method as explained in Section 6.3. As the model of the system at the online stage is unknown, a meta-trained network is used to compute the next state and update the reference trajectory. While the system continues its operation, runtime data are collected to fine-tune the meta-network for future predictions. The fine-tuned prediction network is then used to make predictions about the future states and state uncertainties of the system. These predictions are used within a runtime replanning approach to find a safe trajectory if the original desired trajectory is deemed unsafe with the fault that the system is experiencing.

7.4.1 Offline Training for Future State and Uncertainty Predictions

During the offline stage, in order to learn model for state predictions, we first collect training data with a rich set of trajectories using various faulty UAVs. Then, we train a meta-network which is easy to fine-tune at runtime using a small number of collected data.

Data Collection

During the offline stage, first we create a dataset using the data collected from a UAV with actuator fault from a discrete fault set $\mathcal{F}$ while it is following different trajectories using our reference trajectory update method. The reference trajectory update method is applied as explained in Section 6.3; however, as the model of the faulty system is assumed to be known during training, its next state is simulated as if it follows the original trajectory:

$$\hat{\boldsymbol{x}}(k+1) = f'(\boldsymbol{x}(k), \boldsymbol{u}(k)) \tag{7.4}$$

where $\boldsymbol{u}(k) = g(\boldsymbol{x}(k), \boldsymbol{x}_\tau(k+1))$. Using the position part of the next state $(\hat{\boldsymbol{p}}(k+1))$, the reference input to the system is updated as follows:

$$\boldsymbol{r}(k+1) = \boldsymbol{p}_\tau(k+1) + \boldsymbol{c}(k+1) \tag{7.5}$$

The update term $\boldsymbol{c}$ is computed as follows:

$$\boldsymbol{c}(k+1) = \kappa_p \boldsymbol{d}(k+1) + \kappa_d(\boldsymbol{d}(k+1) - \boldsymbol{d}(k)) + \kappa_i \left(\sum_{t=0}^{t=k} \boldsymbol{d}(t) + \boldsymbol{d}(k+1) \right) \tag{7.6}$$

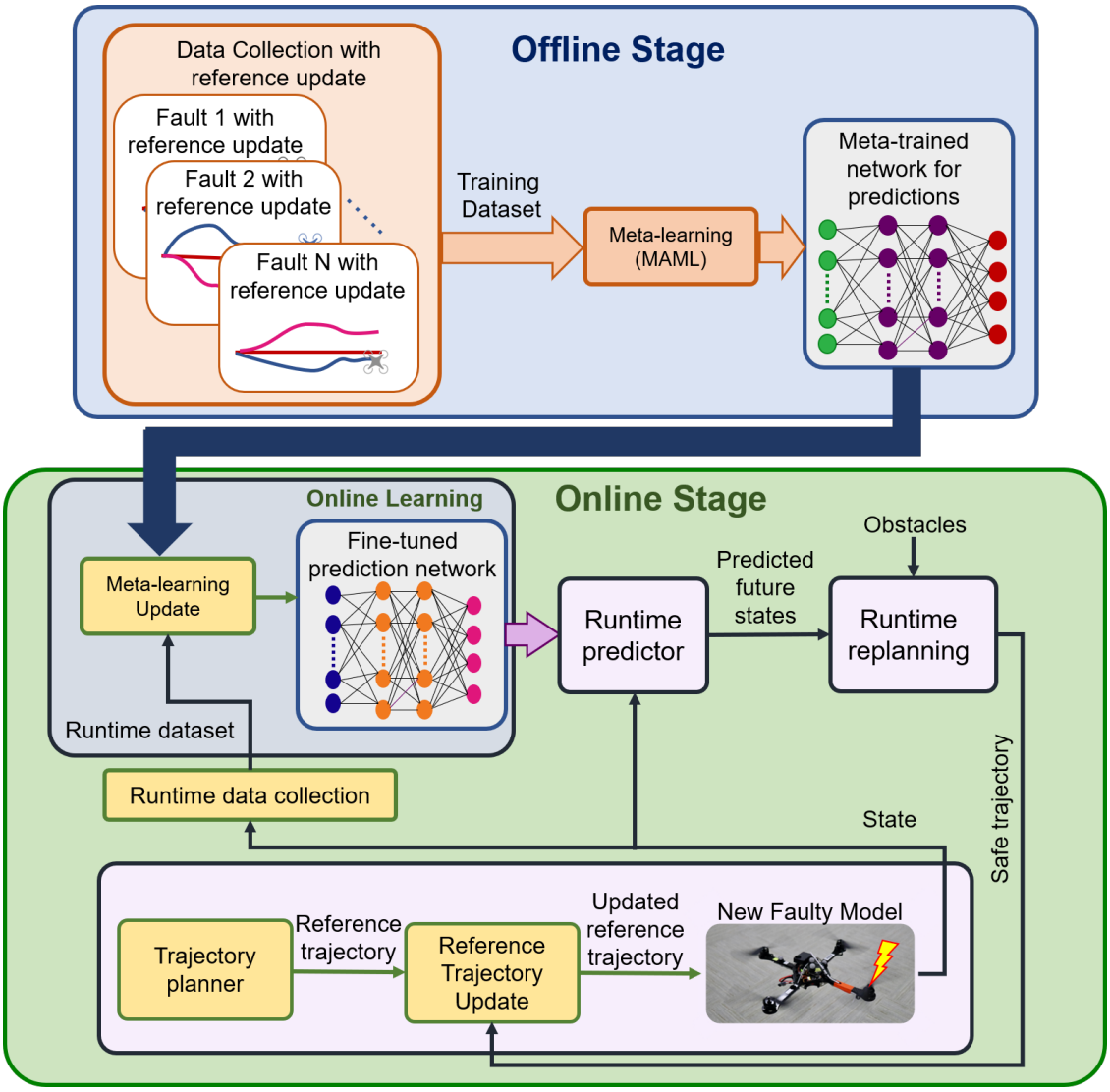


Figure 7.1: Meta-learning-based future state prediction and replanning framework for systems under unknown faults.

where κ_p, κ_d and κ_i are positive proportional, derivative and integral gains respectively. $\boldsymbol{d}(k+1)$ is the deviation of $\hat{\boldsymbol{p}}(k+1)$ from the path:

$$\boldsymbol{d}(k+1) = \check{\boldsymbol{p}}_\tau(k+1) - \hat{\boldsymbol{p}}(k+1) \tag{7.7}$$

where $\check{\boldsymbol{p}}_\tau(k+1)$ is the closest point on the trajectory τ:

$$\check{\boldsymbol{p}}_\tau(k+1) = \underset{\boldsymbol{p}_\tau(t), t\in\{0,\ldots,T(\tau_j)\}}{\arg\min} \|\hat{\boldsymbol{p}}(k+1) - \boldsymbol{p}_\tau(t)\| \tag{7.8}$$

The reference velocity to the system is also updated accordingly:

$$\boldsymbol{r}_v(k+1) = \frac{\boldsymbol{r}(k+1) - \boldsymbol{r}(k)}{\Delta k} \tag{7.9}$$

Given the updated reference position $\boldsymbol{r}$ and reference velocity $\boldsymbol{r}_v(k+1)$, the system generates control input using its fixed controller:

$$\boldsymbol{u}'(k) = g(\boldsymbol{x}(k), \boldsymbol{x}_r(k+1)) \tag{7.10}$$

where $\boldsymbol{x}_r$ is the new desired state with the updated reference position ($\boldsymbol{r}$) and velocity ($\boldsymbol{r}_v$). The system is driven with this updated input with an added noise $\boldsymbol{\eta}(k)$:

$$\boldsymbol{x}(k+1) = f'(\boldsymbol{x}(k), \bar{\boldsymbol{u}}(k) + \boldsymbol{\eta}(k)) \tag{7.11}$$

The actuator noise $\boldsymbol{\eta}(k) \sim \mathcal{N}(\boldsymbol{\mu}_\eta, \boldsymbol{\sigma}_\eta)$ is sampled from a normal distribution with mean $\boldsymbol{\mu}_\eta$ and standard deviation $\boldsymbol{\sigma}_\eta$.

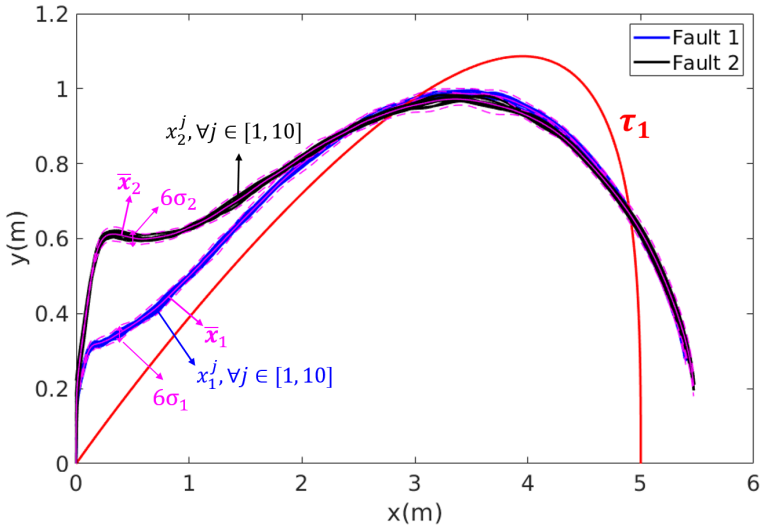
For the training dataset, we generate a set of desired trajectories that the faulty system follows by using the updated reference trajectory and control inputs given in (7.5) and (7.10) respectively. During the data collection, each trajectory is run N times. For each sampled run, the mean of the actuator noise is sampled from a normal distribution to capture the behavior of the system under various uncertainties: $\boldsymbol{\mu}_\eta \sim \mathcal{N}(\bar{\boldsymbol{\mu}}, \boldsymbol{\sigma}_\mu)$.

For each trajectory, we compute the mean and standard deviation of the N sampled paths for each fault $\mathcal{F}_i \subset \mathcal{F}$ for every trajectory $\tau \subset \mathcal{T}$:

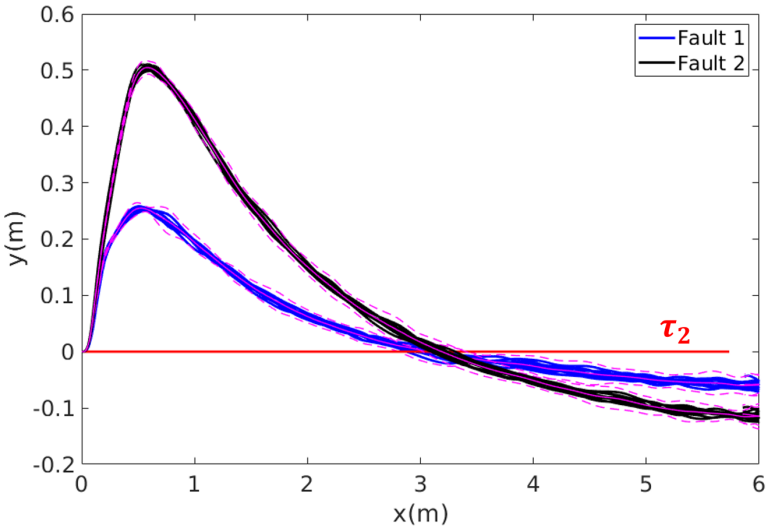
$$\bar{\boldsymbol{x}}_i(k) = \frac{\sum_{j=1}^{N} \boldsymbol{x}_i^j(k)}{N} \qquad \boldsymbol{\sigma}_i(k) = \sqrt{\frac{\sum_{j=1}^{N} |\boldsymbol{x}_i^j(k) - \bar{\boldsymbol{x}}_i(k)|^2}{N-1}} \qquad \forall k \in [0, T_\tau], \forall i \in \{1, \cdots, |\mathcal{F}|\} \tag{7.12}$$

In Figure 7.2 we show two different sample desired trajectories ($\boldsymbol{\tau_1}$ and $\boldsymbol{\tau_2}$) from our training trajectory dataset that are followed by two different faulty UAVs applying reference trajectory update method to reduce their tracking error. The first faulty UAV has reduced thrust in one propeller: $\mathcal{F}_1 \longrightarrow T_1' = T_1 - 0.25\text{N}$ and the second faulty system has more degradation on the same propeller: $\mathcal{F}_2 \longrightarrow T_1' = T_1 - 0.5\text{N}$. Roll and pitch angles of both systems are limited too: $\phi \leq \frac{\pi}{18}, \theta \leq \frac{\pi}{18}$. Each trajectory is run $N = 10$ times with different actuator noise sampled as explained above. Blue curves in Figure 7.2(a) show these N sampled paths for the first fault and magenta curve in the middle shows the mean of these samples ($\bar{\boldsymbol{x}}_1$). Dashed magenta curves show the uncertainty around the mean of the samples with $\pm 3\boldsymbol{\sigma}_i$. Figure 7.2(b) show the paths of the same faulty UAVs following another training trajectory ($\boldsymbol{\tau_2}$). During training, we create 100 training trajectories using minimum-jerk trajectory generation [55] with different final position, initial and

final velocities. Determining the training data size for training an accurate meta-model is an open problem and beyond the scope of this work. This framework leaves the choice of training data size to the user.



(a) Training trajectory 1 with two different faulty systems.



(b) Training trajectory 2 with two different faulty systems.

Figure 7.2: Sample training trajectories with two different faulty systems.

Meta-network Training

The purpose of meta-learning is to train an easily adaptable model to predict the future positions and position uncertainties of a faulty system. We denote this learning model as h which takes the history of the system's positions and velocities, history of desired positions and velocities, the future desired positions and velocities

as inputs and gives the future positions and the position uncertainty as an output. A training input $\boldsymbol{x}_h^i \in \mathbb{R}^{100}$ and training output $\boldsymbol{y}_h^i \in \mathbb{R}^{12}$ with the fault $\mathcal{F}_i \subset \mathcal{F}$ are constructed as follows:

$$\boldsymbol{x}_h^i(k) = \begin{bmatrix} \bar{x}_i(k-T+1:k)^\mathsf{T} - \bar{x}_i(k-T)\vec{\mathbf{1}} \\ \bar{y}_i(k-T+1:k)^\mathsf{T} - \bar{y}_i(k-T)\vec{\mathbf{1}} \\ \bar{v}_i^x(k-T+1:k)^\mathsf{T} \\ \bar{v}_i^y(k-T+1:k)^\mathsf{T} \\ x_\tau(k-T+1:k)^\mathsf{T} - \bar{x}_i(k-T)\vec{\mathbf{1}} \\ y_\tau(k-T+1:k)^\mathsf{T} - \bar{y}_i(k-T)\vec{\mathbf{1}} \\ v_{x,\tau}(k-T+1:k)^\mathsf{T} \\ v_{y,\tau}(k-T+1:k)^\mathsf{T} \\ x_\tau(k+\delta_H:\delta_H:k+H)^\mathsf{T} - \bar{x}_i(k-T)\vec{\mathbf{1}} \\ y_\tau(k+\delta_H:\delta_H:k+H)^\mathsf{T} - \bar{y}_i(k-T)\vec{\mathbf{1}} \\ v_{x,\tau}(k+\delta_H:\delta_H:k+H)^\mathsf{T} \\ v_{y,\tau}(k+\delta_H:\delta_H:k+H)^\mathsf{T} \end{bmatrix} \boldsymbol{y}_h^i(k) = \begin{bmatrix} \bar{x}_i(k+\delta_H:\delta_H:k+H)^\mathsf{T} - \bar{x}_i(k-T)\vec{\mathbf{1}} \\ \bar{y}_i(k+\delta_H:\delta_H:k+H)^\mathsf{T} - \bar{y}_i(k-T)\vec{\mathbf{1}} \\ 3\max(\sigma_i^x(k:k+H)) \\ 3\max(\sigma_i^y(k:k+H)) \end{bmatrix}$$

$$\forall \tau \subset \mathcal{T}, \qquad \forall k \in \{1, \cdots, T(\tau)\} \tag{7.13}$$

where $\bar{\boldsymbol{p}}_i(k) = [\bar{x}_i(k), \bar{y}_i(k)]$ and $\bar{\boldsymbol{v}}_i(k) = [\bar{v}_i^x(k), \bar{v}_i^y(k)]$ are the position and velocity components of the mean state $\bar{\boldsymbol{x}}_i(k)$ respectively, $\sigma_i^x(k)$ and $\sigma_i^y(k)$ are the x and y position components of the standard deviation $\boldsymbol{\sigma}_i(k)$ respectively. During training, the system runs at 40Hz with time step $\delta k = 0.025$s. We use $T = 10$ past data to predict the future states, and we set the future horizon for the predictions H to 50 steps with increments of $\delta_H = 10$steps. The training output consists of the future states of the system relative to its previous positions, and the maximum uncertainty over the course of the future horizon. It should be noted that the training needs to be performed with the same horizon that will be used to make predictions at runtime.

The dataset for meta-learning training $\mathcal{D}_i^H$ for fault $\mathcal{F}_i$ contains the training input matrix $\boldsymbol{X}_h^i \in \mathbb{R}^{100 \times M_i}$ and output matrix $\boldsymbol{Y}_h^i \in \mathbb{R}^{12 \times M_i}$, with the columns $\boldsymbol{x}_h^i$ and $\boldsymbol{y}_h^i$ respectively. M_i is the number of data samples for fault $\mathcal{F}_i$. The training dataset for meta learning contains the dataset for each fault: $\mathcal{D}_i^H \subset \mathcal{D}^H$.

The purpose of meta-learning is to learn a model represented by a parametrized function $h_{\boldsymbol{\phi}}$ that maps the model input to the output. We use MAML[20] as a meta-learning algorithm to train the network. During the offline training, the model parameters $\boldsymbol{\phi}$ are meta-optimized according to Equation 1 in [20]:

$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \beta \nabla_{\boldsymbol{\phi}} \sum_{\mathcal{F}_i \subset \mathcal{F}} \mathcal{L}_{\mathcal{F}_i}(h_{\phi - \alpha \nabla_{\boldsymbol{\phi}} \mathcal{L}_{\mathcal{F}_i}(h_{\boldsymbol{\phi}})}) \tag{7.14}$$

This meta-optimization allows the parameters to quickly be fine-tuned with a few data at runtime. The loss function used during this training is given as follows:

$$\mathcal{L}_{\mathcal{F}_i}(h_{\boldsymbol{\psi}}) = \sum_{\boldsymbol{x}_h^i, \boldsymbol{y}_h^i \in \mathcal{D}_i^H} \|\boldsymbol{h}_\psi(\boldsymbol{x}_h^i) - \boldsymbol{y}_h^i\|_2^2 \tag{7.15}$$

where $\boldsymbol{x}_h^i$ and $\boldsymbol{y}_h^i$ are given in Eq.(7.13).

7.4.2 Online Meta-Network Update

At runtime, the UAV may experience a new fault which is not included in the training set. As the system's model under this new fault is unknown, the system uses a meta-trained network to update its reference trajectory as explained in Section 6. While the system moves under new fault with the reference update method, it collects K_p consecutive data from its position and velocity sensors to update the offline meta-trained model for future state predictions. The online learning dataset for prediction meta-learning model consists of inputs $\boldsymbol{X}_h^* \in \mathbb{R}^{100\times K_p}$ and outputs $\boldsymbol{Y}_h^* \in \mathbb{R}^{12\times K_p}$ which are constructed based on the observations at runtime:

$$\boldsymbol{x}_h^*(k) = \begin{bmatrix} x^*(k-T+1:k)^\mathsf{T} - x^*(k-T)\vec{\mathbf{1}} \\ y^*(k-T+1:k)^\mathsf{T} - y^*(k-T)\vec{\mathbf{1}} \\ v_x^*(k-T+1:k)^\mathsf{T} \\ v_y^*(k-T+1:k)^\mathsf{T} \\ x_\tau^*(k-T+1:k) - x^*(k-T)\vec{\mathbf{1}}^\mathsf{T} \\ y_\tau^*(k-T+1:k) - y^*(k-T)\vec{\mathbf{1}}^\mathsf{T} \\ v_{x,\tau}^*(k-T+1:k)^\mathsf{T} \\ v_{y,\tau}^*(k-T+1:k)^\mathsf{T} \\ x_\tau^*(k+\delta_H:\delta_H:k+H)^\mathsf{T} - \bar{x}_i(k-T)\vec{\mathbf{1}} \\ y_\tau^*(k+\delta_H:\delta_H:k+H)^\mathsf{T} - \bar{y}_i(k-T)\vec{\mathbf{1}} \\ v_{x,\tau}^*(k+\delta_H:\delta_H:k+H)^\mathsf{T} \\ v_{y,\tau}^*(k+\delta_H:\delta_H:k+H)^\mathsf{T} \end{bmatrix}^\mathsf{T} \quad \boldsymbol{y}_h^*(k) = \begin{bmatrix} x^*(k+\delta_H:\delta_H:k+H)^\mathsf{T} - x^*(k-T)\vec{\mathbf{1}} \\ y^*(k+\delta_H:\delta_H:k+H)^\mathsf{T} - y^*(k-T)\vec{\mathbf{1}} \\ \sigma_x \\ \sigma_y \end{bmatrix} \tag{7.16}$$

for $k \in \{T+1, \cdots, T+K_p\}$ where $\boldsymbol{p}^* = [x^*, y^*]$ and $\boldsymbol{v}^* = [v_x^*, v_x^*]$ are the position and velocity of the UAV with an unknown fault at runtime. $\boldsymbol{p}_\tau^* = [x_\tau^*, y_\tau^*]$ and $\boldsymbol{v}_\tau^* = [v_{x,\tau}^*, v_{y,\tau}^*]$ are desired trajectory positions and velocities respectively. σ_x and σ_y are initial assigned uncertainties in x and y directions respectively and they are initially set to a value larger than the mean of the observed uncertainties during training. By using this runtime dataset, the meta-learning prediction model is fine-tuned using gradient descent updates and

a fine-tuned model $h_{\boldsymbol{\phi}^*}$ with updated parameters $\boldsymbol{\phi}^*$ is obtained. This fine-tuned model is used to make predictions for the future states of the system for a given horizon and these predictions are utilized to replan the trajectories if unsafe situations are detected.

Runtime Validation

After the initial meta-network update at runtime, the runtime inputs are compared to the training inputs to assess if a further meta-network update is necessary or not. The distance between the runtime input and training inputs with training faults is calculated:

$$d^i_{\mathcal{F}}(k) = \min_{\boldsymbol{x}^i_h \in \text{col}(\boldsymbol{X}^i_h)} \|\boldsymbol{x}^*_h(k) - \boldsymbol{x}^i_h\| \qquad \forall i \in \{1, \cdots, |\mathcal{F}|\} \qquad \forall k \in \{T + K_p, \cdots, T(\tau)\} \tag{7.17}$$

If the minimum distance between the observed test input and the training inputs are larger than a given threshold, the system re-tunes its meta-trained network:

$$s_H(k) = 1 \qquad \text{if } \min_{i \in \{1, \cdots, |\mathcal{F}|\}} (d^i_{\mathcal{F}}(k)) > \xi_H \tag{7.18}$$

where s_H is a binary variable that enables re-updating the meta-trained network at runtime using the last K_p runtime training inputs and ξ_H is a user defined threshold.

We also constantly monitor the observed stat to check if it is outside of the predicted reachable region. If so, the network is re-tuned:

$$s_H(k) = 1 \qquad \text{if } \boldsymbol{p}(k) \not\subset \tilde{R}_{\boldsymbol{p}}(k) \tag{7.19}$$

where $\tilde{R}_{\boldsymbol{p}}(k)$ is a region where the system is predicted to reach at time k:

$$\tilde{R}_{\boldsymbol{p}}(k) = \cup\{\boldsymbol{p}| \ |\boldsymbol{p} - \tilde{\boldsymbol{p}}(k)| \leq \tilde{\boldsymbol{\zeta}}(k)\} \tag{7.20}$$

7.4.3 Runtime Replanning for Safety

After updating the meta-trained network, the future states and state uncertainties of the system is predicted using the fine-tuned network:

$$\begin{bmatrix} \tilde{x}(k + \delta_H : \delta_H : k + H)^\mathsf{T} \\ \tilde{y}(k + \delta_H : \delta_H : k + H)^\mathsf{T} \\ \tilde{\sigma_x}(k + \delta_H : \delta_H : k + H)^\mathsf{T} \\ \tilde{\sigma_y}(k + \delta_H : \delta_H : k + H)^\mathsf{T} \end{bmatrix} = \boldsymbol{h}^*_\phi(\boldsymbol{x}^*_h(k)) + \begin{bmatrix} x^*(k - T)\vec{\mathbf{1}} \\ y^*(k - T)\vec{\mathbf{1}} \\ 0 \\ 0 \end{bmatrix} \tag{7.21}$$

At runtime, the predicted set based on the updated meta-trained model are used to proactively detect unsafe situations. Given an environment with a set of static obstacles $\mathcal{O}$, the regions that the system is predicted to reach, which are computed as in (7.20), are checked for collision:

$$s^*(k+t) = \begin{cases} 0 & \text{if } \tilde{R}_{\boldsymbol{p}}(k+t) \cap \boldsymbol{\mathcal{O}} \neq \emptyset \\ 1 & \text{otherwise} \end{cases} \qquad \forall t \in \{\delta_H, 2\delta_H, \cdots, H\} \tag{7.22}$$

At time k, if it is detected that $s^*(k+t) = 0$ (for $t \in \{\delta_H, 2\delta_H, \cdots, H\}$), the systems trajectory is replanned. We use a sampling based replanning in which a waypoint around the original unsafe desired trajectory point is generated and tested for safety until a safe waypoint is found as outlined in Algorithm 1.

Algorithm 1 Trajectory Replanning

```
Time k
Initialize the trajectory: τ
Assess safety s*(k+t) based on (7.22)
while s*(k+t) = 0 do
    Sample update distance d_s U_[0,d̄_s]
    Sample a waypoint p_w = p_τ(k+t) + d_s v⃗_d
    Replan trajectory τ with p_w
    Make future state predictions with new trajectory τ
    Compute s*(k+t) based on (7.22) with the new trajectory
    Increase d̄_s
end while
Return updated trajectory τ
```

where $\vec{v}_d$ is the unit vector orthogonal to the direction of the desired movement. The replanning algorithm is applied until the desired trajectory reaches the goal location.

7.5 Simulation Results

To validate the proposed approach, we created a UAV navigation case study in an obstacle cluttered environment. In this case study, we use a 12-dimensional quadrotor UAV model with a baseline PID controller and the actuator faults are simulated by reducing the thrust in one propeller and limiting the roll and pitch angles of the UAV. The faults used during the offline training are given in Table 7.1.

Training data for this case study are collected as explained in Section 7.4.1 using minimum-jerk trajectories with various final positions, initial and final velocity pairs. Training data include 100 training trajectories. A neural network for next state prediction and reference trajectory update is trained using MAML as explained in Section 6.3.1 and the network is fine-tuned at runtime as presented in Section 6.3.2. The network used for reference update consists of 3 layers with 100 nodes. Using the same training data, a network making

Table 7.1: Fault types used during simulations.

Training fault name	Thrust reduction (N)	Roll and pitch angle limit
$\mathcal{F}_1$	0.25	$\frac{\pi}{12}$
$\mathcal{F}_2$	0.50	$\frac{\pi}{12}$
$\mathcal{F}_3$	0.25	$\frac{\pi}{18}$
$\mathcal{F}_4$	0.50	$\frac{\pi}{18}$
Test fault name	Thrust reduction (N)	Roll and pitch angle limit
$\mathcal{F}_1^*$	0.80	$\frac{\pi}{24}$

predictions about the future states and state uncertainties of the system is trained as explained in Section 7.4.1 and it is updated over time at runtime using $K_p = 20$ observed data as presented in Section 7.4.2. The network used for future state predictions consists of five hidden layers with 100 nodes.

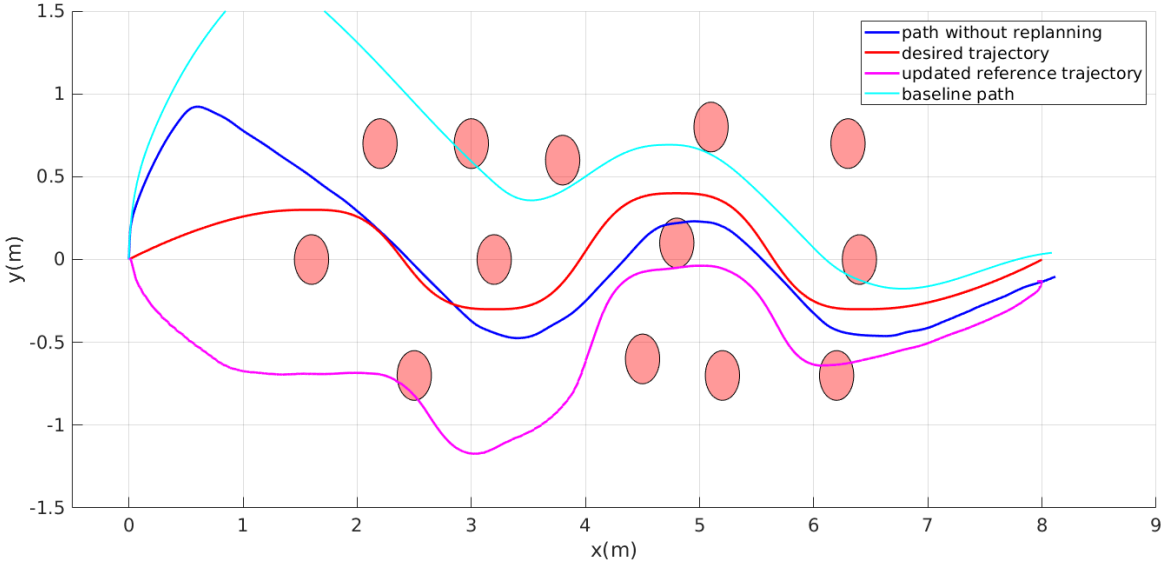
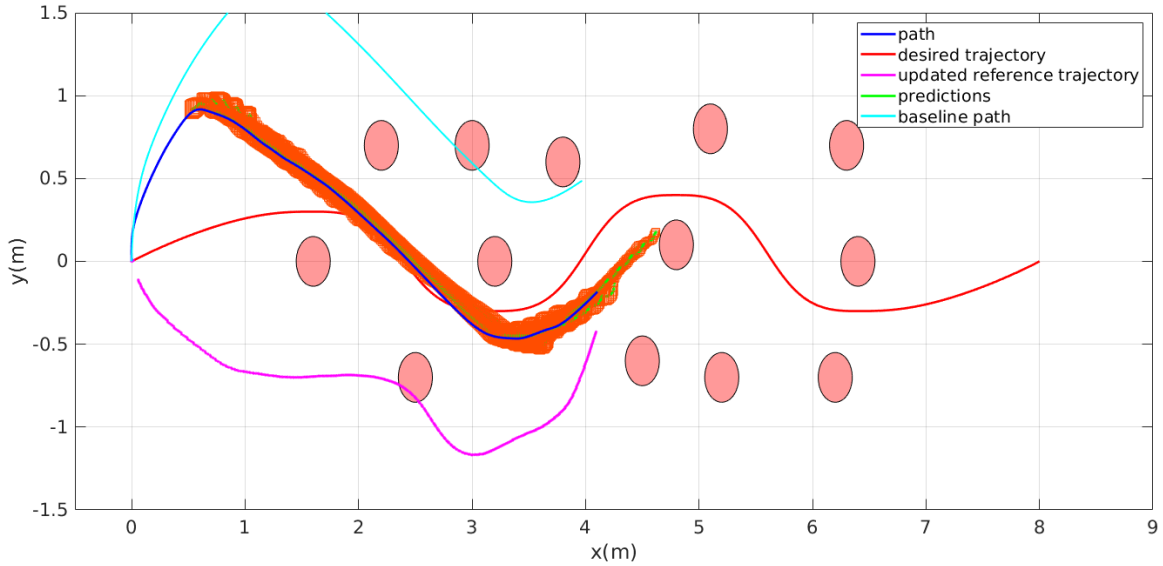
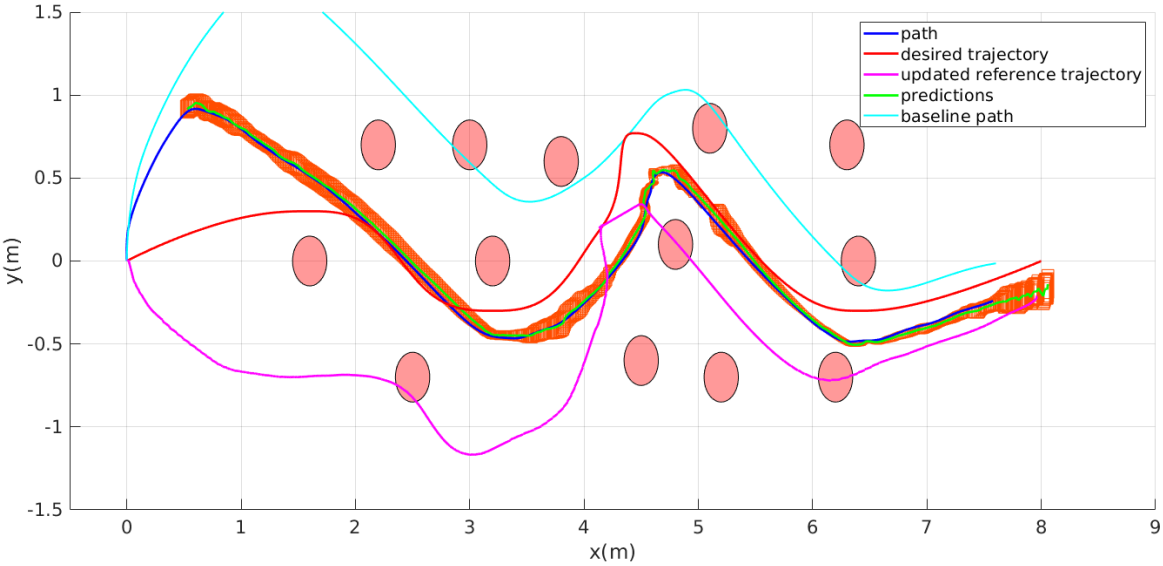


Figure 7.3: Path of a faulty UAV with meta-learning-based state predictions and replanning

At runtime, the UAV is tasked to move to its goal location $\boldsymbol{p}_g = [8.0, 0.0, 1.0]$m while following an obstacle-free trajectory computed using minimum-jerk trajectory generation [55] under the unforeseen fault $\mathcal{F}_1^*$ given in Table 7.1. As the system moves along its desired trajectory, it applies reference input update method explained in Section 6.3.3. In Figure 7.3, the blue curve shows the faulty system's path, cyan curve shows the path of the system without the reference trajectory update method, red curve shows the desired trajectory and the magenta curve shows the updated reference trajectory. Even though the system follows its desired trajectory with less deviation while using the reference update method, due to the nature of the fault and the tuning of the reference update method, it still deviates from its desired trajectory significantly. This deviation causes the system to collide with an obstacle located at $[4.8, 0.4, 1]$m. To prevent this collision, we leverage the prediction network which is trained using meta-learning.



(a) Collision is detected based on the future state predictions (green curve) and uncertainties (orange rectangles) .



(b) Desired trajectory (red curve) is replanned according to Algorithm 1 and the collision is prevented.

Figure 7.4: Path of a faulty UAV with meta-learning-based state predictions and replanning.

In Figure 7.4, the behavior of the system with the future state predictions and replanning approach is depicted. The system starts its operation and predicts the future states (shown by the green curve) and state uncertainties (shown by the orange rectangles). Based on these predictions, it detects that the system may collide with an obstacle at the instant shown in Figure 7.4(a) and finds a safe waypoint to go and replans the trajectory according to Algorithm 1. Figure 7.4(b) shows the final desired trajectory and the path of the system. As can be noticed, by replanning the trajectory, the system avoided all the obstacles in the

environment.

7.6 Discussions

In this chapter, we have presented our current efforts on monitoring the performance of reference update approach explained in Section 6. We leveraged meta-learning to train an easy to tune network to make state and state uncertainty predictions for a system with reference update method and use these predictions to detect if an unsafe situations during a future horizon. With a sampling-based method, we proactively replan the system's original desired trajectory to prevent predicted unsafe situations.

Limitations: This work complements the reference trajectory update approach by adding a safety monitor and replanning scheme. As the predictions for the future states are made based on a meta-trained and fine-tuned network, the performance of the monitor is based on the performance of the training. If the training data is not enough or the network is not well trained, the predictions can become inaccurate. Our framework includes runtime validation by comparing the runtime input with training inputs and monitoring the predictions' accuracy as the system moves. This validation technique helps the system re-tune the trained networks for better predictions, but it does not provide formal assurance guarantees.

Future Work: The replanning scheme used in this framework is based on sampling waypoints and assessing their safety, which can take a long time if a safe waypoint is not found within a limited number of iterations. To overcome this issue, a learning technique which directly produces safe waypoints can be developed. Additionally, we use a fixed horizon for the future state predictions, but a recursive approach can be developed to make predictions for variable horizon.

Chapter 8

Conclusions and Future Work

In this dissertation, we have focused on the problem of runtime monitoring and proactive planning for safe autonomous mobile operations under various uncertainties. First, we have introduced the concept of self/event-triggered scheduling for sensor monitoring operations to minimize computation associated with acquiring and processing sensory data. Reachability analysis has been used to compute the set of states that are reachable by the system under unknown disturbances and noises when the sensors are not monitored. With this approach, we demonstrated that the unnecessary computation for sensor data monitoring can be minimized while still satisfying safety constraints. In order to minimize the computation related to reachability analysis, we leveraged machine learning techniques that allows fast and accurate predictions of the system's future states. With this fast reachability in conjunction with runtime monitoring and recovery techniques, we achieved safe planning under disturbances while decreasing the expensive online computation of reachable sets. To further minimize the computation by completely bypassing reachability computation at runtime, we introduced a novel assured planning framework that leverages verified neural networks to imitate the safety decisions of reachability analysis. This framework allowed us to limit the usage of reachability analysis to solely design time and compute safety-guaranteed paths in milliseconds which would take minutes with the use of traditional reachability analysis tools.

As autonomous systems may face disturbances and model changes that have not been experienced before and outside of the training distribution, we studied online learning techniques to deal with such unforeseen uncertainties. Specifically, we used Gaussian Process regression to estimate the unknown payload mass that a robot is tasked to carry and to predict its effects on the system behavior at runtime. These estimations allow the system to perform proactive replanning to prevent unsafe situations. During proactive planning, the system also updates its offline trained models with the runtime observations. With this method, we have

shown that the system stays safe even under disturbances larger than the training bounds and improves its decision making over time. To deal with unforeseen system model changes such as component faults, we leveraged meta-learning to learn the faulty model at runtime and to update the reference trajectory followed by the system accordingly. With this technique, we demonstrated that the tracking performance of the system with an unforeseen fault is improved.

Throughout this dissertation, we discussed extensive simulation implementations and state-of-the-art aerial robot experiments. These applications demonstrate the applicability of our frameworks to solve real world robotic problems.

With this dissertation, we have demonstrated that runtime monitoring is an essential component for proactive and safe planning and control of autonomous robotic systems operating under uncertainties. We believe that this research is still at an early stage and there are important challenges before making robotics systems truly "autonomous". Future work can focus on extending the approaches presented in this dissertation to address such challenges. One immediate problem towards full autonomy is to deal with complex dynamic behaviors. Such behaviors are specifically prominent in complex tasks such as autonomous driving and social navigation in the presence of humans. For example, in autonomous driving, future states of other dynamic agents, especially when driven by humans, are affected by multiple factors, making it challenging to be accurately predicted. Furthermore, autonomous systems typically rely on their perception modules to sense their environments, which are also subject to uncertainties due to sensory noise, occlusions, or perception inaccuracies. The techniques that we presented in this dissertation can be extended to learn the complex models of dynamic agents and provide safe plans while incorporating perception uncertainties.

Our research presented in this dissertation mainly focuses on runtime monitoring and planning for single autonomous systems; however, multi-vehicle systems have also been gaining attention. Both homogeneous (e.g., robotic swarms) and heterogeneous multi-vehicle systems [44] have been increasingly used to handle complex tasks in which one vehicle may not be enough to complete. These multi-vehicle systems are subject to additional challenges such as communication uncertainties. Computational efficiency becomes even more critical as these vehicles also need to assess their neighboring agents' behavior. Guaranteeing the safety of vehicles during such operations is vital to complete a given task, and future work could also focus on scaling the presented approaches to multi-vehicle systems.

Autonomous robots have shown an immense potential to efficiently perform complex tasks and improve lives both in work and home environments. The runtime monitoring and proactive planning techniques explained in this dissertation take these systems one step closer to assured autonomy. I believe future interdisciplinary research that combines control theory, machine learning, and formal verification will continue making these systems more robust, proactive, efficient, and safer on the journey towards full autonomy.

Bibliography

[1] A. Ahmadzadeh et al. "Stable multi-particle systems and application in multi-vehicle path planning and coverage". In: *46th IEEE Conference on Decision and Control*. Dec. 2007, pp. 1467–1472.

[2] S. R. Ahmadzadeh, P. Kormushev, and D. G. Caldwell. "Multi-objective reinforcement learning for AUV thruster failure recovery". In: *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. 2014, pp. 1–8.

[3] Ibrahim Ahmed, Marcos Quinones-Grueiro, and Gautam Biswas. "Complementary Meta-Reinforcement Learning for Fault-Adaptive Control". In: *arXiv preprint arXiv:2009.12634* (2020).

[4] S. Bansal et al. "Hamilton-Jacobi reachability: A brief overview and recent advances". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. Dec. 2017, pp. 2242–2253.

[5] N. Bezzo et al. "A Cooperative Heterogeneous Mobile Wireless Mechatronic System". In: *IEEE/ASME Transactions on Mechatronics* 19.1 (Feb. 2014), pp. 20–31.

[6] N. Bezzo et al. "Online planning for energy-efficient and disturbance-aware UAV operations". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, pp. 5027–5033.

[7] Y. Bouzid, Y. Bestaoui, and H. Siguerdidjane. "Quadrotor-UAV optimal coverage path planning in cluttered environment with a limited onboard energy". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 979–984. DOI: 10.1109/IROS.2017.8202264.

[8] Manuel Castillo-Lopez et al. "Model Predictive Control for Aerial Collision Avoidance in Dynamic Environments". In: *2018 26th Mediterranean Conference on Control and Automation (MED)*. 2018, pp. 1–6. DOI: 10.1109/MED.2018.8442967.

[9] Ender Çetin et al. "Drone Navigation and Avoidance of Obstacles Through Deep Reinforcement Learning". In: *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. 2019, pp. 1–7. DOI: 10.1109/DASC43569.2019.9081749.

[10] Jing Chen, Tianbo Liu, and Shaojie Shen. "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1476–1483. DOI: 10.1109/ICRA.2016.7487283.

[11] Mo Chen et al. "Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems". In: *IEEE Transactions on Automatic Control* 63.11 (2018), pp. 3675–3688. DOI: 10.1109/TAC.2018.2797194.

[12] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. "Flow*: An Analyzer for Non-linear Hybrid Systems". In: *Computer Aided Verification*. Ed. by Natasha Sharygina and Helmut Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 258–263.

[13] Girish Chowdhary et al. "A Bayesian nonparametric approach to adaptive control using Gaussian Processes". In: *52nd IEEE Conference on Decision and Control*. 2013, pp. 874–879. DOI: 10.1109/CDC.2013.6759992.

[14] Robin Deits and Russ Tedrake. "Efficient mixed-integer planning for UAVs in cluttered environments". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 42–49. DOI: 10.1109/ICRA.2015.7138978.

[15] J. Ding et al. "Hybrid Systems in Robotics". In: *IEEE Robotics Automation Magazine* 18.3 (Sept. 2011), pp. 33–43.

[16] B. Djeridane and J. Lygeros. "Neural approximation of PDE solutions: An application to reachability computations". In: *Proc. of the 45th IEEE Conference on Decision and Control*. Dec. 2006, pp. 3034–3039.

[17] Eric van Doorn et al. "Statistics of wind direction and its increments". In: *Physics of Fluids* 12.6 (2000), pp. 1529–1534.

[18] Magnus Egerstedt et al. "On the regularization of Zeno hybrid automata". In: *Systems and Control Letters* 38 (1999), pp. 141–150.

[19] Fan Fei et al. "Learn-to-Recover: Retrofitting UAVs with Reinforcement Learning-Assisted Flight Control Under Cyber-Physical Attacks". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)* (2020), pp. 7358–7364.

[20] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. 2017, pp. 1126–1135.

[21] N. Gandhi et al. "Self-Reconfiguration in Response to Faults in Modular Aerial Systems". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2522–2529.

[22] S. S. Ge and Y. J. Cui. "Dynamic Motion Planning for Mobile Robots Using Potential Field Method". In: 13.3 (Nov. 2002), pp. 207–222. ISSN: 0929-5593. DOI: 10.1023/A:1020564024509. URL: https://doi.org/10.1023/A:1020564024509.

[23] Antoine Girard and George J. Pappas. "Approximate bisimulation relations for constrained linear systems". In: *Automatica* 43.8 (2007), pp. 1307–1317. ISSN: 0005-1098.

[24] S. L. Herbert et al. "FaSTrack: A modular framework for fast and guaranteed safe motion planning". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. Dec. 2017, pp. 1517–1522. DOI: 10.1109/CDC.2017.8263867.

[25] Sylvia Herbert et al. *Scalable Learning of Safety Guarantees for Autonomous Systems using Hamilton-Jacobi Reachability*. 2021. arXiv: 2101.05916 [cs.RO].

[26] Sylvia L. Herbert et al. "Reachability-Based Safety Guarantees using Efficient Initializations". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. 2019, pp. 4810–4816. DOI: 10.1109/CDC40024.2019.9029575.

[27] Zhiwei Hou, Peng Lu, and Zhangjie Tu. "Nonsingular terminal sliding mode control for a quadrotor UAV with a total rotor failure". In: *Aerospace Science and Technology* 98 (2020), p. 105716. ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2020.105716. URL: http://www.sciencedirect.com/science/article/pii/S1270963819316414.

[28] Nursultan Imanberdiyev et al. "Autonomous navigation of UAV by using real-time model-based reinforcement learning". In: *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2016, pp. 1–6. DOI: 10.1109/ICARCV.2016.7838739.

[29] R. Ivanov et al. "Verisig: verifying safety properties of hybrid systems with neural network controllers". In: *Proceedings of the 22nd International Conference on Hybrid Systems: Computation and Control* (2019).

[30] Girish Joshi and Girish Chowdhary. "Deep Model Reference Adaptive Control". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. 2019, pp. 4601–4608. DOI: 10.1109/CDC40024.2019.9029173.

[31] A. Al-Kaff et al. "Monocular vision-based obstacle detection/avoidance for unmanned aerial vehicles". In: *IEEE Intelligent Vehicles Symposium (IV)*. June 2016, pp. 92–97. DOI: 10.1109/IVS.2016.7535370.

[32] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894. DOI: 10.1177/0278364911406761.

[33] L.E. Kavraki et al. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580. DOI: 10.1109/70.508439.

[34] Azarakhsh Keipour, Mohammadreza Mousaei, and Sebastian Scherer. “Automatic Real-time Anomaly Detection for Autonomous Aerial Vehicles”. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 5679–5685. DOI: 10.1109/ICRA.2019.8794286.

[35] *keras*. https://keras.io.

[36] Oussama Khatib. “Real-Time Obstacle Avoidance for Manipulators and Mobile Robots”. In: *The International Journal of Robotics Research* 5.1 (1986), pp. 90–98. DOI: 10.1177/027836498600500106.

[37] Soonho Kong et al. “dReach: δ-Reachability Analysis for Hybrid Systems”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Christel Baier and Cesare Tinelli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 200–205.

[38] Shreyas Kousik et al. “Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots”. In: *arXiv preprint arXiv:1809.06746* (2018).

[39] A. A. Kurzhanskiy and P. Varaiya. “Ellipsoidal Toolbox (ET)”. In: *Proceedings of the 45th IEEE Conference on Decision and Control*. Dec. 2006, pp. 1498–1503.

[40] Alex A. Kurzhanskiy and Pravin Varaiya. “Ellipsoidal Techniques for Reachability Analysis of Discrete-Time Linear Systems”. In: *IEEE Transactions on Automatic Control* 52.1 (2007), pp. 26–38. DOI: 10.1109/TAC.2006.887900.

[41] Steven M. Lavalle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. 1998.

[42] Thomas Lew et al. *Safe Active Dynamics Learning and Control: A Sequential Exploration-Exploitation Framework*. 2021. arXiv: 2008.11700 [cs.RO].

[43] Q. Li et al. “Deep neural networks for improved, impromptu trajectory tracking of quadrotors”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 5183–5189.

[44] Tony X. Lin, Esen Yel, and Nicola Bezzo. “Energy-aware Persistent Control of Heterogeneous Robotic Systems”. In: *2018 Annual American Control Conference (ACC)*. 2018, pp. 2782–2787. DOI: 10.23919/ACC.2018.8431238.

[45] Björn Lindqvist et al. “Nonlinear MPC for Collision Avoidance and Control of UAVs With Dynamic Obstacles”. In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6001–6008. DOI: 10.1109/LRA.2020.3010730.

[46] S. Liu et al. “Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D Complex Environments”. In: *IEEE Robotics and Automation Letters* 2.3 (July 2017), pp. 1688–1695. ISSN: 2377-3766. DOI: 10.1109/LRA.2017.2663526.

[47] Yu Liu and Gang Tao. “Multivariable MRAC for aircraft with abrupt damages”. In: *2008 American Control Conference*. 2008, pp. 2981–2986. DOI: 10.1109/ACC.2008.4586949.

[48] Yu Liu, Gang Tao, and Suresh M. Joshi. “Modeling and Model Reference Adaptive Control of Aircraft with Asymmetric Damage”. In: *Journal of Guidance, Control, and Dynamics* 33.5 (2010), pp. 1500–1517. DOI: 10.2514/1.47996. eprint: https://doi.org/10.2514/1.47996. URL: https://doi.org/10.2514/1.47996.

[49] S. Lloyd. “Least squares quantization in PCM”. In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[50] T. T. Mac et al. “Improved potential field method for unknown obstacle avoidance using UAV in indoor environment”. In: *IEEE 14th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. Jan. 2016, pp. 345–350.

[51] Anirudha Majumdar and Russ Tedrake. “Funnel libraries for real-time robust feedback motion planning”. In: *The International Journal of Robotics Research* 36.8 (2017), pp. 947–982.

[52] Nick Malone et al. “Stochastic Reachability Based Motion Planning for Multiple Moving Obstacle Avoidance”. In: *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*. HSCC ’14. Berlin, Germany: ACM, 2014, pp. 51–60. ISBN: 978-1-4503-2732-9. DOI: 10.1145/2562059.2562127. URL: http://doi.acm.org/10.1145/2562059.2562127.

[53] L. Mazzara. “Risk-aware path planning and replanning algorithm for UAVs”. In: 2018.

[54] Christopher D McKinnon and Angela P Schoellig. “Meta Learning With Paired Forward and Inverse Models for Efficient Receding Horizon Control”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3240–3247.

[55] D. Mellinger and V. Kumar. “Minimum snap trajectory generation and control for quadrotors”. In: *IEEE International Conference on Robotics and Automation*. May 2011, pp. 2520–2525.

[56] N. Michael et al. “The GRASP Multiple Micro-UAV Testbed”. In: *IEEE Robotics Automation Magazine* 17.3 (Sept. 2010), pp. 56–65.

[57] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. “Methods for interpreting and understanding deep neural networks”. In: *Digital Signal Processing* 73 (2018), pp. 1–15.

[58] M. W. Mueller and R. D'Andrea. "Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 45–52.

[59] Anusha Nagabandi et al. "Learning to adapt: Meta-learning for model-based control". In: *Proc. of ICLR* (2019).

[60] S. R. Nekoo, J. Á. Acosta, and A. Ollero. "Collision Avoidance of SDRE Controller using Artificial Potential Field Method: Application to Aerial Robotics*". In: *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2020, pp. 551–556. DOI: 10.1109/ICUAS48674.2020.9213984.

[61] Michael Otte and Emilio Frazzoli. "RRT X: Real-Time Motion Planning/Replanning for Environments with Unpredictable Obstacles". In: *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 461–478.

[62] Ross T Palomaki et al. "Wind estimation in the lower atmosphere using multirotor aircraft". In: *Journal of Atmospheric and Oceanic Technology* 34.5 (2017), pp. 1183–1191.

[63] Harris Papadopoulos, Vladimir Vovk, and Alexander Gammerman. "Regression conformal prediction with nearest neighbours". In: *Journal of Artificial Intelligence Research* 40 (2011), pp. 815–840.

[64] Jungwon Park and H. Jin Kim. "Fast Trajectory Planning for Multiple Quadrotors using Relative Safe Flight Corridor". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 596–603. DOI: 10.1109/IROS40897.2019.8968502.

[65] G. A. S. Pereira, S. Choudhury, and S. Scherer. "A framework for optimal repairing of vector field-based motion plans". In: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. June 2016, pp. 261–266. DOI: 10.1109/ICUAS.2016.7502525.

[66] P. Pettersson and P. Doherty. "Probabilistic Roadmap Based Path Planning for an Autonomous Unmanned Aerial Vehicle". In: 2004.

[67] A. Punjani and P. Abbeel. "Deep learning helicopter dynamics models". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 3223–3230. DOI: 10.1109/ICRA.2015.7139643.

[68] Ahmed Hussain Qureshi et al. "Motion Planning Networks: Bridging the Gap Between Learning-Based and Classical Motion Planners". In: *IEEE Transactions on Robotics* 37.1 (2021), pp. 48–66. DOI: 10.1109/TRO.2020.3006716.

[69] Carl Edward Rasmussen. "Gaussian processes for machine learning". In: MIT Press, 2006.

[70] Carl Edward Rasmussen and Hannes Nickisch. "Gaussian Processes for Machine Learning (GPML) Toolbox". In: *The Journal of Machine Learning Research* 11 (Dec. 2010), pp. 3011–3015. ISSN: 1532-4435.

[71] SM Richards et al. “Adaptive-Control-Oriented Meta-Learning for Nonlinear Systems”. In: *Robotics science and systems*. 2021.

[72] Vicenç Rúbies Royo et al. “Classification-based Approximate Reachability with Guarantees Applied to Safe Trajectory Tracking”. In: *CoRR* abs/1803.03237 (2018). URL: http://arxiv.org/abs/1803.03237.

[73] M. C. P. Santos et al. “A Novel Null-Space-Based UAV Trajectory Tracking Controller With Collision Avoidance”. In: *IEEE/ASME Transactions on Mechatronics* 22.6 (Dec. 2017), pp. 2543–2553. ISSN: 1083-4435. DOI: 10.1109/TMECH.2017.2752302.

[74] M. L. Schrum and M. C. Gombolay. “When Your Robot Breaks: Active Learning During Plant Failure”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 438–445.

[75] Mariah L Schrum et al. “Meta-active Learning in Probabilistically-Safe Optimization”. In: *arXiv preprint arXiv:2007.03742* (2020).

[76] Hoseong Seo et al. “Robust Trajectory Planning for a Multirotor against Disturbance based on Hamilton-Jacobi Reachability Analysis”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 3150–3157. DOI: 10.1109/IROS40897.2019.8968126.

[77] D.H. Shim, Hoam Chung, and S.S. Sastry. “Conflict-free navigation in unknown urban environments”. In: *IEEE Robotics Automation Magazine* 13.3 (2006), pp. 27–33. DOI: 10.1109/MRA.2006.1678136.

[78] S. Singh et al. “Robust online motion planning via contraction theory and convex optimization”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 5883–5890. DOI: 10.1109/ICRA.2017.7989693.

[79] Abhik Singla, Sindhu Padakandla, and Shalabh Bhatnagar. “Memory-Based Deep Reinforcement Learning for Obstacle Avoidance in UAV With Limited Environment Knowledge”. In: *IEEE Transactions on Intelligent Transportation Systems* 22.1 (2021), pp. 107–118. DOI: 10.1109/TITS.2019.2954952.

[80] Sean Summers et al. “A Stochastic Reach-Avoid Problem with Random Obstacles”. In: *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*. HSCC ’11. Chicago, IL, USA: Association for Computing Machinery, 2011, pp. 251–260. ISBN: 9781450306294. DOI: 10.1145/1967701.1967738. URL: https://doi.org/10.1145/1967701.1967738.

[81] S. Sun et al. “High-Speed Flight of Quadrotor Despite Loss of Single Rotor”. In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3201–3207.

[82] S. Sun et al. “Incremental Nonlinear Fault-Tolerant Control of a Quadrotor With Complete Loss of Two Opposing Rotors”. In: *IEEE Transactions on Robotics* (2020), pp. 1–15.

[83] Jun Tang et al. “Optimized artificial potential field algorithm to multi-unmanned aerial vehicle coordinated trajectory planning and collision avoidance in three-dimensional environment”. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 233.16 (2019), pp. 6032–6043. DOI: 10.1177/0954410019844434.

[84] *Tensorflow*. https://www.tensorflow.org.

[85] D. Tzoumanikas, Q. Yan, and S. Leutenegger. “Nonlinear MPC with Motor Failure Identification and Recovery for Safe and Aggressive Multicopter Flight”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 8538–8544.

[86] Abraham P. Vinod, Baisravan Homchaudhuri, and Meeko M. K. Oishi. “Forward stochastic reachability analysis for uncontrolled linear systems using Fourier Transforms”. In: *CoRR* abs/1610.04550 (2016). URL: http://arxiv.org/abs/1610.04550.

[87] Ory Walker et al. “A Deep Reinforcement Learning Framework for UAV Navigation in Indoor Environments”. In: *2019 IEEE Aerospace Conference*. 2019, pp. 1–14. DOI: 10.1109/AERO.2019.8742226.

[88] Keyu Wu et al. “Achieving Real-Time Path Planning in Unknown Environments Through Deep Neural Networks”. In: *IEEE Transactions on Intelligent Transportation Systems* (2020), pp. 1–10. DOI: 10.1109/TITS.2020.3031962.

[89] Weiming Xiang et al. “Verification for machine learning, autonomy, and neural networks survey”. In: *arXiv preprint arXiv:1810.01989* (2018).

[90] E. Yel and N. Bezzo. “Reachability-based Adaptive UAV Scheduling and Planning in Cluttered and Dynamic Environments”. In: *Workshop on Informative Path Planning and Adaptive Sampling at ICRA*. May 2018. URL: http://robotics.usc.edu/~wippas/program.html.

[91] Esen Yel and Nicola Bezzo. “A Meta-Learning-based Trajectory Tracking Framework for UAVs under Degraded Conditions”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (to appear)*. 2021, ©IEEE.

[92] Esen Yel and Nicola Bezzo. “Fast run-time monitoring, replanning, and recovery for safe autonomous system operations”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, 1661–1667, ©IEEE. DOI: 10.1109/IROS40897.2019.8968498.

[93] Esen Yel and Nicola Bezzo. “GP-based Runtime Planning, Learning, and Recovery for Safe UAV Operations under Unforeseen Disturbances”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, 2173–2180, ©IEEE. DOI: 10.1109/IROS45743.2020.9341641.

[94] Esen Yel, Tony X Lin, and Nicola Bezzo. "Computation-Aware Adaptive Planning and Scheduling for Safe Unmanned Airborne Operations". In: *Journal of Intelligent & Robotic Systems* 100.2 (2020), 575–596, ©Springer. DOI: 10.1007/s10846-020-01192-2.

[95] Esen Yel, Tony X. Lin, and Nicola Bezzo. "Reachability-based self-triggered scheduling and replanning of UAV operations". In: *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2017, 221–228, ©IEEE. DOI: 10.1109/AHS.2017.8046382.

[96] Esen Yel, Tony X. Lin, and Nicola Bezzo. "Self-triggered Adaptive Planning and Scheduling of UAV Operations". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, 7518–7524, ©IEEE. DOI: 10.1109/ICRA.2018.8463205.

[97] Esen Yel et al. "Assured runtime monitoring and planning: Toward verification of neural networks for safe autonomous operations". In: *IEEE Robotics & Automation Magazine* 27.2 (2020), 102–116, ©IEEE. DOI: 10.1109/MRA.2020.2981114.

[98] S. Zhou and A. P. Schoellig. "Active Training Trajectory Generation for Inverse Dynamics Model Learning with Deep Neural Networks". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. 2019, pp. 1784–1790. DOI: 10.1109/CDC40024.2019.9029973.

[99] Y. Zhou, A. Raghavan, and J. S. Baras. "Time varying control set design for UAV collision avoidance using reachable tubes". In: *IEEE 55th Conference on Decision and Control (CDC)*. Dec. 2016, pp. 6857–6862.