A ROS Package for UAV Run Time Assurance with In-the-Loop Reachability

Christian Llanes* and Samuel Coogan †
Georgia Institute of Technology, Atlanta, GA

This work describes an open source software package for run time assurance (RTA) of UAVs to verify safety in the form of collision avoidance. An operator designs a primary controller with possible learning-enabled components or with human inputs. Learning-based control design is inherently unverified and the RTA supervises the control behavior during the learning process. The proposed RTA package guarantees collision avoidance of obstacles while acting as a supervisor for an operator's primary controller. The RTA mechanism uses control barrier functions (CBFs) with reachability analysis of the UAV dynamics to detect unsafe control actions from the primary controller and solves an optimization problem to minimally adjust desired control inputs to ensure that collision-bound trajectories are avoided. We use the Robot Operating System (ROS) middleware as a framework for designing the software package. We describe the main underlying algorithm and its implementation as a ROS2 package, and we demonstrate its use in hardware experiments.

I. Introduction

Unmanned aerial vehicles (UAVs) have become increasingly accessible over the past decade as a result of cheap, small scale, and high computing capabilities of onboard computers. An increase in the number of UAVs in operation over dense urban environments increases the probability and impact of accidents involving UAVs. As a result, there is a growing need for safety verification and assurances of UAVs operating in dense urban environments. One method for guaranteeing safety of a controlled dynamical system is to filter a potentially unsafe primary controller at run time. This approach is referred to as run time assurance (RTA) [1]. There are several ways to design an RTA mechanism and in this work we use control barrier functions (CBFs) [2, 3] to minimally adjust a desired control input while ensuring forward invariance of a predefined safe subregion of the state space.

Classical CBF formulations lead to forward invariance for an a priori defined subregion of the state space. Designing a subset that is viable can be challenging when considering control constraints and the relative degree of a dynamical system. The works [4–6] present an RTA mechanism that uses CBFs with a simulation of the system dynamics under a backup control policy for a user-defined prediction horizon. The simulation, which is done prior to evaluating the CBF constraint, allows for designing a smaller and usually more conservative forward invariant viability kernel. Lyapunov functions are one method for designing the viability kernel. This enables easier verification of the viability kernel and from the simulation of the backup control policy we can verify larger safe forward invariant regions of the state space if the system returns back to the viability kernel within some prediction horizon. For systems with disturbances the resulting simulation is a reachable set and computing reachable sets online is generally computationally intractable. An approach proposed in [7] is to use the theory of mixed monotone systems to compute hyperrectangular over-approximations of the system's reachable set. This approach requires computing a single solution of a related embedding system that relates to the extreme solutions of the original system. Using this approach we considerably reduce the computational complexity of the RTA algorithm which enables a fast in-the-loop reachability analysis at run time for highly dynamic systems such as UAVs.

Our approach to designing an RTA for collision avoidance of quadrotor UAVs is as follows. First, we design a backup control strategy that is continually simulated for a user-defined prediction horizon that slows the vehicle down and returns the vehicle to the initial position in the prediction horizon. Since we consider unknown disturbances in the UAV dynamics, the system evolution is a *reachable set* of possible future states. Secondly, the objective of our RTA mechanism is to ensure that the system's reachable set never intersects with obstacles. To do this we use CBFs to construct inequality constraints for an optimization problem that aims to minimize the 2-norm of the difference between the desired control input and the applied input.

*Graduate Research Assistant, Electrical and Computer Engineering Department, AIAA Student Member
†Associate Professor, Electrical and Computer Engineering Department

The main contribution of this paper is a software package that implements an RTA mechanism that supervises a primary objective controller and filters control input commands that would lead to a collision with obstacles. The Robot Operating System (ROS) middleware is used as a framework for designing the software package due to its message-passing services and modular design. The RTA is implemented as a ROS2 software package for collision avoidance that interfaces with a flight management unit (FMU) running PX4 autopilot. The user provides a primary controller in the form of a ROS2 node to reach an objective with possible obstacles in the desired trajectory. The primary controller publishes control input commands to the RTA ROS2 node which filters the control commands to ensure collision-bound trajectories with obstacles are avoided. Other RTA frameworks [8] have been developed on ROS systems for safe distributed mobile robots using the Simplex architecture [9, 10].

In our prior work [11], we demonstrated a similar RTA designed for collision avoidance between two multirotors. For the case study in that work we construct the backup control policy as nonlinear springs coupled between multirotors in the Y-Z coordinate axes. We presented a successful hardware demonstration to filter an unsafe primary controller that would have led to a collision between the multirotors if not for our RTA mechanism. This motivates the present work for obstacle avoidance where we now consider a single multirotor controlled by a primary controller with some objective, *e.g.*, flying within an urban environment. The goal of the RTA mechanism is to predict possibly unsafe control decisions from the primary controller and minimally alter them to ensure the vehicle avoids obstacles while satisfying the objective. This objective is particularly relevant for the emerging topic of urban air mobility (UAM), where UAVs carrying cargo or passengers operate in an urban environment. Safety verification for UAM is an important subsystem that is crucial for meeting flight regulations of autonomously controlled aerial vehicles. This paper provides a proof of concept for creating an RTA software package that requires minimal configuration for interfacing with a primary controller.

The rest of this paper is organized as follows: We begin with a theoretical discussion for the algorithm and provide an implementation for the algorithm in pseudocode. We then discuss UAV dynamics and the backup control strategy used in the RTA algorithm for reachability analysis. Then, we provide a description for the software package architecture. This includes a discussion of the file directory and a block diagram for the ROS2 RTA software package. Finally, we demonstrate our results via a hardware demonstration of the ROS2 software package for a quadcopter UAV with a position controller and obstacles in the reference trajectory.

II. Notation

We denote vector entries via subscript, *i.e.*, x_i for $i \in \{1, \cdots, n\}$ denotes the i^{th} entry of $x \in \mathbb{R}^n$. Given $x, y \in \mathbb{R}^n$ with $x_i \leq y_i$ for all i,

$$[x,\ y] := \{z \in \mathbb{R}^n \ \mid\ x_i \leq z_i \leq y_i \text{ for all } i\}$$

denotes the hyperrectangle with endpoints x and y, and

$$\langle\!\langle x,\ y \rangle\!\rangle := \{z \in \mathbb{R}^n \ \mid\ z_i \in \{x_i,\ y_i\} \text{ for all } i\}$$

denotes the finite set of 2^n vertices of $[x,\ y]$. We also allow $x_i \in \mathbb{R} \cup \{-\infty\}$ and $y_i \in \mathbb{R} \cup \{\infty\}$ so that $[x,\ y]$ defines an *extended hyperrectangle*, that is, a hyperrectangle with possibly infinite extent in some coordinates.

Let $(x,\ y)$ denote the vector concatenation of $x,\ y \in \mathbb{R}^n$, *i.e.*, $(x,\ y) := [x^T\ y^T]^T \in \mathbb{R}^{2n}$. Given $a = (x, y) \in \mathbb{R}^{2n}$ with $x_i \leq y_i$ for all i, we denote by $[\![a]\!]$ the hyperrectangle formed by the first and last n components of x, *i.e.*, $[\![a]\!] := [x,\ y]$, and similarly $\langle\!\langle a \rangle\!\rangle := \langle\!\langle x,\ y \rangle\!\rangle$.

III. Algorithm Overview

Reachability in RTA involves verifying that the system's reachable set is fully contained in a viability kernel for some time in the prediction horizon. In Fig. 1 a rectangular unsafe region is defined in red and an elliptical viability kernel is defined in green. A mixed monotonicity derived rectangular over-approximation of the system's reachable set is depicted in pink. In this section, we begin by defining the theoretical results from our prior work [11] along with the RTA algorithm as pseudocode used in the software package.

We consider an affine-in-control and affine-in-disturbance nondeterministic dynamical system model of a UAV of the form

$$\dot{x} = f(x) + g_1(x)u + g_2(x)w \tag{1}$$

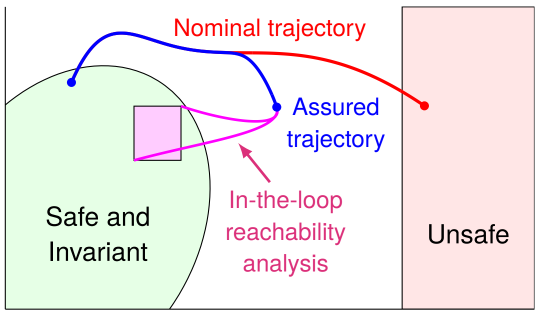


Fig. 1 An example of run time assurance with a reachability analysis to ensure the system's reachable set is fully contained in the safe and invariant subregion of the state space.

where $x \in \mathcal{X} \subset \mathbb{R}^n$ is the state, $u \in \mathcal{U} \subseteq \mathbb{R}^m$ is the control input and $w \in \mathcal{W} \subset \mathbb{R}^p$ is the disturbance, and we assume throughout this section that the state space $\mathcal{X}$ is an extended hyperrectangle and the disturbance space $\mathcal{W} := [\underline{w}, \overline{w}]$ is a hyperrectangle.

The proposed approach to run time assurance described below utilizes a backup control strategy $\mathbf{u}^{\mathrm{b}} : \mathcal{X} \to \mathcal{U}$ that results in closed-loop backup dynamics written as

$$\dot{x} = F^{\mathrm{b}}(x, w) := f(x) + g_1(x)\mathbf{u}^{\mathrm{b}}(x) + g_2(x)w. \tag{2}$$

Computing the reachable set for Eq. (2) through Monte Carlo approaches online is computationally intractable. We apply the theory of mixed monotone systems to decompose the dynamics and evaluate a singular trajectory of the extreme solutions that approximates the true reachable set as a hyperrectangle. The corners of the hyperrrectangle are defined by the pair $(x, \hat{x})$. The backup dynamics defined by Eq. (2) are mixed monotone with respect to a decomposition function d. For more information on mixed monotone systems we direct the reader to the tutorial paper[12] and our past work that details the use of mixed monotonicity for RTA [11]. Given a decomposition function d,

$$\begin{bmatrix} \dot{x} \\ \dot{\widehat{x}} \end{bmatrix} = e(x, \widehat{x}) := \begin{bmatrix} d(x, \underline{w}, \widehat{x}, \overline{w}) \\ d(\widehat{x}, \overline{w}, x, \underline{w}) \end{bmatrix} \tag{3}$$

is the *embedding system relative to* d and e is the *embedding function relative to* d. Note that the embedding system contains no disturbances as d is evaluated only at the extrema of $\mathcal{W} = [\underline{w}, \overline{w}]$.

We denote by $\Phi^{\mathrm{b}}(t;\, x, \mathbf{w})$ the state of Eq. (2) reached at time $t \geq 0$ when starting from state x at time 0 and when evolving subject to the disturbance signal $\mathbf{w} : [0, t] \to \mathcal{W}$. Furthermore, we denote by $\Phi^{e}(t; a)$ the state of Eq. (3) reached at time $t \geq 0$ when starting from state $a \in \mathcal{X} \times \mathcal{X}$ at time zero. As a shorthand notation we write $\Phi^{e}_{t}(x) := \Phi^{e}(t; (x, x))$ for $x \in \mathcal{X}$.

Additionally, we denote

$$R^{\mathrm{b}}(t;\, x) := \{\Phi^{\mathrm{b}}(t;\, x,\, \mathbf{w}) \in \mathcal{X} \mid \mathbf{w} : [0,\, t] \to \mathcal{W}\} \tag{4}$$

as the time-t backup reachable set of Eq. (2) from initial state $x \in \mathcal{X}$. We assume given a viability kernel $S^{\mathrm{b}} = \{x \in \mathcal{X} \mid h(x) \geq 0\} \subset \mathcal{X} \setminus \mathcal{X}_{\mathrm{u}}$ where $h : \mathbb{R}^n \to \mathbb{R}$ is assumed to be a continuously differentiable and concave function. The goal of our RTA mechanism is to ensure that the backup reachable set is always fully contained in the viability kernel $R^{\mathrm{b}}(t;\, x) \subseteq S^{\mathrm{b}}$ for all $t \leq T_{\mathrm{b}}$ where T_{b} is a user defined backup prediction horizon.

Define

$$\gamma^{\mathrm{ideal}}(t;\, x) := \inf_{z \in [\![\Phi^{e}_{t}(x)]\!]} h(z) = \min_{z \in \langle\!\langle \Phi^{e}_{t}(x) \rangle\!\rangle} h(z), \tag{5}$$

for a hyperrectangular reachable set at time t in the embedding trajectory $[\![\Phi^{e}_{t}(x)]\!]$ and where the second equality comes from the concavity on h. Further define

$$\Psi^{\mathrm{ideal}}(x) := \sup_{0 \leq \tau \leq T_{\mathrm{b}}} \gamma^{\mathrm{ideal}}(\tau;\, x) \tag{6}$$

for a prediction horizon T_{b}. CBF algorithms require the utilized functions to be differentiable, whereas γ^{ideal} and Ψ^{ideal} are generally not differentiable due to the minimum operator in Eq. (5). Therefore, we use a numerically stable

Algorithm 1 Runtime Assurance for Nondeterministic Control Systems

```
input      : Current State $x \in \mathcal{X}$.
           : Desired control policy $\mathbf{u}^{\mathrm{d}} : \mathcal{X} \to \mathbb{R}^m$.
           : Previous maximizer time $t^*_{k-1} \in \mathbb{R}$.
output     : Assured control input $\mathbf{u}^{\mathrm{RTA}} \in \mathcal{U}$.
predefined : Class-$\mathcal{K}$ function $\alpha : \mathbb{R} \to \mathbb{R}$.

 1: function $\mathbf{u}^{\mathrm{RTA}}$ =RTA($\mathbf{u}^{\mathrm{d}}, x, t^*_{k-1}$)
 2:     if $t^*_{k-1}$ initialized then
         timevec ← Point_Dist($t^*_{k-1}$)
 3:     else timevec ← linspace($0, T_{\mathrm{b}}$)
 4:     for all $t \in$ timevec compute $\Phi^e_t$ from (3)
 5:     $\Gamma \leftarrow \varnothing$
 6:     for all $t \in$ timevec append $\gamma(t; x)$ from (9) to $\Gamma$
 7:     [$\Psi(x)$, max_psi_indx] ← max($\Gamma$)
 8:     $t^* \leftarrow$ timevec[max_psi_indx]
 9:     compute: $\frac{\partial \Psi}{\partial x}(x)$ as in (11)
10:     compute: $u^* = \arg\min_{u \in [\underline{u}, \overline{u}]} ||u - u^{\mathrm{d}}||_2^2$
           s.t. $\frac{\partial \Psi}{\partial x}(x)(f(x) + g_1(x)u + g_2(x)w) \geq -\alpha(\Psi(x))$
           $\forall w \in \langle\langle \underline{w}, \overline{w} \rangle\rangle$
11:     if Program feasible then return $u^*$
12:     else return $\mathbf{u}^{\mathrm{b}}(x)$
13: end function
```

continuously differentiable soft-min function known as the Log-Sum-Exponential (LSE) that approximates $\min \mathcal{S}$. For some fixed parameter $p > 0$ that controls the tightness of the approximation, define

$$\mathrm{LSE}(\mathcal{S}) = -\frac{1}{p} \log \sum_{s \in \mathcal{S}} \exp(-p \cdot s). \tag{7}$$

We further denote the LSE over a set of evaluations of a barrier function h on the corners of a hyperrectangle a by

$$\mathrm{LSE}_h(a) := \mathrm{LSE}(\, \{h(z) \mid z \in \langle\langle a \rangle\rangle \}\,). \tag{8}$$

The LSE provides a differentiable relaxation for γ^{ideal} and Ψ^{ideal} which we denote by γ and Ψ given by

$$\gamma(t;\, x) := \mathrm{LSE}_h(\Phi^e_t(x)) \tag{9}$$

$$\Psi(x) := \sup_{0 \leq \tau \leq T_{\mathrm{b}}} \gamma(\tau;\, x). \tag{10}$$

Next, we differentiate $\Psi(x)$ with respect to x to obtain

$$\frac{\partial \Psi}{\partial x}(x) = \frac{\partial \gamma}{\partial x}(t^*, x) = \frac{\partial \mathrm{LSE}_h}{\partial a}(\Phi^e_{t^*}(x)) \frac{\partial \Phi^e_{t^*}}{\partial x}(x). \tag{11}$$

where t^* is the time that achieves the maximum in Eq. (10) and the first equality holds by [13, Theorem 1]. The derivative $\frac{\partial \mathrm{LSE}_h}{\partial a}(\cdot)$ is computed from Eq. (7) and Eq. (8) and is evaluated at the embedding state $\Phi^e_{t^*}(x)$ at time $t^* \leq T_{\mathrm{b}}$. The derivative $\frac{\partial \Phi^e_{t^*}}{\partial x}(x)$ is computed efficiently via a sensitivity matrix $S(t) \in \mathbb{R}^{2nxn}$ as explained in [14].

We provide a pseudocode for the proposed RTA software package in Algorithm 1. A point distribution function, `Point_Dist`, is designed by the user. For the proposed software package, we use an efficient logarithm-based point distribution algorithm that uses the maximizer t^*_{k-1} in Eq. (10) from the previous iteration. The convex optimization problem defined in line 10 is solved using an operator splitting quadratic program solver [15]. The algorithm decides when to intervene via the Class-$\mathcal{K}$ function α and in our implementation we use a linear function.

IV. UAV Dynamics

In this section, we present the UAV system dynamics and the backup control policy used for the UAV collision avoidance software package.

The twelve states for modeling multirotor dynamics are position $\boldsymbol{p} \in \mathbb{R}^3$, velocity $\boldsymbol{v} \in \mathbb{R}^3$, orientation using Euler angles $\boldsymbol{E} \in [-\pi, \pi]^3$, and body angular rates $\boldsymbol{\omega} \in \mathbb{R}^3$. In expanded form, the states appear as

$$\begin{aligned} \boldsymbol{x} &= [\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{E}, \boldsymbol{\omega}]^T \\ \boldsymbol{p} &= [p_x, p_y, p_z] \\ \boldsymbol{v} &= [v_x, v_y, v_z] \\ \boldsymbol{E} &= [\phi, \theta, \psi] \\ \boldsymbol{\omega} &= [\omega_x, \omega_y, \omega_z]. \end{aligned} \tag{12}$$

The control input vector for the multirotor is $\boldsymbol{u} = [c, M_x, M_y, M_z]^T$ where $c \in \mathbb{R}$ is the mass-normalized collective motor thrust and M_x, M_y, M_z are the moments on the multirotor in the body-fixed frame. We use the standard North-East-Down (NED) inertial reference frame $\{\vec{e}_x, \vec{e}_y, \vec{e}_z\}$ and the standard aerospace forward-right-down (FRD) vehicle body-fixed frame $\{\vec{b}_x, \vec{b}_y, \vec{b}_z\}$. The multirotor dynamics are modeled as a rigid body with dynamics

$$\begin{aligned} \dot{\boldsymbol{p}} &= \boldsymbol{v} \\ \dot{\boldsymbol{v}} &= g\vec{e}_z - \boldsymbol{R}_B^E \vec{b}_z c \\ \dot{\boldsymbol{E}} &= \boldsymbol{\Gamma}(\boldsymbol{E})^{-1}\boldsymbol{\omega} \\ \dot{\boldsymbol{\omega}} &= \boldsymbol{J}^{-1}(\boldsymbol{M} - \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega}) \end{aligned} \tag{13}$$

where the matrix $\boldsymbol{J} \in \mathbb{R}^{3\times 3}$ is the moment of inertia tensor and the matrix $\boldsymbol{R}_B^E$ and $\boldsymbol{\Gamma}(\boldsymbol{E})$ are defined below. For most multirotors, we simplify the moment of inertia tensor by observing that multirotors are symmetric about their principal axes. Therefore, all non diagonal elements of $\boldsymbol{J}$ are zero. The moment of inertia tensor with the symmetry assumption for the multirotor is given by

$$\boldsymbol{J} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}. \tag{14}$$

The transformation matrix $\boldsymbol{R}_B^E \in \mathbb{R}^{3\times 3}$ is the inverse or transpose of the aerospace direction cosine matrix (DCM) yaw-pitch-roll sequence mapping vectors in the body-fixed frame to the earth frame. The matrix $\boldsymbol{\Gamma}(\boldsymbol{E})$ is obtained by applying intermediate frame rotations to the Euler rates giving the resultant matrix

$$\boldsymbol{\Gamma}(\boldsymbol{E}) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \tag{15}$$

that transforms Euler rates to vehicle body angular rates.

In this work, we assume that an inner loop controller can track desired body angular rates at much faster timescales as opposed to the other states. Therefore, we assume we have instantaneous control of the body angular rates which allows us to turn the body angular rates from states to control inputs. This assumption leads to system dynamics that are now nine-dimensional given by

$$\begin{aligned} \dot{\boldsymbol{p}} &= \boldsymbol{v} \\ \dot{\boldsymbol{v}} &= g\vec{e}_z - \boldsymbol{R}_B^E \vec{b}_z c \\ \dot{\boldsymbol{E}} &= \boldsymbol{\Gamma}(\boldsymbol{E})^{-1}\boldsymbol{\omega} \end{aligned} \tag{16}$$

and the resulting control input vector is $\boldsymbol{u} = [c, \omega_x, \omega_y, \omega_z]^T$ for this nine-dimensional system. This particular control assumption has proven successful in the works [16–18].

The backup control policy for the multirotor UAV is designed to slow the vehicle down and return to the current position which is assumed to be safe. The first step in the backup control policy is to define in Eq. (17) the desired normalized forces on the vehicle given by

$$\begin{aligned} a_{\mathrm{y}} &= k_{\mathrm{p}}(x_{\mathrm{des}} - x) - bv_x \\ a_{\mathrm{y}} &= k_{\mathrm{p}}(y_{\mathrm{des}} - y) - bv_y \\ a_{\mathrm{z}} &= k_{\mathrm{p}}(z_{\mathrm{des}} - z) - g - bv_z. \end{aligned} \tag{17}$$

Then, we define the desired acceleration direction $\boldsymbol{a}_{\mathrm{dir}} = \boldsymbol{a}/||\boldsymbol{a}||_2$ as a normalization of the desired acceleration vector on the multirotor. We then apply intermediate rotations using the DCM to obtain pitch and roll commands given by

$$\begin{aligned} \zeta_1 &= \boldsymbol{R}_E^B(0, 0, \psi)\boldsymbol{a}_{\mathrm{dir}} \\ \theta_{\mathrm{des}} &= \operatorname{atan2}(-\zeta_{1_{\mathrm{x}}}, -\zeta_{1_{\mathrm{z}}}) \\ \zeta_2 &= \boldsymbol{R}_E^B(0, \theta_{\mathrm{des}}, 0)\zeta_1 \\ \phi_{\mathrm{des}} &= \operatorname{atan2}(\zeta_{2_{\mathrm{y}}}, -\zeta_{2_{\mathrm{z}}}). \end{aligned} \tag{18}$$

The resulting control input commands for the backup control policy for a desired yaw angle of zero are

$$\begin{aligned} c &= ||\boldsymbol{a}||_2 \\ \omega_x &= -k_{\mathrm{p_{roll}}}(\phi - \phi_{\mathrm{des}}) \\ \omega_y &= -k_{\mathrm{p_{pitch}}}(\theta - \theta_{\mathrm{des}}) \\ \omega_z &= -k_{\mathrm{p_{yaw}}}\psi. \end{aligned} \tag{19}$$

V. Run Time Assurance ROS2 Software Package for Collision Avoidance

The main contribution of this work is to provide a UAV RTA ROS2 software package named `mav_obstacle_avoidance`[*]. The choice of ROS2 is due to its popularity in robotics and an existing bridge for PX4 autopilot that allows for directly publishing to the FMU. The PX4-ROS2 bridge is called `px4_ros_com` and it allows connectivity between the message-passing services on ROS2 and the FMU. This bridge allows for ROS2 development and direct access to control and status messages on the FMU. ROS2 provides a modular framework for separating the primary controller software from the RTA software. The modular connectivity is done through message-passing services provided by ROS2. A diagram is provided in Fig. 2 with the relevant ROS2 nodes in this work. An operator provides a primary controller in the form of a ROS2 node (e.g. `primary_controller_node`) and publishes control commands to `mav_obstacle_avoidance_node` through the `rta_control` message topic. The `px4_ros_com` node contains additional subscriber and publishers that are not included in Fig. 2 because they are not used for obstacle avoidance.

The file directory tree for the `mav_obstacle_avoidance` software package is outlined in Fig. 3. The main folders are `config`, `include`, `launch`, and `src`. The `config` folder contains parameter files for initialization. The `include` folder contains header files. The `launch` folder contains launch files for the ROS2 system necessary for specifying parameter files and executables. The `src` folder contains the main source and library files for the software package. Compilation is handled by CMake and configured through the `CMakeLists.txt` file.

[*]The ROS2 software package `mav_obstacle_avoidance` is publicly available through the GaTech FACTS Lab Github: `github.com/gtfactslab/Llanes_AIAA2023`.

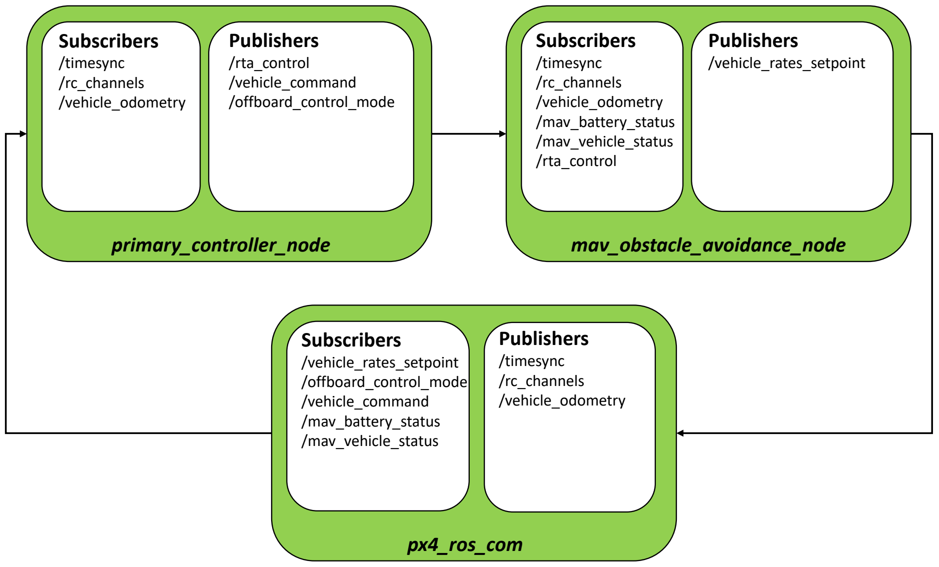
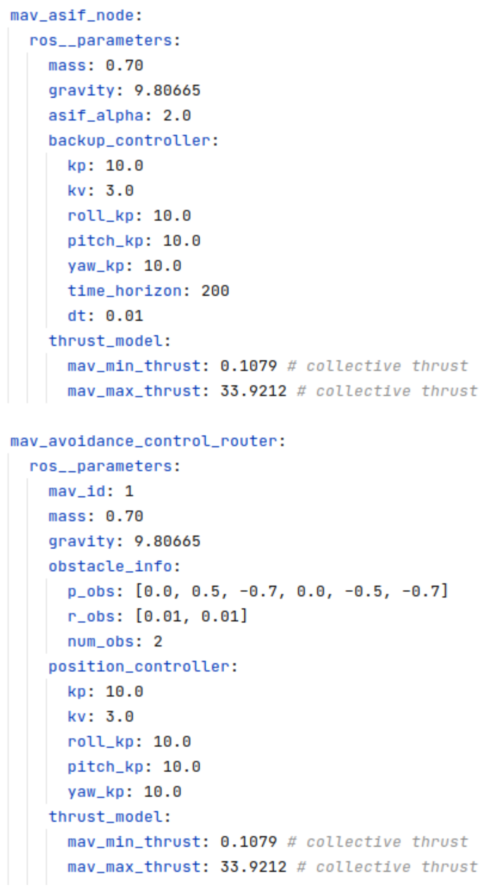


Fig. 2 A diagram illustrating the ROS2 node structure with publishers and subscribers for each node.

```
mav_obstacle_avoidance
├─ config
│  └─ params.yaml
├─ include
│  └─ mav_obstacle_avoidance
│     ├─ mav_asif_util.hpp
│     ├─ mav_avoidance_control_router.hpp
│     └─ mav_util.h
├─ launch
│  └─ mav_obstacle_avoidance_launch.py
├─ src
│  ├─ lib
│  │  ├─ mav_asif_util.cpp
│  │  └─ mav_avoidance_control_router.hpp
│  └─ mav_obstacle_avoidance_control_router.hpp
├─ CMakeLists.txt
└─ package.xml
```

Fig. 3 Directory tree for the `mav_obstacle_avoidance` software package.

Algorithm parameters are defined outside the source code for fast tuning without the need to recompile source code. The parameters are defined in a YAML file called `params.yaml`. The parameter file is shown in Fig. 4. For multiagent capabilities we define a `mav_id` parameter that is unique for each vehicle. This id is used to uniquely identify each vehicle in the ROS2 network. The obstacle primitives are defined as spheres and we plan to extend this further for Lidar measurements. Obstacles are defined through the `p_obs` and `r_obs` parameters for position and radius. To add a new obstacles the user extends the `p_obs` vector with the x, y, and z coordinates of the new obstacle and the radius in the `r_obs` vector. Controller gains can also be adjusted through the `backup_controller` parameters.



```yaml
mav_asif_node:
  ros__parameters:
    mass: 0.70
    gravity: 9.80665
    asif_alpha: 2.0
    backup_controller:
      kp: 10.0
      kv: 3.0
      roll_kp: 10.0
      pitch_kp: 10.0
      yaw_kp: 10.0
      time_horizon: 200
      dt: 0.01
    thrust_model:
      mav_min_thrust: 0.1079 # collective thrust
      mav_max_thrust: 33.9212 # collective thrust

mav_avoidance_control_router:
  ros__parameters:
    mav_id: 1
    mass: 0.70
    gravity: 9.80665
    obstacle_info:
      p_obs: [0.0, 0.5, -0.7, 0.0, -0.5, -0.7]
      r_obs: [0.01, 0.01]
      num_obs: 2
    position_controller:
      kp: 10.0
      kv: 3.0
      roll_kp: 10.0
      pitch_kp: 10.0
      yaw_kp: 10.0
    thrust_model:
      mav_min_thrust: 0.1079 # collective thrust
      mav_max_thrust: 33.9212 # collective thrust
```

Fig. 4 Parameters for the `mav_obstacle_avoidance` software package defined in `params.yaml`.

The software package is run through a ROS2 launch file in the launch folder. After compiling the package the launch file can be run by the command:

```
$ ros2 launch mav_obstacle_avoidance mav_obstacle_avoidance_launch.py
```

VI. Results

To test our RTA mechanism we construct a primary controller similar to the backup controller in Section IV for a quadrotor. The goal of the primary controller is to track a desired circular trajectory. However, two obstacles on opposite sides of the circular trajectory would lead to a collision if the primary controller is allowed to control the quadrotor directly. The goal of our RTA mechanism is to detect the collision and smoothly alter the control input to avoid the obstacles in the path. We provide simulation results and a hardware demonstration of the proposed ROS2 RTA software package.

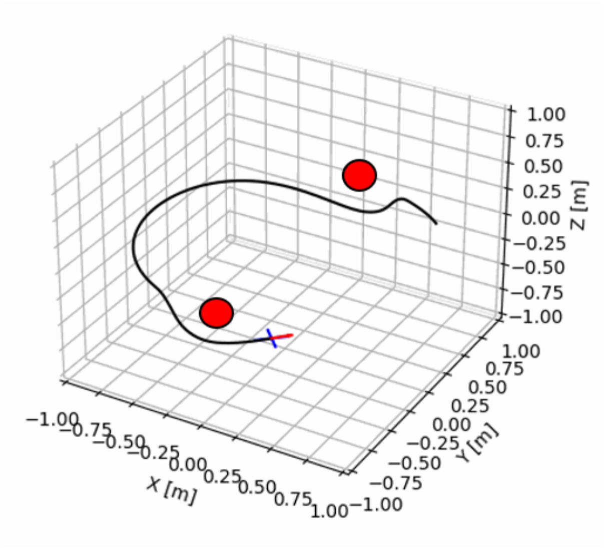


Fig. 5 Simulation results for a quadrotor using our RTA mechanism to supervise a primary controller while avoiding two obstacles intersecting a desired circular trajectory.

In Fig. 5 we provide simulation results of our RTA mechanism avoiding two spherical obstacles in a desired circular trajectory. The primary controller is nominally tracking the circular trajectory while our RTA mechanism filters the control inputs from the primary controller to ensure collisions are avoided with the obstacles. Our RTA mechanism only steps in when the quadrotor approaches an obstacle and detects a collision-bound trajectory.

After successfully demonstrating collision avoidance in a simulation environment we next turn our attention to a hardware demonstration. For the hardware demonstration, we use the same primary controller as in the simulation results, a 0.5m radius circular trajectory, and one spherical obstacle.

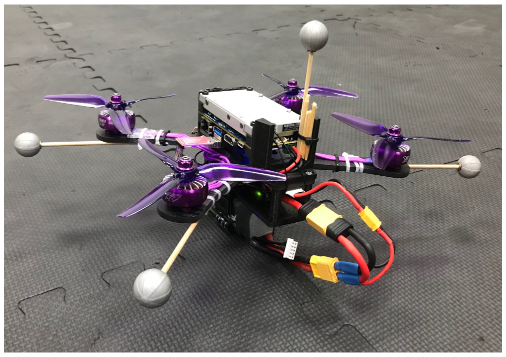
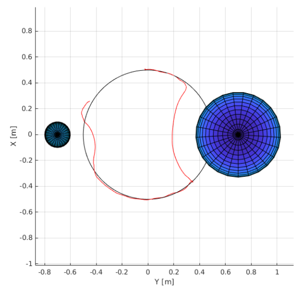
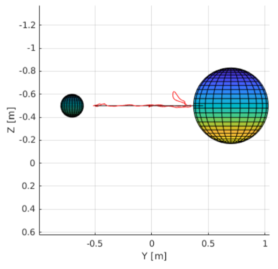


Fig. 6 A photograph of the quadrotor used in the hardware demonstration.

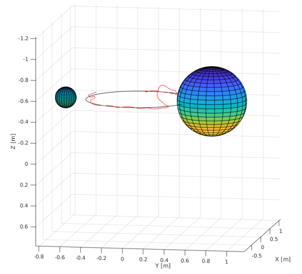
In Fig. 6 is a photograph of a 700g quadrotor used for the hardware demonstration. This quadrotor contains a Jetson TX2 onboard computer for running ROS2 and a flight controller running PX4 autopilot for controlling the motors and state estimation. The ROS2-PX4 bridge is handled by `px4_ros_com` as discussed in Section V. A ROS2 node receives external position measurements from OptiTrack motion capture cameras and passes the data to PX4 through `px4_ros_com` for fusion with the PX4 extended Kalman filter (EKF). State estimation from the EKF is passed back to ROS2 for use in the primary controller and our RTA ROS2 node. Control input commands in the form of collective thrust and body angular rates from our RTA ROS2 node is commanded through the `vehicle_rates_setpoint` message. This message is received by PX4 and a low-level body angular rate controller in PX4 tracks the desired body angular rates.



(a) Top View

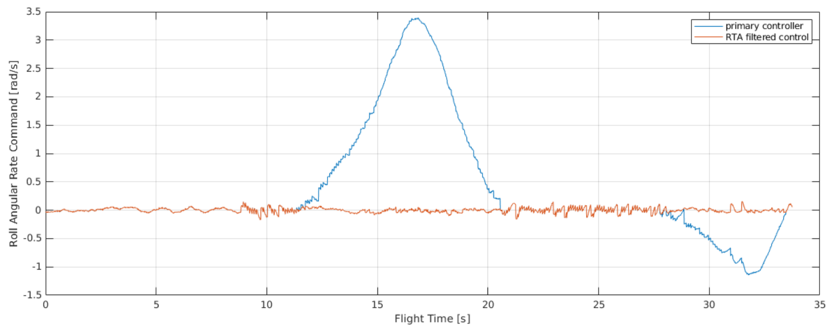


(b) Side View

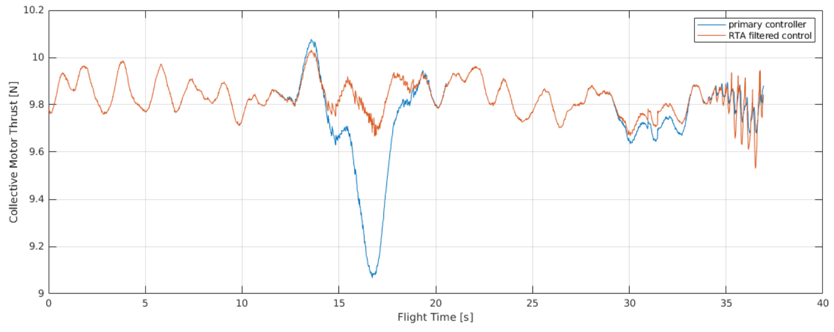


(c) Isometric View

Fig. 7 Flight trajectory plots of a quadcopter demonstrating collision avoidance with a spherical obstacle using our ROS2 RTA software package.



(a) Roll Angular Rate Plot



(b) Collective Normalized Thrust Plot

Fig. 8 Roll rate and normalized thrust plots of a quadcopter demonstrating collision avoidance with a spherical obstacle using our ROS2 RTA software package.

The flight demonstration is performed by initially taking off and stabilizing at the static position setpoint (x = 0.5m, y = 0, z = −0.5m). Then, our RTA software package is activated to supervise and filter input commands from the primary controller. Finally, the primary controller is commanded to track a circular trajectory parameterized by time. The trajectory plots from the first flight are plotted in Fig. 7. The midflight trajectory depicts a succesful avoidance maneuver performed by the ROS2 RTA software package after detecting a collision-bound trajectory from the primary controller tracking the circular trajectory. Normalized collective thrust and roll rate flight data is plotted for the primary controller and proposed RTA software package in Fig. 8. The RTA algorithm passes through the primary controller input commands to the quadcopter within the first 10 seconds of the flight until the quadcopter is predicted to collide into an obstacle. After the collision-bound trajectory is detected we see a disparity between the primary controller and RTA mechanism until about 20 seconds of flight after the obstacle is cleared. We then see another disparity at the 30 second flight time as the second obstacle is approached. Any primary controller designed by the user can be replaced in this flight experiment and the RTA mechanism will step in when needed to avoid obstacles. This enables supervision of learning-based controllers to guarantee safety while in their initial learning phase.

VII. Conclusion

In this work we provide a plug-and-play ROS2 software package for run time assurance of UAVs to verify safety in the form of collision avoidance. The proposed run time assurance algorithm computes reachable sets for a backup control policy and ensures the system's reachable set is never intersecting with obstacles. This guarantees that for all possible disturbances there is never a case where the vehicle collides with obstacles. We implement the proposed algorithm in a ROS2 software package for an operator to use for supervising their primary controllers and guaranteeing that collision-bound trajectories with obstacles are avoided. We demonstrate our run time assurance software package in hardware for collision avoidance of a quadrotor UAV commanded to track a circular trajectory with obstacles in the path. The vehicle successfully avoids obstacles using our RTA software package while meeting the objective of tracking the circular trajectory.

Acknowledgments

This work is supported in part by the NASA University Leadership Initiative (ULI) under grant number 80NSSC20M0161.

References

[1] Hobbs, K., Mote, M., Abate, M., Coogan, S., and Feron, E., "Run Time Assurance for Safety-Critical Systems: An Introduction to Safety Filtering Approaches for Complex Control Systems," , 2021.

[2] Ames, A. D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., and Tabuada, P., "Control Barrier Functions: Theory and Applications," *2019 18th European Control Conference (ECC)*, 2019, pp. 3420–3431. https://doi.org/10.23919/ECC.2019.8796030.

[3] Ames, A. D., Grizzle, J. W., and Tabuada, P., "Control barrier function based quadratic programs with application to adaptive cruise control," *53rd IEEE Conference on Decision and Control (CDC)*, 2014, pp. 6271–6278. https://doi.org/10.1109/CDC.2014.7040372.

[4] Gurriet, T., Mote, M., Ames, A. D., and Feron, E., "An Online Approach to Active Set Invariance," *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 3592–3599. https://doi.org/10.1109/CDC.2018.8619139.

[5] Gurriet, T., Mote, M., Singletary, A., Feron, E., and Ames, A. D., "A Scalable Controlled Set Invariance Framework with Practical Safety Guarantees," *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 2046–2053.

[6] Gurriet, T., Singletary, A., Reher, J., Ciarletta, L., Feron, E., and Ames, A., "Towards a Framework for Realizable Safety Critical Control through Active Set Invariance," *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 98–106. https://doi.org/10.1109/ICCPS.2018.00018.

[7] Abate, M., and Coogan, S., "Enforcing Safety at Runtime for Systems with Disturbances," *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 2038–2043. https://doi.org/10.1109/CDC42340.2020.9304203.

[8] Shivakumar, S., Torfah, H., Desai, A., and Seshia, S. A., "SOTER on ROS: A Run-Time Assurance Framework on the Robot Operating System," *Runtime Verification*, edited by J. Deshmukh and D. Ničković, Springer International Publishing, Cham, 2020, pp. 184–194.

[9] Rivera, J. G., and Danylyszyn, A. A., "Formalizing the Uni-processor Simplex Architecture," Tech. rep., Carnegie Mellon University School of Computer Science, 1995.

[10] Bak, S., Johnson, T. T., Caccamo, M., and Sha, L., "Real-Time Reachability for Verified Simplex Design," *2014 IEEE Real-Time Systems Symposium*, 2014, pp. 138–148. https://doi.org/10.1109/RTSS.2014.21.

[11] Llanes, C., Abate, M., and Coogan, S., "Safety from Fast, In-the-Loop Reachability with Application to UAVs," *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, 2022. https://doi.org/10.1109/ICCPS54341.2022.00018.

[12] Coogan, S., "Mixed Monotonicity for Reachability and Safety in Dynamical Systems," *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 5074–5085. https://doi.org/10.1109/CDC42340.2020.9304391.

[13] Hogan, W., "Directional derivatives for extremal-value functions with applications to the completely convex case," *Operations Research*, Vol. 21, No. 1, 1973, pp. 188–209.

[14] Seywald, H., and Kumar, R., "Desensitized optimal trajectories," Vol. 93, 1996, pp. 103–113.

[15] Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S., "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, Vol. 12, No. 4, 2020, pp. 637–672. https://doi.org/10.1007/s12532-020-00179-2, URL https://doi.org/10.1007/s12532-020-00179-2.

[16] Foehn, P., and Scaramuzza, D., "Onboard State Dependent LQR for Agile Quadrotors," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 6566–6572. https://doi.org/10.1109/ICRA.2018.8460885.

[17] Foehn, P., Romero, A., and Scaramuzza, D., "Time-optimal planning for quadrotor waypoint flight," *Science Robotics*, Vol. 6, No. 56, 2021, p. eabh1221. https://doi.org/10.1126/scirobotics.abh1221, URL https://www.science.org/doi/abs/10.1126/scirobotics.abh1221.

[18] Falanga, D., Foehn, P., Lu, P., and Scaramuzza, D., "PAMPC: Perception-Aware Model Predictive Control for Quadrotors," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–8. https://doi.org/10.1109/IROS.2018.8593739.