2023 UAV Chase Challenge Manual

January 3, 2023

Updated February 3, 2023

Table of Contents

Motivations

Autonomous drones must use computer vision algorithms that are able to run on on-board hardware to ensure safety and reliability. The Unmanned Aerial Vehicle Chase Challenge (UAVCC) encourages teams to develop new low-power computer vision algorithms for drones, and ways to employ these algorithms to be able to track objects. This challenge will involve developing a complete control system to track a rover through a miniature city constructed in Purdue's Unmanned Aerial Systems (UAS) Test Facility.

Competition Overview

The UAV Chase Competition is separated into two stages: first, solutions will be submitted to the Research Infrastructure for Real-Time Onboard Vision Enabled Robots (RI4ROVER) server and tested in a simulation of the competition field. In the simulation, solutions will be used to guide a drone to follow a rover through a miniature city. Second, teams that develop successful solutions in the simulator will be provided resources to develop a control system for an actual drone (the PX4 Vision), and will use their solutions to follow a remote controlled car through a model city in the Purdue Unmanned Aerial Systems Test Facility.

Solution Grading

Solutions will be graded based on the average distance between the rover and the drone. Having a distance of exactly 1 meter will result in the greatest score. If the distance is greater than 5 meters, then the score for this frame will be zero. The UAS Test Facility's motion capture system will obtain ground-truth data regarding the position of the rover and the drone so that competitors may be properly scored. Competitors will be provided the drone's exact position from the motion capture system, but will not have access to the position of the rover.

Stage 1: Simulated Trials

The Gazebo simulator (Version 9) will be used to test solutions. Instructions on using Gazebo for Software-In-The-Loop simulation may be found in the readme for PX4 Vision's object avoidance package. Being able to run the object avoidance package in Gazebo can be a good starting place for developing your control system, as it will involve installing all the necessary technologies to build a control system in ROS. More on the software used may be found below.

There will be four simulated trials which are designed to reflect the four physical trials that will be tested at the UAS Test Facility. It will be expected that you run a single launch script to allow the drone to autonomously track the rover in each of these trials. The trials are as described below.

Simulated Trial 1

In Trial 1, the drone will need to follow a simulated rover with a red ball attached to the back. The rover will travel along a straight line. The drone will begin facing the rover in the direction of its movement. The rover will move forward at a varying speed, and the drone must follow its movements.

Simulated Trial 2

In trial 2, the drone will need to follow a simulated rover with a red ball attached to the back. The rover will drive on a road that is at least 2 feet wide through a city. The drone is expected to follow the rover while avoiding buildings.

Simulated Trial 3

In Trial 3, the drone will need to follow a simulated rover that does not have any distinguishing colors on it. To accomplish this, you may train a model to detect the rover using images taken in the simulator. The rover will drive along a straight line in the same configuration defined in trial 1. There will be buildings along the side of the track.

Simulated Trial 4

In Trial 4, the drone will need to follow a simulated rover that does not have any distinguishing colors on it. To accomplish this, you may train a model to detect the rover using images taken in the simulator. The rover will drive around city blocks in a similar manner to trial 2. The drone must follow the rover through the city without colliding into buildings. Distractions such as other rovers may be introduced.

GitHub Repository

Simulators reflecting trials may be found here:
https://github.com/jgoppert/auav_2022_sample

Please note that testing of solutions may include changes to the path of the rover to ensure robustness of solutions. To test your solution's robustness before submitting, it is encouraged that you set different paths for the rover and account for errors in the control system.

Stage 2: Physical Trials

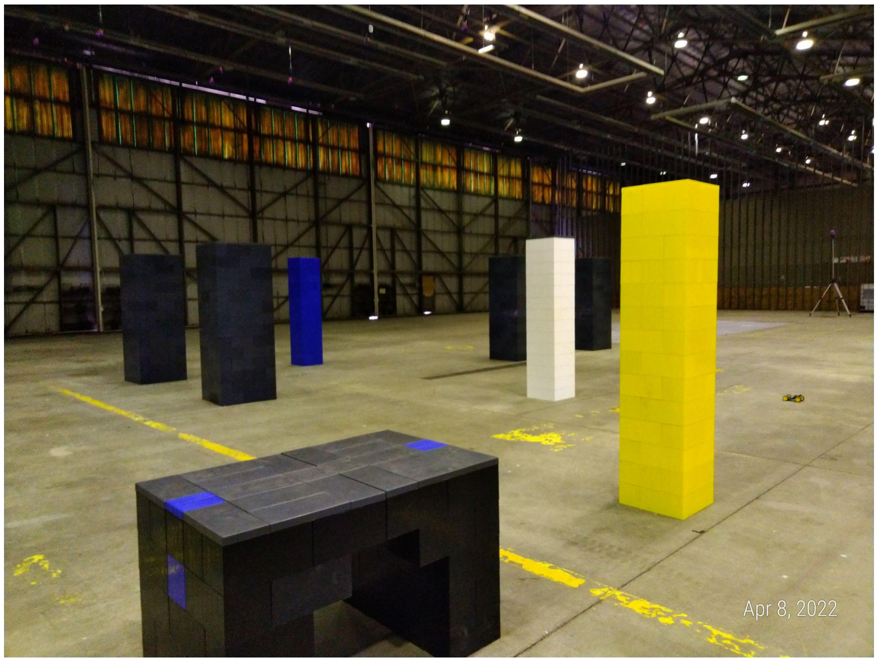


Figure 1. A model of a city similar to what will be used in Trials 2 and 4, built in the UAS Testing Facility

Procedure

Teams selected to continue past the simulated trial phase will be invited to launch their working solutions at the Purdue University UAS Research and Test Facility (PURT). A PX4Vision will be available for competitors to deploy their solutions on. The onboard computer will be an Intel Up board running Ubuntu 20.04 and ROS noetic. The staff at the facility will assist any competitors with updating code to run on the provided hardware if necessary. Competitors are also welcome to bring their own drones to the facility to use. Drones brought by competitors are required to adhere to the following rules:

- The vehicle must have a wheelbase of 500mm or less
- A test flight must be performed prior to use to demonstrate manual control and the ability to turn off the motors
- If the drone utilizes any other sensors aside from that used for vision, please let the organizers know to ensure no interference with airport operations would be possible

Position information will also be made available to competitors' drones through the Qualisys motion capture system installed in the PURT facility. Data will be transmitted to the drones over wifi.

A $2,000 travel grant will be available for each team that travels to the PURT facility for the physical trials. For international teams who qualify and wish to come, remember to secure all documents required for travel to the United States (such as passports and visas).

For teams that cannot travel to the PURT facility, solutions can be submitted and operated by facility staff. This process will be done over Zoom during the competition's preparation and trial phases.

Physical Trial 1

In Trial 1, the drone will need to follow a rover with a red ball attached to it. The rover will travel along a straight line. The drone will begin facing the drone in the direction of its movement. The rover will move forward and backward at varying speeds, and the drone must follow its movements.

Physical Trial 2

In trial 2, the drone will need to follow a rover with a red ball attached to it. The rover will drive on a road that is at least 2 feet wide through a city. The drone is expected to follow the rover while avoiding buildings.

Physical Trial 3

In Trial 3, the drone will need to follow a rover that does not have any distinguishing colors on it. To accomplish this, you may train a model to detect the rover using images taken in the simulator. The rover will drive along a straight line in the same configuration defined in trial 1. There will be buildings along the side of the track.

Physical Trial 4

In Trial 4, the drone will need to follow a rover that does not have any distinguishing colors on it. To accomplish this, you may train a model to detect the rover using images taken in the simulator. The rover will drive around city blocks in a similar manner to trial 2. The drone must follow the rover through the city without colliding into buildings. Distractions such as other rovers may be introduced.

Hardware

PX4 Vision Development Kit

The PX4 Vision is a quadcopter drone that consists of two computers: A Pixhawk 4 flight controller, and an UpCore companion computer. It also comes with a Structure Core depth camera, capable of producing high-resolution depth information. A PX4 Vision Development Kit is what will be available to competitors who do not bring their own drones.

Additional PX4 Information

PX4 Development Kit:
https://docs.px4.io/master/en/complete_vehicles/px4_vision_kit.html
Structure Core Depth Camera:
https://structure.io/structure-core

Intel Neural Compute Stick 2

An Intel Neural Compute Stick 2 may be used onboard the drone to facilitate larger models. More information may be found on the product website.

Intel Realsense D435i

To support object detection algorithms that use RGB images, an Intel Realsense may be used in place of the Structure Core depth camera for Trials 3 and 4. More information may be found on the product website.

Software

Software Requirements

The PX4 Avoidance package with software-in-the-loop simulation will work on Ubuntu 18.04 with ROS Melodic. Some have experienced issues with running PX4 Avoidance on machines with Ubuntu 20.04 and ROS Noetic. As of now, the simulator solution grader is using ROS Melodic. We encourage competitors to begin their development using the PX4 Avoidance package with Ubuntu 18.04 and ROS Melodic.

ROS

Robot Operating System, or ROS, is a middleware used to connect, organize and deploy different pieces of software for controlling robots. Scripts can be broken up into concurrently operating independent parts called nodes, which can be helpful for developing modular systems that can respond to changes in data from many different sensors. Nodes exchange information over topics, by either advertising over them or subscribing to them.

The sample solution uses ROS Melodic on the PX4 Vision drone's Upcore flight computer. The solution teams develop will take the form of a ROS node that may be used with other ROS packages and run on the companion computer. ROS nodes can communicate with the flight computer to receive telemetry and control the drone by using the MAVROS package.

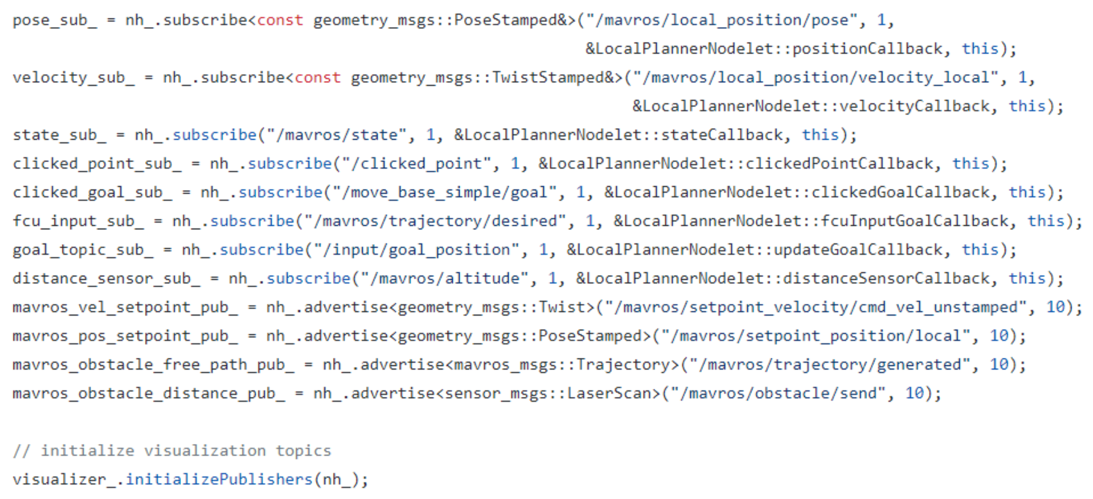
Additional ROS Information

The ROS documentation and tutorials are available on the ROS website: [ROS.org | Powering the world's robots]

MAVLink and MAVROS

MAVLink is a protocol used for communicating with drones. MAVROS is a ROS package that provides a communication node for the ROS system, which publishes information from the drone's sensors over topics. MAVROS serves as a bridge between ROS topics (the way information is transferred between nodes) and MAVLink, the way that information can be transferred to/from the drone's firmware.

The following topics are important to the system. Below is an example of the use of MAVROS in C for the object avoidance local planner node.



```
pose_sub_ = nh_.subscribe<const geometry_msgs::PoseStamped&>("/mavros/local_position/pose", 1,
                                                             &LocalPlannerNodelet::positionCallback, this);
velocity_sub_ = nh_.subscribe<const geometry_msgs::TwistStamped&>("/mavros/local_position/velocity_local", 1,
                                                                  &LocalPlannerNodelet::velocityCallback, this);
state_sub_ = nh_.subscribe("/mavros/state", 1, &LocalPlannerNodelet::stateCallback, this);
clicked_point_sub_ = nh_.subscribe("/clicked_point", 1, &LocalPlannerNodelet::clickedPointCallback, this);
clicked_goal_sub_ = nh_.subscribe("/move_base_simple/goal", 1, &LocalPlannerNodelet::clickedGoalCallback, this);
fcu_input_sub_ = nh_.subscribe("/mavros/trajectory/desired", 1, &LocalPlannerNodelet::fcuInputGoalCallback, this);
goal_topic_sub_ = nh_.subscribe("/input/goal_position", 1, &LocalPlannerNodelet::updateGoalCallback, this);
distance_sensor_sub_ = nh_.subscribe("/mavros/altitude", 1, &LocalPlannerNodelet::distanceSensorCallback, this);
mavros_vel_setpoint_pub_ = nh_.advertise<geometry_msgs::Twist>("/mavros/setpoint_velocity/cmd_vel_unstamped", 10);
mavros_pos_setpoint_pub_ = nh_.advertise<geometry_msgs::PoseStamped>("/mavros/setpoint_position/local", 10);
mavros_obstacle_free_path_pub_ = nh_.advertise<mavros_msgs::Trajectory>("/mavros/trajectory/generated", 10);
mavros_obstacle_distance_pub_ = nh_.advertise<sensor_msgs::LaserScan>("/mavros/obstacle/send", 10);

// initialize visualization topics
visualizer_.initializePublishers(nh_);
```

Figure 2. Published and subscribed topics for the local planner node in the avoidance package

Additional MAVLink and MAVROS Information

More information on MAVROS and MAVLink for the PX4 Vision may be found in the PX4 development guide: https://docs.px4.io/main/en/ros/ros1.html

Using MAVROS for offboard control on the PX4Vision: https://docs.px4.io/main/en/ros/mavros_offboard_cpp.html

The MAVROS Github repo may be found here: https://github.com/mavlink/mavros

QGroundControl

QGroundControl serves as a ground control station that can provide instructions to the drone's flight controller, the Pixhawk 4. It can be used in SITL simulation or when controlling the physical drone. Examples for its use include manually controlling the drone with a joystick or planning routes for travel. QGroundControl does not depend on ROS, and communicates with the drone over MAVLink directly. QGroundControl is also a critical tool when flying the drone in the physical trials as it allows the user to calibrate the drone's sensors and controls, receive telemetry data, and change the parameters set on the Pixhawk flight controller.

RViz

RViz is a visualization program that can be run alongside Gazebo and QGroundControl. RViz provides visualizations of the sensor data received over ROS, and can be used to see what the drone is doing. Note that RViz is a visualization program and not a simulator in itself. This means that RViz may still be used in the physical trials to view sensor data.

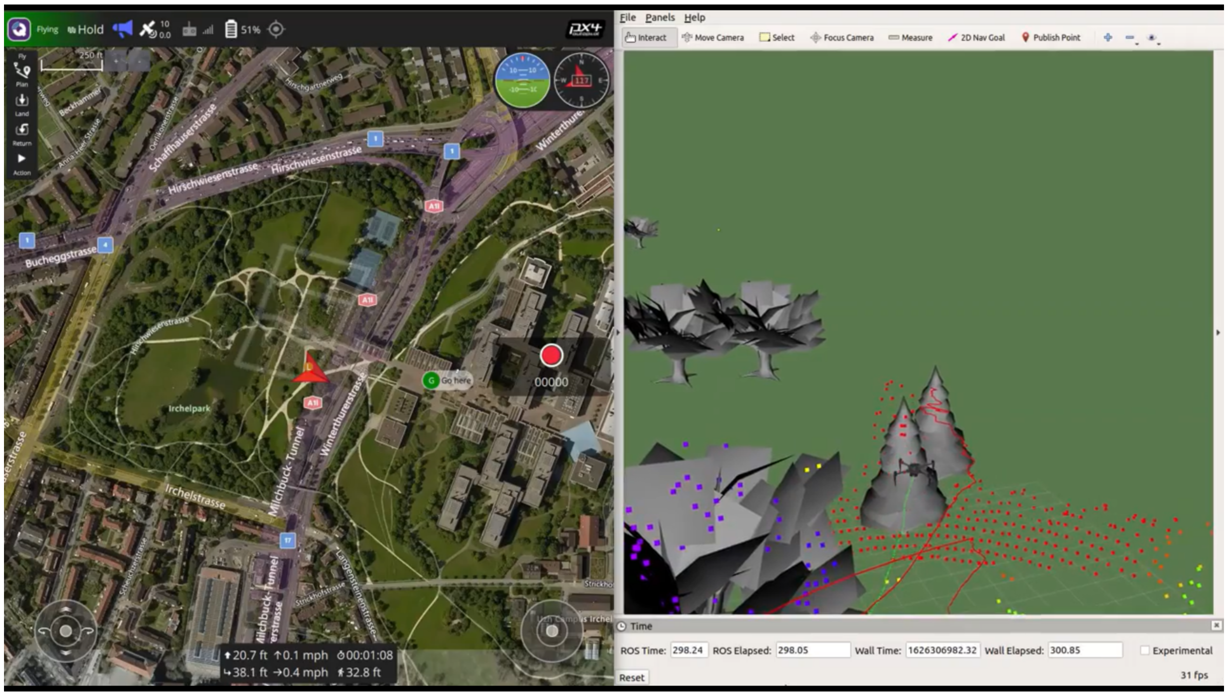


Figure 3. QGroundControl (left) and RViz (right) while running the object avoidance node in a Gazebo SITL simulation

ROS Packages

ROS Packages include nodes and topics that can be added to an existing ROS system. They can be installed in a catkin workspace. More information on setting up a ROS environment and installing ROS packages can be found on the ROS website in the tutorials section.

Additionally, an https://index.ros.org/packages/.

PX4 Avoidance

This package uses the Structure Core depth camera on the PX4 Vision to recognize obstacles in its path and plot courses around them. This can be used to prevent the drone from colliding into objects while tracking the ground robot in testing or competition.

The github repo for this package also includes a helpful tutorial for setting up your ROS environment with MAVROS, and for running software-in-the-loop simulation. I encourage you to read the readme if you are setting up a ROS environment.

Additional Information

Github: PX4/PX4-Avoidance: PX4 avoidance ROS node for obstacle detection and avoidance. (github.com)

MAVROS

Includes nodes for communication over MAVLink. Required to communicate with the drone software, in practice or in simulation, and control it with ROS nodes. More information on MAVROS is included in the section on MAVROS and MAVLink.

Additional Information

Github: https://github.com/mavlink/mavros

Recommended Workflow for ROS Development

We recommend the following workflow for teams to begin ROS development.

- Meet once to set up software and environment on all developer machines, including ROS, Pviz, Gazebo, Firmware, and all relevant ROS packages. Solve any issues with environment setup in this meeting. To verify that everything is working properly, members should be able to run the local_planner launch script in the avoidance package. If they are able to do this and set the drone's goal position in Rviz or QGroundControl, then all dependencies have been installed correctly.
- Create a Github repository for the project, including relevant packages, nodes, and launch files. Keep release, development, and feature branches. Ensure the devel, build, and log directories are added to gitignore.
- Keep a launch file that will run all necessary nodes. When you submit your solution, indicate the path of the launch file so it may be easily run.

Errors and Solutions

If you encounter any errors during the installation, development, or running of any of these technologies, please send a message in the LPCVC 2023 UAV Chase Challenge Slack or create an issue on the uavcc-simulator GitHub.