microsoft / **AirSim-NeurIPS2019-Drone-Racing** Public

Drone Racing @ NeurIPS 2019, built on Microsoft AirSim

microsoft.github.io/airsim-neurips2019-drone-racing/

MIT license

301 stars 81 forks

Star Watch

Code Issues 26 Pull requests 1 Actions Projects Security Insights

master

madratman Merge pull request #159 from microsoft/users/GitHubPolicy... ... on Aug 12, 2022 115

View code

README.md

Game of Drones: A NeurIPS 2019 Competition

Note: This repository is not being maintained any more. Please use AirSim Drone Racing Lab.

Quickstart

- Website
- Register
- Competition guidelines
- Linux and Windows Binaries
- Python API, airsimneurips PyPI package



Note: If you use this repository in your research, please cite our pre-print, AirSim Drone Racing Lab.

```
@article{madaan2020airsim,
  title={AirSim Drone Racing Lab},
  author={Madaan, Ratnesh and Gyde, Nicholas and Vemprala, Sai and Brown, Matthew
and Nagami, Keiko and Taubner, Tim and Cristofalo, Eric and Scaramuzza, Davide and
Schwager, Mac and Kapoor, Ashish},
  journal={arXiv preprint arXiv:2003.05654},
  year={2020}
}
```

Downloading and running AirSim Binaries

Downloading

- Final round binaries and environments (v1.1)
    - tl;dr:
        - [Linux] Use the download_final_round_binaries.sh script
    - Long version:
        - Download the v1.1 Linux or Windows `AirSim.zip` , and unzip it.
        - Download your qualifier environments (shipped in pakfiles) - `Final_Tier_1_and_Tier_2.pak` and `Final_Tier_3.pak` .
        - Move the environment pakfiles into `AirSim/AirSimExe/Content/Paks` .
        - Download and move the `settings.json` file to `~/Documents/AirSim/settings.json` .
        - Use `airsimneurips` >= 1.2.0
- Qualifier binaries and environments (v1.0)

    - tl;dr:
        - [Linux] Use the download_qualification_binaries.sh script
    - Long version:
        - Download the v1.0 Linux or Windows `AirSim.zip`, and unzip it.
        - Download your qualifier environments (shipped in pakfiles) - `Qual_Tier_1_and_Tier_3.pak` and `Qual_Tier_2.pak`.
        - Move the environment pakfiles into `AirSim/AirSimExe/Content/Paks`.
        - Download and move the `settings.json` file to `~/Documents/AirSim/settings.json`.

- Training binaries and environments (v0.3):

    - tl;dr:
        - [Linux] Use the download_training_binaries.sh script
    - Long version:
        - Download the v0.3 Linux or Windows `AirSim.zip`, and unzip it.
        - Download training environments (shipped in pakfiles) - `Soccer_Field.pak`, `ZhangJiaJie.pak`, and `Building99.pak`.
        - Move the environment pakfiles into `AirSim/AirSimExe/Content/Paks`.
        - Download and move the `settings.json` file to `~/Documents/AirSim/settings.json`.

Notes:

- `Source code (zip)` or `Source code (tar.gz)` might not be up-to-date with the master branch of this repository. It can be lagging by `n commits to master since this release`, specified on the released page.
  For the code on this repository, it's best to just `git clone`.

- List of disabled APIs in qualification binaries: The following APIs on the server side in the qualification binaries. You should see an error message pop up in the terminal message when you call these. They do work in the training binaries:

    - `simSetObjectPose`
    - `simSetVehiclePose`
    - `simSetObjectScale`
    - `simGetObjectScale`
    - `simSetSegmentationObjectID`
    - `simGetSegmentationObjectID`
    - `simPause`
    - `simContinueForTime`

Running

- Linux
  - Open a terminal window, `cd to AirSim_Training/` or `AirSim_Qualification` directory, and enter the following command:

    ```
    ./AirSimExe.sh -windowed -opengl4
    ```

  - Running headless (with rendering of images enabled):

    ```
    DISPLAY= ./AirSimExe.sh -opengl4
    ```

  - To disable rendering completely for training planning and / or control policies, you can use:

    ```
    -./AirSimExe.sh -nullrhi
    ```

    Note that `simGetImages` will not work with this option.
  - To increase speed of `simGetImages` / increase speed of Unreal Engine's game thread;
    - Add the `"ViewMode": "NoDisplay"` to your settings.json file, or use this file directly.
      This disables rendering in the main viewport camera.
      Then run the binary with the following options.

      ```
      ./AirSimExe.sh -windowed -NoVSync -BENCHMARK
      ```

      You can also use the Unreal console commands `Stat FPS`, `Stat UnitGraph`, `r.VSync`, `t.maxFPS`. See Issue #111 for more details.
- Windows
  - Navigate to the `AirSim/` directory, and double-click `run.bat` (or `AirSimExe.exe -windowed`)

Docker

- Prerequisites:
  - Install docker-ce.
  - Complete the desired post-installation steps for linux after installing docker.
    At the minimum, the page tells you how torun docker without root, and other useful setup options.
  - Install nvidia-docker2.

- Dockerfile:
  We provide a sample dockerfile you can modify.
  It downloads the training and qualification binaries automatically, and installs the python client.
  By default, it uses Ubuntu 18.04 and CUDA 10.0 with OpenGL, and is build on top of nvidia/cudagl:10.0-devel-ubuntu18.04.
  This can be changed of course, as explained in the following section.

- Building the docker image:
  You can use build_docker_image.py to build the dockerfile above (or your own custom one)
  Usage (with default arguments)

```
cd docker/;
python3 build_docker_image.py \
       --dockerfile Dockerfile \
       --base_image nvidia/cudagl:10.0-devel-ubuntu18.04 \
       -- target_image airsim_neurips:10.0-devel-ubuntu18.04
```

- Running the docker image: See docker/run_docker_image.sh to run the docker image:
  Usage

  - for running default image, training binaries, in windowed mode:
    `$ ./run_docker_image.sh "" training`
  - for running default image, qualification binaries, in windowed mode:
    `$ ./run_docker_image.sh "" qualification`
  - for running default image, training binaries, in headless mode:
    `$ ./run_docker_image.sh "" training headless`
  - for running default image, qualification binaries, in headless mode:
    `$ ./run_docker_image.sh "" qualification headless`
  - for running a custom image in windowed mode, pass in you image name and tag:
    `$ ./run_docker_image.sh DOCKER_IMAGE_NAME:TAG`
  - for running a custom image in headless mode, pass in you image name and tag, followed by "headless":
    `$ ./run_docker_image.sh DOCKER_IMAGE_NAME:TAG headless`

AirSim API

- To control your drone and get information from the environment, you will need the `airsimneurips` API, which is accessible via Python.
  We recommend you used python >= 3.6. Python 2.7 will go out of support soon

- To install the Python API, do a :

```
pip install airsimneurips
```

- See quick overview of the API below
- The API is documented at airsimneurips API doc
- Resources
  - Going through both open and closed issues in this repository might answer some of your questions. The search bar on top left can prove useful.
  - AirSim upstream API and examples can also be of use. However, please note that the main AirSim repo's API is not used in the competition (there's some overlap and some differences), however is a good learning resource.

Submitting Results and Leaderboard - Qualification Round

- For the qualification round, we have one race track for each tier. The relevant binaries (v1.0) are available for linux and windows
  - Tier 1: This is in the Soccer Field environment.
    THe race track is in the `Qual_Tier_1_and_Tier_3.pak` pakfile
  - Tier 2: This is in the ZhangJiaJie environment.
    The race track is in the `Qual_Tier_2.pak` pakfile.
  - Tier 3: This is again in the Soccer Field environment.
    The race track is in the `Qual_Tier_1_and_Tier_3.pak` pakfile.
- How to generate logfiles for each tier:
  - Loading level and starting race:
    - Please update your airsimneurips pythonclient (should be >=1.0.0).
    - Calling `simStartRace(race_tier=1, 2, or 3)` generates the appropriate log files.
    - Tier 1:

```
airsim_client.simLoadLevel('Qualifier_Tier_1')
airsim_client.simStartRace(1)
```

    - Tier 2:

```
airsim_client.simLoadLevel('Qualifier_Tier_2')
airsim_client.simStartRace(2)
```

- Tier 3:

```
airsim_client.simLoadLevel('Qualifier_Tier_3')
airsim_client.simStartRace(3)
```

  - As Tier 2 focuses on perception and Tier 3 focuses on both perception and planning, note that `simGetObjectPose` returns noisy gate poses, after `simStartRace(2)` and `simStartRace(3)` is called.
  - As soon as `simStartRace(1)` or `simStartRace(3)` is called, `drone_2` (MSR opponent racer) will start flying.
  - See `baseline_racer.py` for sample code. The previous bullet points are being called in wrapper functions in the following snippet in `baseline_racer.py`:

```
baseline_racer.load_level(args.level_name)
if args.level_name == "Qualifier_Tier_1":
    args.race_tier = 1
if args.level_name == "Qualifier_Tier_2":
    args.race_tier = 2
if args.level_name == "Qualifier_Tier_3":
    args.race_tier = 3
baseline_racer.start_race(args.race_tier)
```

- To submit your results to the leaderboard:
  - Navigate to the submission site, enter your team name in the proper field, and upload any number of race logs.
    It's ok to make a submission for as little as a single track and/or a single tier.
    You can find race logs inside of `AirSimExe/Saved/Logs/RaceLogs` in your downloaded binary folder.
    Please read the race monitoring section in the competition guidelines for more details.
  - The leaderboard will publish the results of a drone that is named `drone_1` (call `generate_settings_file.py` to generate an AirSim settings file, as done for the `baseline_racer` below.
  - Please submit a PDF file in the `report` section to help us verify the honesty of your submission for the Nov 21st deadline. Please summarize your approach for all tiers you make a submission for, with appropriate citations. The report PDF size should not exceed 10 MB, and should be a maximum of 4 pages in length. We leave the exact format of the report to your description, but the IEEE template is a good choice.
  - We have emailed you a private key, which should be entered in the `Team ID` field. This helps us verify it was your team who indeed made the submission.
  - The leaderboard is updated once per day at 2100 PST.
    If you do not see your results after 24 hours, please email us with your team name and

submitted log files.

Submitting Results and Leaderboard - Final Round

- For the final round, we have one race track for each tier. The relevant binaries (v1.1) are available for linux and windows
    - Tier 1: This is in the Soccer Field environment.
      THe race track is in the `Final_Tier_1_and_Tier_2.pak` pakfile
    - Tier 2: This is in the Soccer Field environment.
      The race track is in the `Final_Tier_1_and_Tier_2.pak` pakfile.
    - Tier 3: This is again in the ZhangJiaJie environment.
      The race track is in the `Final_Tier_3.pak` pakfile.
- How to generate logfiles for each tier:
    - Loading level and starting race:
        - Please update your airsimneurips pythonclient (should be >=1.2.0).
        - Calling `simStartRace(race_tier=1, 2, or 3)` generates the appropriate log files. You can only run `tier N` races in `Final_Tier_N` levels.
        - Tier 1:

```
airsim_client.simLoadLevel('Final_Tier_1')
airsim_client.simStartRace(tier=1)
```

        - Tier 2:

```
airsim_client.simLoadLevel('Final_Tier_2')
airsim_client.simStartRace(tier=2)
```

        - Tier 3:

```
airsim_client.simLoadLevel('Final_Tier_3')
airsim_client.simStartRace(tier=3)
```

    - As Tier 2 focuses on perception and Tier 3 focuses on both perception and planning, note that `simGetObjectPose` returns noisy gate poses.
    - As soon as `simStartRace(tier=1)` or `simStartRace(tier=3)` is called, `drone_2` (MSR opponent racer) will start flying.

- See `baseline_racer.py` for sample code. The previous bullet points are being called in wrapper functions in the following snippet in `baseline_racer.py`:

```
baseline_racer.load_level(args.level_name)
baseline_racer.start_race(args.race_tier)
```

- To submit your results to the final leaderboard:
  - Navigate to the submission site, enter your team name in the proper field, and upload any number of race logs.
    It's ok to make a submission for as little as a single track and/or a single tier.
    You can find race logs inside of `AirSimExe/Saved/Logs/RaceLogs` in your downloaded binary folder.
    Please read the race monitoring section in the competition guidelines for more details.
  - The leaderboard will publish the results of a drone that is named `drone_1` (call `generate_settings_file.py` to generate an AirSim settings file, as done for the `baseline_racer` below.
  - Please submit a PDF file in the `report` section to help us verify the honesty of your submission by the Dec 5th, 2359 PST deadline. Please summarize your approach for all tiers you make a submission for, with appropriate citations. The report PDF size should not exceed 10 MB, and should be a maximum of 6 pages in length. We leave the exact format of the report to your descrition, but the IEEE template is a good choice.
  - We have emailed you a private key, which should be entered in the `Team ID` field. This helps us verify it was your team who indeed made the submission.
  - The final leaderboard is updated once per day at 2100 PST.
    If you do not see your results after 24 hours, please email us with your team name and submitted log files.

Sample code

- Plan and move on a minimum jerk trajectory using ground truth poses of gates:
  - Generate an AirSim settings.json file (same as the one provided in releases)

```
$ cd baselines;
$ python generate_settings_file.py
```

  - Start the AirSim Neurips binary, as explained above
  - Run the code!

```
$ python baseline_racer.py \
        --enable_viz_traj \
        --enable_viz_image_cv2 \
        --planning_baseline_type all_gates_at_once \
```

```
        --planning_and_control_api moveOnSpline \
        --level_name ZhangJiaJie_Medium \
        --race_tier 1
```

Usage is:

```
$ python baselines/baseline_racer.py -h
usage: baseline_racer.py [-h]
        [--level_name {Soccer_Field_Easy,Soccer_Field_Medium,ZhangJiaJie_Medium,B
        [--planning_baseline_type {all_gates_at_once,all_gates_one_by_one}]
        [--planning_and_control_api {moveOnSpline,moveOnSplineVelConstraints}]
        [--enable_viz_traj] [--enable_viz_image_cv2]
        [--race_tier {1,2,3}]
```

- Plan a Game Theoretic Plan (GTP) trajectory for an ego drone based on an estimate of the opponent drone's behavior.
    - Generate an AirSim settings.json file

```
$ cd baselines;
$ python generate_settings_file.py
```

    - Start the AirSim Neurips binary, as explained above
    - Run the GTP code!

```
$ python baseline_racer_gtp.py \
       --blocking_behavior \
       --plot_gtp \
       --enable_viz_traj \
       --level_name Qualifier_Tier_1
```

    - This method is an Iterative Best Response (IBR) trajectory planning technique. In IBR, first the trajectories of both drones are initialized as straight down the track at maximum speed (to win the game!). The opponent trajectory is then held constant while we solve for the ego trajectory via Model Predictive Control (MPC) optimization (details in gtp.py). Then, we hold the ego trajectory constant and solve for a guess of the opponent's trajectory in the same fashion. If after some iterations, the solution convereges (i.e., the resulting trajectories stop changing), we have reached a Nash equilibrium over the space of trajectories. That is to say, either agents can not unilaterally change their trajectory to increase their own performance. This implementation is a heuristic based on the original method proposed in the paper below (PDF here).
        - R. Spica, D. Falanga, E. Cristofalo, E. Montijano, D. Scaramuzza, and M. Schwager, "A Real-Time Game Theoretic Planner for Autonomous Two-Player Drone Racing", in the Proccedings of Robotics: Science and Systems (RSS), 2018.

Quick API overview

We added some new APIs (marked with 💚) to AirSim for the NeurIPS competition binaries.

Loading Unreal Engine environments

- `simLoadLevel(level_name)` 💚
  Possible values for `level_name` are:
  - `"Soccer_Field_Easy"`, `"Soccer_Field_Medium"`, `"ZhangJiaJie_Medium"`, `"Building99_Hard"` in the training binaries (`v0.3`).
  - `"Qualification_Tier_1"`, `"Qualification_Tier_2"`, `"Qualification_Tier_3"` in the qualification binaries (`v1.0`).
  - `"Final_Tier_1"`, `"Final_Tier_2"`, `"Final_Tier_3"` in the final round binaries (`v1.1`). Before trying this, please ensure you've downloaded the corresponding training (`v0.3`) / qualifier (`v1.0`) / final round (`v1.0`) binaries, as described above
- UI Menu
  - Press `F10` to toggle the level menu
  - Click your desired level. (Note: the UI lists all the pakfiles in the `AirSim/AirSimExe/Content/Paks` directory. Ensure you downloaded the pakfile, if you are not able to see a particular environment)

Race APIs:

- Start a race: `simStartRace(tier=1/2/3)` 💚
- Reset race: `simResetRace()` 💚
- Check if racer is disqualified: `simIsRacerDisqualified()` 💚
- Get index of last gate passed: `simGetLastGatePassed()` 💚
- Disable generation of logfiles by race APIs: `simDisableRaceLog` 💚

Lower level control APIs:

- FPV like Angle rate setpoint APIs:
  - `moveByAngleRatesThrottleAsync` 💚
  - `moveByAngleRatesZAsync` 💚 (stabilizes altitude)
- Angle setpoint APIs:
  - `moveByRollPitchYawThrottleAsync` 💚
  - `moveByRollPitchYawZAsync` 💚 (stabilizes altitude)

- RollPitchYawrate setpoint APIs:
  - `moveByRollPitchYawrateThrottleAsync` 💚
  - `moveByRollPitchYawrateZAsync` 💚 (stabilizes altitude)

Medium level control APIs:

- Velocity setpoints
  - `moveByVelocityAsync`
  - `moveByVelocityZAsync` (stabilizes altitude)
- Position setpoints
  - `moveToPosition`
  - `moveOnPath`
  - `moveToZAsync`

High level control APIs:

- Minimum jerk trajectory planning (using ethz-asl/mav_trajectory_generation), and trajectory tracking (using a pure pursuit like controller minimizing position and velocity errors), with position setpoints. Optionally use the `*lookahead*` parameters to start new trajectory from a point sampled `n` seconds ahead for trajectory being tracked currently.
  - `moveOnSplineAsync` 💚
- Minimum jerk trajectory planning (using ethz-asl/mav_trajectory_generation), and trajectory tracking (using a pure pursuit like controller minimizing position and velocity errors), with position setpoints and corresponding velocity constraints. Useful for making a drone go through a gate waypoint, while obeying speed and direction constraints. Optionally use the `*lookahead*` parameters to start new trajectory from a point sampled `n` seconds ahead for trajectory being tracked currently.
  - `moveOnSplineVelConstraintsAsync` 💚
- Clear and stop following current trajectory.
  - `clearTrajectory` 💚

Gain setter APIs:

- `setAngleRateControllerGains` 💚
- `setAngleLevelControllerGains` 💚
- `setVelocityControllerGains` 💚
- `setPositionControllerGains` 💚

- `setTrajectoryTrackerGains` 💚

APIs to help generate gate detection datasets:

- Object pose setter and getter:
  - `simSetObjectPose`
  - `simGetObjectPose`
- Object scale setter and getter:
  - `simSetObjectScale` 💚
  - `simGetObjectScale` 💚
- Object segmentation ID setter and getter:
  - `simGetSegmentationObjectID`
  - `simSetSegmentationObjectID`
- Listing all the objects in the scene:
  - `simListSceneObjects` 💚
- Gate specific APIs:
  - `simGetNominalGateInnerDimensions` 💚
  - `simGetNominalGateOuterDimensions` 💚

Questions

Please open a Github Issue on **this** repository (not AirSim) for any technical questions w.r.t. the Neurips competition.

Releases 16

Final Round Binaries - Windows Latest
on Nov 22, 2019

+ 15 releases

Packages

No packages published

Contributors 8

Environments 1

github-pages Active

Languages

Python 90.6% Shell 8.0% Dockerfile 1.4%