User Manual for Mcity AV Challenge

Updated on June 19, 2024. Version 1.9.9

Introduction

Welcome to the User Manual for the Mcity Autonomous Vehicle (AV) Challenge! This manual serves as a comprehensive guide to assist participants in preparations for the challenge. To streamline the configuration process and maintain uniformity across development and testing phases, we primarily employ Docker images to handle simulated testing environments and communication for developing and testing AV decision-making modules. Inside this manual, you will find detailed instructions to jump-start your participation in the challenge. This includes information on software installation, initialization of Docker containers, integration of your AV decision-making module, execution of tests, and the submission process for your entry. Additionally, we offer insights into the criteria for evaluation and the anticipated scheduling for the entirety of the challenge.

Configuration and Installation

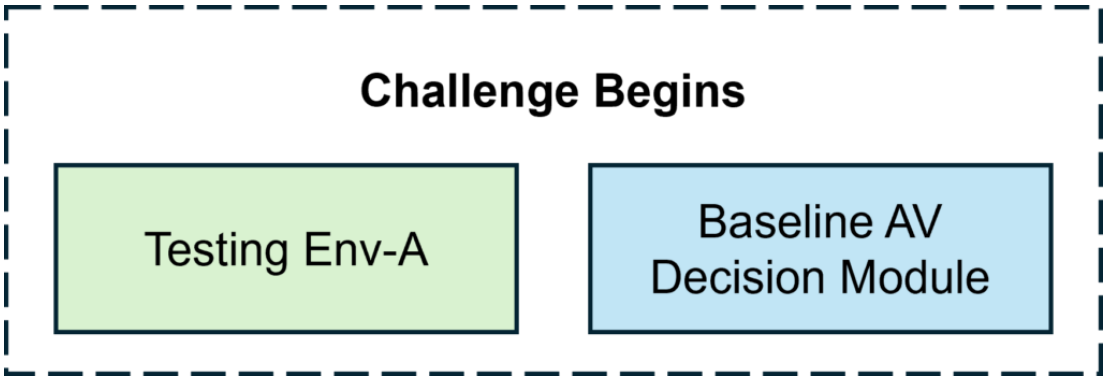
Pre-requirements

1. Operating system: Ubuntu 22.04 system is preferred and tested for this challenge setup.
2. Software:
    - Integrated Development Environment: Visual Studio Code is recommended for code development.
    - Docker Desktop: To install Docker Desktop on your operating system, please follow this link.
    - Docker account: Participants are required to establish a Docker account and set up a private repository to house the Docker image that contains their developed AV decision-making module. To activate the sharing capabilities, it is essential for participants to upgrade to Docker Pro. Additionally, participants should add Mcity Docker account <mcityavchallenge> as a collaborator on their repository to facilitate the contest entry and evaluation process.
3. Hardware**:** The Docker image containing the developed AV decision-making module will be executed on **Amazon Elastic Compute Cloud (EC2)**. The instance type utilized is c6a.2xlarge featuring 8 cores and 16 GB memory. Participants should test the developed AV decision-making module with similar computational requirements to ensure it runs normally with the Mcity AWS cloud environment.

> 💡 **Currently, our platform does not provide support for GPU capabilities. Therefore, it is essential to ensure that your algorithm is designed to operate without the need for GPU resources.**

Installation

Participants will obtain two Docker images for developing and testing their AV decision-making module:



- **Docker Image 1 with Testing Env-A** (named test_env_A) is a public Docker image with the Mcity simulated testing environment that can help participants test the developed AV decision-making module.
- **Docker Image 2 with Baseline AV Decision-making Module** (named baseline_av)**:** This Docker image provided by Mcity includes a baseline AV decision-making module that allows the AV to follow a designated route and interact with Testing Env-A in Docker image 1 through a specified communication protocol. We also provide a high-definition SUMO-format map of the Mcity testing facility, along with an example route for the challenge.

To obtain the two Docker images, please follow the instructions below:

1. Download files from the GitHub repository, which outlines where to pull the Docker images and how to configure them. The current workspace will have the following structure:

```
mcity_av_challenge_workspace/
|__av_decision_making_module/
|_____initial_information/
|________mcity.net.xml # Mcity High-definition map in SUMO format
|________route.csv # waypoints of the example testing route for AV
|_____main.py # Main file to launch the developed AV decision-making module
|_____main.sh # Shell script to run the file "main.py" in loop
|_____visualization_tool.py # Main file to visualize the trajectories
|__output/
|_____trajectory_data/ # Recorded trajectories of simulated AV and other
background vehicles
|_____trajectory_videos/ # Visualizaton videos of the recorded trajectories
|__utils/
|_____download_test_results.py # Main script to download the test results
every week
|__docker-compose-development.yml # YAML file used to configure the
development environment for participants based on Docker
|__README.md
```

3. Pull the Docker images and launch the Docker containers using the following command:

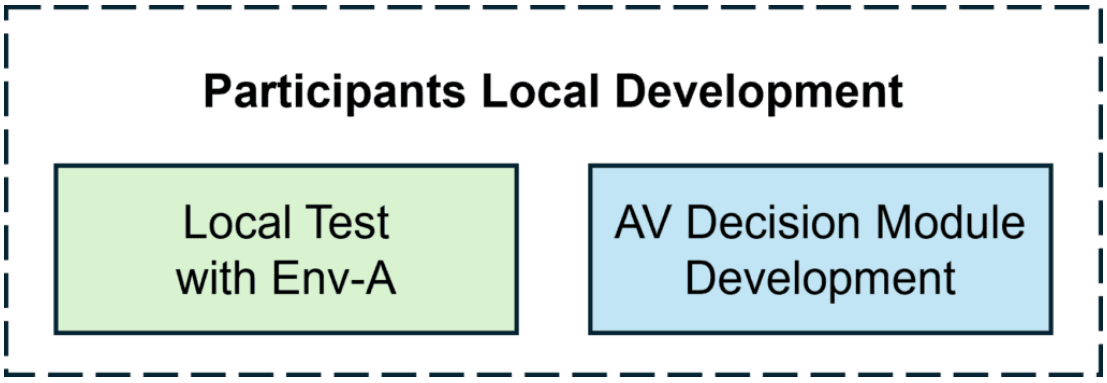
```
docker compose -f docker-compose-development.yml up -d
```

After a brief interval, two Docker containers will launch. By using `docker ps`, you will be able to view the status of both containers. You can also use `docker logs` to monitor the logs of these two containers. As per our configuration,

- One Docker container, named “test_env_a,” initializes the Docker image “test_env_a.” This container is responsible for launching the Mcity simulated testing environment and maintains the communication with the other container, “avalgo,” through Redis.
- The other Docker container, referred to as “avalgo,” initializes the Docker image “baseline_av”; however, it does not automatically run any programs. Please refer to the **Usage** section for running the baseline AV decision-making module as an example. The main codes and data are stored in the directory “/app,” and its structure is as follows:

```
/app/
|__av_decision_making_module/
|_____initial_information/
|________mcity.net.xml
|________route.csv
|_____main.py
|_____main.sh
|_____visualization_tool.py
|__output/
|_____trajectory_data/
|_____trajectory_videos/
```

Usage



1. Launch the Docker container ”avalgo” (Docker image “baseline_av”) and conduct tests with the Docker container “test_env_a” (Docker image “test_env_a”).
   a. At first, initiate the two Docker containers by

```
docker compose -f docker-compose-development.yml up -d
```

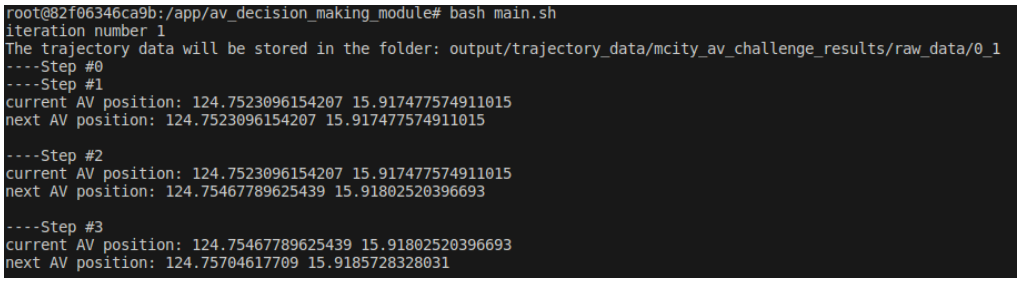
b. Before starting the baseline AV decision-making module, enter the Docker container “avalgo” by

```
docker exec -it avalgo /bin/bash
```

c. Run the following command inside this container to start the baseline AV decision-making module:

```
cd /app/av_decision_making_module && bash main.sh
```

The baseline AV decision-making module will start to run, and you will be able to see the following output:



After the test, participants can find the logged data within the folder named "output/trajectory_data” in the local workspace.

d. After testing, stop the baseline AV decision-making module via Ctrl-C. In the meantime, the Mcity testing environment will automatically restart.

> 💡 **If you want to remove all the Docker images, containers, and network created by `docker compose up`, please first exit the container and then use the following command: `docker compose -f docker-compose-development.yml down --rmi all`.**

2. Regarding the communication between these two Docker containers, we implement Redis for real-time information exchange.

   a. Inside the Docker container "test_env_a,” a Redis server is automatically activated to transmit information about the Mcity simulated testing environment, This includes data about background vehicles and states of traffic signals.

   b. Within the Docker container "avalgo," a Redis client is set up to obtain information about the Mcity simulated testing environment and send the outcomes of the developed AV decision-making module to the "test_env_a" container.

> 💡 **Ensure the local machine does not have the Redis server running during tests since the Docker container “test_env_a” has launched a Redis server.**

3. AV decision-making module development and integration.

a. A baseline AV decision-making module is provided as an example. To prepare for developing and integrating the AV decision-making module, participants are instructed to carefully review this example file step by step.

- First, import necessary libraries and utility functions.

```
import csv
import math
import numpy as np
import time

from mrav.mcity_mr_av import (
    MRAVTemplateMcity,
)
# This Python class is a basic component for any developed AV
# decision-making module and the user should inherit from it.
```

- Implement the developed AV decision-making module.

```
class AVDecisionMakingModule(MRAVTemplateMcity):
    """This is an example AV decision making module that reads a logged
trajectory from a file and follows it."""

    def initialize_av_algorithm(self):
        """This function will be used to initialize the developed AV dd
ecision-making module. In this example, we read the predefined trajecto
ry from a file."""
        trajectory = []
        with open("/baseline_av_data/baseline_av_trajectory.csv", "r")
as f:
            reader = csv.reader(f)
            trajectory = []
            for row in reader:
                orientation = float(row[3])
                if orientation > math.pi:
                    orientation -= 2 * math.pi
                trajectory.append(
                    {
                        "x": float(row[1]),
                        "y": float(row[2]),
                        "orientation": orientation,
                        "velocity": float(row[4]),
                    }
                )
        self.trajectory = {
            "x_vector": np.array([point["x"] for point in trajectory]),
```

```
            "y_vector": np.array([point["y"] for point in trajectory]),
            "orientation_vector": np.array(
                [point["orientation"] for point in trajectory]
            ),
            "velocity_vector": np.array([point["velocity"] for point in
trajectory]),
        }
        self.trajectory_index = 0

    def derive_planning_result(self, step_info):
        """This function will be used to compute the planning results b
ased on the observation from "step_info". In this example, we find the
closest point in the predefined trajectory and return the next waypoint
as the planning results."""
        # parse the step_info
        av_state = step_info["av_info"]
        tls_info = step_info["tls_info"]
        av_context_info = step_info["av_context_info"]
        # find the closest point in the predefined trajectory
        current_x = av_state["x"]
        current_y = av_state["y"]
        if self.trajectory_index > len(self.trajectory["x_vector"]) -
1:
            next_x = self.trajectory["x_vector"][-1]
            next_y = self.trajectory["y_vector"][-1]
        else:
            next_x = self.trajectory["x_vector"][self.trajectory_index]
            next_y = self.trajectory["y_vector"][self.trajectory_index]

        print("current AV position:", current_x, current_y)
        print("next AV position:", next_x, next_y)
        planning_result = {
            "timestamp": time.time(),
            "time_resolution": 0.1,
            "next_x": self.trajectory["x_vector"][self.trajectory_inde
x],
            "next_y": self.trajectory["y_vector"][self.trajectory_inde
x],
            "next_speed": self.trajectory["velocity_vector"][self.traje
ctory_index],
            "next_orientation": self.trajectory["orientation_vector"][
                self.trajectory_index
            ],
        }
```

```
        self.trajectory_index += 1
        return planning_result
```

- Launch the AV decision-making module.

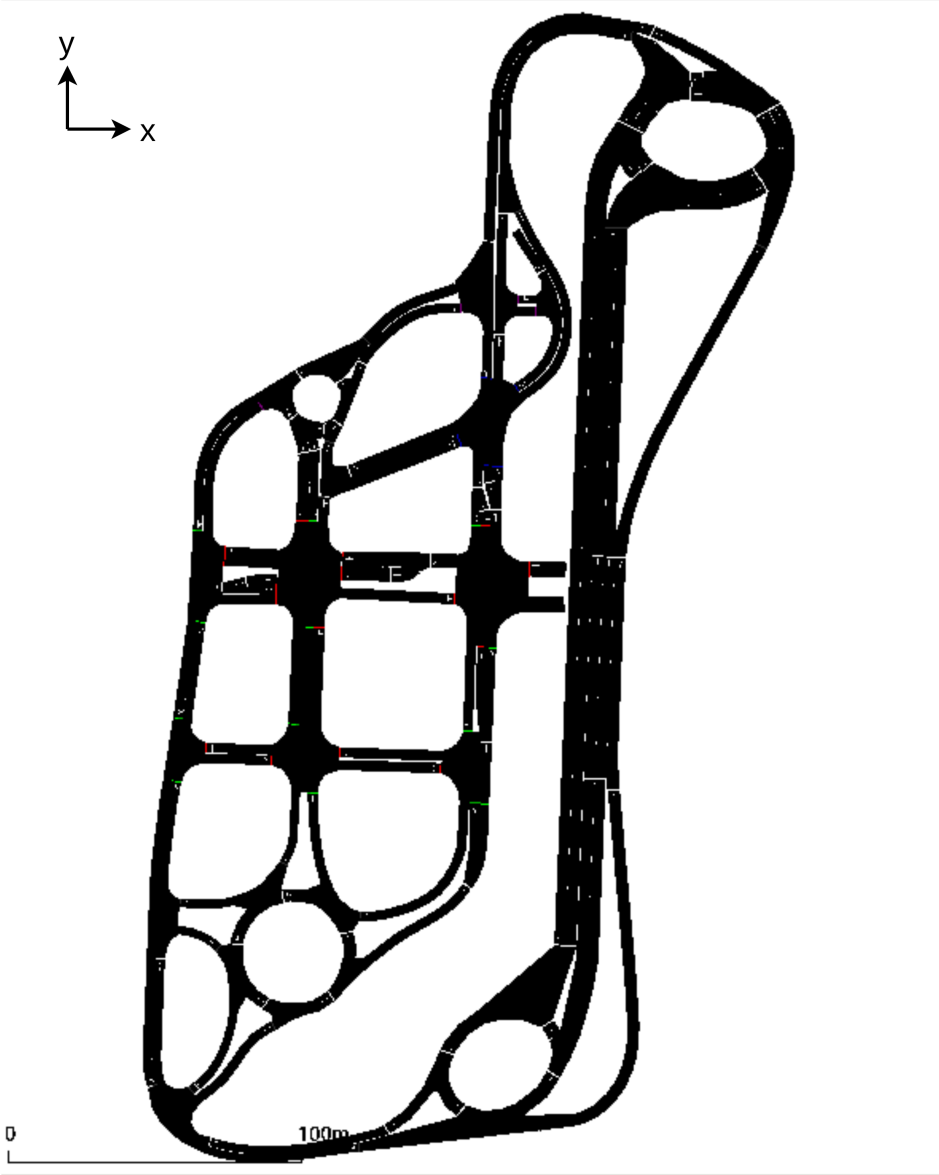
```
# Create an instance of the AV decision-making module and run it
av_decision_making_module = AVDecisionMakingModule()
av_decision_making_module.run()
```

b. Understand the Mcity High-definition map in SUMO format and the AV testing route.

- Mcity High-definition map in SUMO format: This map is located in "mcity_av_challenge_workspace/av_decision_making_module/initial_information/mcity.net.xml." To visualize the map, it is suggested to install SUMO following this guidance and run the following command:

```
sumo-gui -n /path/to/mcity.net.xml
```

  Participants will be able to see the SUMO map:



With this graphical user interface (GUI), participants can easily access essential map information like road speed limits, widths, etc. Please refer to this link to get familiar with the operation of the GUI. Moreover, there's a handy Python tool called Sumolib designed specifically for managing SUMO networks. Participants can install it using the command `pip3 install sumolib`.

- Initial status for AV: The starting point is located at coordinates (124.752310, 15.917478) within the SUMO reference frame. The AV will start with its orientation as 0.326795 radians and its velocity as 0 meters per second.
- AV Testing Route for the Challenge: The AV testing route file is stored at "av_decision_making_module/initial_information/route.csv." It comprises a series of waypoints in SUMO coordinates, with each consecutive pair of waypoints spaced 0.1 meters apart.

AV Testing Route for the Challenge

> 💡 **Please note the scoring will not be influenced by how well the AV decision-making module can track the waypoints.**

c. Install the necessary dependencies and libraries inside the Docker container "avalgo" to develop participants' AV decision-making module.

- To install some Python packages under this Ubuntu OS, please run:

```
pip3 install <Python package name>
```

- To install some software packages under this Ubuntu OS, please run:

```
apt install <software package name>
```

d. Develop the AV decision-making module by **customizing the Python script** found at "av_decision_making_module/main.py" under the current workspace.

> 💡 **Please note the local directory "av_decision_making_module" has been mounted to the folder "/app/av_decision_making_module" of the Docker container "avalgo." It means any changes to the local directory "av_decision_making_module" will also be effective for the folder "/app/av_decision_making_module" inside the container. Participants should add any necessary source codes or data to the local directory "av_decision_making_module." They will automatically appear inside the Docker container "avalgo."**

Specifically, participants need to modify two main functions of the "AVDecisionMakingModule" Python class:

- `initialize_av_algorithm` **method**

  This method does not have any input or output. You can use it to complete the necessary configurations for the developed AV decision-making module.

- `derive_planning_result` **method**

  This method takes a dictionary called `step_info` as input, which holds the latest simulation environment details sent from the Docker container "test_env_a" via Redis. The AV decision-making module then utilizes this information to calculate the `planning_result`, which is then automatically transmitted back to the Docker container "test_env_a."

  - `step_info` is a dictionary with the following keys:
    - `av_context_info`: A dictionary providing the background vehicle information in the simulation. Each key within this dictionary denotes a unique name assigned to background vehicles (named "BV_<unique ID>"), and the corresponding value is also a dictionary containing the following keys:
      - 'x': <float> x-coordinate of the vehicle center within the SUMO coordinate system (meters).
      - 'y': <float> y-coordinate of the vehicle center within the SUMO coordinate system (meters).
      - 'length': <float> Vehicle length (meters).
      - 'width': <float> Vehicle width (meters).
      - 'height': <float> Vehicle height (meters).
      - 'orientation': <float> Vehicle orientation (radians, from $-\pi$ to π, with 0 pointing to the east and $\pi/2$ pointing to the north).
      - 'yaw_rate': <float> Vehicle yaw rate (radians per second, and the positive direction is counter-clockwise).
      - 'speed_long': <float> Longitudinal speed of the vehicle (meters per second).
      - 'speed_lat': <float> Lateral speed of the vehicle (meters per second, and the positive direction is to the left).

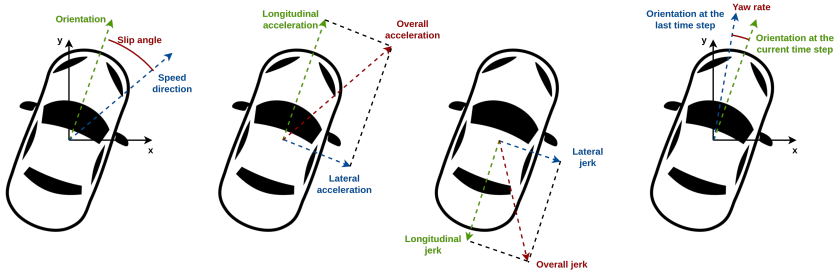
- 'accel_long': <float> Longitudinal acceleration of the vehicle (meters per second squared).
- 'accel_lat': <float> Lateral acceleration of the vehicle (meters per second squared, and the positive direction is to the left).
- 'edge_id': <string> Edge ID of the vehicle (in SUMO map).
- 'lane_id': <string> Lane ID of the vehicle (in SUMO map).
- 'leading_info': {'is_leading_cav': <bool> Whether this vehicle is leading the AV, 'distance': <float/None> If this vehicle is leading AV, this value will be the distance between AV and this vehicle, else it will be None}.

- `av_info`: The information regarding the AV will also be transmitted after initializing the AV control, following the same structure as the value of `av_context_info`.
- `tls_info`: A dictionary providing the traffic signal information in the Mcity testing environment. It contains the following keys:
  - 'next_tls_id': <string> Name of the closest traffic light.
  - 'distance_to_next_tls': <float> Distance from AV to the closest traffic light along the route (meters).
  - 'next_tls_state': <string> State of the closest traffic light, with 'G' or 'g' representing the green light, 'Y' or 'y' representing the yellow light, and 'R' or 'r' representing the red light.
  - 'tls_info': A dictionary contains all the traffic light information within the simulated world. The keys are the traffic light name, and the values are {'tls_state': <string> State of the traffic light, following the definition of traffic light state in SUMO}.

- `planning_result`, returned by the `derive_planning_result` method, is a dictionary with the following keys:
  - 'timestamp': <float> Current timestamp (seconds).
  - 'time_resolution': <float> Time resolution of the planning results (seconds). Currently, we only support 0.1s time resolution. So, **please make sure the time resolution of your planning results is 0.1s.**
  - 'next_x': <float> The x coordinate of the vehicle center in the next 0.1s within the SUMO coordinate system (meters).
  - 'next_y': <float> The next y coordinate of the vehicle center in the next 0.1s within the SUMO coordinate system (meters).
  - 'next_speed': <float> The desired overall velocity in the next 0.1s (meters per second).
  - 'next_orientation': <float> The planned orientation in the next 0.1s (in radians, $-\pi$ to π, with 0 pointing to the east and $\pi/2$ pointing to the north).

> 💡 **Please ensure that the data format for `planning_result` should remain unchanged.**

> 💡 **Please note that `planning_result` will be transmitted to the Docker container "test_env_a" via Redis to simulate the AV in the testing environment without any modification. Therefore, the planning result should comply with certain constraints related to vehicle dynamics, specifically:**
>
> **1. Slip angle: -15 ~ 15 (degrees);**
> **2. Overall acceleration (combination of longitudinal and lateral acceleration): -7.06 ~ 2.87 (meters per second squared);**
> **3. Overall jerk (combination of longitudinal and lateral jerk): -10 ~ 10 (meters per second cubed);**
> **4. Yaw rate: -50 ~ 50 (degrees per second);**
> **The definitions of these variables are illustrated in the following figure. For ease of use, our system will automatically verify if the planning results adhere to the specified constraints. Should notifications like "** *Warning: The overall jerk (24) is higher than the maximum jerk (10)* **" emerge in the command line output, it indicates a breach of constraints, necessitating participants to adjust their planning results accordingly. Failure to do so will result in the current test not succeeding, which will, in turn, affect the overall score.**



e. Conduct preliminary tests to debug the developed AV decision-making module by running the command inside the "avalgo" Docker container:

```
cd /app/av_decision_making_module
bash main.sh
```

For each test, there are three common results:

i. If the AV decision-making module runs normally, the terminal will output:

```
Exiting...
The trajectory data have been stored in the folder: output/trajectory_data/mcity_av_challenge_results/raw_data/0_1
```

ii. If there are some errors within the AV decision-making module, such as importing some uninstalled modules, the terminal will output:

```
An unexpected error occurred: No module named 'UninstalledModule'
The trajectory data have been stored in the folder: output/trajectory_data/mcity_av_challenge_results/raw_data/0_1
```

iii. If the whole program is stopped by Ctrl-C, the terminal will output:

```
Received interrupt signal. Exiting...
The trajectory data have been stored in the folder: output/trajectory_data/mcity_av_challenge_results/raw_data/0_1
```

Under different situations, participants are given guidance on locating trajectory data. In this context, the trajectory data is stored in the folder "output/trajectory_data/mcity_av_challenge_results/raw_data/0_1."

4. Visualize the AV and BV trajectories for verification.

   We offer a tool to visualize the logged AV and BV trajectory data, which has been installed inside the "avalgo" Docker container. This tool can assist participants in assessing whether the developed AV decision-making module runs as expected.

   - The logged data will be stored in a folder named "output/trajectory_data" within the current workspace. The folder's structure is as follows:

     ```
     mcity_av_challenge_workspace/output/trajectory_data/
     |__mcity_av_challenge_results/
     |_____raw_data/
     |________0_1/
     |___________fcd.xml # Trajectory data
     |___________...
     |________0_2/
     |________...
     ```

   - Below is the guidance on how to utilize the visualization tool. Please first enter the Docker container "avalgo" by running `docker exec -it avalgo /bin/bash`. Then, please run the following command inside the container:

     ```
     # If you want to visualize all the trajectories:
     python3 /app/av_decision_making_module/visualization_tool.py
     # If you want to visualize one specific trajectory:
     # Please clarify the path of the folder containing the trajectory data,
     # And the folder path should be printed out after the lates test.
     python3 /app/av_decision_making_module/visualization_tool.py --trajectory_
     folder output/trajectory_data/mcity_av_challenge_results/raw_data/0_1
     ```

     Eventually, the videos will be generated within a local directory named 'output/trajectory_videos' organized with the structure detailed below:

     ```
     mcity_av_challenge_workspace/output/trajectory_videos/
     |__0_1.mp4 # Trajectory video
     ```

```
|__0_2.mp4
|__...
```

> 💡 **To playback the videos, you may need to adjust the permissions of the data folder by executing this command:**
>
> `sudo chown -R $(id -u):$(id -g) output`.

- Video example

> https://prod-files-secure.s3.us-west-2.amazonaws.com/07cfc0c4-5aac-453a-bcec-0610db1f74ea/3f85e104-6130-4f66-84f4-c24a890d926b/example.mp4

5. To safeguard your intellectual property, it is recommended to obscure your source code using a tool like Pyarmor. After obfuscation, it's advisable to retest the AV decision-making module to ensure that it still operates as intended.
6. Upload the Docker image to Docker Hub for the contest.
   a. Set up a Docker Hub account under **<participant_dockerhub_account>.**
   b. Establish a private repository named **private_repo_<team_id>**.

   > 💡 **A team ID will be assigned after registering for this AV challenge.**

   c. Copy the latest version of the source code and data within the local directory "av_decision_making_module" to the Docker container "avalgo" by running:

   ```
   docker exec avalgo rm -rf /app/av_decision_making_module_user
   docker cp av_decision_making_module avalgo:/app/av_decision_making_module_user
   ```

   > 💡 **Make sure to remove the old code and data before copying the latest version.**

   d. Export the Docker container "avalgo" which contains the developed AV decision-making module as a customized Docker image with **the tag to be your team ID** under your private repository:

   ```
   docker commit avalgo <participant_dockerhub_account>/private_repo_<team_id>:tag_<team_id>
   ```

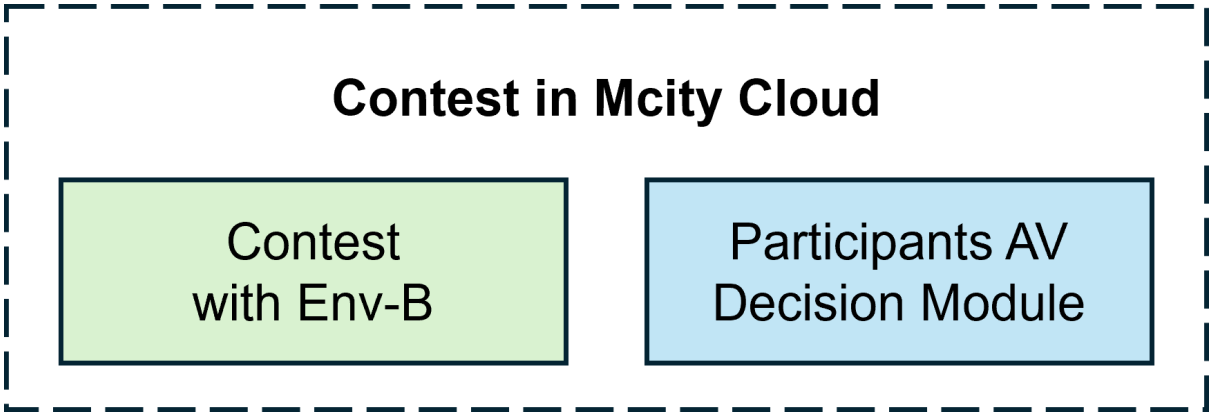
   e. Push the Docker image to your private repository through the following command:

```
docker push <user_dockerhub_account>/private_repo_<team_id>:tag_<team_id>
```

f. Set Mcity Docker Hub account <**mcityavchallenge**> as the collaborator of your private repository and share your Docker Hub account name with the Mcity AV Challenge team through email <mcity-av-challenge@umich>. Then, we will have access to your Docker image for assessment.

Additional Information

- Contest in Mcity Cloud



Every Monday at 0:00 AM (EST), Mcity retrieves participants' latest Docker image from Docker Hub, which contains their AV decision-making module. This image is then uploaded to the Mcity cloud for assessment within Env-B, a private environment featuring different traffic conditions from Env-A.

In Env-B, the AV is expected to complete the testing routes safely and efficiently. Each testing round involves a substantial volume of testing loops to accurately evaluate performance using scoring functions.

AV performance will be assessed across five dimensions:

- **Safety:** crash rate, crash severity, safety metrics like Post Encroachment Time (PET)
- **Rule-compliance:** off-route rate, red-light running rate, speeding rate
- **Efficiency:** average traveling speed
- **Comfort:** longitudinal and lateral acceleration and jerk
- **Trajectory Completion:** successfully finishing the predefined route without deviation and dynamics feasibility violations (i.e., violations of acceleration/deceleration, jerk, yaw rate, and slip angle limits)

- **Leaderboard Updates and Evaluation**

  The leaderboard will be updated weekly. It will reflect the current standings of the top 10 teams and their scores, including the total score and the scores for safety, efficiency, and comfort, respectively.

  All teams are able to access the Mcity S3 bucket to retrieve evaluation data of their own, including score, example failure cases with trajectories, etc., for review and algorithm refinement. This will

enable participants to refine and resubmit their AV decision module prior to the challenge's conclusion. To download all the content automatically, please run the following command:

```
# Download test data in participant's directory
# There is one specific dependency, which can be installed using
# "pip install boto3"
python utils/download_test_results.py -t <team_id> -r <round_number> -k <uniq
ue_key>
```

Then, participants will be able to see the evaluation results in the local workspace, which has the following structure:

```
|__test_data/
|_____round1/
|________0_1_*/
|___________fcd.mp4 # Trajectory video
|___________fcd.xml # Trajectory data
|________... # There are totally ten trajectories to help participants improv
e their algorithm
|________all_score.csv # Details about the scores
|________critical_trajectory.csv # Details about the selected trajectories
```

- **Q & A**
- **Support**

  Should you face any challenges while submitting or need assistance with navigating the simulation environment, feel free to open an issue on our GitHub repository. Alternatively, you can reach out to the Mcity support team via email for any inquiries.

> 🙌 Your contribution to the Mcity AV Challenge is invaluable, and we strive to make the submission process as smooth as possible. We look forward to your innovative solutions and wish you the best of luck in the competition.