Getting start

NETH CARLA Challenge

Outline

2.1 SOURCE CODE STRUCTURE

2.2 CAMERA SENSOR

2.3 DISPLAY

2.4 CARLA CONTROL API

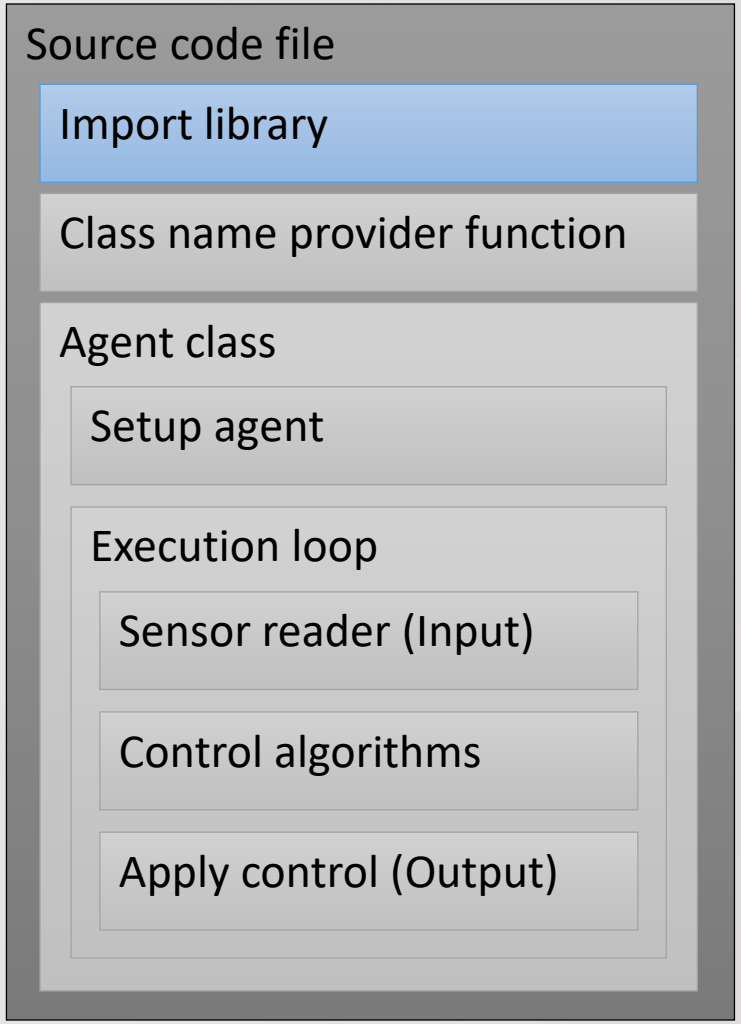
2.1 Source code structure

Source code file

- Import library
- Class name provider function
- Agent class
  - Setup agent
  - Execution loop
    - Sensor reader (Input)
    - Control algorithms
    - Apply control (Output)

```python
import carla
from leaderboard.autoagents.autonomous_agent import AutonomousAgent

# For get class name
def get_entry_point():
    return 'Agent_2_1' # Change this to match your class name below

class Agent_2_1(AutonomousAgent):
    def setup(self, path_to_conf_file):
        """
        Setup the agent parameters
        """
        pass

    def run_step(self, input_data, timestamp):
        """
        Execute one step of navigation.
        """
        # Edit your code here

        # Return control value to CARLA server
        control = carla.VehicleControl()
        control.steer = 0.0          # A scalar value to control the vehicle steering [-1.0, 1.0]. Default is 0.0.
        control.throttle = 0.0       # A scalar value to control the vehicle throttle [0.0, 1.0]. Default is 0.0.
        control.brake = 0.0          # A scalar value to control the vehicle brake [0.0, 1.0]. Default is 0.0.
        control.hand_brake = False   # Determines whether hand brake will be used. Default is False.
        control.reverse = False      # Determines whether the vehicle will move backwards. Default is False.
        return control

```

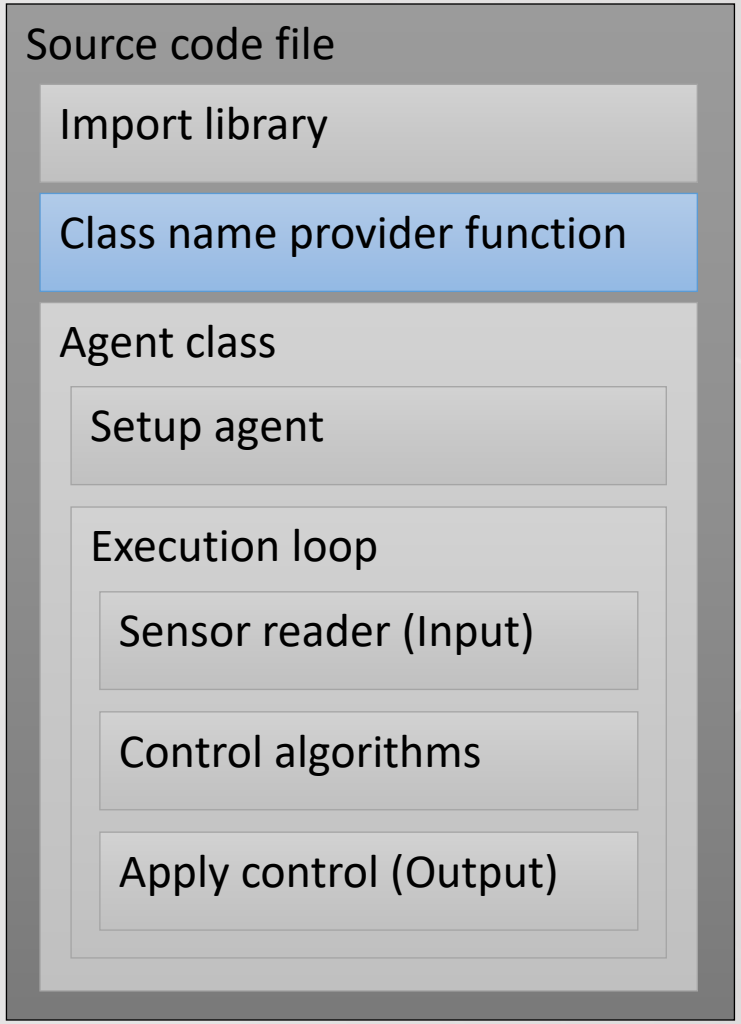
2.1 Source code structure (Import library)




```python
import carla
from leaderboard.autoagents.autonomous_agent import AutonomousAgent
```

Import 2 necessary library for Agent. There are “carla” and “AutonomousAgent”. You can import your library here.

2.1 Source code structure (Class name provider function)




```
4   # For get class name
5   def get_entry_point():
6       return 'Agent_2_1' # Change this to match your class name below
```

This function is providing the agent class name to the main program.
So, the return string must match with agent class name below.

2.1 Source code structure (Agent class)

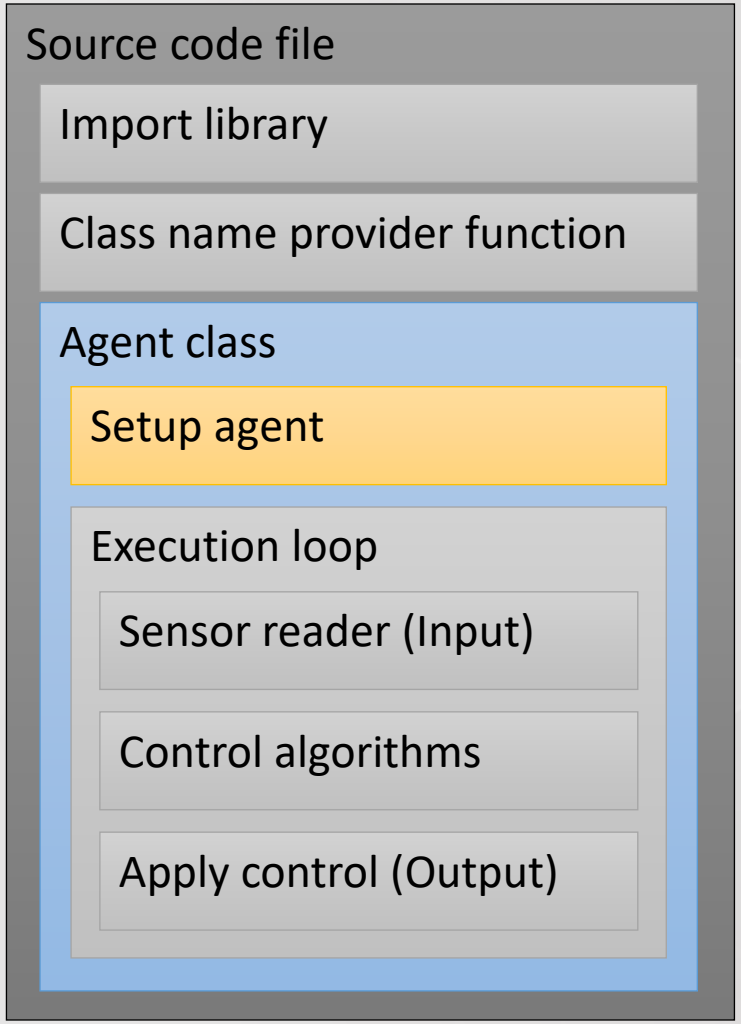
Source code file

- Import library
- Class name provider function
- Agent class
  - Setup agent
  - Execution loop
    - Sensor reader (Input)
    - Control algorithms
    - Apply control (Output)

This class is our agent. It is consisted of “setup” and “run_step” function.
We write code here to control our agent.

```python
class Agent_2_1(AutonomousAgent):
    def setup(self, path_to_conf_file):
        """
        Setup the agent parameters
        """
        pass

    def run_step(self, input_data, timestamp):
        """
        Execute one step of navigation.
        """
        # Edit your code here

        # Return control value to CARLA server
        control = carla.VehicleControl()
        control.steer = 0.0          # A scalar value to control the vehicle steering [-1.0, 1.0]. Default is 0.0.
        control.throttle = 0.0       # A scalar value to control the vehicle throttle [0.0, 1.0]. Default is 0.0.
        control.brake = 0.0          # A scalar value to control the vehicle brake [0.0, 1.0]. Default is 0.0.
        control.hand_brake = False   # Determines whether hand brake will be used. Default is False.
        control.reverse = False      # Determines whether the vehicle will move backwards. Default is False.
        return control

```

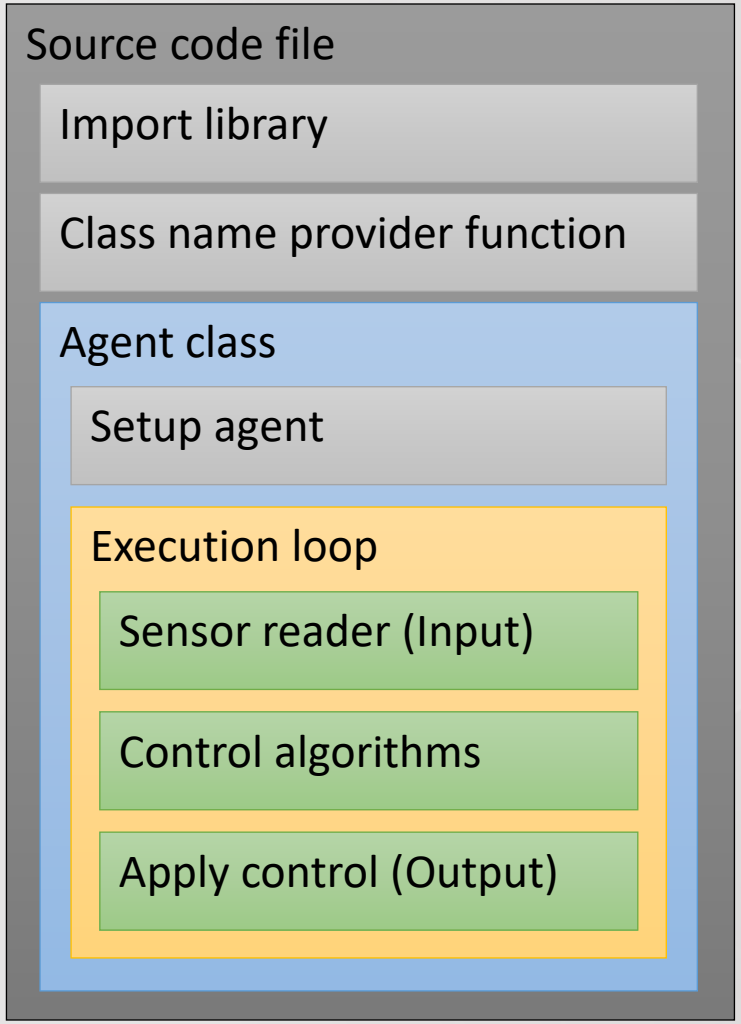
2.1 Source code structure (Setup agent)




```
9     def setup(self, path_to_conf_file):
10        """
11        Setup the agent parameters
12        """
13        pass
14
```

Setup function execute once before execution loop.
You can get parameters in config file by read “path_to_conf_file” file.
“path_to_conf_file” can be set at “--agent-config” argument in “run_agent.bat”.

2.1 Source code structure (Execution loop)



Execution loop execute every time step until finish route or timeout.
It has 3 steps.

1. Sensor reader : We can get the sensors data from “input_data”
   and get current time step from “timestamp”.
2. Control algorithms : We write agent control code here.
3. Apply control : We send control command through “control” variable.

For more details about “Sensor reader” and “Control algorithms”. We will talk later.

```python
    def run_step(self, input_data, timestamp):
        """
        Execute one step of navigation.
        """
        # Edit your code here

        # Return control value to CARLA server
        control = carla.VehicleControl()
        control.steer = 0.0          # A scalar value to control the vehicle steering [-1.0, 1.0]. Default is 0.0.
        control.throttle = 0.0       # A scalar value to control the vehicle throttle [0.0, 1.0]. Default is 0.0.
        control.brake = 0.0          # A scalar value to control the vehicle brake [0.0, 1.0]. Default is 0.0.
        control.hand_brake = False   # Determines whether hand brake will be used. Default is False.
        control.reverse = False      # Determines whether the vehicle will move backwards. Default is False.
        return control
```

2.2 Camera sensor (Attribute)

Camera senser provide a streaming of BGRA 32-bit image with following attributes.

Attribute	Value	Remark
Mounting position (x, y, z)	(0.45, 0.0, 2.15)	Reference from origin of agent vehicle.
Mounting angle (roll, pitch, yaw)	(0.0, 0.0, 0.0)	Reference from origin axis of agent vehicle.
Image size (width, height)	(800, 600)	Image size in pixels.
Camera FOV	100	Horizontal field of view in degrees.

Note: All attribute value are fix.

2.2 Camera sensor (How to get data 1/2)

Camera sensor provide an array of frame and image.

- Frame is index of sensor data.
- Image is BGRA 32-bit image.

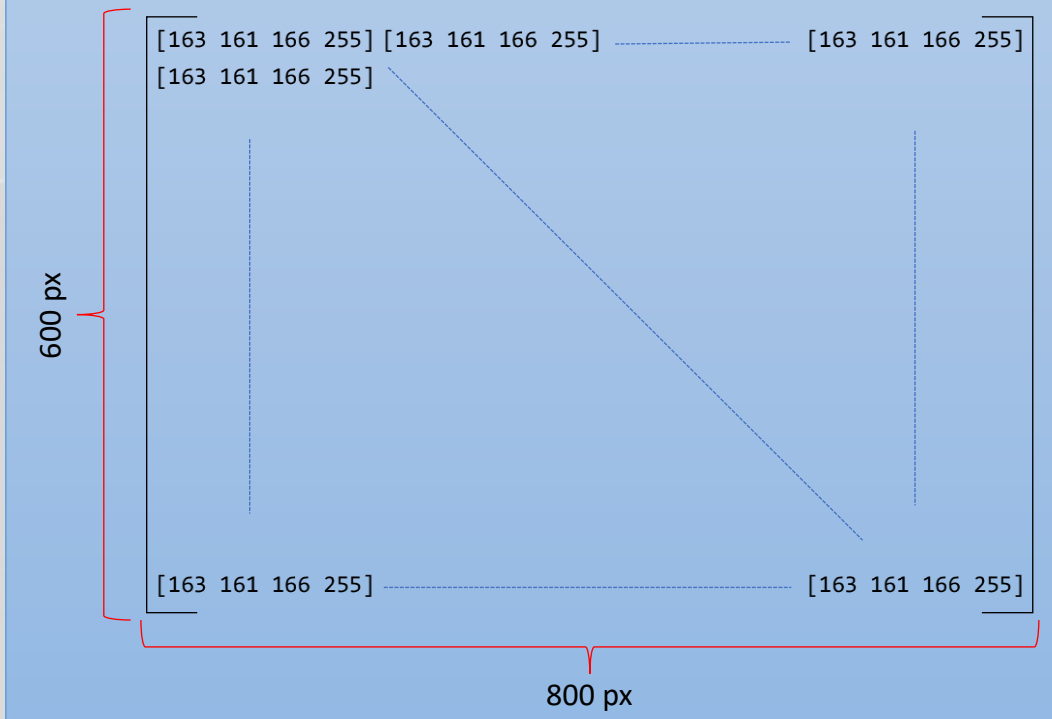
How to get value

- `[frame, image] = input_data['CAMERA']`
- `frame = input_data['GPS'][0]`
- `image = input_data['GPS'][1]`
- `[B, G, R, A] = image[0][0]`

Sensor Data

(204, image)

Image (Blue, Green, Red, Alpha)



2.2 Camera sensor (How to get data 2/2)

“agent_22.py” is shown how to get camera sensor from input_data in runstep loop.

```
15      def run_step(self, input_data, timestamp):
16          """
17          Execute one step of navigation.
18          """
19          # Edit your code here
20
21          # frame = Frame of sensor data
22          # image = 800x600 BGRA 32-bit/pixel image
23          [frame, image] = input_data['CAMERA']
24          print(f"frame: {frame}, image: {image}")
```

2.3 Display (1/2)

Pygame library made for create 2D game with python. It has a lot of tools for 2D rendering. So, we use pygame library to display steaming of camera sensor.

For more information about pygame. Please refer to https://www.pygame.org/docs/

2.3 Display (2/2)

We create Display class and initial it in setup stage. Then call it every run step loop for rendering. Please refer to “agent_23.py”.

```python
class Display():
    """
    Class to display the video stream from front camera.
    """

    def __init__(self):
        self.__width = 800
        self.__height = 600
        self.__surface = None

        pygame.init() # Initialize pygame
        self.__display = pygame.display.set_mode((self.__width, self.__height), pygame.HWSURFACE | pygame.DOUBLEBUF) # Create the window and set the size
        pygame.display.set_caption("Agent 23") # Set the window caption

    def render(self, input_data):
        """
        Render the image from the front camera
        """
        # Did the user click the window close button?
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit() # Close the window

        image = input_data['CAMERA'][1][:, :, -2::-1] # Get the image from the sensor data and convert it to RGB

        self.__surface = pygame.surfarray.make_surface(image.swapaxes(0, 1)) # Convert the image to a pygame surface and display it
        if self.__surface is not None:
            self.__display.blit(self.__surface, (0, 0)) # Display the image on the screen
        pygame.display.flip() # Update the display
```

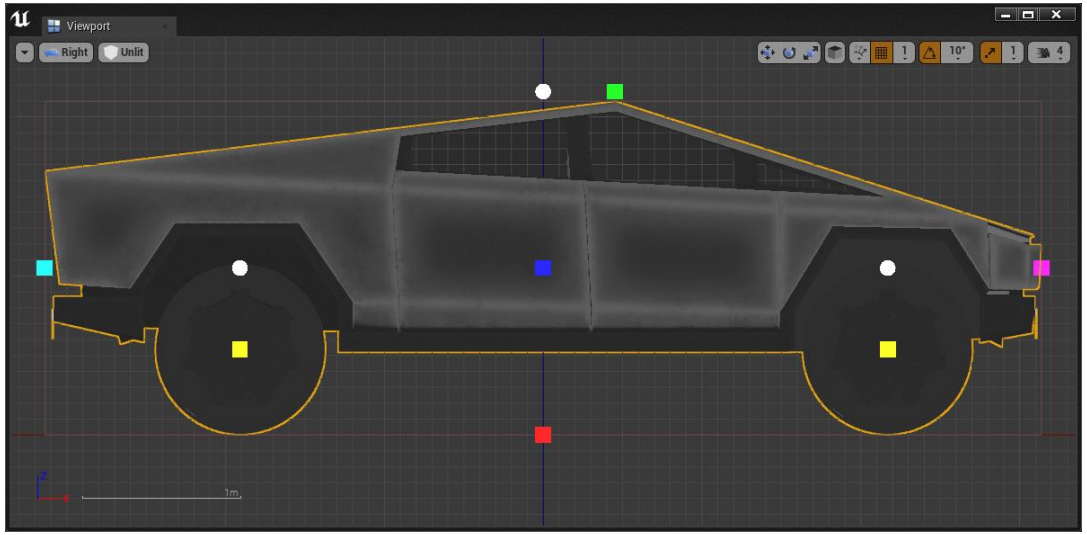
2.4 CARLA control API

We can control our agent by using following Control API.

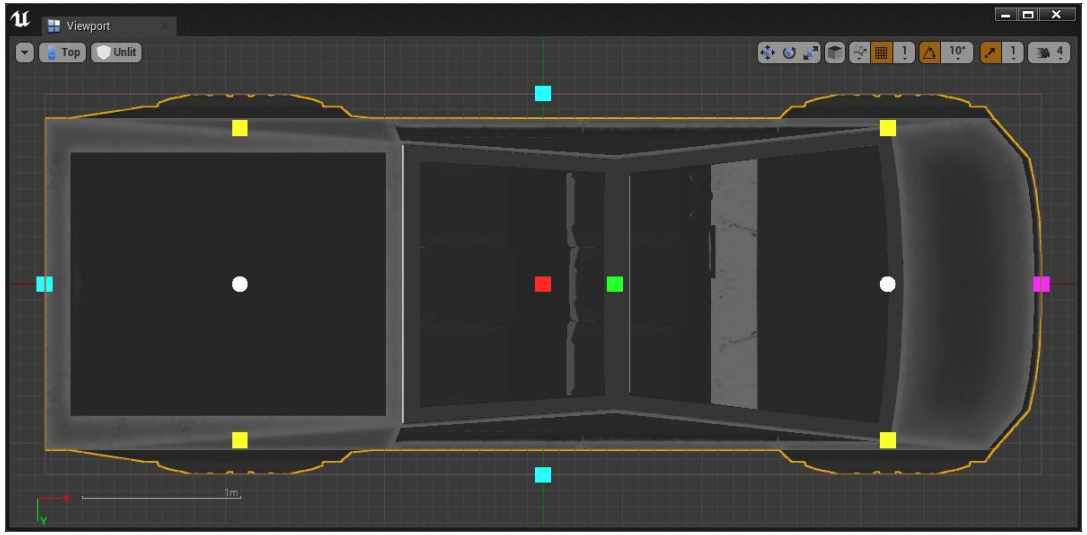
Variable	Type	Value Range	Description
steer	float	[-1.0, 1.0]	A scalar value to control the vehicle steering. Default is 0.0.
throttle	float	[0.0, 1.0]	A scalar value to control the vehicle throttle. Default is 0.0.
brake	float	[0.0, 1.0]	A scalar value to control the vehicle brake. Default is 0.0.
hand_brake	bool	[False, True]	Determines whether hand brake will be used. Default is False.
reverse	bool	[False, True]	Determines whether the vehicle will move backwards. Default is False.

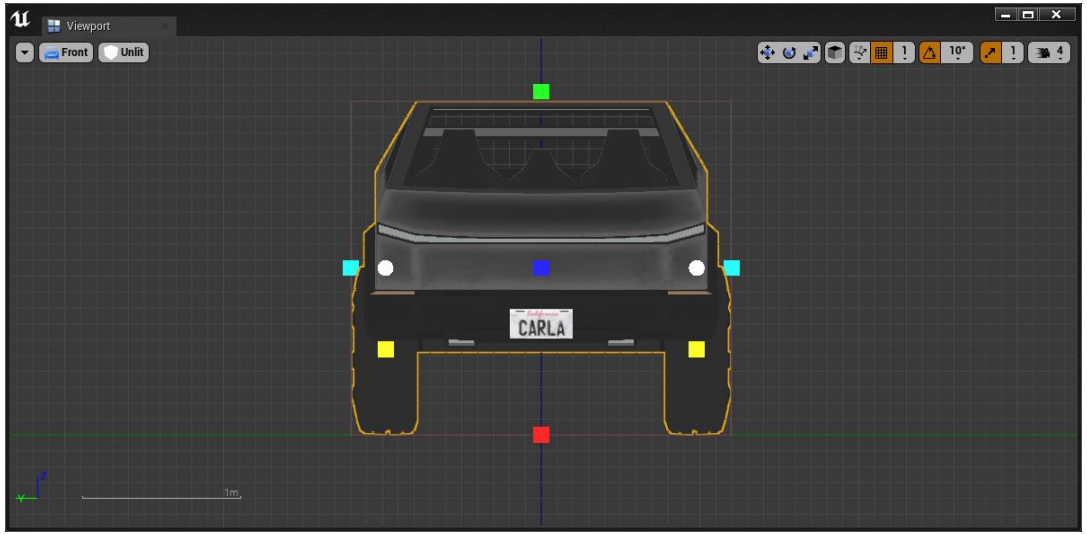
We show simple control by apply 100% of throttle in “agent_24.py”. Agent vehicle will go strength forward until terminate by exceeding waypoints.

Cybertruck dimension

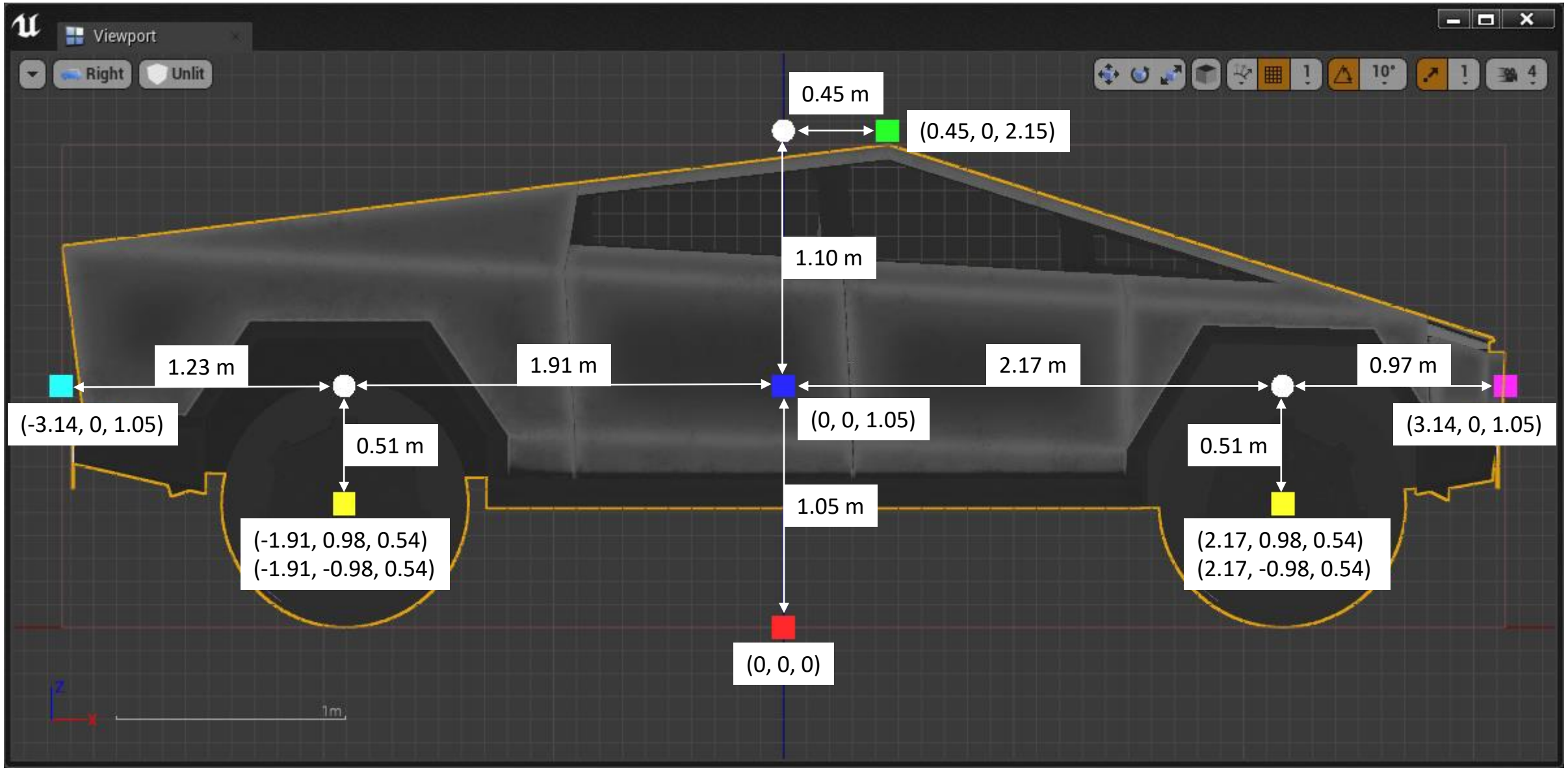


- Origin
- Car center
- Sensors
- Wheels
- Car front
- Car back and side
- Reference point

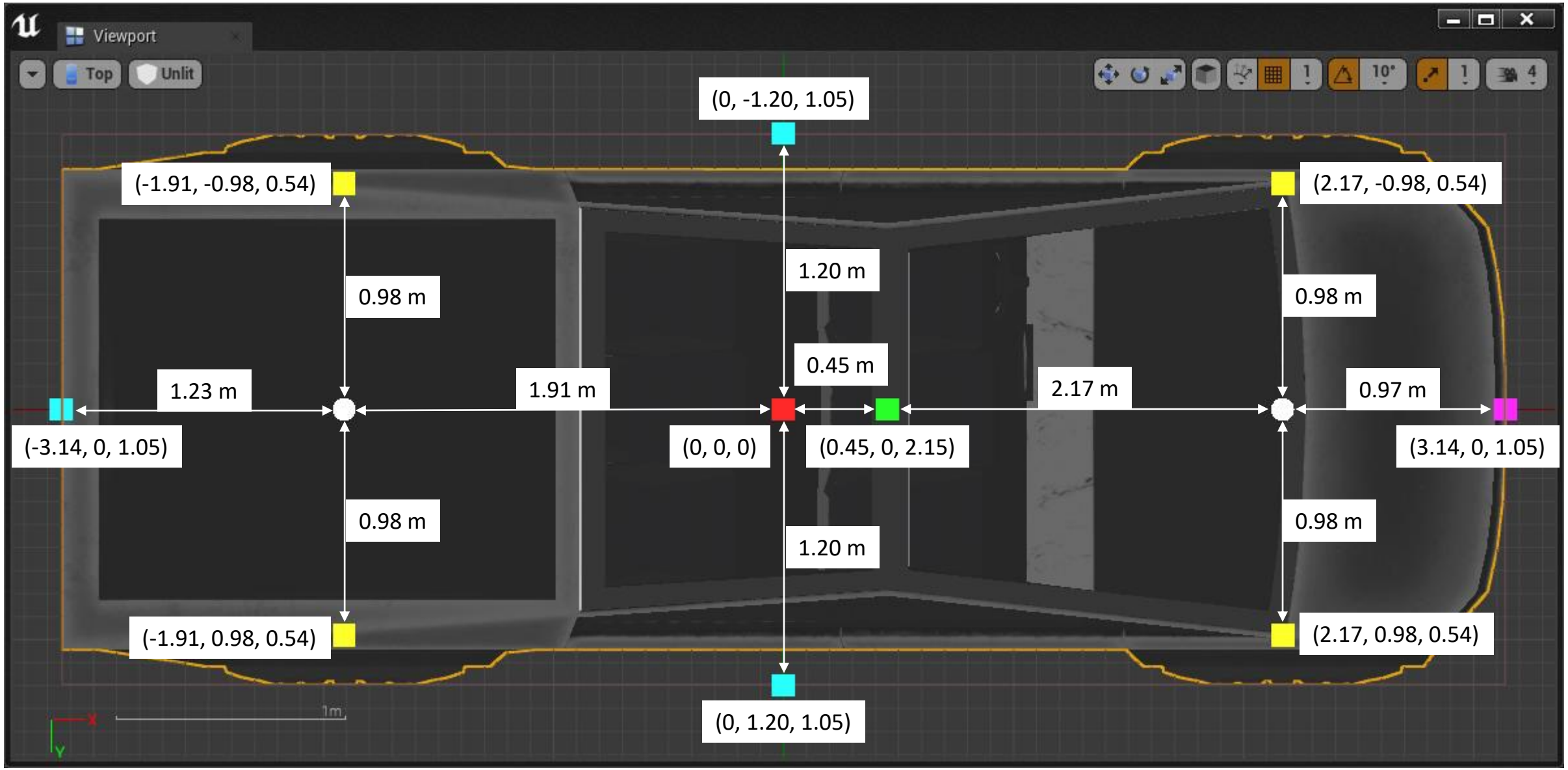




Cybertruck dimension (side view)



Cybertruck dimension (top view)



Cybertruck dimension (front view)