ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Институт компьютерных наук и технологий

Отчет о прохождении учебной практики (ознакомительной)

Курякин Данила Александрович
(Ф.И.О. обучающегося)

1 курс магистратуры, гр.3540901/12001

09.04.01 «Информатика и вычислительная техника»
(Направление подготовки (код и наименование)

Место прохождения практики: Высшая школа интеллектуальных систем и суперкомпьютерных технологий (ВШИСиСТ) ИКНТ ФГАОУ ВО «СПбПУ» с использованием электронного обучения и дистанционных образовательных технологий.

Сроки практики: с 14 января 2022 по 25 января 2022 г.

Руководитель практики:

Нестеров С.А., старший преподаватель ВШИСиСТ ИКНТ
(Ф.И.О., уч.степень, должность)

Оценка (зачет): _____

Руководитель практики: / С.А. Нестеров /

Обучающийся: /Д. А. Курякин/

Дата: 25.01.2022

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Институт компьютерных наук и технологий

ИНДИВИДУАЛЬНЫЙ ПЛАН (ЗАДАНИЕ И ГРАФИК) ПРОВЕДЕНИЯ ПРАКТИКИ

Ф.И.О. обучающегося Курякин Данила Александрович

Направление подготовки 09.04.01 «Информатика и вычислительная техника»

(код/наименование)

Профиль 09.04.01.20 «Проектирование интеллектуальных компьютерных систем»

(код/наименование)

Вид практики: учебная

Тип практики: ознакомительная

Место прохождения практики: Высшая школа интеллектуальных систем и суперкомпьютерных технологий (ВШИСиСТ) ИКНТ ФГАОУ ВО «СПбПУ». При необходимости руководителями практики обеспечивается организация практики на базе университета с использованием электронного обучения и дистанционных образовательных технологий.

Руководитель практики: Нестеров С.А., старший преподаватель ВШИСиСТ ИКНТ

(Ф.И.О., уч.степень, должность)

Рабочий график ознакомительной практики

Сроки практики: с **14.01.2022** г. по **25.01.2022** г.

№ п/п	Этапы (периоды) практики	Вид работ	Сроки прохождения этапа (периода) практики
1	Организационный этап	Установочная лекция для разъяснения целей, задач, содержания и порядка прохождения практики, выдача сопроводительных документов по практике Основная тема: Подготовка тестового перекрестка в 3D-симуляторе CARLA со стереокамерами.	14.01.2022 г.

2	Основной этап	1. Разобраться с API 3D-сиаулятора Carla для языка программирования Python. 2. С помощью языка программирования Python расставить стереокамеры на перекрёстке и вывести с них картинку. 3. Запустить трафик.	14.01.2022 - 24.01.2022 г.
3	Заключительный этап	Подготовка отчета	14.01.2022 - 24.01.2022 г.
		Защита отчета по практике (аттестация)	25.01.2022 г

Обучающийся: _____ /Д. А. Курякин/

Руководитель практики: _____ /С.А. Нестеров/

Содержание

Введение

Разработчики транспортных систем требуется учитывать очень большое количество данных факторов, алгоритмы поведения в тех или иных ситуациях в основном не прописываются «вручную» программистами, а формируются в процессе обучения. Для этого может использоваться реальное тестирование, при котором на заранее известном участке записываются данные с сенсоров, а потом воспроизводятся на разрабатываемой модели. Но при таком подходе не всегда можно записать все возможные ситуации. Поэтому совместно с реальным тестированием применяется тестирование на 3D симуляторе.

Исследователи под руководством Владлена Колтуна (Vladlen Koltun) из Центра Компьютерного Зрения в Барселоне создали симулятор с открытым кодом под названием CARLA. В нем имитируется городская среда со зданиями, пешеходами, автомобилями и другими объектами, а также меняющаяся погода. Для рендеринга в симуляторе используется бесплатный для некоммерческого использования движок Unreal Engine 4. Разработчики алгоритмов для беспилотных автомобилей могут подключать к симулятору свои алгоритмы через специальный API. На данный момент в симуляторе доступны несколько сенсоров: обычная камера, камера глубины и сегментирующая камера, классифицирующая объекты, а также лидар, радар и GPS. Также доступен API для подключения сторонних датчиков.

1. Задание

- Разобраться с API 3D-сиаулятора Carla для языка программирования Python.
- С помощью языка программирования Python расставить стереокамеры на перекрёстке и вывести с них картинку.
- Запустить трафик.

2. Установка и настройка

Для настройки тестовой площадки использовался Carla симулятор 0.9.13 для ОС Windows из репозитория [1]. Этот репозиторий содержит множество различных готовых версий Carla, а также документацию к ним. Также использовался язык программирования Python 3.8.

Для запуска сервер Carla симулятора требуется распаковать скачанный архив, перейди в директорию ввести в консоль:

```
CarlaUE4.exe
```

тогда запустится сервер и графическое окно с летающей камерой. Управление движением камеры осуществляется с помощью клавиш “W”, “A”, “S’, “D”, а управление камерой с помощью мыши и зажатой правой кнопкой. Для ускорения перемещения нужно прокрутить колесико мыши вверх, а для замедления в низ.

При запуске Carla можно вести аргументы. Например:

```
CarlaUE4.exe -carla-rpc-port=2000 -quality-level=Hight
```

аргумент carla-rpc-port устанавливает порт, через который доступен сервер, quality-level настраивает качество графики. Если количество видео памяти на компьютере меньше 6 гигабайт, то требуется ввести аргумент dx11 для запуска с DirectX 11.

В директории root/examples есть примеры программ на python. Например, если запустить:

python manual_control.py

то запустится сценарий, где можно управлять автомобилем. Также в этом сценарии есть примеры сенсоров, которые можно выбирать кнопками 1–9, обработка коллизий и отображаются сообщения о пересечении дорожной разметки. Если запустить программу:

python generate_traffic.py

то появится трафик.

3. Настройка тестовой площадки

Для настройки тестовой площадки были определены координаты перекрестка, с помощью программы на Python, приведенной в приложение 1. В программе определяются координаты камеры и угол поворота по трем осям.

В приложении 2 приведена программа, расставляющая стереокамеры. По координатам, которые могут определяться в программе из приведенной в приложении 1, определяется нужный перекресток. Зная перекресток можно определить координаты светофоров. В этом симуляторе под светофором подразумевается столб с нескольким трёхцветным табло светофора. На рис. 1 показан пример перекрестка с 4 светофорами. Также в симуляторе есть перекрестки с другим количеством светофоров.

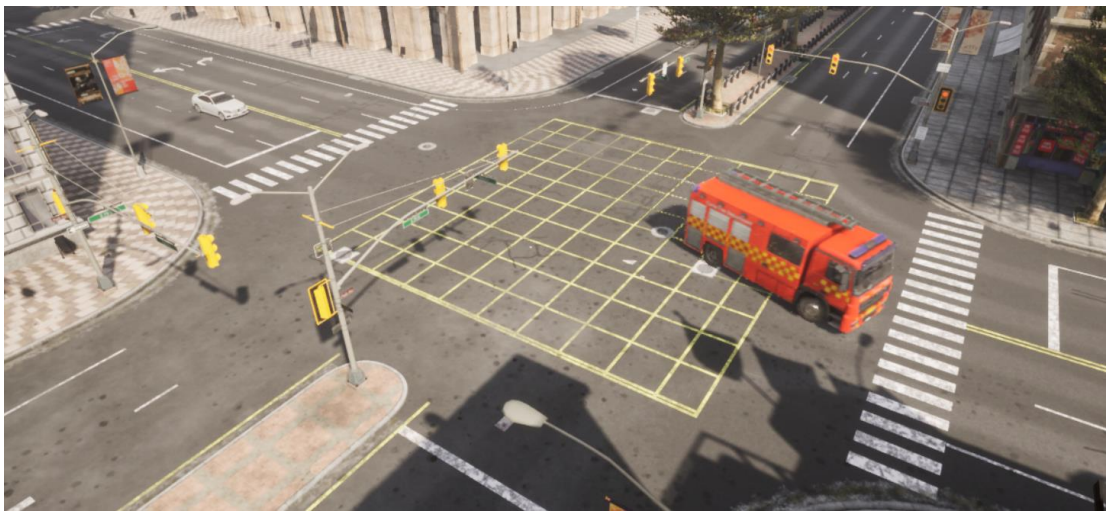

Рис 1. Пример перекрестка

В написанной программе на один светофор в общем ставится по 2 стереокамеры или 4 простых камеры так как одна стереокамера состоит из 2-х простых цветных камер. Одна стереокамера смотрит в сторону огней светофора, а вторая в противоположную сторону. На рис. 2 показан вывод картинок с камер которые расставлены на перекрестке из рис. 1.

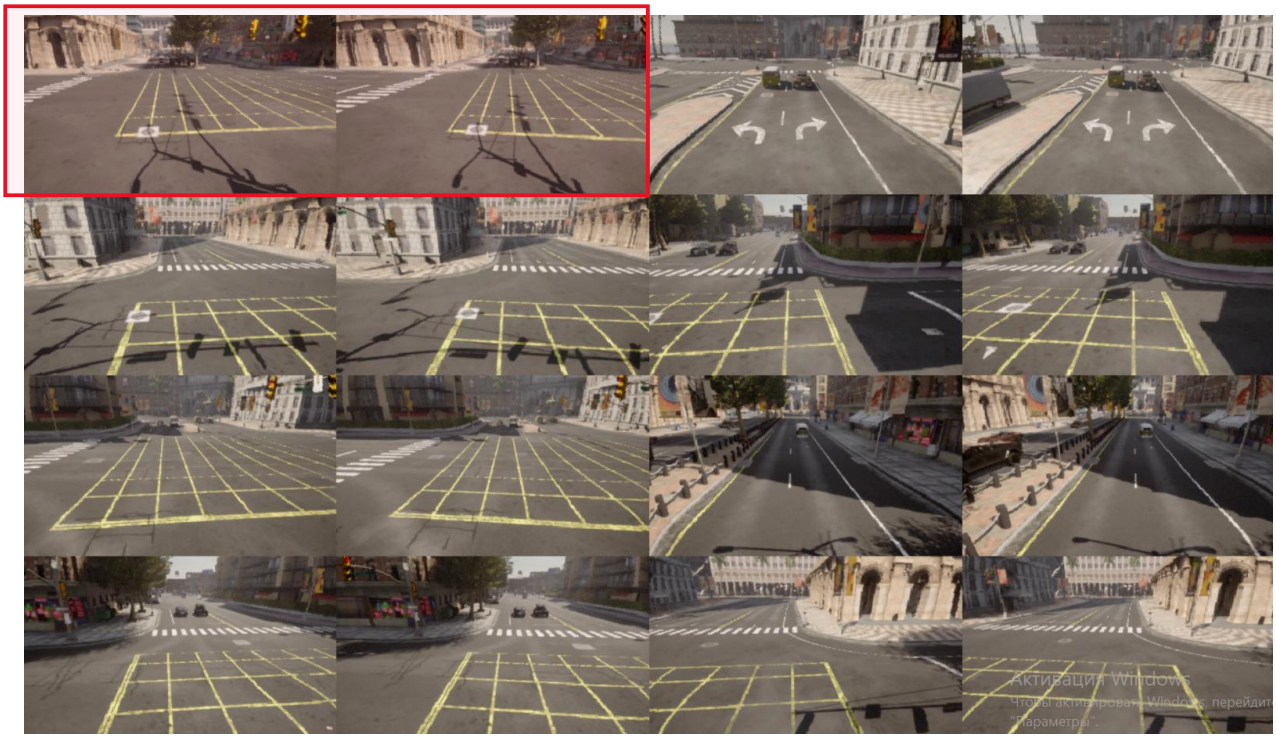

Рис. 2. Картинка с камер

Одна строка это картинки с камер, которые размещенных на одном столбе. Камеры по парно объединены. Красным квадратом выделены

картинки с одной стереокамеры. Одна стереокамера смотрит в сторону движения, а вторая против.

Для удобной настройки тестовой площадки был создан файл с настройками в json формате. В файле можно изменить координаты перекрестка, угол обзора камеры, параметры искажения камеры, угол наклона к дороге, смешение одной камеры относительно другой, угол между двумя камерами, смешение камер между собой.

4. Описание программ

Код созданных программ приведен в приложении 1–3 или в репозитории [3]. Всего были созданы 2 программы для определения координат перекрестка и для добавления камер на светофоры и был создан файл с настройками в формате json.

4.1. Файл add_camera_on the_junction.py

Рассмотрим файл add_camera_on the_junction.py, который представлен в листинге 2. Он состоит из 2-х классов DisplayManager, RGBCamera и запускающих функций main, run_simulation, read_json.

4.1.1. Класс RGBCamera

Класс RGBCamera отвечает за создание камеры, обновление картинки и удаление камеры. Содержит в себе положение по трем координатам, положение в окне приложения, за которое отвечает класс DisplayManager, настройки камеры.

4.1.2. Класс DisplayManager

Класс DisplayManager отвечает за отображение картинок с камер. Сдержит в себе окно приложения, список камер.

4.1.3. Запуск программы

В функции main считываются аргументы ip адрес и порт указанные при запуске программы. Затем осуществляется подключение клиента к серверу по ip адресу и порту, которые узаконены в аргументах или по умолчанию. Дальше

вызывается функция run_simulation. В этой функции считываются настройки и координаты перекрестка с json файла, файл приведен в листинге 3; по считанным координатам определяется перекресток и количество столбов со светофорами; создается окно для отрисовки картинок с камер, в зависимости от количество столбов настраивается сетка окна; устанавливаются и камеры на столбах и добавляются в окно для отрисовки изображения; в конце создается цикл для перерисовки изображения с камер и ожидания завершения программы.

4.2. Файл get_coordinate_spectator.py

Файл get_coordinate_spectator.py приведен в листинге 1. Он состоит из функции main в которой по указанному ip адресу и порту происходит подключение клиента к серверу. Затем в цикле считываются координаты летающей камеры и выводятся в консоль.

4.3. Файл sittings.json

Файл sittings.json состоит из настроек. Описание настроек файла приведено ниже:

- junction_coordinate – координаты перекрестка по x, y, z осям. Для каждой оси тип float
- width – ширина окна приложения. Тип int.
- height – высота окна приложения. Тип int.
- baseline – смещение одной камеры относительно другой. Тип float.
- angel – угол между камерами. Тип float.
- orientationю.tilt – наклон камеры к дороге. Тип float.
- intrinsic – смещение левой и правой камеры относительно нормального положения по осям yaw, pitch, roll. Тип float.
- fov – угол обзора камеры. Тип float.
- distortion – параметры искажения камеры подробнее в [2]. Для каждого параметра тип float.

Выводы

При выполнение практического задания был получен опыт работы с 3d симулятором Carla: запуск с различными настройками, работа с API для Python. Были созданы две программы: для определения координат перекрестка и для расставления стереокамер по светофорам и вывода с них картинку. Также был создан файл с настройками в формате json.

Во время выполнения задания было выяснено, что светофоры на перекрестке сгруппированы их координаты можно получить, зная координаты перекрестка. Также можно получить координаты любого объекта на карте. У Carla симулятора есть редактор карт, есть удобный доступ к сенсорам и объектам сенсоры можно устанавливать на движущиеся и статические объекты на карте.

Источники

1. Интернет-ресурс. Загрузчик Carla 0.9.13. https://github.com/carla-simulator/carla/blob/master/Docs/download.md

2. Интернет-ресурс. Документация Carla 0.9.13. https://carla.readthedocs.io/en/0.9.13/tuto_G_retrieve_data/

3. Интернет-ресурс. Репозиторий проекта. https://github.com/danila-kuryakin/carla_test_area

Приложение

Приложение 1. Определение координат летающей камеры.

Листинг 1. get_coordinate_spectator.py

```
import glob
import os
import sys
import time

try:
    sys.path.append(glob.glob('./PythonAPI/carla/dist/carla-*%d.%d-%s.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass

import carla

_HOST_ = '127.0.0.1'
_PORT_ = 2000
_SLEEP_TIME_ = 2


def main():
    client = carla.Client(_HOST_, _PORT_)
    client.set_timeout(2.0)
    world = client.get_world()

    while (True):
        t = world.get_spectator().get_transform()
        coordinate_str = "(x,y,z) = ({},{},{})".format(t.location.x, t.location.y, t.location.z)
        rotation_str = "(yaw, pitch, roll) = ({},{},{})".format(t.rotation.yaw, t.rotation.pitch,
t.rotation.roll)
        print('*'*20)
        print(coordinate_str)
        print(rotation_str)
        time.sleep(_SLEEP_TIME_)


if __name__ == '__main__':
    main()
```

Приложение 2. Код для размещения камер.

Листинг 2. add_camera_on the_junction.py

```
import glob
import json5 as json
import os
```

```
import sys
import argparse
import time
import numpy as np
import carla
import cv2 as cv
from matplotlib import pyplot as plt
from carla import Location, Rotation, Transform

try:
    sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass

try:
    import pygame
    from pygame.locals import K_ESCAPE
    from pygame.locals import K_q
except ImportError:
    raise RuntimeError('cannot import pygame, make sure pygame package is installed')


class DisplayManager:
    def __init__(self, grid_size, window_size):
        pygame.init()
        pygame.font.init()
        self.display = pygame.display.set_mode(window_size, pygame.HWSURFACE |
pygame.DOUBLEBUF)

        self.grid_size = grid_size
        self.window_size = window_size
        self.sensor_list = []

    def get_window_size(self):
        return [int(self.window_size[0]), int(self.window_size[1])]

    def get_display_size(self):
        return [int(self.window_size[0]/self.grid_size[1]),
int(self.window_size[1]/self.grid_size[0])]

    def get_display_offset(self, gridPos):
        dis_size = self.get_display_size()
        return [int(gridPos[1] * dis_size[0]), int(gridPos[0] * dis_size[1])]

    def add_sensor(self, sensor):
        self.sensor_list.append(sensor)

    def get_sensor_list(self):
        return self.sensor_list
```

```
    def render(self):
        if not self.render_enabled():
            return

        for s in self.sensor_list:
            s.render()

        pygame.display.flip()

    def destroy(self):
        for s in self.sensor_list:
            s.destroy()

    def render_enabled(self):
        return self.display != None


class RGBCamera:
    def __init__(self, world, display_man, transform, options, display_pos):
        self.surface = None
        self.world = world
        self.display_man = display_man
        self.display_pos = display_pos
        self.sensor = self.init_sensor(transform, options)
        self.sensor_options = options

        self.display_man.add_sensor(self)

    def init_sensor(self, transform, options):
        camera_bp = self.world.get_blueprint_library().find('sensor.camera.rgb')
        disp_size = self.display_man.get_display_size()
        camera_bp.set_attribute('image_size_x', str(disp_size[0]))
        camera_bp.set_attribute('image_size_y', str(disp_size[1]))

        camera_bp.set_attribute('fov', str(options['fov']))

        for key in options['distortion'].keys():
            camera_bp.set_attribute(key, str(options['distortion'][key]))

        camera = self.world.spawn_actor(camera_bp, transform, attach_to=None)
        camera.listen(self.save_rgb_image)
        self.camera = camera

        return camera

    def get_sensor(self):
        return self.sensor

    def get_image(self):
        self.camera.listen(lambda image: cv.imshow('Frame', image))
```

```
    def save_rgb_image(self, image):
        image.convert(carla.ColorConverter.Raw)
        array = np.frombuffer(image.raw_data, dtype=np.dtype("uint8"))
        array = np.reshape(array, (image.height, image.width, 4))
        array = array[:, :, :3]
        array = array[:, :, ::-1]

        if self.display_man.render_enabled():
            self.surface = pygame.surfarray.make_surface(array.swapaxes(0, 1))


    def render(self):
        if self.surface is not None:
            offset = self.display_man.get_display_offset(self.display_pos)
            self.display_man.display.blit(self.surface, offset)

    def destroy(self):
        self.sensor.destroy()


def run_simulation(client):
    json_data = read_json("settings.json")

    display_manager = None

    try:
        # Getting the world and
        world = client.get_world()

        map = world.get_map()

        wpJunction = map.get_waypoint(Location(json_data['junction_coordinate']['x'],
                                               json_data['junction_coordinate']['y'],
                                               json_data['junction_coordinate']['z']))

        junction = wpJunction.get_junction()
        if junction:
            traffic_lights = world.get_traffic_lights_in_junction(junction.id)
            traffic_lights_len = len(traffic_lights)
            print('Number of traffic lights: ', traffic_lights_len)
        else:
            print('The junction not found')
            exit(0)

        display_manager = DisplayManager(grid_size=[traffic_lights_len, 4],
window_size=[json_data['width'], json_data['height']])
        i = 0
        for tl in traffic_lights:
            if i >= traffic_lights_len:
                break
            print(tl.get_transform())
```

```
        transform = tl.get_transform()
        if -1.0 < transform.rotation.yaw < 1.0:
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x - 7, y=transform.location.y,
z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw + 90 + json_data['angel'] +
json_data['intrinsic']['left']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['left']['pitch'],
                                roll=json_data['intrinsic']['left']['roll'])), json_data, display_pos=[i, 0])
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x - 7 - json_data['baseline'],
y=transform.location.y, z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw + 90 - json_data['angel'] +
json_data['intrinsic']['right']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['right']['pitch'],
                                roll=json_data['intrinsic']['right']['roll'])), json_data, display_pos=[i, 1])
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x - 7 - json_data['baseline'],
y=transform.location.y, z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw - 90 + json_data['angel'] +
json_data['intrinsic']['left']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['left']['pitch'],
                                roll=json_data['intrinsic']['left']['roll'])), json_data, display_pos=[i, 2])
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x - 7 , y=transform.location.y,
z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw - 90 - json_data['angel'] +
json_data['intrinsic']['right']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['right']['pitch'],
                                roll=json_data['intrinsic']['right']['roll'])), json_data, display_pos=[i, 3])

        elif 89.0 < transform.rotation.yaw < 91.0:
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x, y=transform.location.y - 7,
z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw + 90 + json_data['angel'] +
json_data['intrinsic']['left']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['left']['pitch'],
                                roll=json_data['intrinsic']['left']['roll'])), json_data, display_pos=[i, 0])
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x, y=transform.location.y - 7 -
json_data['baseline'], z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw + 90 - json_data['angel'] +
json_data['intrinsic']['right']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['right']['pitch'],
                                roll=json_data['intrinsic']['right']['roll'])), json_data, display_pos=[i, 1])
```

```
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x, y=transform.location.y - 7-
json_data['baseline'], z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw - 90 + json_data['angel'] +
json_data['intrinsic']['left']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['left']['pitch'],
                                roll=json_data['intrinsic']['left']['roll'])), json_data, display_pos=[i, 2])
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x, y=transform.location.y - 7 ,
z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw - 90 - json_data['angel'] +
json_data['intrinsic']['right']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['right']['pitch'],
                                roll=json_data['intrinsic']['right']['roll'])), json_data, display_pos=[i, 3])

        elif 179.0 < transform.rotation.yaw < 181.0:
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x + 7, y=transform.location.y,
z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw + 90 + json_data['angel'] +
json_data['intrinsic']['left']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['left']['pitch'],
                                roll=json_data['intrinsic']['left']['roll'])), json_data, display_pos=[i, 0])
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x + 7  + json_data['baseline'],
y=transform.location.y, z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw + 90 - json_data['angel'] +
json_data['intrinsic']['right']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['right']['pitch'],
                                roll=json_data['intrinsic']['right']['roll'])), json_data, display_pos=[i, 1])
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x + 7+ json_data['baseline'],
y=transform.location.y, z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw - 90 + json_data['angel'] +
json_data['intrinsic']['left']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['left']['pitch'],
                                roll=json_data['intrinsic']['left']['roll'])), json_data, display_pos=[i, 2])
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x + 7 , y=transform.location.y,
z=transform.location.z + 4),
                       Rotation(yaw=transform.rotation.yaw - 90 - json_data['angel'] +
json_data['intrinsic']['right']['yaw'],
                                pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['right']['pitch'],
                                roll=json_data['intrinsic']['right']['roll'])), json_data, display_pos=[i, 3])

        else:
```

```
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x, y=transform.location.y + 7,
z=transform.location.z + 4),
                                Rotation(yaw=transform.rotation.yaw + 90 + json_data['angel'] +
json_data['intrinsic']['left']['yaw'],
                                         pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['left']['pitch'],
                                         roll=json_data['intrinsic']['left']['roll'])), json_data, display_pos=[i, 0])
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x, y=transform.location.y + 7 +
json_data['baseline'], z=transform.location.z + 4),
                                Rotation(yaw=transform.rotation.yaw + 90 - json_data['angel'] +
json_data['intrinsic']['right']['yaw'],
                                         pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['right']['pitch'],
                                         roll=json_data['intrinsic']['right']['roll'])), json_data, display_pos=[i, 1])
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x, y=transform.location.y + 7+
json_data['baseline'], z=transform.location.z + 4),
                                Rotation(yaw=transform.rotation.yaw - 90 + json_data['angel'] +
json_data['intrinsic']['left']['yaw'],
                                         pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['left']['pitch'],
                                         roll=json_data['intrinsic']['left']['roll'])), json_data, display_pos=[i, 2])
            RGBCamera(world, display_manager,
                      Transform(Location(x=transform.location.x, y=transform.location.y + 7 ,
z=transform.location.z + 4),
                                Rotation(yaw=transform.rotation.yaw - 90 - json_data['angel'] +
json_data['intrinsic']['right']['yaw'],
                                         pitch=json_data['orientation']["tilt"] +
json_data['intrinsic']['right']['pitch'],
                                         roll=json_data['intrinsic']['right']['roll'])), json_data, display_pos=[i, 3])
            i = i + 1

        #Simulation loop
        call_exit = False
        while True:
            world.tick()

            # Render received data
            display_manager.render()

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    call_exit = True
                elif event.type == pygame.KEYDOWN:
                    if event.key == K_ESCAPE or event.key == K_q:
                        call_exit = True
                        break

            if call_exit:
                break
```

```
    finally:
        if display_manager:
            display_manager.destroy()



def read_json(path):
    with open(path, "r") as read_file:
        json_data = ''.join(line for line in read_file if not line.startswith('//'))
        return json.loads(json_data)


def main():
    argparser = argparse.ArgumentParser(
        description='CARLA Sensor')
    argparser.add_argument(
        '--host',
        metavar='H',
        default='127.0.0.1',
        help='IP of the host server (default: 127.0.0.1)')
    argparser.add_argument(
        '-p', '--port',
        metavar='P',
        default=2000,
        type=int,
        help='TCP port to listen to (default: 2000)')

    args = argparser.parse_args()

    try:
        client = carla.Client(args.host, args.port)
        client.set_timeout(5.0)

        run_simulation(client)

    except KeyboardInterrupt:
        print('\nCancelled by user. Bye!')


if __name__ == '__main__':

    main()
```

Приложение 3. Файл с настройками

Листинг 3. settings.json

```
{
 "junction_coordinate" : {"x" : -46.656982421875, "y" : 21.270511627197266, "z" : 0},
 "width" : 1280,
 "height" : 740,
 "baseline": 0.5,
```

```
  "angel" : 5,
  "orientation" :
  {
    "tilt" : -20
  },
  "intrinsic" : {
    "left" : {
      "yaw" : 0,
      "pitch" : 0,
      "roll" : 0
    },
    "right" : {
      "yaw" : 0,
      "pitch" : 0,
      "roll" : 0
    }
  },
  "fov" : 120,
  "distortion" :
  {
    "lens_circle_falloff": 2.0,
    "lens_circle_multiplier": 2.0,
    "lens_k" : -3.0,
    "lens_kcube" : -1.0,
    "lens_x_size" : 0.2,
    "lens_y_size" : 0.08
  }
}
```