МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное

учреждение высшего образования Российской Федерации

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Инженерно-технологическая академия

Институт радиотехнических систем и управления

Кафедра радиотехнических и телекоммуникационных систем

Отчет

по итоговому проекту по курсу «Прикладное программирование на Python в РЭС»

Выполнила ст. гр. РТсо3-12

Есаева О. Г.

Таганрог

2025

1. Введение

Цель проекта — демонстрация навыков работы с робототехническими фреймворками, такими как ROS2, инструментами контейнеризации (Docker), системой контроля версий (Git) и средствами визуализации данных с сенсоров (RViz2 и image_viewer). В рамках проекта реализуется универсальный скрипт на Python, автоматически обрабатывающий входные параметры, определяющие сценарий работы: тип сенсора, способ обработки и структуру git-репозитория.

Проект создается на виртуальной машине VirtualBox, где установлена ОС Ubuntu 22.04, являющаяся рекомендованной для работы с ROS2. Контейнер ROS2 запускается внутри этой ОС, и взаимодействие с внешней системой визуализации происходит через X11 forwarding.

2. Генерация варианта задания

Генератор вариантов реализован на Python с использованием встроенного модуля `random`. Он создает три случайных целых числа, которые определяют:

- Тип сенсора, с которым будет проводиться работа.
- Форму вывода данных: либо частота сообщений, либо содержимое.
- Архитектуру git-репозитория проекта.

Этот подход обеспечивает уникальность задания для каждого запуска и эмулирует систему экзаменационного билета.

Пример работы генератора:

```
first = random.randint(0, 3)
second = random.randint(0, 1)
third = random.randint(0, 2)
print(f"Ваш вариант: {first} {second} {third}")
```

---

3. Работа с сенсорными данными

ROS2 позволяет абстрагироваться от физического оборудования за счет работы с записанными bag-файлами (`rosbag2`). Эти файлы содержат записи сообщений из разных топиков и могут быть воспроизведены в реальном времени.

Работа с датасетами:

Скачанные архивы датасетов содержат записи с различных сенсоров:

- Один архив содержит `Image`-сообщения от камеры и радара.
- Второй — `PointCloud2`-сообщения от лидара и радара.

Для определения содержания:

ros2 bag info rosbag2_2025_04_04-20_51_13

В зависимости от результата выбирается нужный топик для подписки и визуализации.

4. Обработка сообщений ROS2

ROS2 предоставляет расширенную модель подписки и публикации. В данном проекте реализована подписка на один из четырех топиков. Далее, в зависимости от параметра варианта:

- Если указана частота (`0`), используется внутренний таймер ROS2 (`create_timer`) для вывода количества полученных сообщений за интервал (1 секунда).
- Если указано содержимое (`1`), в колбэке выводится информация о типе и структуре полученного сообщения.

Реализация подписки:

```
self.subscription = self.create_subscription(
   self.msg_type,
   self.topic,
   self.callback,
   10)
```

5. Визуализация данных

Визуализация выполняется средствами ROS2:

- Для изображений используется `image_tools:image_viewer`, который отображает поток изображений в отдельном окне.
- Для облаков точек — `rviz2`, визуализатор с 3D-интерфейсом.

В зависимости от типа сенсора, визуализатор запускается в отдельном потоке, чтобы не блокировать основной цикл подписки и обработки.

Особенности:

- `rviz2` требует ручной настройки отображения (`Add -> PointCloud2`, установка правильного топика).
- `image_viewer` автоматически привязывается к нужному топику через `--ros-args -t`.

6. Docker-контейнер

Контейнеризация необходима для обеспечения воспроизводимости и независимости проекта от системы хоста. Скрипт автоматически проверяет, запущен ли он в контейнере (через `/` точку входа) и в случае необходимости:

- Генерирует Dockerfile.
- Перемещает текущий скрипт внутрь контейнера.
- Выполняет сборку через `docker build`.
- Запускает контейнер с настройками сети и доступом к X11 для визуализации.

Особенности Docker-конфигурации:

- Используется официальный образ `osrf/ros:humble-desktop`.
- Устанавливаются дополнительные зависимости: `image-tools`, `rviz2`, `git`, `pip`.
- Указан `CMD` для запуска основного скрипта внутри контейнера.

7. Git-репозиторий

Репозиторий создается и наполняется автоматически, в соответствии с заданным вариантом. Используются команды:

```
git init
git checkout -b branch_name
git add file.txt
git commit -m "..."
```

Типы репозиториев:

- Вариант `0`: последовательные коммиты в одной ветке `main`.

- Вариант `1`: две ветки (`main` и `feature`) с двумя коммитами каждая и последующим merge.
- Вариант `2`: три ветки (`dev`, `test`, `feature`) с одним коммитом, которые затем сливаются в `main`.

История проекта визуализируется с помощью:

git log --oneline --graph --all

В результате можно продемонстрировать навыки ведения ветвления и управления слияниями.

8. Используемые технологии

Проект демонстрирует интеграцию и владение следующими инструментами и технологиями:

- **Python 3.10** — основной язык программирования.
- **ROS2 Humble** — робототехнический фреймворк нового поколения.
- **Docker** — контейнеризация для обеспечения переносимости.
- **Git** — система контроля версий и управления ветками.
- **RViz2** — 3D-визуализатор ROS2-сообщений.
- **image_tools** — пакет для отображения изображений.
- **Ubuntu 22.04** — стабильная версия ОС с поддержкой ROS2.
- **VirtualBox** — гипервизор для создания виртуальной машины.

9. Заключение

Проект реализует автоматизированную систему, которая в зависимости от случайного набора параметров:

- Выбирает сенсор.
- Подписывается на соответствующий топик и обрабатывает сообщения.
- Визуализирует поток данных.

- Запускается внутри Docker-контейнера.
- Создает git-репозиторий с заданной структурой.

Такая система может служить основой для обучения и тестирования студентов, изучающих ROS2 и системы автономных роботов. Благодаря контейнеризации, проект полностью переносим и может быть легко воспроизведен в любой среде.

Возможные направления расширения:

- Добавление параметров командной строки.
- Расширение типов сенсоров.
- Интеграция с реальными устройствами.
- Автоматическая настройка RViz2

```
wizeo@wizard:~$ sudo docker images
[sudo] password for wizeo:
REPOSITORY           TAG       IMAGE ID       CREATED          SIZE
ros2-humble-custom   latest    ae524ed97a34   45 minutes ago   1.82GB
ubuntu               22.04     c42dedf797ba   12 days ago      77.9MB
hello-world          latest    74cc54e27dc4   3 months ago     10.1kB
wizeo@wizard:~$
```

ПРИЛОЖЕНИЕ. Код

```python
import random

def generate_variant():
    sensor_type = random.randint(0, 3)
    data_mode = random.randint(0, 1)
    git_mode = random.randint(0, 2)
    print(f"Вариант: {sensor_type} {data_mode} {git_mode}")
    return sensor_type, data_mode, git_mode

if __name__ == "__main__":
    generate_variant()



import rclpy
from rclpy.node import Node
import random
import subprocess
```

```
import time
import os
import sys
import threading
from sensor_msgs.msg import Image, PointCloud2

class SensorProcessor(Node):
    def __init__(self, sensor_type, output_type):
        super().__init__('sensor_processor')
        self.sensor_type = sensor_type
        self.output_type = output_type
        self.msg_count = 0

        sensor_types = {
            0: ("Radar Image", "/radar/image", Image),
            1: ("Camera Image", "/camera/image", Image),
            2: ("Lidar PointCloud", "/lidar/points", PointCloud2),
            3: ("Radar PointCloud", "/radar/points", PointCloud2)
        }

        self.sensor_name, self.topic, self.msg_type =
sensor_types[sensor_type]
        self.get_logger().info(f"Processing: {self.sensor_name}")

        self.subscription = self.create_subscription(
            self.msg_type,
            self.topic,
            self.callback,
            10)

        self.timer = self.create_timer(1.0, self.timer_callback)

        self.visualization_thread =
threading.Thread(target=self.launch_visualization)
        self.visualization_thread.daemon = True
        self.visualization_thread.start()

    def callback(self, msg):
        self.msg_count += 1
        if self.output_type == 1:
            if self.sensor_type in [0, 1]:
                self.get_logger().info(f"Image:
{msg.height}x{msg.width}, Encoding: {msg.encoding}")
            else:
                self.get_logger().info(f"PointCloud: Width={msg.width},
Height={msg.height}")
```

```
    def timer_callback(self):
        if self.output_type == 0:
            self.get_logger().info(f"Frequency: {self.msg_count} Hz")
        self.msg_count = 0

    def launch_visualization(self):
        time.sleep(2)
        if self.sensor_type in [0, 1]:
            subprocess.run(['ros2', 'run', 'image_tools',
'image_viewer', '--ros-args', '-t', self.topic])
        else:
            subprocess.run(['rviz2'])

def generate_variant():
    return random.randint(0, 3), random.randint(0, 1), random.randint(0,
2)

def play_dataset(sensor_type):
    dataset = "rosbag2_2025_04_04-20_51_13" if sensor_type in [0, 1]
else "output_bag"
    if not os.path.exists(dataset):
        print("Датасет не найден. Загрузите датасет")
        return None
    return subprocess.Popen(["ros2", "bag", "play", dataset])

def setup_repository(repo_type):
    os.system("rm -rf final_repo")
    os.makedirs("final_repo")
    os.chdir("final_repo")
    os.system("git init")

    if repo_type == 0:
        for i in range(5):
            with open(f"file{i}.txt", "w") as f:
                f.write(f"File {i}")
            os.system(f"git add file{i}.txt && git commit -m 'Commit
{i}'")

    elif repo_type == 1:
        with open("main.txt", "w") as f:
            f.write("Main 1")
        os.system("git add . && git commit -m 'Main commit 1'")
        os.system("git checkout -b feature")
        with open("feature.txt", "w") as f:
            f.write("Feature")
```

```
        os.system("git add . && git commit -m 'Feature commit'")
        os.system("git checkout main")
        with open("main2.txt", "w") as f:
            f.write("Main 2")
        os.system("git add . && git commit -m 'Main commit 2'")
        os.system("git merge feature -m 'Merge feature'")

    elif repo_type == 2:
        for branch in ["dev", "test", "feature"]:
            os.system(f"git checkout -b {branch}")
            with open(f"{branch}.txt", "w") as f:
                f.write(f"{branch} branch")
            os.system("git add . && git commit -m 'Commit on {branch}'")
            os.system("git checkout main")
        for branch in ["dev", "test", "feature"]:
            os.system(f"git merge {branch} -m 'Merge {branch}'")

    os.chdir("..")

def main():
    sensor_type, output_type, repo_type = generate_variant()
    print(f"Variant: {sensor_type} {output_type} {repo_type}")

    setup_repository(repo_type)
    play_proc = play_dataset(sensor_type)
    if play_proc is None:
        return

    rclpy.init()
    node = SensorProcessor(sensor_type, output_type)
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        print("В процессе...")
    finally:
        if play_proc:
            play_proc.terminate()
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()

# Dockerfile
FROM osrf/ros:humble-desktop
```

```
RUN apt-get update && apt-get install -y \
    python3-pip \
    git \
    ros-humble-image-tools \
    ros-humble-rviz2 \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /app
COPY . /app
RUN pip3 install -r requirements.txt || true

ENTRYPOINT ["python3", "sensor_processor.py"]



git init
git checkout -b sensor_node
git commit -am "sensor node"

git checkout main
git checkout -b docker_setup
git commit -am "Docker setup"

git checkout main
git checkout -b bag_replay
git commit -am "Bag replay added"

git checkout main
git merge sensor_node
git merge docker_setup
git merge bag_replay
```