Nama : Gusti Gratia Delpiera

NRP : 5026231097

Laporan Penugasan PX4 Simulation (Tugas 1)

1. Langkah pertama setelah membuat package dan menambahkan dependencies serta executable yang diperlukan pada file Cmakelist.txt dan package.xml adalah membuat file CPP di dalam folder src. Menambahkan header yang diperlukan, untuk membuat pola yang ditentukan disini saya menggunakan rclcpp, offboard_control_mode, trajectory_setpoint, vehicle_command, untuk include dari px4_msgs dan beberapa header dari standard library seperti iostream dan chrono

```
#include "rclcpp/rclcpp.hpp"
#include "px4_msgs/msg/offboard_control_mode.hpp"
#include "px4_msgs/msg/trajectory_setpoint.hpp"
#include <px4_msgs/msg/vehicle_command.hpp>
#include <px4_msgs/msg/vehicle_control_mode.hpp>
#include <chrono>
#include <iostream>
```

2. Langkah kedua membuat class Node, yang saya beri nama circle_shape, untuk mempublish offboard control drone.

```
class OffboardControl : public rclcpp::Node {
  public:
    OffboardControl() : Node("circle_shape") {
      trajectory_setpoint_publisher_ = this-
>create_publisher<px4_msgs::msg::TrajectorySetpoint>("fmu/in/trajectory_setpoi
nt", 10);
      offboard_control_mode_publisher_ = this-
>create_publisher<px4_msgs::msg::OffboardControlMode>("fmu/in/offboard_control
_mode", 10);
      vehicle_command_publisher_ = this-
>create_publisher<VehicleCommand>("/fmu/in/vehicle_command", 10);
      offboard_setpoint_counter_ = 0
      yawcount = 1;
      auto timer_callback = [this]() -> void {
        if (offboard_setpoint_counter_ == 100) {
          this-
>publish_vehicle_command(VehicleCommand::VEHICLE_CMD_DO_SET_MODE, 1, 6);
          this->arm();
        }
        if (offboard_setpoint_counter_ < 190){
          publish_offboard_control_mode();
          takeoff();
        }
```

```
        else if(offboard_setpoint_counter_ >= 190 &&
offboard_setpoint_counter_ < 440){
          publish_offboard_control_mode();
          publish_trajectory_setpoint(0.1-yawcount*0.0255);
          yawcount++;
        }
        else if (offboard_setpoint_counter_ == 500){
          this->land();
        }
        if (offboard_setpoint_counter_ == 650) {
          this->disarm();
        }
        if (offboard_setpoint_counter_ < 651){
          offboard_setpoint_counter_++;
        }

      };
      timer_ = this->create_wall_timer(100ms, timer_callback);
    }

    void arm();
    void land();
    void takeoff();
    void disarm();

  private:
    rclcpp::Publisher<px4_msgs::msg::TrajectorySetpoint>::SharedPtr
trajectory_setpoint_publisher_;
    rclcpp::Publisher<px4_msgs::msg::OffboardControlMode>::SharedPtr
offboard_control_mode_publisher_;
    rclcpp::Publisher<VehicleCommand>::SharedPtr
vehicle_command_publisher_;//t
    rclcpp::TimerBase::SharedPtr timer_;
    std::atomic<uint64_t> timestamp_;
    uint64_t offboard_setpoint_counter_;
    uint64_t yawcount;
    void publish_offboard_control_mode();
    void publish_trajectory_setpoint(float yaw);
    void publish_vehicle_command(uint16_t command, float param1 = 0.0, float
param2 = 0.0);

};
```

Disini, saya membuat prototype dari empat buah method untuk public, yaitu untuk takeoff, land, arm, dan disarm. Kemudian tiga buah method private untuk publish offboard control mode, trajectory setpoint, dan vehicle command (arm/disarm/land).

Kemudian saya menginisialisasi untuk publisher dengan create_publisher pada Node, dan juga offboard setpoint counter, sebagai counter untuk offboard mode, dan yawcount untuk mengatur sudut yaw pada drone ketika offboard.

```
trajectory_setpoint_publisher_ = this-
>create_publisher<px4_msgs::msg::TrajectorySetpoint>("fmu/in/trajectory_setpoi
nt", 10);
offboard_control_mode_publisher_ = this-
>create_publisher<px4_msgs::msg::OffboardControlMode>("fmu/in/offboard_control
_mode", 10);
vehicle_command_publisher_ = this-
>create_publisher<VehicleCommand>("/fmu/in/vehicle_command", 10);
```

Pada bagian private saya juga mendeklarasikan publisher dalam framework ROS2,

```
    rclcpp::Publisher<px4_msgs::msg::TrajectorySetpoint>::SharedPtr
trajectory_setpoint_publisher_;
    rclcpp::Publisher<px4_msgs::msg::OffboardControlMode>::SharedPtr
offboard_control_mode_publisher_;
    rclcpp::Publisher<VehicleCommand>::SharedPtr
vehicle_command_publisher_;//t
    rclcpp::TimerBase::SharedPtr timer_;
```

3. Sebelum masuk ke dalam lambda function, saya terlebih dahulu untuk membuat method arm, disarm, land, takeoff, publish vehicle command, trajectory setpoint, dan offboard control mode.

```
void OffboardControl::arm()
{
  publish_vehicle_command(VehicleCommand::VEHICLE_CMD_COMPONENT_ARM_DISARM,
1.0);
  RCLCPP_INFO(this->get_logger(), "Arm command send");
}

void OffboardControl::land(){
  publish_vehicle_command(VehicleCommand::VEHICLE_CMD_NAV_VTOL_LAND, 0.0);
  RCLCPP_INFO(this->get_logger(), "Land command send");
}

void OffboardControl::disarm()
{
  publish_vehicle_command(VehicleCommand::VEHICLE_CMD_COMPONENT_ARM_DISARM,
0.0);
  RCLCPP_INFO(this->get_logger(), "Disarm command send");
}
```

Untuk tiga method disini (arm, disarm, land), saya menggunakan vehicle command yang tersedia pada px4_msgs, yang menggunakan mavlink messages sebagai protocol komunikasi.

```
void OffboardControl::publish_offboard_control_mode()
{
  OffboardControlMode msg{};
  msg.position = true;
  msg.velocity = false;
  msg.acceleration = false;
  msg.attitude = false;
  msg.body_rate = false;
  msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
  offboard_control_mode_publisher_->publish(msg);
}
```

Dan saya membuat publisher untuk offboard control, yang bernilai true untuk msg adalah msg.position. Karena kita akan mengubah posisi dari UAV.

```
void OffboardControl::publish_trajectory_setpoint(float yaw)
{
  TrajectorySetpoint msg{};
  float radius = 3.0;
  float omega = 0.2514;
  double time_now = static_cast<double>(this->get_clock()-
>now().nanoseconds()) / 1e9;
  float x = radius * cos(omega*time_now)-3;
  float y = radius * sin(omega*time_now);
  msg.position = {-x, y,-4.0};
  msg.yaw = yaw;
  msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
  trajectory_setpoint_publisher_->publish(msg);
}
```

Selanjutnya untuk trajectory setpoint, karena lintasan berbentuk lingkaran, maka saya membuat beberapa variable sebagai berikut :

a. float radius = 3.0;
   Mendefinisikan variabel radius yang menyimpan nilai 3.0, yang merupakan jari-jari lingkaran.
b. float omega = 0.2514;
   Mendefinisikan variabel omega yang menyimpan nilai 0.2514, yang merupakan kecepatan sudut (dalam radian per detik).
c. double time_now = static_cast<double>(this->get_clock()->now().nanoseconds()) / 1e9;
   menghitung waktu saat ini dalam detik.
d. this->get_clock()->now()
   digunakan untuk mendapatkan waktu saat ini dari node ROS 2 yang sedang berjalan.
e. nanoseconds() digunakan untuk mendapatkan waktu dalam satuan nanodetik.
f. static_cast<double>
   mengonversi nilai waktu (yang biasanya dalam tipe std::chrono::nanoseconds) menjadi tipe data double.
g. / 1e9

digunakan untuk mengonversi waktu dari nanodetik menjadi detik.

h. float x = radius * cos(omega*time_now)-3;
   baris ini menghitung koordinat x posisi objek pada lintasan melingkar.
i. cos(omega*time_now)
   Digunakan untuk menghitung nilai cosinus dari sudut yang berubah tergantung waktu (dalam radian).
j. Radius * cos(omega*time_now)
   mengalikan hasil cosinus dengan jari-jari lingkaran untuk mendapatkan koordinat x pada lingkaran.
k. (-3) merupakan penyesuaian posisi x, supaya titik Tengah lingkaran berada di koordinat (3,0).
l. float y = radius * sin(omega*time_now);
   Menghitung koordinat y posisi objek pada lintasan melingkar.
m. sin(omega*time_now) digunakan untuk menghitung nilai sinus dari sudut yang berubah tergantung waktu (dalam radian).
n. radius * sin(omega*time_now) mengalikan hasil sinus dengan jari-jari lingkaran untuk mendapatkan koordinat y pada lingkaran.
o. msg.position = {-x,y,-3,0}
   -x untuk drone berputar berlawanan jarum jam.

Terakhir yaw saya ambil dari parameter float yaw.

```
void OffboardControl::takeoff()
{
  TrajectorySetpoint msg{};
  msg.position = {0.0,0.0,-4.0};
  msg.yaw = 0.0;
  msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
  trajectory_setpoint_publisher_->publish(msg);
}
```

Untuk takeoff, menggunakan msg trajectory setpoint, namun yang ditentukan hanyalah altitude saja (z) yang bernilai -4, artinya 4 meter dengan frame NED.

```
void OffboardControl::publish_vehicle_command(uint16_t command, float param1,
float param2)
{
  VehicleCommand msg{};
  msg.param1 = param1;
  msg.param2 = param2;
  msg.command = command;
  msg.target_system = 1;
  msg.target_component = 1;
  msg.source_system = 1;
  msg.source_component = 1;
  msg.from_external = true;
  msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
  vehicle_command_publisher_->publish(msg);
}
```

Selanjutnya publish vehicle command untuk mempublish vehicle cmd pada method2 sebelumnya seperti VEHICLE_CMD_COMPONENT_ARM_DISARM, dan VEHICLE_CMD_NAV_VTOL_LAND.

4. Pada main program, digunakan untuk spin Node Offboard Control.

```
int main(int argc, char *argv[]) {
  std::cout << "Starting offboard control node..." << std::endl;
  setvbuf(stdout, NULL, _IONBF, BUFSIZ);
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<OffboardControl>());

  rclcpp::shutdown();
  return 0;
}
```

5. Pada lambda function di dalam class Node :

```
      offboard_setpoint_counter_ = 0
      yawcount = 1;
      auto timer_callback = [this]() -> void {
        if (offboard_setpoint_counter_ == 100) {
          this-
>publish_vehicle_command(VehicleCommand::VEHICLE_CMD_DO_SET_MODE, 1, 6);
          this->arm();
        }
        if (offboard_setpoint_counter_ < 190){
          publish_offboard_control_mode();
          takeoff();
        }
        else if(offboard_setpoint_counter_ >= 190 &&
offboard_setpoint_counter_ < 440){
          publish_offboard_control_mode();
          publish_trajectory_setpoint(0.1-yawcount*0.0255);
          yawcount++;
        }
        else if (offboard_setpoint_counter_ == 500){
          this->land();
        }
        if (offboard_setpoint_counter_ == 650) {
          this->disarm();
        }
        if (offboard_setpoint_counter_ < 651){
          offboard_setpoint_counter_++;
        }

      };
      timer_ = this->create_wall_timer(100ms, timer_callback);
```

jika offboard setpoint counter bernilai 100 ms (10 detik), maka ganti mode menjadi offboard, dan publish arm command.
Setelah itu publish offboard control mode, dan takeoff. Setelah takeoff, publish trajectory setpoint, di mana tadi sudah ditentukan untuk bergerak secara sirkular, dan untuk parameter yaw, saya menggunakan yawcount yang akan berkurang setiap miliseconds dan membuat drone bergerak seolah mengorbit titik Tengah dari lingkaran. Setelah satu putaran maka drone berhenti dan landing, beberapa detik setelah landing drone akan di disarm.

KENDALA :
Kesulitan dalam menentukan yaw angle dari drone yang bertambah seiring waktu sehingga kurang sempurna ketika drone ingin menghadap ke arah titik Tengah lingkaran. Karena drone bergerak melingkar mengikuti waktu ketika simulasi dimulai (time_now).

LINK VIDEO LAPORAN :
https://drive.google.com/file/d/1F-aafT9DvYlP0B6ss8FEyoqHrZ-epNA6/view?usp=sharing