착륙 시 하강속력의 사용자 지정

1. 개요
2. 사용자 입력 받기
3. 입력값을 속도로 반영하기

연세대학교
Space-Y 항공팀 제어부
2025/09/30

1. 개요

"LiDAR_및_ArUco_마커를_활용한_정밀착륙_시뮬레이션"에서 비행체의 착륙 시 하강속력은 사용자 입력으로 변화시킬 수 없다. 더불어, 하강속도를 결정하는 변수값이 속력과 직접적인 대응 관계에 있지 않다. 이러한 요인으로 인해 발생할 수 있는 사용자 오류 및 사고 방지와 유연한 비행 실험을 위해 하강속력을 사용자가 직접 지정할 수 있도록 한 `landing_test_vel` 노드의 코드를 설명한다.

2. 사용자 입력 받기

ROS 노드는 일반적인 `loop` 형태가 아니라 특정 시간마다 전체 코드가 반복해서 실행되는 구조이기 때문에 초기값 설정을 위해 `std cin >> variable;`과 같은 명령의 사용은 적합하지 않다. ROS 노드에서 초기값 설정을 위해서는 파라미터를 정의해주어 노드 실행 시 해당 파라미터의 값을 지정해주는 방법을 사용한다. 주석은 파란색, 추가된 코드는 빨간색이다.

```
#include ...

class LandingTest : public rclcppp::Node {
  public:
    LandingTest() : Node( "landing" ) {
      odom_sub_ = ...
      ...
      //declare ROS parameter called  "descent_vel_param"
      this->declare_parameter<float>( "descent_vel_param" , 0.0f);
      ...
      auto timer_callback = [this]() -> void {
        ...
        ...
        //pass value of  "descent_vel_param"  to float variable  "descent_vel_"
        descent_vel_ = (float)this->get_parameter( "descent_vel_param" ).as_double;
        ...
      }
      ...
  private:
    ...
    float descent_vel_ = 0;    //declare variable to use for flight control
    ...
```

ROS 파라미터는 파라미터용 데이터 타입을 갖고 있기 때문에 나중에 변수로서 계산에 사용하기 위해서는 알맞은 데이터 타입으로 변환해줘야 한다.

이후 아래의 명령어와 같이 일반적인 ROS 노드 실행 명령 뒤에 `--ros-args...`를 추가하면 `descent_vel_param`이라는 파라미터에 `<value>`값이 지정된다. 실제 실행할때는 `<value>`를 0.5와 같이 원하는 값으로 대체하면 된다.

```
$ ros2 run flight_control landing_test_vel --ros-args -p descent_vel_param:=<value>
```

3. 입력값을 속도로 반영하기

"LiDAR_및_ArUco_마커를_활용한_정밀착륙_시뮬레이션"에서는 좌표 기반 기체 제어를 통해 정밀착륙을 완수했다. 그러나 사용자에게 속도를 입력받아 기체 제어를 이루기 위해서는 속도 기반 제어를 통한 기체 제어가 이루어져야한다. 이에 따라 착륙 제어 함수인 land()를 아래와 같이 수정했다. 수정 내용은 빨간색, 주석은 파란색이다.

```
...
//needed for defining NaN
#include<cmath>
#include<limits>
...
void LandingTest::land() {
  TrajectorySetpoint msg{};
  Eigen::Quaternionf q(curr_odom.q[0], curr_odom_.q[1], curr_odom_.q[2], curr_odom_.q[3]);
  q.normalize();

  //coefficients for planar speed
  iter_ratio_ = descent_vel_*acc_alt_*0.15;
  float k = -acc_alt_*iter_ratio_;

  Eigen::Vector3f target_pos_FRD (0, 0, 0);
  if(desired_x_ != 0 || desired_y_ != 0)
    target_pos_FRD = {desired_y_*k, desired_x_*k, 0};

  Eigen::Vector3f target_pos_NED = q*target_pos_FRD;
  target_pos_NED.normalize(); //make ∠(Coordinate) to unit vector to use for speed
  Eigen::Vector3f target_vel_NED = iter_ratio_*target_pos_NED; //calculate planar speed

  if(/*aircraft' s altitude is low enough*/) {
    //send land command via VEHICLE_CMD_NAV_LAND
  }

  float nan = std::numeric_limits<float>::quiet_NaN();
  msg.position = {nan, nan, nan}; //should assign NaN to position for no position control
  msg.velocity = {target_vel_NED[0], target_vel_NED[1], descent_vel_};
  msg.timestamp = //clock;
  trajectory_setpoint_publisher_->publish(msg);
}
```

속도 기반 제어를 통한 착륙 시 좌표 기반 제어보다 다소 불안정하다는 단점이 있다. 정확성에 영향을 줄 정도는 아니지만 코드 구조 수정을 통해 안정성 개선이 이루어진다면 보다 신뢰도 높은 정밀 착륙을 실행할 수 있을 것이다.