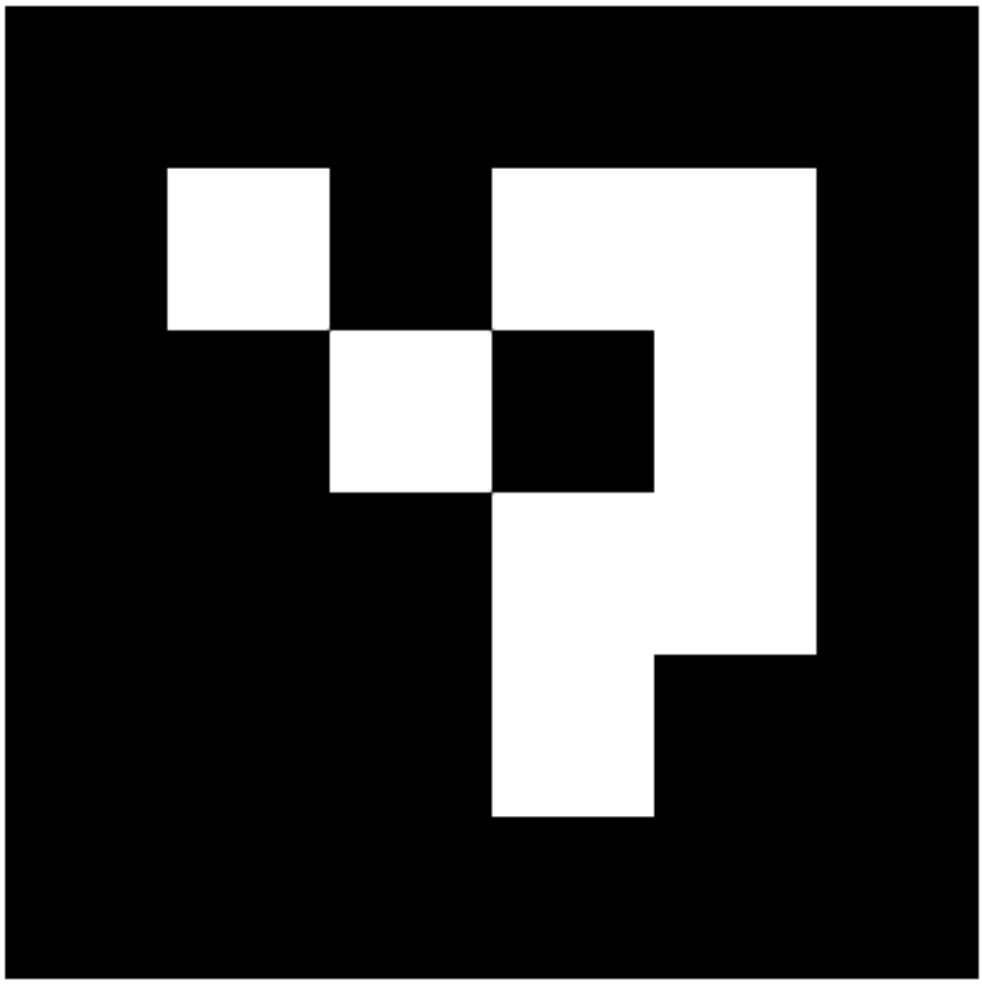
LiDAR 및 ArUco 마커를 활용한 정밀착륙 시뮬레이션

연세대학교
Space-Y 항공팀 제어부
2025/09/29

1. 개요

회전익 비행체를 목표 착륙지점으로 유도할 때 GPS, 비콘 등 다양한 방법을 사용할 수 있다. 그 중 ArUco 마커를 사용한 유도방법도 많이 사용된다. ArUco 마커는 검은색 테두리의 사각형에 흰색 사각형의 무늬가 있는 마커로, 탐지율이 비교적 높고 OpenCV 등을 활용하면 쉬운 탐지가 가능하다. 해당 시뮬레이션에서는 아래의 4×4 ID0 ArUco 마커를 사용한다.



시뮬레이션 환경은 '개발 환경 통일안(20250925)'와 동일하다. 해당 문서는 시뮬레이션 환경에서 LiDAR 정보를 받아와 착륙을 수행하는 방법에 대해 기술하고 있다. 실(實)기체에서는 회로 구성에 따라 LiDAR 정보를 읽는 방법이 다를 수도 있다.

한 줄이 긴 코드들이 있어 문서의 여백을 좁혔다...

2. 비행체 모델

Gazebo 모델은 .sdf 파일로 작성돼있다. 해당 파일들의 경로는 Gazebo 버전에 따라 조금씩 다르며, Gazebo Harmonic에서 PX4(v16.0.0) 모델은 아래의 경로에서 확인할 수 있다.

```
~/PX4-Autopilot/Tools/simulation/gz/models/
```

해당 시뮬레이션에서는 x500_lidar_down에 카메라를 결합하여 사용할 것이다. x500_lidar_down/안에는 model.cfg, model.sdf, 두가지 파일이 있다. 새로운 파일을 만들면 이후 CMakeLists.txt 등의 파일도 수정해야하기 때문에 model.sdf를 수정하여 과정을 간소화했다. 전문이 아래와 같이 되도록 수정하면 된다. 수정 부분은 빨간색 글씨로, 주석은 파란색 글씨로 표기했다.

```
<?xml version="1.0" encoding="UTF-8"?>
<sdf version='1.9'>
  <model name='x500_lidar_down'>
    <self_collide>false</self_collide>
    <include merge='true'>
      <uri>x500</uri>
    </include>
    <include merge='true'>
      <uri>model://LW20</uri>
      <pose relative_to="base_link">0 0 -0.079 0 1.57 0</pose>
    </include>
    <include merge='true'>
      <uri>model://mono_cam</uri>
      <pose0 0 .10 0 1.5707 0 </pose>
      <name>mono_cam</name>
    </include>
    <joint name="lidar_model_joint" type="fixed">
      <parent>base_link</parent>
      <child>lw20_link</child>
      <pose relative_to="base_link">-0 0 0 0 0 0</pose>
    </joint>
    <joint name="lidar_sensor_joint" type="fixed">
      <parent>base_link</parent>
      <child>lidar_sensor_link</child>
    </joint>
    <joint name="CameraJoint" type="fixed">
      <parent>base_link</parent>
      <child>camera_link</child>
      <pose relative_to="base_link">0 0 0 0 1.5707 0</pose>
    </joint>
    <link name="lidar_sensor_link">
      <!-- IMPORTANT: change first element of the link pose coordinate from 0 to 2>
      <!-- if not camera covers lidar and accurate value is not measured>
      <pose relative_to="base_link">2 0 -0.05 0 1.57 0</pose>
...
...
...
```

이후 파일을 저장하면 된다. PX4에서는 Gazebo를 실행할때마다 빌드를 하기 때문에 파일 수정 후 따로 빌드를 해줄 필요가 없다.

3. LiDAR 데이터 받아오기

3-1. Gazebo 토픽 확인

아래 명령어를 통해 Gazebo 실행 후 LiDAR 토픽이 잘 발행되는지 확인한다.

```
$ make px4_sitl gz_x500_lidar_down
```

LiDAR 토픽은 ROS2 토픽이 아닌 Gazebo 토픽으로 실행되기 때문에, 발행 중인 토픽 종류 와 내용 확인은 아래의 명령어로 수행한다.

```
$ gz topic -l                      ← lists Gazebo topics
$ gz topic -e -t <topic name>   ← echoes selected Gazebo topic
```

발행 중인 토픽 종류 중 /world/default/model/x500_lidar_down_0/link/lidar_sensor_link/sensor/lidar/scan/points의 내용을 확인해보면 아래와 같이 뜬다.

```
header {
  stamp {
    sec: 42
    nsec: 900000000
  }
  data {
    key: "frame_id “
    value: "lidar_sensor_link “
  }
  data {
    key: "seq"
    value: "2017"
  }
}
field {
  name: "x"
  datatype: FLOAT32
  count: 1
}
...
...
...
height: 1
width: 1
point_step: 32
row_step: 32
data: "\223B5>\000\000\000\000\000\000\000\000\000 ... \000"
is_dense: true
```

해당 토픽이 잘 발행되는 것을 확인했으면 ROS 토픽으로 바꿔준다.

3-2. ROS 토픽 변환하기

토픽 변환을 위해서 ros-gz-bridge를 사용한다. apt를 사용하여 설치하면 편하지만, 설정된 버전과 현재 사용 중인 버전이 다른 경우 bridge가 제대로 작동하지 않기 때문에 source에서 바로 설치하는 방법을 소개한다. 아래의 링크를 따라하면 된다.

https://github.com/gazebosim/ros_gz/tree/humble?tab=readme-ov-file#ros

일반적인 ROS 패키지 설치와 똑같다. workspace 이름은 꼭 ws가 아니라 마음대로 해도 된다. 해당 문서에서는 혼선을 방지하기 위해 ros-gz-bridge를 설치한 workspace는 ros-gz-ws라고 지칭한다.
중간에 아래와 같은 명령을 수행하는 부분이 있다.

```
$ rosdep install -r --from-paths src -i -y --rosdistro humble
```

수행 후 만약 콘솔에 아래와 같은 메시지가 보이면 안된다.

```
ros-gz-bridge: cannot locate rosdep definition for [gz-transport13]
```

이러한 경우 아래 명령 수행 후 다시 rosdep 명령을 수행하면 해결이 될 수도 있다...

```
$ sudo apt autoremove
```

설치가 완료되면 새로운 콘솔에서 아래의 명령어를 실행하여 Gazebo 토픽을 ROS 토픽으로 바꿔준다.

```
$ cd ~/ros-gz-ws
$ source install/setup.bash
$ ros2 run ros_gz_bridge parameter_bridge
/world/default/model/x500_lidar_down_0/link/lidar_sensor_link/sensor/lidar/scan/points@se
nsor_msgs/msg/PointCloud2[gz.msgs.PointCloudPacked
```

문제없이 실행이 되었으면 아래의 명령어로 ROS 토픽으로 내용 누락 없이 잘 변환되고 있는지 확인한다.

```
$ ros2 topic list
$ ros2 topic echo
/world/default/model/x500_lidar_down_0/link/lidar_sensor_link/sensor/lidar/scan/points
```

3-2-1. 에러

Unknown data type[9]와 같은 메시지가 뜨거나 Failed to create bridge for topic ... 과 같은 메시지가 뜨면 ros-gz-bridge의 버전이 개발환경과 맞지 않을 확률이 크다.

bridge 구축은 잘 되지만 ros2 topic echo에서 아무 데이터가 뜨지 않으면 토픽의 이름에 오타가 있을 확률이 크다.

4. Node 작성하기

marker_recognition에서 LiDAR 좌표까지 받아오도록 코드를 수정한다. 아래의 코드를 기존 코드의 사이사이에 잘 끼워넣으면 된다. 기존 코드의 일부분도 참고점 개념으로 작성하였다. 작성해야 하는 코드는 빨간색, 주석은 파란색으로 작성했다.

```
...
import struct
...
from sensor_msgs.msg import PointCloud2
...

class MarkerRecognition(Node):
    ...
    def __init__(self) -> None:
    ...

        self._lidar_sub = self.create_subscription(
            PointCloud2,
            "/world/aruco/model/x500_lidar_down_0/link/lidar_sensor_link/sensor/lidar/scan/points",
            self._lidar_cb,
            10
        )
    ...

    def _lidar_cb(self, msg: PointCloud2) -> None:
        raw = bytes(msg.data)
        first_four = raw[0:4] # ← PointCloud2 gives data in little endian and needs conversion
        self._altitude = struct.unpack('<f', first_four)[0] - 0.17701
        self.get_logger().info(f"calculated altitude: {self._altitude:.04f}")
```

저장 후 빌드하자. 파이썬 노드는 CMakeLists.txt와 package.xml에 따로 dependency를 작성할 필요가 없기에 해당 패키지에서 더 이상 수정할 파일은 없다.

landing_test에서는 marker_recognition에서 보낸 LiDAR 데이터를 고도 데이터로 사용하도록 수정한다. 이번에도 기존 코드의 일부분도 같이 작성하였다. 수정해야 하는 코드는 빨간색, 주석은 파란색으로 작성했다.

```
#include ...
#include ...
...

class LandingTest : public rclcpp::Node{
  public:
    LandingTest() : Node("landing") {
    ...
      desired_setpoint_sub_ = this->create_subscription<...>(...) {
            desired_x_ = msg->point.x;
            desired_y_ = msg->point.y;
            acc_alt_ = -msg->point.z;   //point.z>0 whereas acc_alt_ should be negative since it is in NED frame
      });

void LandingTest::land() {
  ...
  //acc_alt_ = curr_odom_.position[2];   //delete or comment out this line
  ...
}
...
```

저장하고 빌드하면 된다. 디렉토리나 디펜던시 변동사항이 없기에 CMakeLists.txt나 package.xml는 수정하지 않아도 된다.

5. 시뮬레이션 실행하기

5-1. QGC 실행하기

5-2. uXRCE Agent 실행하기

5-3. Gazebo 실행하기

아래의 명령어를 활용하여 실행한다. 착륙지점 표시를 위한 ArUco 마커가 있어야 하기에 aruco.world를 사용한다.

```
$ PX4_GZ_WORLD=aruco make px4_sitl gz_x500_lidar_down
```

5-4. ros-gz-bridge 실행하기

아래의 명령어를 활용하여 실행한다. 토픽의 이름이 다를 수도 있으니 실행 전에 확인해본다.

```
$ cd ~/ros-gz-ws
$ source install/setup.bash
$ ros2 run ros_gz_bridge parameter_bridge
/world/aruco/model/x500_lidar_down_0/link/lidar_sensor_link/sensor/lidar/scan/points@sens
or_msgs/msg/PointCloud2[gz.msgs.PointCloudPacked
```

5-5. marker_recognition 노드 실행하기

그냥 실행하면 노드가 물리 카메라에 접근하기 때문에 아래의 명령어를 활용하여 가상 카메라에 접근하도록 설정한다.

```
$ ros2 run imagery_processing marker_recognition --ros-args -p camera_source:="udpsrc port=5600 !
application/x-rtp, encoding-name=H264 ! rtph264depay ! h264parse ! avdec_h264 ! videoconvert ! appsink"
```

5-6. landing_test 노드 실행하기