AER1217: Development of Unmanned Aerial Vehicles
Lab 3: Georeferencing Using UAV Payload Data

1 Introduction

This is the third lab in a series designed to complement the lecture material and help you with the course project. In this lab, you will be using the Parrot AR.Drone 2.0 bottom-facing camera (64° diagonal FOV, 640px × 360px) to detect and locate circular targets of interest on the ground. The bottom camera's intrinsic matrix and distortion coefficients are the following:

$$K = \begin{bmatrix} 674.84 & 0.00 & 298.61 \\ 0.00 & 674.01 & 189.32 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$$

$$d = \begin{bmatrix} 0.159121 & -0.549986 & 0.003377 & 0.001831 & 0.000000 \end{bmatrix}$$

The data from the Vicon motion capture system and the images from the quadrotor will be used to find each target's location on the ground within the inertial Vicon fixed reference frame. The extrinsic transformation matrix from the vehicle body frame to the camera frame is assumed to be

$$T_{CB} = \begin{bmatrix} 0.0 & -1.0 & 0.0 & 0.0 \\ -1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}.$$

1.1 Lab Objectives

This lab requires you to design an image processing algorithm that will assign georeferenced coordinates to the targets of interest based on the position and attitude of the quadrotor. The targets of interest are **six green circles** within an area on the ground bounded by $x \in [-2.0, 2.0]$ m and $y \in [-2.0, 2.0]$ m. To perform the georeferencing task, you are required to design a trajectory for the quadrotor to scan the area and collect the Vicon and AR.Drone bottom image data during your lab time slot. The image processing algorithm for georeferencing will be developed based on the data you collected. You may find the OpenCV library useful for this task. This lab is split up into two separate sections:

1. Design and implementation of the trajectory for data collection.
2. Implementation of a georeferencing algorithm that analyzes images and AR.Drone pose data to locate the circular targets of interest on the ground.

2 Lab Components

2.1 Requirements

For this lab, you are required to locate the centroid of targets of interest in the Vicon space using the quadrotor. There will be a **six targets (green circles)** on the ground for which you are required to provide the Vicon position coordinates based on the AR.Drone pose (position and attitude) and the images it captures. The targets will be within an area on the ground bounded by $x \in [-2.0, 2.0]$ m and $y \in [-2.0, 2.0]$ m.

2.2 Data Collection

You are required to design and implement a trajectory for data collection. You are free to choose any trajectory within the following position bounds: $x \in [-2.0, 2.0]$ m, $y \in [-2.0, 2.0]$ m, and $z \in [0.0, 2.0]$ m. The bottom-facing camera on the AR.Drone will be used to collect image data. To toggle the camera channel, you will need to call the ROS service `/ardrone/setcamchannel` or `/ardrone/togglecam`. Setting the channel to 1 activates the bottom-facing camera, and channel to 0 activates the front-facing camera. The service can be called either in the terminal or in the Python code.

In the terminal, ROS service can be called:

```
rosservice call /ardrone/togglecam     or
rosservice call /ardrone/setcamchannel 0
rosservice call /ardrone/setcamchannel 1
```

In the Python code, ROS service can be called through this code snippet:

```
from ardrone_autonomy.srv import CamSelect

...

rospy.wait_for_service("ardrone/setcamchannel")
try:
    switchcam = rospy.ServiceProxy("ardrone/setcamchannel",CamSelect)
    switchcam(1) # for bottom-facing, switchcam(0) for front-facing
except rospy.ServiceException as e:
    print("Service call failed:  %s"%e)
```

During the flight, you need to collect the bottom image data from the AR.Drone and the Vicon information which allows you to develop the image processing and georeferencing algorithms after the flight experiment. The two corresponding ROS topics are: `/ardrone/bottom/image_raw` and `/vicon/ARDroneCarre/ARDroneCarre`.

It is recommended to collected data through rosbag recording. The `rosbag record` command records the data from a running ROS system into a `.bag` file. In the terminal, the command for recording the above two ROS topics is:

```
rosbag record /ardrone/bottom/image_raw /vicon/ARDroneCarre/ARDroneCarre
```

To conduct the real-world experiments, you need to transfer your codes in the `aer1217_ardrone_simulator` folder, developed in Lab2, to the `aer1217_ardrone_vicon` folder. The files need to be transfered and modified includes:

- `scripts/ros_interface.py`,
- `scripts/position_controller.py`,
- `scripts/desired_positions.py`,
- other equivalent, customized developed scripts, and
- `launch/ardrone_vicon.launch`

Some general tips about the real-world experiment are as follows:

- Ensure the wired connection (Ethernet) is connected to receive Vicon data,
- Ensure the wireless connection is established with Parrot AR.Drone,
- Ensure the AR.Drone is detected by the Vicon system and the rostopic `/vicon/ARDroneCarre/ARDroneCarre` is publishing data,
- Remember to switch the camera view to the bottom-facing camera, and
- Ensure the images and Vicon data are collected during the flight.

2.3 Georeferencing

The georeferencing process includes two steps:

- Detect the target positions through image processing.
  In this lab, the target positions are referred to the centers of the green circles on the ground.
- Transform the detected target positions from the bottom camera frame into the Vicon inertial frame.

In this lab, you are allowed to use the image processing functions provided by the OpenCV library to process the image data. To interface ROS and OpenCV, you will find `cv_bridge` package useful, which converts ROS image messages into OpenCV images. The Python tutorial for `cv_bridge` can be found in here. The default supported version of OpenCV for ROS Kinetic is OpenCV.3. Both OpenCV.3 and the `cv_bridge` package are already installed during the Lab1 environment setup.

Since there exists motion noises during the flight experiments, not all image frames can provide accurate target location results. The quality of the image data also depends on the desired trajectory and the motion control performance during the flights. You may need to select a subset of the collected image frames for georeferencing. Moreover, the asynchronous nature of real-world sensor data streams will lead to a time offset between the arrival times of the image data and the Vicon data. Hence, a data synchronization process is essential to achieve accurate georeferencing results. Additionally, to improve the robustness and accuracy of the algorithm, you should use more than one image per target to estimate the target positions in your algorithm design.

3 Deliverable and Grading

Your deliverable for this lab includes (1) a short report of **maximum four pages** documenting your results, and (2) the source code that your team wrote or modified. You are encouraged to write code in a modular way.

This lab is worth 15%. Please submit the following in a .zip format, to Quercus, latest by **11:59 PM, March 21, 2022**:

- **Trajectory generation** (2%):
  - Design of a path that could cover the area on the ground bounded by $x \in [-2.0, 2.0]$ m and $y \in [-2.0, 2.0]$ m. Demonstrate the trajectory in simulation.
- **Data collection** (3%)
  - Collection of the `.bag` file including bottom image from the AR.Drone and Vicon data for post-lab processing.
- **Code** (3%):
  - Image (post)-processing (1%).
  - Target localization (2%)

  Please comment your code and provide a `readme.txt` with proper explanation of the code structure.
- **PDF report** (7%) containing the following:

- Overview of the algorithm used to locate targets, including the equations used. (2%)
  You should use more than one image per target to improve your estimation.
- Georeferencing results. (5%)
  Summary of the targets found and their estimated positions.
  The grading for the target position accuracy follows the formula:

$$\text{Score} = \sum_{i=1}^{N} \frac{5}{N} \beta_i \min\left\{1, \frac{0.25}{R_i}\right\}$$

  where N is the total number of targets, β_i equals to 1 if the i-th target is detected (0 otherwise), and R_i is the distance between the identified position and actual position of the target in meters.