# ELEC 5660: Introduction to Aerial Robotics
# Project 2: Phase 1

Assigned: Mar. 19, 2024 Due: 11:59 PM Mar. 29, 2024

## 1 Project Work

In Project 2 Phase 1, you need to implement the 3D-2D pose estimation [1] algorithm learned in the lecture to estimate the camera's poses with images. This is an individual project, which means you must complete it by yourself.

### 1.1 Environment Setup

You can finish this project using Ubuntu 16.04/18.04 in your local computer. Otherwise, you will be configuring and using a Docker environment. The advantage of using Docker is that you can deploy the programming environment needed for this assignment across multiple platforms, such as Windows and Linux, without the need of complex environment configuration steps. Please refer to the Github Repo for specific installation and configuration instructions.

### 1.2 Tasks

Three tasks are required for this phase including:

1. Calculating the camera's pose corresponding to every image.
2. Publishing camera pose information in the message form of **`nav_msgs/Odometry`**.
3. Plotting these poses with **rqt_rviz**, which is a visualization tool in ROS.
4. Comparing your result with the reference.

### 1.3 Note

1. You are **NOT** allowed to use any OpenCV functions in your implementation.
2. **IMPORTANT**: If you reference or use any external sources, please clearly indicate this in your report or code comments. Please explain how your work differs from the referenced work.
3. If your code framework is different from the template provided, please make sure to give a **README** file to explain how to run your code. If we are unable to run your code directly, we will not be able to grade your assignment.

4. If you are unfamiliar with ROS, please refer to the **ROS tutorials** for learning materials. If you have any difficulties or questions, please do not hesitate to contact your TAs.

### 1.4 Details

You will be provided with a ROS package named `tag_detector`, where a serial of points and their positions will be calculated with images. You need to implement this project based on the point and position array. You can follow the below procedures to prepare for your coding.

1. Put `aruco-1.2.4` and `tag_detector` in your workspace (such as `/home/worksapce/catkin_ws/src/`). If you are configuring your Docker environment according to our tutorial, you can put both folders in your local path. Changes made to the files in Docker will be synchronized and updated in the local machine.
2. Install aruco (following `aruco-1.2.4/README`)
3. Setup your ROS environment and compile the `tag_detector` package (catkin_make).
4. Find `bag_tag.launch` in `tag_detector/launch`, and `images.bag` in `tag_detector/bag`. You can use `images.bag` to verify that your code is bug-free. We will use a separate bag to test your submitted code.
5. Use `bag_tag.launch` and `images.bag` to run this package (roslaunch `bag_tag.launch`).
6. Read the comments in `tag_detector/src/tag_detector_node.cpp` **carefully**.
7. Add your code into `tag_detector/src/tag_detector_node.cpp`.
8. Note that the pose you calculated is $(\mathbf{t_{cw}}, \mathbf{R_{cw}})$, which represents the pose of world frame respecting to the camera frame.

### 1.5 Basic use of rviz

1. open one terminal, input:roscore; open another termianl, input: rosrun rviz rviz.
2. click add button, add your topic.
3. change the frame to ”world” (see Figure.1).
4. change the color of your odometry and reference odometry.

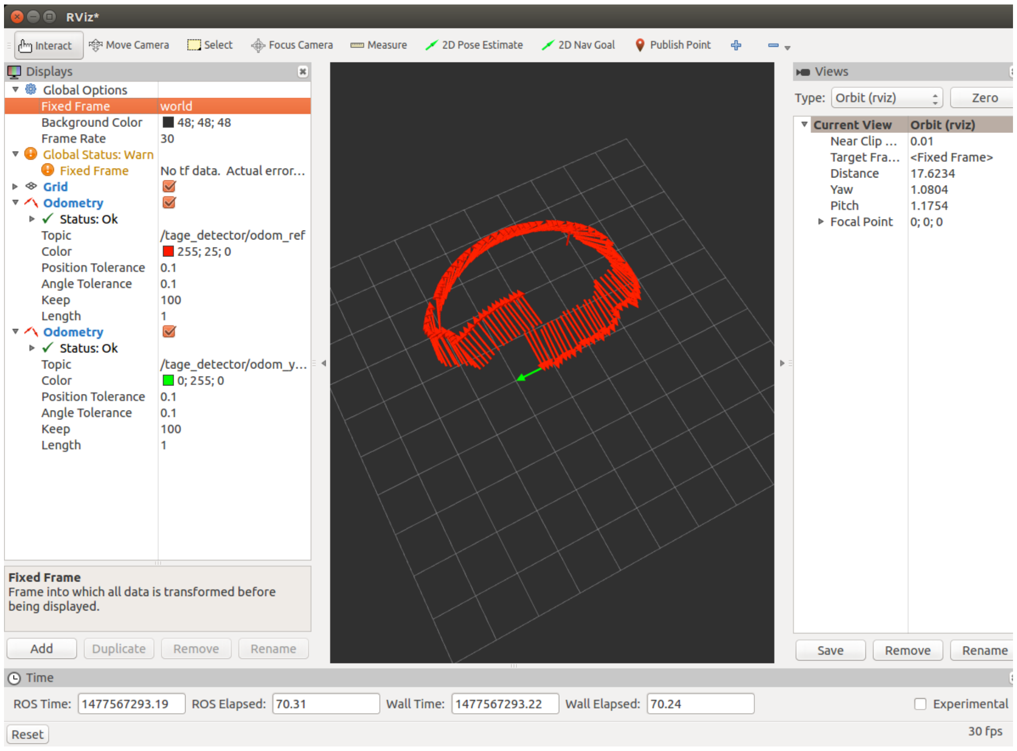
## 2 Submission

When you finish the assignment you may submit your code and documents on **canvas** before **Mar. 29, 2024 23:59:00**. The project name for this assignment is titled “**`proj1phase2_YOUR_NAME.zip`**”.

Please cite the paper, GitHub repo, or code url if you use or reference the code online. Please keep academic integrity. **Plagiarism** is not tolerated in this course.

Your submission should contain:

1. A **maximum 2-page** document including:



```
Global Options
  Fixed Frame            world
  Background Color       48; 48; 48
  Frame Rate             30
Global Status: Warn
  Fixed Frame            No tf data.  Actual error...
Grid                     ✓
Odometry                 ✓
  Status: Ok
  Topic                  /tage_detector/odom_ref
  Color                  🟥 255; 25; 0
  Position Tolerance     0.1
  Angle Tolerance        0.1
  Keep                   100
  Length                 1
Odometry                 ✓
  Status: Ok
  Topic                  /tage_detector/odom_y...
  Color                  🟩 0; 255; 0
  Position Tolerance     0.1
  Angle Tolerance        0.1
  Keep                   100
  Length                 1

Fixed Frame
Frame into which all data is transformed before
being displayed.

Type: Orbit (rviz)
Current View             Orbit (rviz)
  Near Clip ...          0.01
  Target Fra...          <Fixed Frame>
  Distance               17.6234
  Yaw                    1.0804
  Pitch                  1.1754
  Focal Point            0; 0; 0

ROS Time: 1477567293.19   ROS Elapsed: 70.31   Wall Time: 1477567293.22   Wall Elapsed: 70.24
30 fps
```

Figure 1: step 3

(a) Figures plotted by **rviz**.

(b) Statistics about your result. (For example, RMS error between the poses you calculate with the reference ones)

(c) Descriptions about your implementation.

(d) Any other things we should be aware of.

2. Files `tag_detector_node.cpp`, as well as any other C++ files you need to run your code.

## References

[1] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.