NTNU
Kunnskap for en bedre verden

Department of Marine Technology

Course Code - TTK33 Aerial Robotics: Autonomy methods and Systems

Open-hardware Eye-like compact pan-tilt camera-IMU system

Authors:
Tihan Mahmud Hossain
Md Shamin Yeasher Yousha
Abhimanyu Bhowmik
Madhushree Sannigrahi

Date 13/12/2024

1 Motivation and Problem Statement

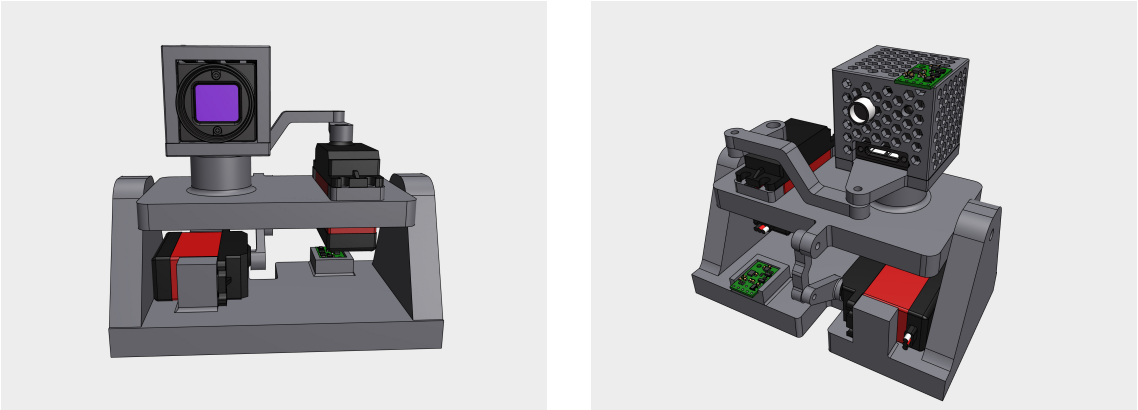
In this project, we aim to develop a compact eye-like open-hardware pan-tilt camera-IMU system. This system will serve two main purposes:

- Active viewpoint control with pan and tilt gimbal using servo motors,
- Image stabilization using the camera feed and IMU data

This project is designed for active vision-based guidance of unmanned aerial vehicles (UAVs). The goal is to integrate hardware and control efficiency with vision stability for precision tasks like targeting precise drops on incipient-stage wildfires. All the codes and details are present in Github repository *Eye-like Pan-tilt Camera-IMU system* 2024.

2 Hardware Description

We received the necessary hardware components from the Autonomous Robots Lab which includes FLIR Blackfly USB3 Colored imaging camera, 2 Polulu MinIMU 9 v6, 2 Servos, Raspberry Pi 4b, and Arduino UNO.



(a) Front View (b) Back View

Figure 1: CAD design for the pan-tilt Gimbal hardware

The animatronics eye-like pan-tilt system is designed in CAD software (Shapr3D) as given in Figure 1 and 3D printed using Ultimaker 2+. This enables the rotation of the camera in horizontal (pan) and verticle (tilt) axes, providing flexibility for viewpoint control. One of the servos is placed at the bottom plate to control the tilt movement and the other is on the top plate for the pan movement. This ensures smooth and precise movements in both axes. The pan mechanism allows the camera to move from 30°to 130°along horizontal axis and the tilt mechanism allows for movement from 20°to 140°along the verticle axis.

For attitude estimation, the gyro and accelerometer data from two different IMUs are used. One IMU is fixed at the bottom base and the other IMU is fixed on the camera top to get the orientation data in the base frame and camera frame respectively. This IMU data is used for the viewpoint and image stabilization in this project.

3 Viewpoint Control

With viewpoint control, we are aiming to keep the camera fixed to one point of view in spite of any movement in the UAV. To achieve this, we first assemble the Arduino, which receives

inputs from the 2 IMUs and processes the gyroscope and accelerometer data to map the desired output angle to control the servos. For calibrated IMU data, the bias is removed from the raw accelerometer and gyro readings. Since we are receiving linear acceleration and angular velocity as measurements, we calculate the pitch (tilt) and yaw (pan) angles using a complimentary filter. Extended Kalman Filter, which provides more stability to the system. For further stabilization of the system, we implemented a low-pass filter which smoothes out the high-frequency noises and maintains a smooth camera movement.

After processing, the smoothed filtered data is used for mapping the desired pan and tilt movement for compensating the body motion. This data is sent to the respective servos for actuation.

4 Image stabilization

For stabilization of the camera feed, we used the Gyroflow application [Eddy 2024]. Using Gyroflow requires the gyro data and lens profile along with the video.

To obtain the camera feed from the FLIR Blackfly camera, we used Sparinnaker SDK [Vision-Solutions 2024] along with its ROS drivers. This provides us with multiple image topics in ROS such as `camera/image_raw`, `camera/image_color`, etc. For this project, we used a feed from `camera/image_color/compressed` which provided us with colored video data. We color-corrected the video feed like balancing exposure, brightness, white balance, etc using `rqt_reconfigure`. The camera color configuration can be found in the `camera_color.config` file in the Github repository.

We calibrated the camera using the ROS Noetic `camera_calibration` package with an 8 x 6 checkerboard, where each block is 2 cm in length. This generates a camera calibration file, which is used by the SDK to build and publish the `camera/camera_info` topic. By doing this calibration we get the camera and lens parameters such as resolution, distortion coefficient, camera matrix, projection, and retrification matrix. These parameters are stored in a **blackfly_camera.json** file and provided to Gyroflow for image stabilization.

For the Gyro data, we connected the Polulu MinIMU to Raspberry Pi GPIO using I^2C communication. This GPIO data is used by a ROS node for calibrating the gyro, calculating and removing the bias from the data, and publishing it into the ROS topic `camera/gyro/sample`.

Since we required camera-synchronized gyro data in a **.csv** format, we developed a ROS package called `video_gyro_recorder` for time-synchronization between the camera node and the gyro node in ROS. This node gives us the video in **.mp4** format and synced gyro data in **.csv** format. The Gyroflow takes this camera and gyro node data along with the blackfly_camera.json file and renders the stabilized video after processing.

4.1 Discussion

The main drawback of Gyroflow is that it operates offline and requires substantial computing power for rendering stabilized videos. We tried to make this process online and since Gyroflow doesn't have any online rendering solution, we attempted using another software package called `virtualgimbal_ros` [Sato 2022]. The ROS nodes for camera and gyro data were interfacing with this package. We could not make it work on Raspberry Pi since it is resource-intensive. Alternatively, we built a ROS network with Raspberry Pi as the master and connected a computer with an NVIDIA GTx GPU for stabilization but the `camera/camera_info` topic was not synced with the `camera/image_color` topic. To overcome this, we recorded bag files and synchronized the `/camera_info` data. Yet, while using `virtualgimbal_ros`, the rendering freezes after 4-5 frames. After investigating further, we realised the repository works only with 1920x1080 undistorted video with a higher IMU refresh rate. Due to time-constraint, we could not develop further on this, so we moved on with the Gyroflow application.

Calibrating IMU was also a huge challenge as the IMU yaw and pitch data was coupled with each other along with the structural constraint, requiring multiple filters for stabilization.

Bibliography

Eddy, Adrian (2024). *Gyroflow*. URL: https://github.com/gyroflow/gyroflow (visited on 6th Dec. 2024).

Eye-like Pan-tilt Camera-IMU system (2024). URL: https://github.com/Tihan-hossain/Eye-like-pan-tilt-camera-IMU-system (visited on 13th Dec. 2024).

Sato, Yoshiaki (2022). *Virtualgimbal ROS*. URL: https://github.com/yossato/virtualgimbal_ros (visited on 19th Sept. 2022).

VisionSolutions, Teledyne (2024). *Sparinnaker SDK*. URL: https://www.teledynevisionsolutions.com/products/spinnaker-sdk/?model=Spinnaker%20SDK&vertical=machine%20vision&segment=iis (visited on 2024).