Project 3b: Circular BullsEye

Team 6: Noob Quaternions

using 2 late days

Abhinav Modi
Masters of Engineering in Robotics
University of Maryland, College Park
Email: abhi1625@umd.edu

Prateek Arora
Masters of Engineering in Robotics
University of Maryland, College Park
Email: pratique@terpmail.umd.edu

I. Introduction

The objective of this project was to takeoff, search for a circular bullseye pattern and then land on the target. Approximate location of the center of the target were given in the form of a Gaussian distribution from the start location. The challenge was to detect the pattern under different lighting conditions and segment it from the background to estimate the position of its center in the camera coordinate frame.

II. Vision Pipeline

The ultimate aim of vision node is to get the pose of the circular tag in the world coordinate system. In order to do so, the circular tag has to segmented from the scene and followed by filtering of the resultant image in order to get a perfect bulls-eye target. The entire pipeline is explained in detail in the following subsections:

A. Tag Segmentation

Due to the presence of multiple features in the image from downward facing camera extracting the tag to detect the bulls-eye is crucial. The property of high contrast of bull-eye target and circular rings is exploited to segment it. All the pixels with intensity values less than 80 percent of maximum intensity value in the image are discarded. Since the target is reflective and shiny, this condition is true for most cases. After this step we have a perfect white rectangle with the circular rings in it as shown in figure 2

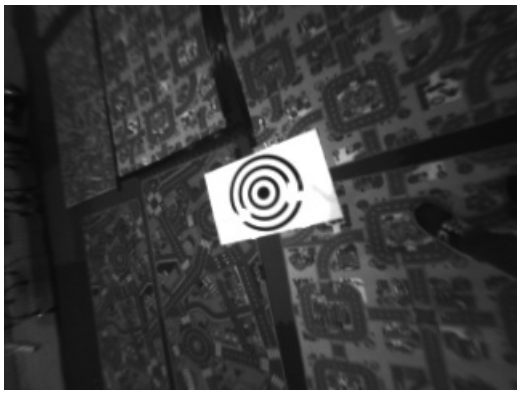

Figure 1: Image from left duo camera

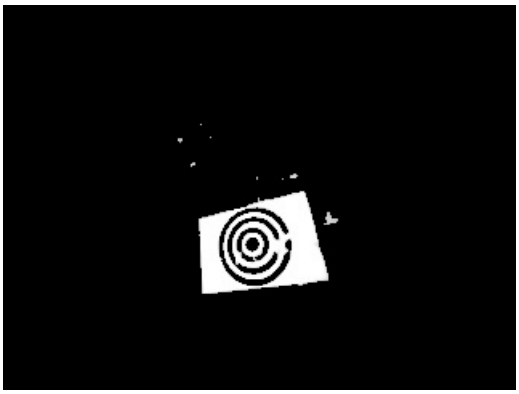

Figure 2: Output image after color thresholding

B. Filtering

Now that we have the tag with rectangular boundary, the process of extracting circular rings is the next step. Using the canny edge detector, an image (figure 3) containing the edges of the tag is obtained. Next, we dilate the edge image and use opencv function "findContours()" to get all the contours. These contours need filtering since there is no clear boundary between the rectangular contour and all the other elliptical contours. In order to get ellipse in all the frames, the outer rectangle needs to be removed.

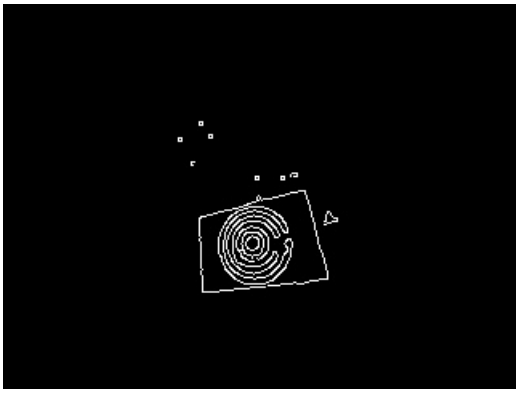

Figure 3: Output of canny edge detector

To filter out the rectangle, the biggest continuous contour (figure 4) detected above is taken and using opencv function

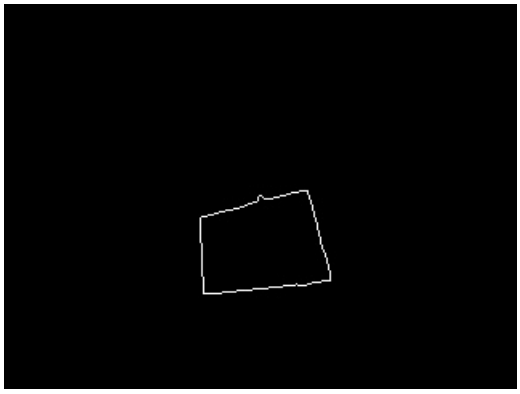

Figure 4: Image of biggest contour after using opencv function "findContours"

"houghLines()", lines are fitted to this contour. We created our own clustering function to get four lines that correspond to rectangle, since we get multiple houghlines (approx 150 lines in every frame) for the same rectangle.
Now that the four lines are obtained remove the outer rectangle in canny image using **half-planes**. A half-plane is a planar region consisting of all points on one side of an infinite straight line, and no points on the other side. After all the aforementioned filtering, an image with just the contours of ellipse are obtained which is shown in figure 5.

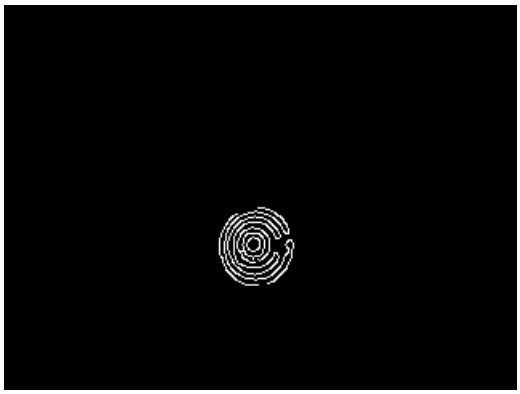

Figure 5: Segmented output of rings of circular tag

C. Ellipse Detection

This part is fairly trivial. Using the opencv function "fitEllipse()", an ellipse is fit to the largest contour in the image obtained in last section. We get the center, length of minor and major axis and the angle of inclination of major axis, theta.

D. Pose Estimation

Assuming the center of the bulls-eye at the center of the ellipse as the world origin, we get the point correspondences as shown in equation 1. Using the point correspondences we compute pose of the camera with respect to the world origin using our own PnP function.

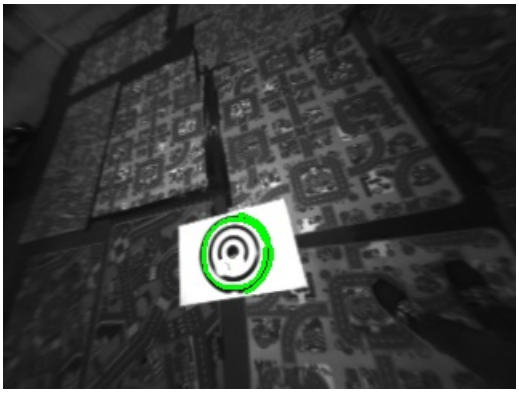

Figure 6: Output of fitEllipse

$$W = \begin{bmatrix} (\rho, 0, 0) \\ (0, \rho, 0) \\ (0, -\rho, 0) \\ (-\rho, 0, 0) \end{bmatrix} \tag{1}$$

$$x_{im} = \begin{bmatrix} (X_c - bsin\theta, Y_c + bcos\theta) \\ (X_c + acos\theta, Y_c + asin\theta) \\ (X_c + bsin\theta, Y_c - bcos\theta) \\ (X_c - acos\theta, Y_c - asin\theta) \end{bmatrix} \tag{2}$$

Figure 7: Pose calculated from PnP

III. Position Following Controller

Bebop's internal controller provides good attitude stabilization and altitude control for maintaining the quad in a state of hover. To control the position in the horizontal x and y directions we designed a simple LQR controller [1] which computes the thrust in the x and y directions to maintain position and even track a given (x,y) position.

A. LQR Controller

The state space models for the horizontal motion are:

$$\begin{bmatrix} \dot{p_x} \\ \dot{v_x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ v_x \end{bmatrix} + \begin{bmatrix} 0 \\ g \end{bmatrix} u_x \tag{3}$$

$$\begin{bmatrix} \dot{p_y} \\ \dot{v_y} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p_y \\ v_y \end{bmatrix} + \begin{bmatrix} 0 \\ g \end{bmatrix} u_y \tag{4}$$

where u_x and u_y are control commands in the x and y directions.

After tuning the Q and R matrices were chosen as R = 20 and

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}$$

The computed gain matrix $K = \begin{bmatrix} 0.2236, & 0.2657 \end{bmatrix}$

To track a particular reference position (r_x, r_y) the control commands u_x and u_y were calculated using

$$u_x = -\begin{bmatrix} 0.2236, & 0.2657 \end{bmatrix} \begin{bmatrix} p_x - r_x \\ v_x \end{bmatrix} \tag{5}$$

$$u_y = -\begin{bmatrix} 0.2236, & 0.2657 \end{bmatrix} \begin{bmatrix} p_y - r_y \\ v_y \end{bmatrix} \tag{6}$$

The reference position was then set in the mission script, the process of which is explained in the next section.

B. State Logic

The whole task from takeoff to landing on target was divided into 3 missions which are listed below:

- **Mission 1**: Takeoff and go to (r_x, r_y).
  Since we were given an initial estimate of the location of the center of the bullseye pattern, the first mission concentrated on taking off and then navigating to that reference location (r_x, r_y) which was given manually. Once the drone reaches within 5cm radius of the given reference location, a flag for starting the vision node was sent to start searching for the target on the ground. This was done to prevent the processor from doing extra computations. This marks the completion of mission-1 and the flag for mission-2 is set True.
- **Mission 2**: Increase Altitude and align with center.
  The vision node utilizes a wide-angled downward facing stereo camera for detecting the target. So if the target location is within approximately 1.5m of the reference location, the target is always visible in the image if the quad is at an appropriate height.
  So during this mission the quad increases altitude to a set reference and once the center of the pattern is estimated aligns with the center and holds position. This marks the completion of mission-2 and mission-3 is started.
- **Mission 3**: Descend, Realign and Land.
  The quad descends from the set altitude, in the previous mission, to a lower reference which was decided based on the minimum height from which the whole pattern can be seen in the image. Now, we re-estimate the center of the pattern and try to align the quad's x-y position with the pattern's center. Once the error in position is within 5cms, the landing command is initiated. This marks the end of mission-3 and thus, the whole task.

The complete trajectory of one of the runs performed during the final demo was plotted in rviz which is shown below:

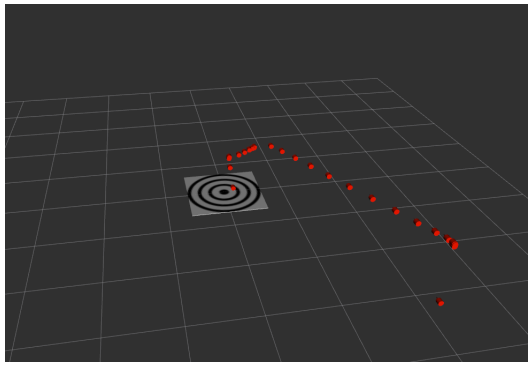

Figure 8: Trajectory plot of the complete mission

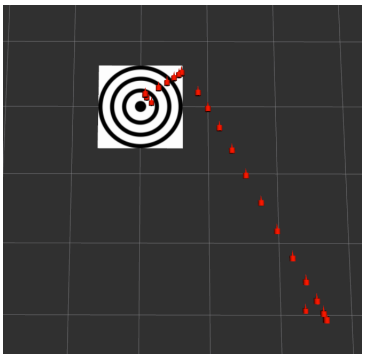

Figure 9: Top view of the trajectory plot

IV. Discussion and Conclusion

A. Odometry Estimation

The controller was highly reliant on the position and velocity estimates. So a very good and stable estimate of the quad's odometry was required. But the estimate of position from the optical flow was very iffy resulting in poor trajectory following. On the other hand, the linear velocity estimates were near perfect. We feel that this is due to the fact that bebop utilizes a small angle assumption due to which the rotation part in the equation of optical flow becomes obselete resulting in the following equation

$$\dot{p} = \frac{1}{Z} \begin{bmatrix} xV_z - V_x \\ yV_z - V_y \end{bmatrix} \tag{7}$$

By integrating these linear velocities we computed the x-y position which was way better than the bebop's odometry estimates to improve the trajectory tracking performance.

B. Velocity saturation

The rate at which we can publish velocity commands to the quadrotor is very low(around 10Hz) therefore the max velocity magnitude being published was clipped so that the quad doesn't overshoot due to a very high velocity command due to a very high error in position. The clipping was done according to the following equation:

vel_cmd = $min\{max\{$ctrl_cmd, -vel_max$\}$, vel_max$\}$

For our purpose vel_max was chosen to be 0.25 m/s.

C. Varying lighting condition

The thresholding performed to recover tag from scene needs to be adaptive for the algorithm to work in all lighting condition. For this we use opencv function for "Contrast Limited Adaptive Histogram Equalization". This made our thresholding exponentially better.

D. Averaging filter

Since vision pipeline is not 100% accurate we implemented a moving average filter to get stable pose estimates. We define a window of 10 observations and take the mean as our final pose estimate.

V. Comments about the submission videos

Bullseye: Both the poses displayed on the image are with respect to the inertial frame of the drone. The origin of inertial frame is the position where drone takes off from the ground.
video link
Bullseye-duo: The red dots in the video submitted denote the end points of major and minor axis and the center. Camera coordinate system is as follows: positive x points into the image, positive y points to the left of image and positive z points upwards.
video link

Bullseye-rviz: video link

References

[1] Axel Reizenstein. Position and trajectory control of a quadcopter using pid and lq controllers, 2017.