ENAE788M Project 3b
Team Bouncing Rainbow Zebras

Erik Holum
Graduate Student
University of Maryland
Email: eholum@gmail.com

Edward Carney
Graduate Student
University of Maryland
Email: carneyedwardj@gmail.com

Derek Thompson
Graduate Student
University of Maryland
Email: derekbt@yahoo.com

***Abstract*—We discuss using a mounted, down facing camera to detect, locate, and ultimately land on a target of circles of known radii. Methods of image processing for circle/ellipse detection are presented. As well as determination of position relative the detected target using known properties of the camera. We discuss our controller, and present results of testing on a Bebop Quadrotor.**

I. Introduction

In this project, we are given the task of landing a Bebop quadrotor on the center of target made of black and white concentric circles. We are provided with a stereo Duo3d camera, which we mount on the nadir-facing side of the drone so that it is as in line with the body fixed z-axis as much as possible.

Using image data from the downward facing camera, we use the OpenCV [1] to locate the the target in the image frame and find the center of the target in pixel coordinates. We discuss the process of deducing the location of the target in real world coordinates using the processed image data. Finally we implement a controller and provide real-world test results of landing the drone on the target.

As per usual, unless otherwise mentioned all code is implemented using ROS [2] and Python. Testing is done using the Bebop Autonomy [3] package.

II. Image Processing and Circle Finding

Due to the high contrast of the black-and-white bullseye target, we were able to perform a simple intensity thresholding on the grayscale image returned from the downward-facing Duo camera to create an accurate binary mask. This thresholding was done by taking the average of the grayscale values above a given threshold value, then setting any pixels that had a value below this shifted average to zero, and any pixels above or equal to the shifted average to full intensity. Figure 1 shows the intensity values for a sample raw image of the target from the Duo camera, note the large number of pixels with the maximal intensity value; by simply setting pixels below the maximal value it was possible to isolate the majority of the target whitespace.

Two separate methods were then used to determine the location of the bullseye target in the image based on the binary mask. The first utilized OpenCV's built-in Hough circle fitting to attempt to isolate the any of the bullseye concentric

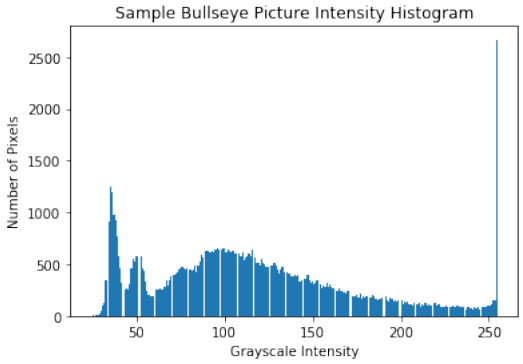

Fig. 1. Grayscale intensity histogram for a sample bullseye target image.

circles. The parameters for this fitting were set to be strict and typically would only return a single circle centered at the target. It was then trivial to export the center of the circle as the target location. In the event that no circle was returned from Hough circle fitting, an alternative method was used to provide direction to the quadcopter. Built-in functions in OpenCV were used to first obtain all contours in the binary image. Small contours were removed, and the remaining contours were processed to fit bounding rectangles to them. The largest rectangle was then isolated, and the center of the whitespace in that rectangle was found. This centerpoint was then exported from the image processing function as the location of the bullseye.

Hough circle fitting was typically more accurate and less noisy, but would occasionally have periods when no circles could be found. The implementation of the whitespace centerpoint allowed us to continue to provide the quadcopter with points for navigation during these periods until the target could be accurately reacquired. The image processing method is shown in Figures 2, 3, and 4.

III. Pose Estimation

Ultimately, we found that the simplest approach was the best approach for providing target position relative the quadrotor. In particular, the radius of the circle returned by the image processing algorithm was too noisy to reliably get major and minor axes for an ellipse, or even a reliable radius of the circle. However, using just the computed center of the circle,

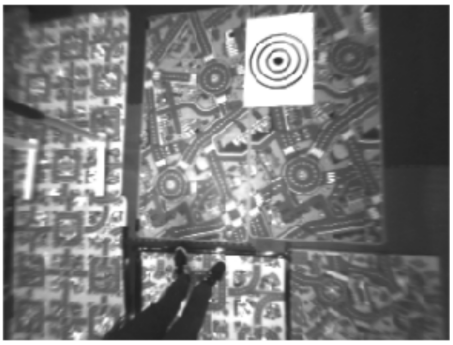

Fig. 2. Raw Duo image sample.

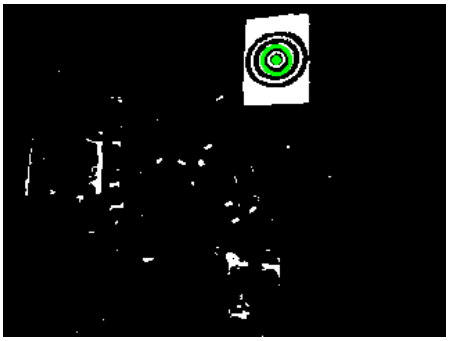

Fig. 3. Hough circle fitting on masked image.

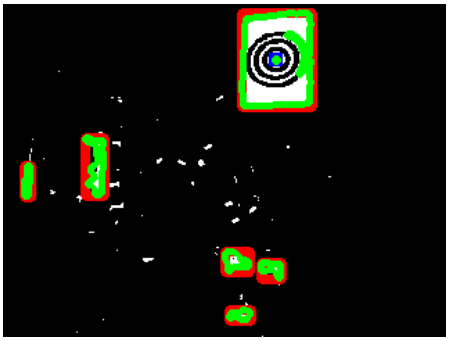

Fig. 4. Contours, bounding rectangles, and whitespace center on masked image.

we are able to get reliable guesses for the *direction* of the target. Ultimately, we are able to rely on the Bebop's own altitude estimation for Z, then use basic trigonometry and similar triangles to produce a heading.

Let f be the focal length of the camera, (c_x, c_y) be the center of the projected ellipse in pixel coordinates. And let θ_x be the angle formed by $[0,0,0]$, $[0,0,f]$, and $[c_x,0,0]$. Likewise, let θ_y be the angle formed by $[0,0,0]$, $[0,0,f]$, and $[0,c_y,0]$. See figure 5.

Then, rather than computing the full estimated position, we only compute the angles θ_x and θ_y as

$$\begin{bmatrix}\theta_x\\ \theta_y\end{bmatrix} = \begin{bmatrix}\arctan 2\,(c_x, f)\\ \arctan 2\,(c_y, f)\end{bmatrix}$$

However, in this case we must deal with the reality of the

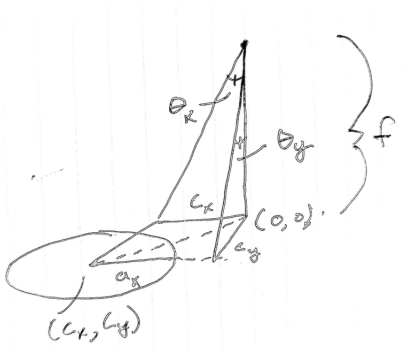

Fig. 5. Projection of the ellipse in the image plane, together with the center of the image frame and the focal length.

distortion of the camera. Using the focal lengths f_x, f_y, and the image center u_x, u_y returned by Kalibr, we can compute the physical world values of θ_x and θ_y as

$$\begin{bmatrix}\theta_x\\ \theta_y\end{bmatrix} = \begin{bmatrix}\arctan 2\,(c_x - u_x, f_x)\\ \arctan 2\,(c_y - u_y, f_y)\end{bmatrix}.$$

Where (c_x, c_y) is the center of the circle in image frame coordinates. Ultimately, we are able to accomplish the task of landing on the target using only these two values, which is discussed further in the Controller section.

After computing the angles θ_x and θ_y the detection node sends these values in an array to the navigation node. When the navigation node receives these values it uses the attitude from the last odometry update to project the position of the target on the ground plane using the equations below.

$$\begin{aligned} R_{att} &= Rot(\phi, \theta, \psi)\\ R_{cam} &= Rot(\theta_x, \theta_y, 0)\\ V_{proj} &= R_{att} * R_{cam} * [0,0,1]^T\\ t &= altitude / V_{proj}[2]\\ X_{bullseye} &= [bebop_x + V_{proj}[0] * t, bebop_y + V_{proj}[1] * t, 0] \end{aligned} \quad (1)$$

These prediction were input to the filter designed for the gate traversal in project 3a to get a final filtered result. The filter used a moving average of the last ten measurement while discarding new measurements whose error was outside a defined threshold.

IV. Controller

The control of the drone used the same state machine implementation as project 3a to maintain the flexibility and ability to extend the drones capabilities to longer and more complex tasks. The navigation process was composed of a total of 5 states. As the time was recorded from takeoff to landing the first and last state were processes to takeoff and land. Each state would send the takeoff or land command until the bebop had registered it was in a state of hover or landed respectively.

Between these processes the states composed of a point navigation process, a hold position process, and a bullseye

navigation process. The point navigation would guide the vehicle to the best estimate of the target with an altitude of 3 meters. The altitude was set high in order to get a wide coverage with the down facing camera. At that altitude the camera was able to see the majority of the test area so we were confident that we would be able to see the target. The point navigation ends when the bebop has reached the desired view point, at which point it moves to the next state where it is commanded to hold its position. This state waits until a target position is confirmed before it enters the final navigation state and moves towards the target. The bullseye navigation is based on the point navigation function with a small change to the altitude control. Instead of the desired altitude being the final desired position of the target, it is a function of the lateral error as defined below

$$Z_{des} = Min\_altitude + Lateral\_error * 1.5 \qquad (2)$$

This projected a cone upwards from the target so if the lateral error was too great the vehicle would raise to keep the target in the field of view and if it was directly over the target it would lower to its minimum altitude. The exit condition for this state used a check on the altitude, lateral error and lateral velocity. If the altitude was within a threshold of the minimum atitude, the lateral error was below a defined error and the magnitude of the lateral velocity was below another defined threshold the state would progress and a land command would be issued.

V. Videos

1) Bullseye Landing: https://youtu.be/VdHa78PFRbo
2) Bullseye Duo: https://youtu.be/0qzBGMySmg8
3) Bullseye Rviz https://youtu.be/LC$_4EnzWDuw$

VI. Important Lessons Learned

A. Limitations of Up board processing power

One of the biggest issues we faced on this project was managing the processes running on the UP board. During testing we noticed consistent overshoot of the target. The vehicle tended to oscillate, never being able to lock in on the target. We finally diagnosed the problem as latency between the image processing and the navigation functions. The navigation functions were receiving the data about the target with about 2 seconds of latency, so by the time the navigation reacted to the sensor information in a certain location the vehicle had already moved a significant distance.

We fixed the problem by moving all the processes onto the board to minimize the amount of data being sent between different processor and ended any extraneous processes such as recording or displaying live data. This solution was enough to allow the vehicle to complete its assignment but did not fix the root of the problem. Going forward we will need to identify the process that is either maximizing the processing power of the board or taking more bandwidth then the board can allow.

Acknowledgment

The authors would like to thank the professors for this course, Nitin J. Sanket and Chahat Deep Singh, as well as Dr. Inderjit Chopra.

References

[1] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[2] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[3] "bebop_autonomy - ros driver for parrot bebop drone." [Online]. Available: https://bebop-autonomy.readthedocs.io/en/latest/index.html