Camera Tracking Team

Final Project

Duseok Choi, Wren Sakai, Jacob Sickafoose

ECE 163: Introduction to Small Scale UAV Theory and Practice
May 30, 2021

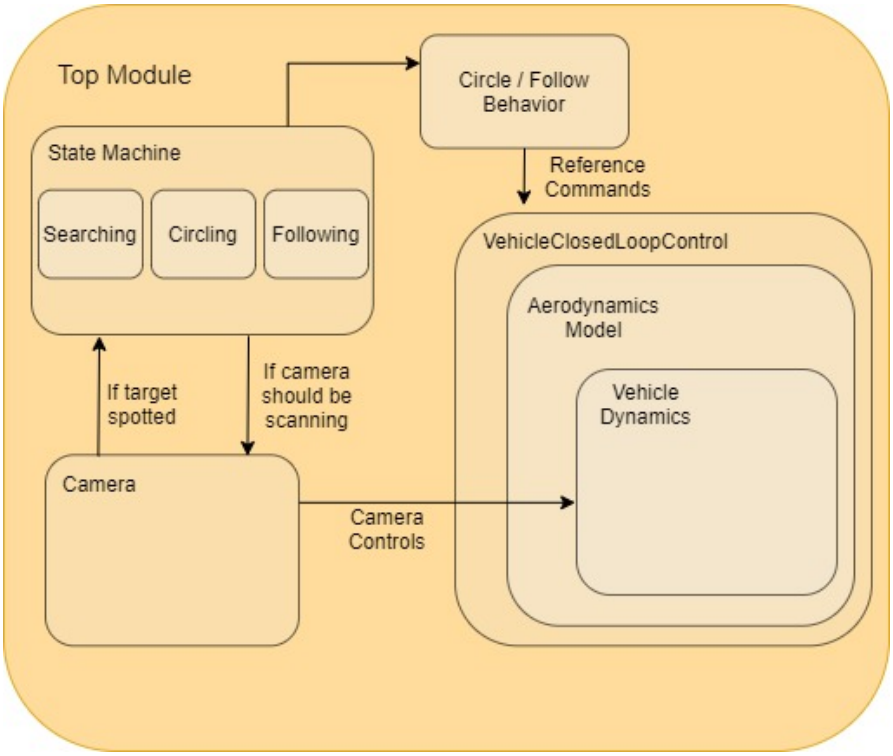
Contents

1 Introduction and Description

One potentially productive application of a UAV would be the search and observation of a target somewhere on the ground. This functionality could be potentially useful for search and rescue, situational awareness, or security applications. Thus, our project is to create modules which allow the UAV to search for a ground target using minimal information, lock onto that target with a camera, and then begin tracking the target as it moves. We would look to keep the previous simulator modules largely intact, but add modules to represent the behavior of a two-axis camera gimbal, as well as UAV behavior for searching and tracking a ground target. A successful module will be able to search for, and acquire a target under a wide variety of different circumstances, including a moving target, and a randomly placed target. Additionally a successful module should have very efficient searching behavior and take as little time as possible to acquire the target

2 Methods Overview

The modifications to the simulator will be limited to the addition of two new modules. The first of these modules will be our top level control module, the second will control the behavior of our onboard camera and gimbal. Our rough design goals are outlined in the following diagram:



The plan is to maintain a state machine inside our top module. This state machine will keep track of whether we are searching for a target, circling a stationary target, or following a moving target. This state machine will update according to input information from the camera, as well as the aircraft's positioning in relation to the target. The idea is to start the UAV out in searching mode, with a general idea of where the target is located. The UAV will then set the course angle to point towards this general location and continue forward at the minimum speed while the camera scans the area. Once the target is located, the state machine will click into follow mode, to move within a certain distance from the target. This distance will be the circling radius and when it's within this distance, the UAV will begin to circle and maintain the radius.

We will also have logic to control the plane's circling/following flight path and pass reference commands into the existing `VehicleClosedLoopControl` module. It remains to be determined whether or not the closed loop module will require modification as well. Ideally, the UAV will be able to hold a constant radius while circling around the target, despite wind blowing it off

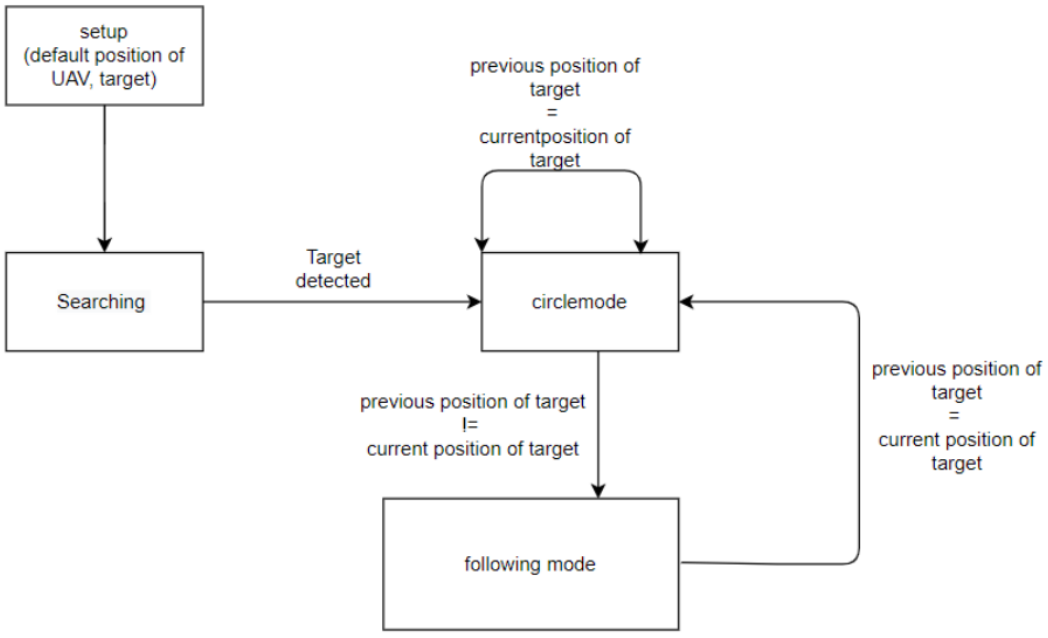
course. While following, the aircraft will also need to maintain a constant follow distance by controlling it's speed.

Finally, our camera module will control the behavior of the camera. The camera will be drawn inside of `VehicleDynamicsModel`, so our camera module will need to provide the angle to draw the camera at. If the state machine is in searching mode, the camera will begin it's spiralling scanning of the ground in search of a target. If the target is detected by the camera within the camera module, it will pass this information out to the state machine to advance the state.

3 Module Implementation Details

3.1 Top Level

The overarching module will keep track of the data being passed into the camera module and the `VehicleClosedLoopControl`. It will also contain the implementation of the following state machine:



When in our `Searching` mode, we know the Δ between the current course angle, and our target with a certain amount of error. In `Searching` mode, the top level module will command the course angle to be

$$CommandedCourseAngle = TargetAngle - CurrentAngle$$

Once our aircraft has flown within the distance we wish to circle the target at, it will switch into `Circle` mode. This mode will hold the craft at our specified radius using the Beard equations. From `Pg. 183, eq. 10.12` and `eq. 10.13`. We define the following variables:

$\lambda = 1$, for clockwise rotation
ρ = orbit radius
d = radial distance from the desired center of the orbit to the MAV
χ_d = desired course angle, $d \to \rho$
χ_0 = desired course angle when the MAV is located on the orbit
χ_c = course command orbit
κ_{path} and κ_{orbit} = constant input, we can change to play with how smooth it's transition to the path is

$$\chi^0 = \phi + \lambda\tfrac{\pi}{2}$$

Step 1
If target is detected, $\rho = d$

$$\chi_d(d-\rho,\lambda) = \chi^0 + \lambda tan^{-1}(\kappa_{orbit}(\tfrac{d-\rho}{\rho}))$$

If $d \to \rho$

$$\chi_d \approx \chi^0 + \lambda\tfrac{\pi}{2}$$

Step 2
Find χ_d from `Step 1`
If $\chi = \chi d$

$$W = \tfrac{1}{2}(d-\rho)^2 \text{ Lyapunov Function}$$

$$\dot{W} = -V_g(d-\rho)sin(tan^{-1}(\kappa_{orbit}(\tfrac{d-\rho}{\rho})))$$

Implying $d \to \rho$ asymptomatically (Camera on `Circle` mode)

$$\chi^c(t) = \phi + \lambda[\tfrac{\pi}{2} + tan^{-1}(\kappa_{orbit}\tfrac{d-\rho}{\rho}))] \text{ Equation } 10.12$$

If the target is detected in range of ρ, the camera enters `Circle` mode. $d(altitude)$, the distance from the camera, must also remain greater than 0. If $d < 0$, the camera would hit the ground. If there is a change in position of the target in range of ρ, we enter `Following` mode.

Algorithm 4 Circular Orbit Following: $[h^c, \chi^c]$ = followOrbit $(\mathbf{c}, \rho, \lambda, \mathbf{p}, \chi)$

Input: Orbit center $\mathbf{c} = (c_n, c_e, c_d)^\top$, radius ρ, and direction λ, MAV position $\mathbf{p} = (p_n, p_e, p_d)^\top$, course χ, gains k_{orbit}, sample rate T_s.

$h^c \leftarrow -c_d$
$d \leftarrow \sqrt{(p_n - c_n)^2 + (p_e - c_e)^2}$
$\varphi \leftarrow \texttt{atan2}(p_e - c_e, p_n - c_n)$
while $\varphi - \chi < -\pi$ **do**
    $\varphi \leftarrow \varphi + 2\pi$
end while
while $\varphi - \chi > \pi$ **do**
    $\varphi \leftarrow \varphi - 2\pi$
end while
Compute commanded course angle using equation (10.13).
return h^c, χ^c

Follow/Search Mode:
`Pg. 179, eq. 10.7` and `Pg. 180, eq. 10.12`

$$\dot{e}_{py} = V_g sin(\chi - \chi_q)$$

e_{py} = path error expressed in the path frame

$$\chi^c(t) = \chi_q - \chi^\infty \tfrac{2}{\pi} tan^{-1}(\kappa_{path} e_{py}),$$
$$\chi^c(t) = \chi_q - \chi^\infty \tfrac{2}{\pi} tan^{-1}(\kappa_{path} e_{py}(t))$$

χ_q is from equation `10.1`

If $\chi_q > 0$, MAV turns right to align with the waypoint path
If $\chi_q < 0$, MAV turns left to align with the waypoint path

The concept of the `Searching` mode, and the `Following` mode is very similar. `Follow` mode should include some implementation for position of target which search mode doesn't have the information for.

Algorithm 3 Straight-line Following: $[h^c, \chi^c]$ = followStraightLine $(\mathbf{r}, \mathbf{q}, \mathbf{p}, \chi)$

Input: Path definition $\mathbf{r} = (r_n, r_e, r_d)^\top$ and $\mathbf{q} = (q_n, q_e, q_d)^\top$, MAV position $\mathbf{p} = (p_n, p_e, p_d)^\top$, course χ, gains $\chi_\infty, k_{\text{path}}$, sample rate T_s.

Compute commanded altitude using equation (10.5).
$\chi_q \leftarrow$ `atan2`(q_e, q_n)
while $\chi_q - \chi < -\pi$ **do**
  $\chi_q \leftarrow \chi_q + 2\pi$
end while
while $\chi_q - \chi > \pi$ **do**
  $\chi_q \leftarrow \chi_q - 2\pi$
end while
$e_{py} \leftarrow -\sin \chi_q (p_n - r_n) + \cos \chi_q (p_e - r_e)$
Compute commanded course angle using equation (10.8).
return h^c, χ^c

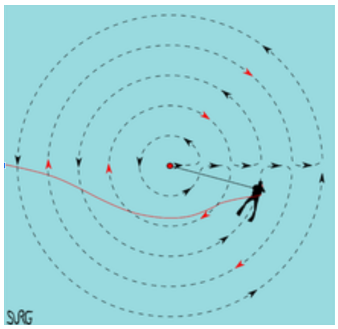
This module will control the behavior of the camera, and will pass the current state into the UAV control module.

3.2 Search Mode

Before the UAV begins to circle and follow a target it must first search for and locate said target. In order to simulate realistic conditions, the NED coordinates of the target will not be provided to the UAV for the purpose of setting a heading. Instead, the UAV is given a "rough" angle between its current heading and the target. This rough heading will simply be the true heading as calculated with the NED coordinates of both vehicles, but with an added noise created by a random number generator. Thus the UAV is only roughly aware of which direction it must travel in. In this case the UAV will set a heading based on the provided rough angle and pass that heading into the original Vehicle Closed Loop controls from previous labs. At this time the UAV will also begin the Camera search behavior until the camera has "detected" the UAV.

3.3 Camera

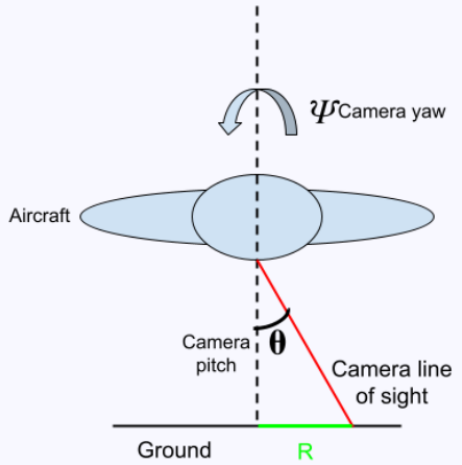
The camera module will create a class that stores the gimbal angles and manage the behavior of the camera during search mode, and target lock mode. Additionally, this module will pass camera data into a slightly modified vehicle dynamics model in order to draw a camera representation in the GUI, which will likely be a simple line or pyramid shape. While in "search" mode, the camera will operate independently of UAV motion, and begin conducting an expanding circle search shape with its viewing direction. This rough idea is shown below (note that the below diagram uses the motion of a SCUBA diver, where our camera's circle will be the point on the ground where the camera is looking).



In this procedure, once the camera hits a maximum radius, it will reset to point down from the UAV and begin to search again. This process will

repeat until the target is within the camera's view. At this point the camera will switch from "search" to "target lock" mode and stay pointing at the target. It will also output a flag so that the UAV control module can switch from "search" to "follow" state.

In order to properly control the camera, we will assume that there is some sort of passive stabilization device. Adding controls to always keep the camera yaw perpendicular to the floor is outside the scope of this project, so we will assume this happens on its own. In other words the yaw of the gimbal will always be assumed as perpendicular to the ground regardless of aircraft position.



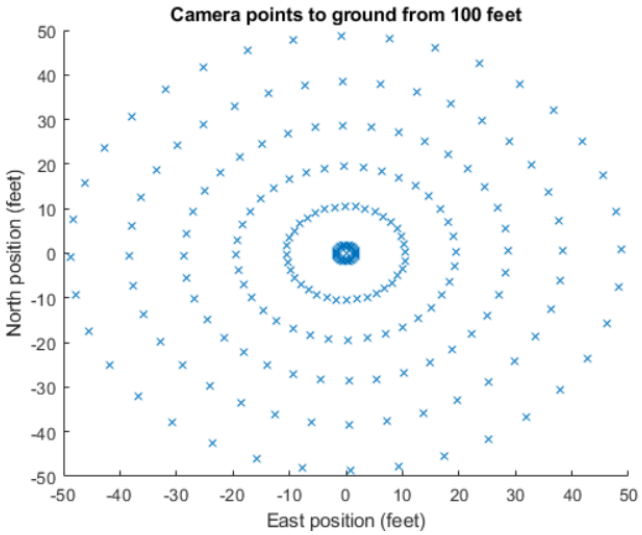
From the above diagram, we assume that our camera has a yaw rotation and a pitch rotation, thus to passively perform an expanding circle search we need to run a full yaw rotation for smaller increments of theta. The ground N and E coordinates of the camera line of sight can be found by finding a polar coordinate where R is the magnitude and yaw is used as the rotational value. R can be found using simple trig, no matter the yaw value, where pd is the altitude of the UAV.

$$R = -\rho d * tan(\Theta)$$

Now the polar coordinate can be converted into an N E cartesian coordinate of camera point from the center of the aircraft.

$$N = R * cos(\psi)$$
$$R = R * sin(sin(\psi)$$

This will produce a pattern that is mapped below. Note that the below figure represents a birds-eye view of the ground where (0,0) represents the point where the camera is located relative to the UAV.



The circular pattern can be achieved with similar code as outlined below except instead of using for loops, the circular pattern would simply update on each iteration of the top level simulation.

```
pd = -100
index = 0;

for i = 1:5:30
    for y = 1:10:360
        R = -pd*tand(i);
        X(index + y) = R*cosd(y);
        Y(index + y) = R*sind(y);

    index = index + 360;
```

In order to simulate the camera "seeing" the target a simple set of conditionals can be used to check if the camera line of sight is within a certain radius of the target.

4 Testing Strategies

Simulating this project is likely going to be done in multiple small chunks for each module. For the camera module, the equations used will likely be graphed on a simple 2D plane in order to visualize the path of the camera to assure it is working properly. Additionally, a point of interest will be added to assure that camera position and behavior adequately handles the presence of a target. Once simple 2D plots are made, the simulator can be run with the UAV fixed in the air or moving in a simple straight line. Thus the camera behavior can be tested in the simplest environment.

Next, the follow and search behavior of the UAV should each be tested on their own. Again, a 2D graph of the N and E position of the UAV can be made to validate the functionality of the modules while also having a very high level of control of all the little variables that define the UAV's motion and position. In the case of the search behavior, a point of interest should be added to the 2D graph in order to test the UAV's flight behavior in specific directions. At a minimum this would include a point of interest in all four rotational quadrants of the UAV to assure it banks towards the target from a variety of different positions. For the encircling behavior, similar tests can be performed in a 2D environment for a variety of UAV positions relative to the target. As with the search procedure, the encircling procedure should be able to properly track the radius of the target from a variety of different trajectories. Once the system was validated graphically, then it would be possible to test them in isolation in the simulator

In theory each of these modules should be able to work in isolation, so testing them separately will likely validate them for use in a single program. However it is likely that some debugging from the simulator may be necessary to assure all the components work well together. If the simulator is failing to run properly with each component combined it might be preferable to run the 2D simulation of the UAV position in python to troubleshoot the entire program.

5 Final Implementation

We will start the project simply trying to locate a target. Once locked onto, we hope to get the plane successfully circling overhead the target. If we finish with enough time, we will potentially move on to moving the target vehicle, and commanding the UAV to follow the target. If we manage this, our long-shot stretch goal would then be to calculate how the camera body affects the dynamics of our aircraft. Until this step, we will treat the camera as though it is both infinitely small and weightless.