MEAM 620 Project 3

Lou Lin, Mary Ibrahim, Gabrielle Merritt

May 8, 2015

Abstract

Methods

Control System

Estimating Flight States To have the Bebop fly any arbitrary trajectory in R^3, a position based controller is necessary. In order to implement a position based controller, we must be able to estimate the position and orientation of the quadrotor. Since the video transferred through wifi is not reliable, using epipolar geometry related technique on the computer's side is impossible. Luckily, the Bebop drone already have onboard attitude and velocity estimator using a combination of inertial sensors and optical flow. Assuming the sensor fusion onboard is reliable, it is reasonable for me to just pull the information from the drone registering my own callback functions in the API.

```
ARCOMMANDS_Decoder_SetCommonCommonStateBatteryStateChangedCallback
(batteryStateChangedCallback, deviceManager);

ARCOMMANDS_Decoder_SetARDrone3PilotingStateFlyingStateChangedCallback
(flyingStateChangedCallback, deviceManager);

ARCOMMANDS_Decoder_SetARDrone3PilotingStateAltitudeChangedCallback
(altitudeCallback, deviceManager);

ARCOMMANDS_Decoder_SetARDrone3PilotingStateAttitudeChangedCallback
(attitudeCallback, deviceManager);

ARCOMMANDS_Decoder_SetARDrone3PilotingStateSpeedChangedCallback
(velocityCallback, deviceManager);
```

The velocity measured is already set in the world frame. Since the Bebop can directly measure altitude, all it's left to do is to numerically integrate the velocity in the x and y direction to obtain my position. We used forward Euler approximation to calculate our position, and the result were reasonable.

Actuator Inputs We are able to send 4 different inputs to the Bebop drone. The

`ARCOMMANDS_Generator_GenerateARDrone3PilotingPCMD()`

function allows me to send desired roll and pitch angles, the yaw rate, and the velocity in the world Z direction. The inner loop from propeller inputs to the desired roll, pitch, yaw and velocity in Z is already close onboard, and we have no access to it unless we hack the firmware, so we will just assume that the inner loop is stable and reacts fast to the reference signals. It is interesting that the API has two different definitions of the body frame. The reference frame for the actuator inputs has a rotation of π about the X axis of the sensor reference frame, therefore the controller needs to be modified accordingly.

Backstepping Control of position Leverging from project 1 phase 2, we can just take a controller in this form and expect the errors converge to zero exponentially.

$$aicommanded = a_{idesired} + K_{di}(V_{idesired} - V_i) + K_{pi}(r_{idesired} - r_i)$$

Then we can linearize about hover and use the commanded acceleration in the X and Y direction to find the desired roll and pitch angles. As long as the onboard attitude controller make these angles converge to the desired values much faster than my position controller, the quadrotor will perform reasonably.

However, the reference frames are defined differently. In the sensor frame, the Z axis points down, and the Z axis points up. In order to compensate for this inconsistency, I have to negate the pitch angle and the desired velocity in the Z direction.

$$\phi_{commanded} = g(a_{xcommanded} * \sin\psi - a_{ycommanded})\cos\psi$$

$$\theta_{commanded} = -g(a_{xcommanded} * \cos\psi + a_{ycommanded})\sin\psi$$

Note that I can only command desired Z velocity instead of a thrust. I must modify the control law for the commanded Z velocity (such is named gaz in the API).

$$gaz = V_{zdesired} + K_{pz}(r_{zdesired} - r_z) \tag{1}$$

Trajectory Generation

Spline Generation for Helix

Code Architecture

Results

1

Graphs