Ground Station For UAV Autopilot

Salvador Vazquez
svazquez@ucsc.edu

Autonomous Systems Lab
Computer Engineering Department
Jack Baskin School of Engineering
University of California-Santa Cruz
CA 95064

SURF-IT 2008

Faculty Mentor: Gabriel Hugh Elkaim
elkaim@soe.ucsc.edu

Table of Contents

Introduction……..……..1

Approach……2

Latitude and Longitude……...2.1

Zoom……...2.2

Setting up the Map…….……..2.3

Createmap…….....2.3.1

Moving the Map……2.4

Setting Local Altitude…….....2.5

Disabling Waypoint Adding……2.6

Adding Waypoints……2.7

Converting Pixels into Degrees……...2.7.1

The Done Button……2.8

Future Work……3

References……4

Acknowledgements……...5

1. Introduction

The overall project goal is to create a robust unmanned aerial vehicle autopilot from the ground up. The ground station is used to control the aircraft which employs the autopilot. The ground station is created to be simple enough that anyone without programming experience can use. The ground station has a simple user interface with buttons that are clearly labeled. The user moves a map to a desired location, sets the local altitude of the location, sets height above local altitude for aircraft and then clicks the map to set waypoints.

2. Approach

The ground station interface was created in Matlab (version R2007a) for simplicity of programming. The structure of the program waits for user interaction before anything happens. The main function simply creates all GUI's and texts and sets the callback to different functions. The map is retrieved from the Google static maps data base and cached. The approach for each callback function is as follows:

2.1 Latitude and longitude

There are two edit text boxes which make up the values for the latitude and longitude in decimal degrees. There is no callback for these edit boxes, they are simply a way to store and show the user what the latitude and longitude of the center of the map is at any given time. They are stored as strings in the edit boxes but converted to numerical values when needed.

2.2 Zoom

The zoom bar has a minimum of 0 and maximum of 17. Google maps can go up to zoom level 21 but only in limited areas. In order to see all places at high zoom levels the maximum is set to 17. The call back for the zoom is set to the same as the set up map callback.

2.3 Setting up the map

The setup map button gathers the latitude, longitude, and zoom then goes to a function called createmap.

2.3.1 Createmap Function

The createmap function is a separate function from the callbacks so it can be used in multiple functions. This function is where the interface actually gets the map from Google. Google static maps API is used to get the map image. In order to use the API one must have the latitude and longitude of the center of the map, the zoom level, and a Google API key. The API key is retrieved from Google after

registration. Those three things along with the size of the image desired, in pixels, is put into a URL to go to a web page that has that image. The format of the URL is http://maps.google.com/staticmap?center=(latitude),(longitude)&zoom=(zoom)&size=(width)x(length)&key=(API key). The createmap function grabs the parameters needed and places it into this format as a string. Once this string is completed, the function does an image read from this address and saves it as an image onto the current axis. Next it sets the callback for when a user presses on the image to the function addpoint, which adds the point and will be described later.

2.4 Moving the map

There are four buttons that move the map up, down, left and right. The callback function for each moves the same just in different directions. The latitude and longitude are converted into numerical values so they can be worked with. Based on calibrations I did, if the zoom level is at a certain number it will add between 0.001 degrees and 10 degrees in the direction the user pressed. The numerical values are then converted back to strings and stored in the edit boxes. Then the function goes to createmap where it creates a new map since the latitude and/or longitude has been changed.

2.5 Setting local altitude

Since there was no time to find a way to obtain the local altitude for every coordinate the user must input the local altitude, assuming he knows it. When the button is pressed a new figure comes out prompting the user to type in the local altitude. The figure is set to modal which means it is always on top of the main figure, since the new figure is so small the user can easily press the main window to hide the small window. The local altitude is typed into an edit box then converted into a numerical value then saved as a global variable. Once the local altitude is set, a display text shows the user what he entered. The problem with this method of getting local altitude is that if the user zooms out enough there may be multiple local altitudes on the map and can be potentially risky for the aircraft. Future revisions should use a different method for this.

2.6 Disabling waypoint adding

The user might not always want to add waypoints, so there is a toggle button that allows him to enable and disable adding new waypoints. When the toggle button is down the value is equal to one and when up the value is zero. The callback simply sets the values to one or zero and labels the button either to “clicking enabled” or “clicking disabled”. When the image is clicked it goes to the addpoint function where it checks the status of the toggle button. If the toggle button is down then it leaves the addpoint function and hence no waypoints are added.

2.7 Adding waypoints

When the user clicks any point on the map image it goes straight to the addpoint function. Everything that has to do with adding waypoints is done in this function or in the conversion functions. The first thing this function does is to check if clicking is disabled by checking the toggle button. If the toggle button is down then it returns. Next thing it does is add a full crosshair as a pointer. The first time the user clicks the screen it changes the pointer to a full crosshair and exits the function. The next time the user clicks the screen it removes the crosshair and goes through the function. Now it gets whether the user right clicked or left clicked and also where the position of the click was in terms of pixels. If the person right clicks and it is not within 5 pixels of another waypoint it increases the waypoint number count then does the conversion of pixels into degrees. If the user right clicks within 5 pixels of another waypoint it erases it, although the erasing is not working after adding the path lines between the waypoints.

2.7.1 Converting pixels into degrees

When converting from pixels into degrees the program goes into a function called wp2latlong. In this function the pixel origin is changed from the top left of the image to the middle of the image. Then uses an adjust function found online from a Google groups forum* post. The code was originally java but was converted into matlab code. The difference in the codes was just the zoom, he used maximum zoom at 21 I used 17 as maximum zoom. The offset is simply half the circumference of the earth in terms of pixels at the highest zoom level. These equations are just modified versions of Mercator projection equations found anywhere online. They are just modified to work with Google's tile system. Google uses a system where at the furthest zoom when the whole Earth can be seen there is one tile that is 256x256 pixels. As you zoom in once you get 2x2 tiles each one is 256x256 pixels, at the next zoom level 4x4 tiles each 256x256 and so on so each map on Google is 2^zoom x 2^zoom tiles. The main modification is changing the use of meters to pixels but the number of pixels in the Google image depends on the zoom level and the radius has to be in terms of pixels when the map is at the furthest zoom.

After the conversions are made into latitude and longitude coordinates the altitude from the altitude slider is added in and stored in a vector called llwaypoints. Next, the circles must be drawn on the map image itself. In order to do that we must place it on the latitude and longitude coordinate and not on the pixel coordinate of the initial click since the map may be moved at any given time. In order to do that the latitude and longitude coordinates must be converted into pixels from the latitude and longitude coordinates every time they are drawn. The function position is the exact inverse of the adjust function. The position function is called before the circles are drawn every time and the vector with the new pixel coordinates is stored in wppos. After that the lines that connect the dots are drawn, which will only happen when there are two or more dots. Then text is added next to each dot to keep track of each waypoint. The dots only get redrawn after the map is clicked again so if the user moves the map the dots will not show up again until the user clicks on the map again. Future work may want to create a separate

function to draw everything and call it every time the map is drawn so the waypoints show up on every movement of the map.

2.8 The Done button

The done button simply draws a line from the last waypoint to the first one and makes the first waypoint the last one, making the aircraft go back to its starting location. Then it disables clicking by changing the toggle button. The problem with doing this is that the toggle button can still be changed to “clicking enabled” and gives error after clicking on the map again. Future work may want to fix it by making the screen clickable again or by not allowing the user to ever click the map again by turning the toggle button into a text box. It is a choice for the developer.

3. Future Work

Future work includes finding a new way of adding the local altitude, having the circles be redrawn every time the map is created, fixing the done button, and testing on flighgear. There are data bases out there that have local altitudes for latitude and longitude coordinates but problems occur with how zoomed out the user is since he may be over several local altitudes. The circles can be redrawn by making a separate function that draws all circles and calling it when the map is redrawn. The developer must decide what to do in the done function. He can completely disable the clicking enabling/disabling toggle button by turning it into a text box or fix the problem of being able to turn the clicking on and off after the done button is pressed. Lastly, once the code is finished it should be tested on flightgear. Flightgear should be connected through UDP for simplicity and so the user can run both flightgear and matlab on the same computer. Along with the source code for the ground station there is another matlab file folder called matlab-fg, which was downloaded on the mathworks file exchange. The author has well commented code on how to connect matlab to flightgear. The comments are in Spanish but can easily be translated into English.

4. Refreences

“Google groups” Google
<http://groups.google.com>

*“Google Static Maps API” Google
<http://code.google.com/apis/maps/documentation/staticmaps/>

5. Acknowledgements

Gabriel Hugh Elkaim, my professor advisor. David Ilstrup and Mariano Lizarraga, who constantly helped me in the lab. The SURF-IT program, especially Richard Hughey and Colt Hagen, for making this project possible for me. The NSF for their funds to the SURF-IT prgram