Gesture Controlled UAV Proposal

Ben Schreck and Lee Gross
10/29/2014

1 Overview

There are currently two types of unmanned aerial vehicles (UAVs): autonomous aircrafts and remotely piloted aircrafts. Remotely piloted aircrafts are tough to control using handheld remotes. We propose an intuitive approach to controlling these UAVs, using hand gestures rather than remotes.

Our project will allow a user to control a quadcopter (a type of UAV) using hand gestures. The operator will make these gestures in front of a Microsoft Kinect device, which can sense both colored light and depth of field. Using a Kinect allows us to define more complex gestures that take advantage of the distance of the operator's hands from the camera. Furthermore, they may allow us to discard the typical colored gloves usually worn by 6.111 students that enable easy hand-tracking by color matching.

When an operator makes a correct gesture, the FPGA will classify it, and send the appropriate signal to the quadcopter's remote controller, which will then send an infrared signal to the quadcopter commanding it to perform a particular action.

Due to time constraints, we chose to limit the number of ways the user can control the quadcopter. Manually holding a quadcopter's altitude constant without complicated controls and avionics is a challenge on its own. Therefore, we decided it is best to only allow the user to turn the quadcopter on and off and control its elevation, leaving out the ability to turn left or right.

2 Implementation

Our design is composed of two main parts: gesture recognition and quadcopter control. The gesture recognition component is responsible for capturing information about the operator in the form of video and depth sensing, recognizing the operator's hands, and then mapping the operator's hand movements to predetermined gestures. The quadcopter control component is responsible for interfacing with the infrared transmitter that controls the quadcopter and sending signals that correspond to the controls indicated by the gestures.

2.1 Gesture Recognition

The gesture recognition component is composed of a five step pipeline shown in figure 1. The first step consists of capturing hand gestures using a Kinect camera. The subsequent steps allow us to detect the location of the hands, transmit the data to the FPGA and determine which gestures correspond to the operator's actions.

2.1.1 Kinect Camera Input

We will be using a Kinect camera to capture the user's hand gestures. The Kinect camera affords us a 3D representation of the space: it provides us with an RGB image stream as well as a depth stream.

The Kinect camera will be connected to a PC and a few computations (discussed in sections 2.1.2 and 2.1.2a) will also be computed on the PC so that we can can use less bandwidth when sending data to the FPGA.

2.1.2 Hand Classifier

This module finds the center of mass of each of the operator's hands. It takes in depth data from the Kinect, and potentially color stream data that has been mapped from RGB (Red, Green Blue) to HSV (Hue, Saturation, Vibrance) color space. These two options are discussed in the following subsections.

2.1.2a Using Kinect's Depth Stream

The depth stream provides with a depth value for each x,y coordinate in the image, corresponding to how far away that particular real world location is from the camera. Ideally, we would be able to capture all necessary information about hand location solely from the Kinect's depth stream. Scanning the depth stream for clusters of coordinates within a particular range will allow us to identify hands. However, in this case we may have difficulty keeping track of the left and right hand as discrete units.

There are two solutions to this problem. The first idea is to divide the viewport into two horizontally adjacent portions. If the majority of a hand is detected on the left side of the viewport, it will be classified as the left hand, otherwise it will be classified as the right hand. The operator is then in charge of keeping each of his or her hands on either side of viewport.

The other option is to have the operator wear different colored gloves on each hand. To take advantage of this option, we will have to use the Kinect's RGB camera as well, described in the following section.

2.1.2b Using Kinect's Color Stream

The color stream's main purpose is to distinguish between left and right hands. To facilitate this, we first use a color space converter to convert from RGB to HSV data. Then, knowing the correct hue associated with the operator's colored gloves, the colorstream submodule of the hand classifier averages the coordinates of the pixels that are of the correct hue value. For example, it will look for all pixels of the color red (meaning they are within a certain range of values) sum the x and y coordinates, and then find the average. This leaves us with the center of mass of each hand.

2.1.4 USB Adapter

This module transmits data from the kinect-connected PC to the FPGA. The information contained in this data will be used to classify gestures, so it will contain the coordinates and depth of each hand of the operator and pass them to the gesture recognition state machine on the FPGA.

2.1.5 Gesture Recognition State Machine

This module will determine which gesture the operator indicated. It will receive the coordinates of the location of the hands and output one gesture from a set of predetermined gestures. Each gesture will have a state machine that checks whether that gesture was the most likely to have been indicated by the operator. We plan to have one gesture that will turn the drone on or off and one gesture that will allow the drone to levitate. This second gesture is actually a set of similar gestures, that each output a particular number, corresponding to how high the operator wants the drone to levitate.

2.2 Quadcopter Control

Quadcopter control entails the conversion of representations of gestures inside the FPGA to signals that the system will feed to the quadcopter remote controller. The controller itself contains analog dials that alter four aspects of quadcopter control: roll, pitch, yaw, and thrust. The definition of all of these terms is somewhat unnecessary for this proposal, since we will only concern ourselves with thrust. The other terms deal with moving the drone forward, backward, and sideways in space. Thrust deals with up-and-down motion only.

We have yet to take apart the controller and understand how these controls are delivered to the quadcopter, so we do not know how we will interface with it. However, assuming the controls are simple analog voltage values, we will simply take apart the control, and wire the thrust input to a digital-to-analog converter connected to the FPGA. Similarly, we will wire digital on/off signals to the on/off switch on the controller. If the controller has pushbuttons, we can replace the pushbuttons with a relay and control the relay with the labkit.

2.3 Additional Features

If time allows, we're also considering adding a couple of additional features. We have a couple of ideas ranging from feedback to the user to gamifying the quadcopter control (the game would be to see how long a user could keep the quadcopter at specified altitudes). Below is a list of features that we will consider to add along with a difficulty measure.

Feature	**Difficulty**
Camera view from quadcopter	5
Feedback of recognized gestures on a computer monitor	3

Additional gestures	2
Gamifying the quadcopter	4

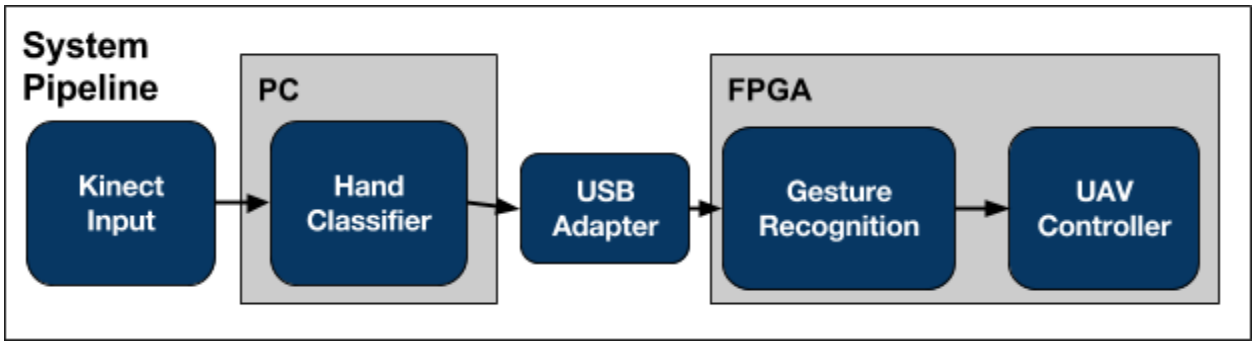


Figure 1 displays the pipeline of the system as well as well each module is located.

3 Testing

We plan on testing each module individually, and then testing the interconnects between them in stages. Many of these tests can be run in parallel.

3.1 Kinect Camera Input Testing

The Kinect input module needs to be tested to see if color and depth stream input can be received by the PC from the Kinect.

3.1a Hand Classifier Testing

After receiving input from the Kinect, we need to test our ability to find the left and right hand of the operator from the Kinect input, on the PC. During this test, we will decide whether to use only the depth stream, or both the depth and color stream, based on how difficult it is to distinguish between the left and right hands. The output of this test will determine whether the operator needs to wear colored glove.s

3.2 USB Adapter Testing

This module can actually be tested in conjunction with 3.1 and 3.1a, because we only need to see if we can transmit data from the PC to the FPGA.

3.3 Gesture Recognition State Machine Testing

This module can be tested independently, because its input can be mocked with fake data corresponding to location of the hands. Here, we need to test whether particular movements of the center of mass of hands can be classified into different gestures.

3.4 Quadcopter Control Testing
This module can also be tested independently, again because its input can be mocked with fake data, this time with signals about how to control the quadcopter. Here, we need to test whether particular control signals can be relayed to the quadcopter.

3.5 Integration Testing
We need to test how each module interacts with each other module. We will accomplish this is in stages.

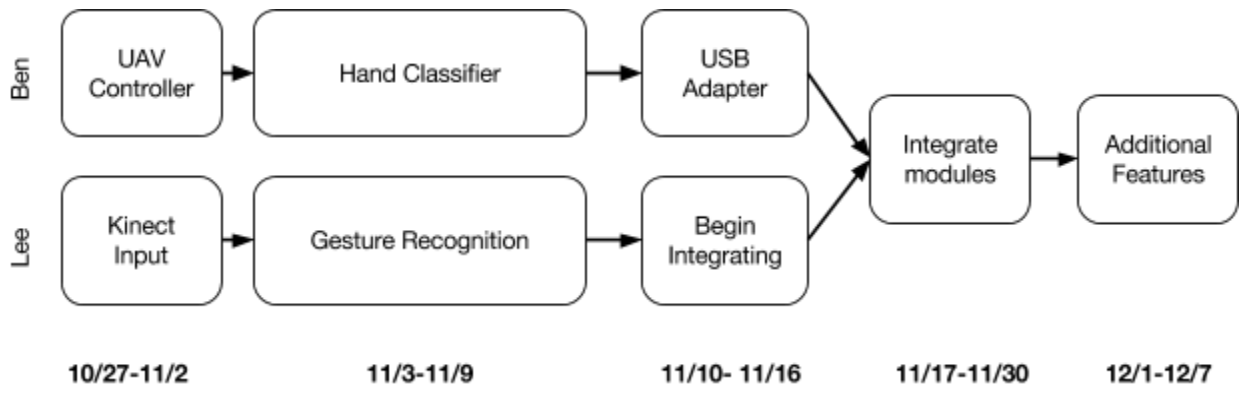
3.5.1 Kinect Input to USB Adapter Testing
Here, we test whether we can take in Kinect input, determine hand locations, and pass that information to the FPGA.

3.5.2 Kinect Input to Gesture Recognition State Machine Testing
Here, we test whether we can take in Kinect input, determine hand locations, pass that information to the FPGA, and classify hand motions as particular gestures.

3.5.3 Gesture Recognition State Machine to Quadcopter Testing
Here, we test whether we can classify gestures from faked data, and feed that to the quadcopter control unit to actually control the quadcopter.

3.5.4 Kinect Input to Quadcopter Control Testing
Here, we test the entire system, making sure that Kinect input can be used to actually control the quadcopter.

4 Timeline
We will begin by first working on the two external modules: interfacing with the drone controller and getting the depth and VGA values from the Kinect. We will then implement the color space converter as well as the center of mass calculation in C++ on a PC in parallel with the gesture recognition module in verilog. Once we built these, we will implement the USB adapter and finally, combine the modules. If time allows, we will also implement additional features. A diagram showing the parallel flow is shown below.



This diagram is an approximation of our schedule but we realize that issues always arise so we've planned to finish the project a week early and assigned extra time to each module. If we finish the goals early, we plan on implementing additional features. The feature we chose to implement will depend on the amount of time we have.

5 Resources

Below are the two items that we need for the project.

Item	Cost
Kinect camera	Free - borrow from friend
UAV	$60