Christoph von Braun
12/6/15
Design HW2: Components & Interfaces

Major components:

- Object recognition on webcam
- Generate 3D coordinate from 2D coordinates

Minor components:

- User interface for displaying program status
- Text logging of all identified coordinates
- Communication between object recognition and coordinate generation

Algorithms used:

SURF, or Speeded Up Robust Features, will be the main algorithm used in the object recognition component. This has not been designed or implemented by me, and instead is called via OpenCV. It requires a still image, and a video feed, and attempts to match the two. I will be using a still photo of the drone, or an identifying label of the drone, to try and detect it flying in my monitored space. The process of calling SURF is non-trivial. First, features must be detected and extracted on the given still image. Once the camera feed is loaded, the same can be applied to every given frame from the video. A matcher is then used to align the patterns from the image to the patterns in the video. I then filter out what are considered to be poor matches, and draw and store what matches remain. As this is not the algorithmic component that I myself design, the associated pseudocode is quite simplistic. It consists just of API calls to OpenCV for the associated SURF functions.

```
detector.detect(still image)
extractor.extract(still image)
while(camera running)
      detector.detect(video)
      extractor.extract(video)
      matcher.match(still image, video)
      vector<DMatch> good_matches;
      for all matches
            if (good match criteria)
                  good_matches.add(match)
      if(good_matches resemble expected shape) //error correction
            draw good_matches
```

During the winter and spring months, I will be designing and implementing the algorithm that takes in *n* sets of 2D coordinates and generates one 3D coordinate. This will be an algorithm of my own design. As I have not yet designed this however, there is no applicable pseudocode to show. There is a general idea of how it will work, but it's not yet finalized.

External library:

The only external library I am using is OpenCV, which is an open source computer vision library with interfaces in C, C++, Python, and Java. I am using the C++ version. My usage of the library will be restricted to the object recognition side of the project. SURF, or Speeded Up Robust Features, is the main pattern recognition algorithm that I will be using to identify the drone from the camera feed. Because my camera feeds are already easily captured within OpenCV, I will also be using the library to create the user interface to display its status. I can easily draw labels around detected objects, as well as display lines connecting the matched regions in SURF.

User interface and user input:

The interface has been designed to be extremely basic. There are no sources of input by a human user; it is simply for informational purposes only. The window currently has one view of what the camera sees. For the full implementation of four cameras, there will be four views in a grid. On each camera's viewport, there will be superimposed matches of any drone detections. A box is drawn around the drone, with a center circle indicated the average point that is taken and sent to be processed. On a left panel is the still image that has been fed into SURF. An example screenshot is shown below. This will be the primary display for demonstration purposes. The drone will be flying around a room, and its tracked location per individual camera will be viewable on screen. I am exploring the best way to show the 3D coordinate once that has been calculated. It will likely be another window (or subset of the primary window) with a simple 3D graph with axis' lengths corresponding to the room's dimension. A line on the graph can represent the drone's movement through the room.

In the program's current state, there is one user input via command line and that is the relative positioning of each camera. Just two integers are accepted, as the cameras are assumed to be in a rectangle. The final version will also require viewing angles. These values will be hard-coded in a configuration file in the final version, once the cameras are mounted.