# Legacy Computer Vision Code

The purpose of this section is to document the code from previous years that the team attempted to use for the vision portion of the project. While the team did eventually end up deciding not to take this approach, which involves blob detection, a lot was learned about the process, and if nothing else, we were required to think about the problem of vision much more thoroughly. The code is split up into multiple source files, each file having its own purpose. While this topic is indeed briefly touched upon in the prototypes section of this document, this section will go into much more depth in examining the code, and describing how it functions. The files used are:

- Camera.h/cpp
- Coord.h/cpp
- LED.h/cpp
- Triangle.h/cpp
- tracker.cpp

## tracker.cpp

### Overview

The file tracker.cpp contains the main control structure for the blob detection and tracking used by previous teams. Julian Brackins (the original author of the file) provided the following description of the file, which is included in the listing of the contents of the file as well:

> This software should hopefully soon be adapted into a ROS publisher that is capable of sending messages to our UAV to indicate the craft's distance from our landing pad situated on our Unmanned Ground Vechicle (UGV). This is done using three equidistant coloured dots, or "blobs" situated on our landing pad. Utilizing OpenCv's libraries for tracking specific colours, these three blobs are recognized by the software's camera feed, with corresponding coordinates in relation to the camera feed's image plane. The software compares these values to the stored points read in from the configuration file in order to compare the observed size of the objects being tracked to the known size of these objects that has been previously calibrated and written to a config file.

```
/**************************************************************************//**
 * @file tracker.cpp
 *
 * @brief SOURCE - Blob detection software for determining distance
 *
 *
 * @section course_section CSC 465
 *
 * @author Julian Brackins
 *
 * @date December March, 10
 *
 * @par Professor:
```

```
 *        Dr. Jeff McGough
 *
 * @par Course:
 *        CSC 465 - M001 - Tues / Thurs - 9:00am
 *
 * @par Location:
 *        McLaury - 304
 *
 * @section program_section Program Information
 *
 * @details
 * This software should hopefully soon be adapted into a ROS publisher that is
 * capable of sending messages to our UAV to indicate the craft's distance from
 * our landing pad situated on our Unmanned Ground Vechicle (UGV). This is done
 * using three equidistant coloured dots, or "blobs" situated on our landing pad.
 * Utilizing OpenCv's libraries for tracking specific colours, these three blobs
 * are recognized by the software's camera feed, with corresponding coordinates
 * in relation to the camera feed's image plane. The software compares these values
 * to the stored points read in from the configuration file in order to compare the
 * observed size of the objects being tracked to the known size of these objects that
 * has been previously calibrated and written to a config file.
 *
 * @section compile_section Compiling and Usage
 *
 * @par Compiling Instructions:
 *      (Linux) - catkin_make
 *
 * @par Usage:
@verbatim
./tracker -[options] [configfile]
@endverbatim
 *
 * @section todo_bugs_modification_section Todo, Bugs, and Modifications
 *
 * @par Modifications and Development Timeline:
@verbatim
Date               Modification
----------------   ------------------------------------------------------------
 * October  21, 2014    Uploading qr reader with distance algorithm
 *
 * November  4, 2014    Adding led stuff
 *
 * February  2, 2015    Set up LED tracking code
 *
 * February  8, 2015    Created Coordinates and Triangle classes, need to start
 *                      working on distance algorithm.
 *
 * February 16, 2015    Triangle is set up and printing, but lawOfCosines
 *                      calculation isn't working right.
 *
 * February 21, 2015    Cleaning out led directory's cmake files so that code
 *                      builds more easily on other machines
 *
 * February 22, 2015    Distance approximations now working through blob
 *                      detection, Now just have to comment it all.... -.-
 *
 * February 28, 2015    A few tweaks
 *
 * March 10, 2015       Retooled tracker system so that it takes in command arguments
 *                       properly. run <tracker -h> to see the command line options.
 *                       Tracker can now determine distance from just two blobs. Need
 *                       to implement config files still
 *
 @endverbatim
 *
 ****************************************************************************/

/****************************************************************************
 *
 * INCLUDE
 *
 ****************************************************************************/


#include <sstream>
#include <string>
#include <iostream>
#include <vector>
#include <unistd.h>
#include "opencv2/highgui/highgui.hpp"
#include "opencv/cv.h"
#include "LED.h"
#include "Coord.h"
#include "Triangle.h"
#include "Camera.h"
#include <math.h>

using namespace cv;
using namespace std;

//default capture width and height
const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;

//max number of objects to be detected in frame
const int MAX_OBJS=50;

//minimum and maximum object area
const int MIN_OBJ_AREA = 15*15;
const int MAX_OBJ_AREA = FRAME_HEIGHT*FRAME_WIDTH/1.5;
//names that will appear at the top of each window
const string winName = "Blob Detection";


int _arg_cam = 0;
//Global flags for command arguments
bool _arg_calibrate = false;
bool _arg_distance  = false;
bool _arg_points    = false;
bool _arg_booleans  = false;
bool _arg_ros       = false;

string _arg_conf     = "hsv.conf";

string intToString(int number);

Coord drawObject(LED light, Mat &frame);
void drawObject(vector<LED> lights, Mat &frame);
```

```cpp
void trackFilteredObject(Mat threshold,Mat HSV, Mat &cameraFeed);
Coord trackFilteredObject(LED lights,Mat threshold,Mat HSV, Mat &cameraFeed);
void morpher(Mat &threshold);
ostream& operator<<(ostream& os, Coord& cd);
void Process_Arguments(int argc, char **argv);


/*****************************************************************************
 * @author Julian Brackins
 *
 * @par Description:
 * Set up infinite loop for program. Sorry for using main() for my whole prog..
 *
 * @param[in] argc - count of args.
 * @param[in] argv - arguments themselves.
 *
 *****************************************************************************/
int main(int argc, char* argv[])
{
    //true to set calibration

    Process_Arguments( argc, argv);


   //Matrix to store each frame of the webcam feed
   Mat cameraFeed;
   Mat threshold;
   Mat HSV;


   //video capture object to acquire webcam feed
   VideoCapture capture;
   //open capture object at location one (default location for usb cam)

       capture.open(_arg_cam);


   //set height and width of capture frame
   capture.set(CV_CAP_PROP_FRAME_WIDTH,FRAME_WIDTH);
   capture.set(CV_CAP_PROP_FRAME_HEIGHT,FRAME_HEIGHT);

   //infinite loop where all operations occur.
   while(1)
   {
      //store image to matrix
      capture.read(cameraFeed);

      //convert frame from BGR to HSV colorspace
      cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);

       //Create LEDs
       LED green("green"), blue("blue"), red("red");

       //Create Triangle
       Triangle camTri;

       //Create the coordinate sets for all three blobs
        Coord p1, p2, p3;

```

```
          //Set up Camera class that calculates distance based on focal length &
          //observed object distance.
          Camera distCam;

         //Set up Red Tracker, send coordinates to p1
         cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
         inRange(HSV,red.getHSVmin(),red.getHSVmax(),threshold);
         morpher(threshold);
         p1 = trackFilteredObject(red,threshold,HSV,cameraFeed);

         //Set up Green Tracker, send coordinates to p2
         cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
         inRange(HSV,green.getHSVmin(),green.getHSVmax(),threshold);
         morpher(threshold);
         p2 = trackFilteredObject(green,threshold,HSV,cameraFeed);

         //Set up Blue Tracker, send coordinates to p3
         cvtColor(cameraFeed,HSV,COLOR_BGR2HSV);
         inRange(HSV,blue.getHSVmin(),blue.getHSVmax(),threshold);
         morpher(threshold);
         p3 = trackFilteredObject(blue,threshold,HSV,cameraFeed);

          //Display Image Feed in window
        imshow(winName,cameraFeed);

          //Set up the three sides of the Triangle
          if(p2.isTracking() && p3.isTracking())
              camTri.setSide("A", p2,p3);
          if(p3.isTracking() && p1.isTracking())
              camTri.setSide("B", p3,p1);
          if(p1.isTracking() && p2.isTracking())
              camTri.setSide("C", p1,p2);

          //Set the A,B,C sides of the triangle in the Camera Class
          distCam.setA( camTri.getSide("A") );
          distCam.setB( camTri.getSide("B") );
          distCam.setC( camTri.getSide("C") );

          distCam.resetBlobs();
          if(p1.isTracking())
              distCam.incBlobs();
          if(p2.isTracking())
              distCam.incBlobs();
          if(p3.isTracking())
              distCam.incBlobs();
          //Print out distance to stdout
          if(_arg_calibrate)
          {

              //cout << "A: " << dist(p2,p3) << " | ";
              ///cout << "B: " << dist(p3,p1) << " | ";
              //cout << "C: " << dist(p1,p2) << endl;
          }
          if(_arg_distance)
          {
              cout << "DISTANCE: " << distCam.getDist() << endl;
          }
          if(_arg_points)
          {
            cout <<   "A: " << camTri.getSide("A") << " | B: " << camTri.getSide("B") << " | C: " << camTri.getSide("C") << endl;
        }
        if(_arg_booleans)
        {
            cout << distCam.blobTotal() << " blobs detected | ";
            cout << "Red  : " << p1.printTracking() << " | ";
            cout << "Green: " << p2.printTracking() << " | ";
            cout << "Blue : " << p3.printTracking() << endl;
        }


      //delay 30ms so that screen can refresh.
      //image will not appear without this waitKey() command
        if (waitKey(30) == 27) //wait for 'esc' key press for 30ms. If 'esc' key is pressed, break loop
        {
            break;
        }
   }
   return 0;
}

/*****************************************************************************
 * @author Julian Brackins
 *
 * @par Description:
 * Convert integers to strings lmaoooo.
 *
 * @param[in] number - an integer lmfaoo.
 *
 * @returns that number as a STRING yo
 *
 *****************************************************************************/
string intToString(int number)
{
   std::stringstream ss;
   ss << number;
   return ss.str();
}

/*****************************************************************************
 * @author Julian Brackins
 *
 * @par Description:
 * Convert integers to strings lmaoooo.
 *
 * @param[in] light - an led light being tracked
 * @param[in] frame - image matrix
 *
 * @returns Set of coordinates in the Coord class structure, giving you (x,y)
 *
 *****************************************************************************/
Coord drawObject(LED light, Mat &frame)
{
    //Draw a circle around the object being tracked

    cv::circle( frame, cv::Point( light.getX(),light.getY() ), 10, cv::Scalar(0,0,255));
    cv::putText(frame, intToString( light.getX() ) + " , " + intToString( light.getY() ),
                cv::Point(light.getX(), light.getY() + 20 ), 1, 1, cv::Scalar(0,255,0));
    Coord point;
    point.setX(light.getX());
    point.setY(light.getY());
    return point;
}

/******************************************************************************
 * @author Julian Brackins
 *
 * @par Description:
 * After the images have been filtered properly, this function
 * handles determining if an image contains a blob that fits in the HSV range
 * of the colour being tracked.
 *
 * @param[in] light - an led light being tracked
 * @param[in] threshold - thresholded image matrix
 * @param[in] HSV - Matrix containing HSV values
 * @param[in] cameraFeed - Matrix containing camera feed
 *
 * @returns Set of coordinates in the Coord class structure, giving you (x,y)
 *
 *****************************************************************************/
Coord trackFilteredObject(LED lights,Mat threshold,Mat HSV, Mat &cameraFeed)
{
    //Maybe change all this so that It doesn't refer to everything as lights...
   LED blob;

   Coord coordinates;

   Mat tempMatrix;
   threshold.copyTo(tempMatrix);

   //findContours params
   vector<vector<Point>> contours;
   vector<Vec4i> hierarchy;

    //find the contours of the image using openCV
   findContours( tempMatrix, contours, hierarchy,CV_RETR_CCOMP,CV_CHAIN_APPROX_SIMPLE );

   //use moments method to find our filtered object
   double refArea = 0;
   bool objectFound = false;

   //Zero out the coordinates
   coordinates.setX(0);
   coordinates.setY(0);

   if (hierarchy.size() > 0)
   {
      int numObjects = hierarchy.size();
      //The filter becomes too noisy if the numObjects is too great...
      if(numObjects < MAX_OBJS)
      {
         for (int i = 0; i >= 0; i = hierarchy[i][0])
         {

            Moments moment = moments((cv::Mat)contours[i]);
            double area = moment.m00;

            //if the area is less than 20 px by 20px then it is probably just noise
            //if the area is the same as the 3/2 of the image size, probably just a bad filter
            //we only want the object with the largest area so we safe a reference area each
            //iteration and compare it to the area in the next iteration.
            if(area>MIN_OBJ_AREA)
            {
               blob.setX(moment.m10/area);
               blob.setY(moment.m01/area);
               blob.setColour(lights.getColour());
               blob.setText(lights.getText());

               //Push onto vector if allowing multiples of one colour
               //blobvec.push_back(blob);
               objectFound = true;
            }
            else objectFound = false;
         }
         //let user know you found an object
         if(objectFound ==true)
         {
            //draw object location on screen
            //use vector version of drawObject if allowing multiples
            coordinates = drawObject(blob,cameraFeed);
             coordinates.setTracking(true);
         }
         else
         {
             coordinates.setTracking(false);
          }
      }
      else putText(cameraFeed,"Adjust Filter, too much noise.",Point(0,50),1,2,Scalar(0,0,255),2);
   }
   return coordinates;
}

/***************************************************************************
 * @author Julian Brackins
 *
 * @par Description:
 * Morphing function to properly erod and dilate the dense array to enhance
 * image visibility.
 *
 * @param[in] threshold - thresholded image matrix
 *
 ***************************************************************************/
void morpher(Mat &threshold)
{
    //Erode and dilate the dense array to make the object clearly visible

    //Construct the erode
   Mat erodeElement  = getStructuringElement( MORPH_RECT, Size(3,3));
   Mat dilateElement = getStructuringElement( MORPH_RECT, Size(8,8));

   erode(threshold, threshold, erodeElement);
   erode(threshold, threshold, erodeElement);
   dilate(threshold, threshold, dilateElement);
   dilate(threshold, threshold, dilateElement);
}

/*****************************************************************************
 * @author Julian Brackins
 *
 * @par Description:
 * Overloaded output stream for the Coord class. Might as well toss this
 * eventually because I'll most likely be switching to printf anyways... hehe.
 *
 * @param[in] os - outstream.
 * @param[in] cd - representatoin of Coord class indicating how output works
 *
 * @returns os - output stream
 *
 *****************************************************************************/
ostream& operator<<(ostream& os,  Coord& cd)
{
    //Overloaded Coord << operator
    if (cd.getX() == 0 || cd.getY() == 0)
        os << "( " << "N/A" << "," << "N/A" << " )";
    else
        os << "( " << cd.getX() << "," << cd.getY() << " )";
    return os;
}

/*****************************************************************************
 * @author Julian Brackins
 *
 * @par Description:
 * Super sophisticated arguments processor that I made back at NASA woo.
 * Handles a bunch of different option flags brought in through command line
 * arguments. The first command argument is the program name (./tracker), the
 * second argument is the list of option flags, and the third option is the
 * configuration file being read in. Each flag that is read in will modify a
 * global flag that will determine how the program is executed.
 *
 * @param[in] argc - count of args.
 * @param[in] argv - arguments themselves.
 *
 *****************************************************************************/
void Process_Arguments(int argc, char **argv)
{
    int c;
    printf("CVTracker: argc=%d, argv=%s\n", argc-1, argv[1]);
    //tokenize each param and check it individually
    while ((c = getopt(argc, argv, "01234cdpbrh")) != -1)
    {
        switch (c)
        {
            case '0':
                _arg_cam = 0;
                printf("CVTracker: using camera 0 (-0)\n");
                break;
            case '1':
                _arg_cam = 1;
                printf("CVTracker: using camera 1 (-1)\n");
                break;
            case '2':
                _arg_cam = 2;
                printf("CVTracker: using camera 2 (-2)\n");
                break;
            case '3':
                _arg_cam = 3;
                printf("CVTracker: using camera 3 (-3)\n");
                break;
            case '4':
                _arg_cam = 4;
                printf("CVTracker: using camera 4 (-4)\n");
                break;
            case 'c':
                _arg_calibrate = true;
                printf("CVTracker: running calibration mode (-c)\n");
                break;
            case 'd':
                _arg_distance = true;
                printf("CVTracker: printing Object Distance to stdout (-d)\n");
                break;
            case 'p':
                _arg_points = true;
                printf("CVTracker: printing RGB point coordinates to stdout (-p)\n");
                break;
            case 'b':
                _arg_booleans = true;
                printf("CVTracker: printing blob tracking booleans (-b)\n");
                break;
            case 'r':
                _arg_ros = true;
                printf("CVTracker: publishing to ROS (-r)\n");
                break;
            case 'h':
                printf("CVTracker: help\n");
                printf("$ %s [-options] [logfile] \n", argv[0]);
                printf("    Please place a - then however many of the following [option] values:\n");
                printf("    -#  select which camera to use ( 0 - 4, 0 is default).\n");
                printf("    -c  run calibration mode.\n");
                printf("    -d  ouput object distance to stdout\n");
                printf("    -p  print RGB point coordinates to stdout\n");
                printf("    -b  print blob tracking booleans (true=tracking / false=not tracking)\n");
                printf("    -r  publish info to ROS\n");
                printf("    -h  display help\n");
                printf("    If no [logfile] specified then the default\n");
                printf("    \"hsv.conf\" will be used.\n");
                exit(0);
                break;
            default:
                //gStandalone = 0;
                printf("CVTracker: ignoring option -%c\n", c);
                break;
        }
    }
    /**
     * Handle the configuration file here.
     **/
        if( argc > 2 )
        {
            _arg_conf = argv[2];
            printf("CVTracker: reading HSV & focal length settings from configuration file:↩
                 \"%s\"\n", argv[2]);
        }
        else
        {
            printf("CVTracker: reading HSV & focal length settings from default file: \"hsv↩
                .conf\"\n");
        }
}
```

# Camera.h/cpp

## Overview

The Camera class is used to take an image feed that has three colored blobs detected on it, and calculate the distance to the target, based on the known size of the target. Within Camera.h, you can see that there are a couple of options for the paper size, and these will need to be changed at the time of compilation so that the system will be able to accurately calculate the distance to the target. The method used to calculate the distance to the target is to simply take some known distance to the target that has a pixel width associated with it, and calculate the focal length for the camera. Then, we just divide the focal length by the pixel distance on our measurement and multiply that result by the known width of the target in order to get the new distance.

$$focal\_length = pixel\_width * known\_dist\_to\_target / known\_width\_of\_target$$

$$measured\_dist\_to\_target = known\_width\_of\_target * focal\_length / pixel\_width$$

### Camera.h

```
#ifndef _CAMERA_H_
#define _CAMERA_H_

#include <string>
#include <iostream>



class Camera
{

public:
   Camera();
   ~Camera();


    void setFocaLength( );
    double getFocalLength() { return focal_length; }
    double getDist( );
    void setA( double val ) { curr_a = val ; }
    void setB( double val ) { curr_b = val ; }
    void setC( double val ) { curr_c = val ; }
    double getA() { return curr_a; }
    double getB() { return curr_b; }
    double getC() { return curr_c; }
    void resetBlobs();
    void incBlobs();
    double blobTotal();
private:
    //double that keeps track of how many
    //blobs that are being detected
    double tracked_blobs;
    double focal_length;

    //SMALL PAPER
    //const double known_obj_width = 5.5;
    //const double known_obj_dist  = 27.0;

    //LARGE PAPER
    const double known_obj_width = 11.5;
    const double known_obj_dist  = 30.5;
    double dist;

    //Original Pixel Widths

    //SMALL PAPER
    //const double orig_a = 141.00;
    //const double orig_b = 139.87;
    //const double orig_c = 142.68;

    //LARGE PAPER
    const double orig_a = 273.827;
    const double orig_b = 276.123;
    const double orig_c = 270.017;

    //Observed Pixel Widths
    double curr_a;
    double curr_b;
    double curr_c;


};

#endif
```

## Camera.cpp

```
#include "Camera.h"


Camera::Camera()
{
    setFocaLength( );
    Camera::resetBlobs();
}

Camera::~Camera()
{
}
```

```

void Camera::setFocaLength( )
{
    double avg_pix = ( orig_a + orig_b + orig_c ) / 3.0;
    focal_length = avg_pix * ( known_obj_dist / known_obj_width );
}

double Camera::getDist( )
{
    double avg_val = 1.0;
    if( blobTotal() >= 3 )
        avg_val = 3.0;
    else
        avg_val = 1.0;
    double avg_pix = ( curr_a + curr_b + curr_c ) / avg_val;
    dist = known_obj_width * ( focal_length / avg_pix );
    return dist;
}


void Camera::resetBlobs( )
{
    tracked_blobs = 0;
}

void Camera::incBlobs( )
{
    tracked_blobs += 1;
}

double Camera::blobTotal()
{
    return tracked_blobs;
}
```

## Coord.h/cpp

### Overview

The Coord class is used to provide coordinate functionality, which is to say that it will be used by later classes to allow them to store coordinates.

#### Coord.h

```
#ifndef _COORD_H_
#define _COORD_H_

#include <string>

class Coord
{
public:
   Coord();
   ~Coord();


```

```cpp

   int getX();
   void setX(int val);

   int getY();
   void setY(int val);

   bool isTracking();
   void setTracking(bool val);
   std::string printTracking();

private:
    bool tracking;
   int  xPos, yPos;

};

#endif
```

## Coord.cpp

```cpp
#include "Coord.h"


Coord::Coord()
{
    setX(0);
    setY(0);
    setTracking(false);
}


Coord::~Coord()
{
}


int Coord::getX()
{
   return Coord::xPos;
}

void Coord::setX(int val)
{
   Coord::xPos = val;
}

int Coord::getY()
{
   return Coord::yPos;
}

void Coord::setY(int val)
{
   Coord::yPos = val;
}


```

```
bool Coord::isTracking()
{
    return Coord::tracking;
}

std::string Coord::printTracking()
{
    if(Coord::tracking)
        return "true";
    else
        return "false";
}

void Coord::setTracking(bool val)
{
    Coord::tracking = val;
}
```

# Triangle.h/cpp

## Overview

The Triangle class is used to allow the program to store a set of three coordinates that make up the triangle of the blobs on the target. The class stores the coordinates of each vertex of the triangle, along with the lengths of each triangle edge. Additionally, it will store the angle between the edges, that can be used to see how "straight on" the image has been taken from.

### Triangle.h

```
#ifndef _TRIANGLE_H_
#define _TRIANGLE_H_

#include <string>
#include <iostream>
#include "Coord.h"
#include "math.h"
#define PI 3.14159265

class Triangle
{
public:
   Triangle();
   Triangle(Coord c1, Coord c2, Coord c3);
   ~Triangle();


   void setPoint(std::string pointName, Coord point);
   void setSide(std::string sideName, Coord firstPoint, Coord secondPoint);
   double calcSide(Coord P1, Coord P2);
   Coord getPoint(std::string pointName);
   double getSide(std::string sideName);

   //find all angles
    void lawOfCosines();
    double radToDeg(double radVal);

    void printTriangle();

private:

    //Reminder:
    //sideA = dist(pointB, pointC);
    //sideB = dist(pointA, pointC);
    //sideC = dist(pointA, pointB);

    //Angle A = angle at pointA,
    //Angle B = angle at pointB,
    //Angle C = angle at pointC,
    double sideA, sideB, sideC;
   Coord pointA, pointB, pointC;
    double angleA, angleB, angleC;


};

#endif
```

## Triangle.cpp

```
#include "Triangle.h"


Triangle::Triangle()
{
    sideA = 0.0;
    sideB = 0.0;
    sideC = 0.0;
}


Triangle::Triangle(Coord c1, Coord c2, Coord c3)
{
    pointA = c1;
    pointB = c2;
    pointC = c3;
}

Triangle::~Triangle()
{
}

void Triangle::setPoint(std::string pointName, Coord point)
{
    if(pointName == "A")
        pointA = point;
    else if(pointName == "B")
        pointB = point;
    else if(pointName == "C")
        pointC = point;
}

void Triangle::setSide(std::string sideName, Coord firstPoint, Coord secondPoint)
{
    double tempSide = 0.0;
```

```cpp

    tempSide = calcSide(firstPoint, secondPoint);

    if(sideName == "A")
        sideA = tempSide;
    else if(sideName == "B")
        sideB = tempSide;
    else if(sideName == "C")
        sideC = tempSide;
}

double Triangle::calcSide(Coord P1, Coord P2)
{
    double xSquared = (P2.getX() - P1.getX()) * (P2.getX() - P1.getX());
    double ySquared = (P2.getY() - P1.getY()) * (P2.getY() - P1.getY());
    return sqrt( xSquared + ySquared );
}

Coord Triangle::getPoint(std::string pointName)
{
    if(pointName == "A")
        return pointA;
    else if(pointName == "B")
        return pointB;
    else if(pointName == "C")
        return pointC;

}

double Triangle::getSide(std::string sideName)
{
    if(sideName == "A")
        return sideA;
    else if(sideName == "B")
        return sideB;
    else if(sideName == "C")
        return sideC;
    else
        return -200.0;
}

void Triangle::lawOfCosines()
{
    //Calculate angleA
    angleA = ( (sideB*sideB) + (sideC*sideC) - (sideA*sideA) ) / (2 * sideB * sideC);
    angleA = acos(angleA);
    angleA = radToDeg(angleA);

    //Calculate angleB
    angleB = ( (sideC*sideC) + (sideA*sideA) - (sideB*sideB) ) / (2 * sideC * sideA);
    angleB = acos(angleB);
    angleB = radToDeg(angleB);

    angleC = 180.0 - angleA - angleB;
}

double Triangle::radToDeg(double radVal)
{
    return radVal * 180.0 / PI;
```

```
}

void Triangle::printTriangle()
{
    lawOfCosines();
    std::cout << "A: " << angleA << " degrees | ";
    std::cout << "B: " << angleB << " degrees | ";
    std::cout << "C: " << angleC << " degrees | ";
    std::cout << "Total: " << angleA+angleB+angleC << " degrees" << std::endl;
}
```

## LED.h/cpp

### Overview

The LED class essentially is there to allow definitions of the different colors that are in the blobs being searched for. While the class is named LED, it would be more accurate to rename it to blobs, since detecting colored LEDs was replaced with detecting colored blobs.

**LED.h**

```
/**************************************************************************//**
 * @file LED.h
 *
 * @author Julian Brackins
 *
 * @brief HEADER - class for LED groupings. HSV values stored in this class.
 *
 *****************************************************************************/

#ifndef _LED_H_
#define _LED_H_

/******************************************************************************
 *
 * INCLUDE
 *
 ******************************************************************************/

#include <string>
#include "opencv/cv.h"
#include "opencv/highgui.h"

/**************************************************************************//**
* @class LED
*
* @author Julian Brackins
*
* @brief LEDs! Not really anymore but who wants to mess with names.
*
*****************************************************************************/
class LED
{
public:
   LED();
    LED(std::string name);
   ~LED();

   int getX()                   { return xPos; }
   void setX(int val)           { xPos = val; }

   int getY()                   { return LED::yPos; }
   void setY(int val)           { LED::yPos = val; }

    cv::Scalar getHSVmin()      { return LED::HSVmin; }
    cv::Scalar getHSVmax()      { return LED::HSVmax; }

    void setHSVmax(cv::Scalar val)  { LED::HSVmax = val; }
    void setHSVmin(cv::Scalar val)  { LED::HSVmin = val; }

    std::string getColour()         { return colour; }
    void setColour(std::string val) { colour = val;  }

    cv::Scalar getText()         { return txtColour; }
    void setText(cv::Scalar val) { txtColour = val;  }

private:
   int xPos, yPos;
   std::string colour;
   cv::Scalar HSVmin, HSVmax;
   cv::Scalar txtColour;
};

#endif //LED_H
```

## LED.cpp

```
/**********************************************************************//**
 * @file LED.cpp
 *
 * @author Julian Brackins
 *
 * @brief SOURCE - class for LED groupings. HSV values stored in this class.
 *
 ***************************************************************************/

#include "LED.h"

/***********************************************************************//**
 * @author Julian Brackins
 *
 * @par Description:
 * LED Constructor. The empty constructor should never be used...
 *
 ***************************************************************************/
LED::LED()
{

}

/***********************************************************************//**
 * @author Julian Brackins
 *
 * @par Description:
```

```
 * LED Constructor for when a name is passed in as a parameter. This decides
 * what colour the LED is.
 *
 ******************************************************************************/
LED::LED(std::string name)
{

   setColour(name);

    //TODO: Adjust HSV vals to something more appropriate...
   if(name=="green")
   {
      setHSVmin(cv::Scalar(32, 55, 0));
      setHSVmax(cv::Scalar(105, 256, 256));

      //BGR value for Green:
      setText(cv::Scalar(0,255,0));
   }
   if(name=="yellow")
   {
      setHSVmin(cv::Scalar(17, 34, 201));
      setHSVmax(cv::Scalar(45, 256, 256));

      //BGR value for Yellow:
      setText(cv::Scalar(0,255,255));
   }
   if(name=="red")
   {
        setHSVmin(cv::Scalar(0,181,0));
        setHSVmax(cv::Scalar(217,256,224));

      //BGR value for Red:
      setText(cv::Scalar(0,0,255));
   }
   if(name=="blue")
   {
      setHSVmin(cv::Scalar(87, 110, 0));
      setHSVmax(cv::Scalar(200, 256, 139));

      //BGR value for Red:
      setText(cv::Scalar(0,0,255));
   }

}

/******************************************************************************//**
 * @author Julian Brackins
 *
 * @par Description:
 * LED Destructor.
 *
 ******************************************************************************/
LED::~LED()
{

}
```

## Final Conclusions About Legacy Code

### Functionality

Fortunately, the legacy code provided by the 2014-2015 UAV/UGV team is definitely functional. However, due to time constraints, that team was unable to "ROSify" the code by creating a ROS publisher for the data being generated within tracker.cpp. While three blobs being detected should be plenty to determine orientation of and distance to the target, after having members complete the Computer Vision course at school, it was decided that it may be worthwhile to pursue a four-blob approach to allow use of some homography techniques that were used in that course. However, this never ended up turning into anything. Eventually, it was decided that it would be best to use the ROS library AR_Track_Alvar, since it is already implemented in ROS, and provides all functionality that would be needed by the team.

### Using the Legacy Code

It is important to understand how to run this code to see the progress made by previous teams. This provided a huge amount of help to the current team, as we were able to have a jumping off point to begin work from, and understand some approaches that worked well, and some things that maybe didn't work as well. To find instructions on running this code, look to the prototypes section of this document, and find the prototypes for Sprint 2, as that will provide information on building and running this code.