%% Grid of Points Image Processing

% Samuel S. Artho-Bentz

%% Clean Up

clc

clear all

close all

%% Read in Photo

myPhoto = imread('img\_7849.jpg');

%% Set system param

% distance to board from system origin

% dboardMin\_mm was measured at 1260. however, it is found that the result

% is VERY dependant on small changes to this

dBoardMin\_mm = 1265;

dBoardMin\_in = dBoardMin\_mm/25.4;

%% Find location of red reference points

% The red dots are 12 inches apart

redDots=myImgStats(myPhoto,1);

%% Calculate a Vertical Scale for pixels per inch

% Find vertical distances between points in pixels

vDist = [redDots(1,2)-redDots(3,2), redDots(2,2)-redDots(3,2), redDots(1,2)-redDots(2,2)];

% Actual distance in inches

vDistRef=[12, 12, 0];

% Apply simple linear regression to find a conversion between pixels and inches in the vertical direction

% Measurement[inches] = vertreg(1) \* Measurement[pixels] + vertreg(2)

vertReg=myLinReg(vDist,vDistRef);

% Verify linear regression is reasonable

vDistCheck=vertReg(1)\* vDist + vertReg(2);

%% Calculate a Horizontal Scale for pixels per inch

% Find Horizontal distances between points in pixels

hDist = [redDots(3,1)-redDots(1,1), redDots(2,1)-redDots(3,1), redDots(2,1)-redDots(1,1)];

% Actual distance in inches

hDistRef=[12, 0, 12];

% Apply simple linear regression to find a conversion between pixels and inches in the vertical direction

% Measurement[inches] = vertreg(1) \* Measurement[pixels] + vertreg(2)

horReg=myLinReg(hDist,hDistRef);

% Verify linear regression is reasonable

hDistCheck=horReg(1)\* hDist + horReg(2);

%% Find all black dots

myGSPhoto=rgb2gray(myPhoto);

myBWPhoto=myGSPhoto<100;

rp=regionprops(myBWPhoto,myGSPhoto,'Centroid');

centroids = cat(1, rp.Centroid);

% Remove the centroids of the red dots from the list

for i = 1:length(centroids)

if i > length(centroids)

break

end

if ((centroids(i,1)>redDots(1,1)\*.9) && (centroids(i,1)<redDots(1,1)\*1.1)) && ((centroids(i,2)>redDots(1,2)\*.9) && (centroids(i,2)<redDots(1,2)\*1.1))

centroids(i,:)=[];

end

if ((centroids(i,1)>redDots(2,1)\*.9) && (centroids(i,1)<redDots(2,1)\*1.1)) && ((centroids(i,2)>redDots(2,2)\*.9) && (centroids(i,2)<redDots(2,2)\*1.1))

centroids(i,:)=[];

end

if ((centroids(i,1)>redDots(3,1)\*.9) && (centroids(i,1)<redDots(3,1)\*1.1)) && ((centroids(i,2)>redDots(3,2)\*.9) && (centroids(i,2)<redDots(3,2)\*1.1))

centroids(i,:)=[];

end

end

%% Find the matching pairs of points by looking for those that are within 150 pixels of each other.

pairIndex = 1;

for i = 1:length(centroids)

for j = 1:length(centroids)

if (i~=j)&& (j>i)

if hypot(centroids(i,1)-centroids(j,1), centroids(i,2)-centroids(j,2))<150

centroidPairs(pairIndex, :) = [i,j];

pairIndex= pairIndex +1;

end

end

end

end

%% Convert centroid coordinates from pixels to inches using the scales above

centroidsInches(:,1) = horReg(1) \* centroids(:,1) + horReg(2);

centroidsInches(:,2) = vertReg(1) \* centroids(:,2) + vertReg(2);

%% Create new variable pair data

for i = 1:length(centroidPairs)

% pairedCoord = [p1x p1y p2x p2y]

pairedCoord(i,:) = [centroidsInches(centroidPairs(i,1),1) centroidsInches(centroidPairs(i,1),2) centroidsInches(centroidPairs(i,2),1) centroidsInches(centroidPairs(i,2),2)];

end

%% Find distance between each pair of points

for i = 1:length(centroidPairs)

pairDistance(i,1) = hypot(pairedCoord(i,1)-pairedCoord(i,3),pairedCoord(i,2)-pairedCoord(i,4));

end

%% use the distance from the board (known)

% define one dot (center bottom) as 'correct'

% use locations of all the dots, along with the commanded angles, to do

% some sort of error? +/- theoretical angle?

%% Reassociate all points with new origin

% New origin at bottom middle pair

middleBottomPair = 15; % Determined manually (az = 0, alt = 17)

middleBottomCoord = [mean([pairedCoord(middleBottomPair, 1), pairedCoord(middleBottomPair,3)]) mean([pairedCoord(middleBottomPair, 2), pairedCoord(middleBottomPair,4)])];

middleBottomWRTZero = [0, tand(17)\*dBoardMin\_in];

% All points wrt 0,0

pairedCoord = [pairedCoord(:,1)-middleBottomCoord(1), -pairedCoord(:,2)+middleBottomCoord(2)+middleBottomWRTZero(2), pairedCoord(:,3)-middleBottomCoord(1), -pairedCoord(:,4)+middleBottomCoord(2)+middleBottomWRTZero(2)];

%% Find average point between each pair

for i=1:length(pairedCoord)

pairedCoord(i,5) = mean([pairedCoord(i,1),pairedCoord(i,3)]);

pairedCoord(i,6) = mean([pairedCoord(i,2),pairedCoord(i,4)]);

end

%% Associate commanded angles to each pair

% Ideally would automatically assign commanded angles to each pair but for

% sake of time, doing it manually. adding [az, alt] to each pair

pairedCoord(1,7:8) = [-10, 35];

pairedCoord(2,7:8) = [-10, 30];

pairedCoord(3,7:8) = [-10, 25];

pairedCoord(4,7:8) = [-10, 20];

pairedCoord(5,7:8) = [-10, 17];

pairedCoord(6,7:8) = [-5, 35];

pairedCoord(7,7:8) = [-5, 30];

pairedCoord(8,7:8) = [-5, 25];

pairedCoord(9,7:8) = [-5, 20];

pairedCoord(10,7:8) = [-5, 17];

pairedCoord(11,7:8) = [0, 35];

pairedCoord(12,7:8) = [0, 30];

pairedCoord(13,7:8) = [0, 25];

pairedCoord(14,7:8) = [0, 20];

pairedCoord(15,7:8) = [0, 17];

pairedCoord(16,7:8) = [5, 17];

pairedCoord(17,7:8) = [5, 20];

pairedCoord(18,7:8) = [5, 25];

pairedCoord(19,7:8) = [5, 30];

pairedCoord(20,7:8) = [5, 35];

pairedCoord(21,7:8) = [10, 17];

pairedCoord(22,7:8) = [10, 20];

pairedCoord(23,7:8) = [10, 25];

pairedCoord(24,7:8) = [10, 30];

pairedCoord(25,7:8) = [10, 35];

%% Associate Theoretical x,y distances with each pair

% pairedCoord = [x1, y1, x2, y2, xavg, yavg, azCmd, altCmd, xTheoretical, yTheoretical]

%

for i=1:length(pairedCoord)

pairedCoord(i,9) = dBoardMin\_in\*tand(pairedCoord(i, 7));

pairedCoord(i,10) = dBoardMin\_in\*tand(pairedCoord(i,8))/cosd(pairedCoord(i,7));

end

%% Calc Percent Error of each point from its theoretical location

% This gives poor comparison values because the range of distances examined

% are so different. Better to back calculate from x1,y1,etc into what the

% angle its looking at is. This will be better for doing error.

for i = 1:length(pairedCoord)

percentErrorDistance(i,1) = abs((pairedCoord(i,1)-pairedCoord(i,9))/pairedCoord(i,9))\*100;

percentErrorDistance(i,2) = abs((pairedCoord(i,2)-pairedCoord(i,10))/pairedCoord(i,10))\*100;

percentErrorDistance(i,3) = abs((pairedCoord(i,3)-pairedCoord(i,9))/pairedCoord(i,9))\*100;

percentErrorDistance(i,4) = abs((pairedCoord(i,4)-pairedCoord(i,10))/pairedCoord(i,10))\*100;

percentErrorDistance(i,5) = abs((pairedCoord(i,5)-pairedCoord(i,9))/pairedCoord(i,9))\*100;

percentErrorDistance(i,6) = abs((pairedCoord(i,6)-pairedCoord(i,10))/pairedCoord(i,10))\*100;

absoluteErrorDistance(i,1) = abs((pairedCoord(i,1)-pairedCoord(i,9)));

absoluteErrorDistance(i,2) = abs((pairedCoord(i,2)-pairedCoord(i,10)));

absoluteErrorDistance(i,3) = abs((pairedCoord(i,3)-pairedCoord(i,9)));

absoluteErrorDistance(i,4) = abs((pairedCoord(i,4)-pairedCoord(i,10)));

absoluteErrorDistance(i,5) = abs((pairedCoord(i,5)-pairedCoord(i,9)));

absoluteErrorDistance(i,6) = abs((pairedCoord(i,6)-pairedCoord(i,10)));

end

testVar = [percentErrorDistance(:,5:6) absoluteErrorDistance(:,5:6)];

%% Calc what angle each point is theoretically looking at.

% theoreticalAngle = [azCmd, altCmd, p1Az, p1Alt, p2Az, p2Alt, pavgAz,

% pavgAlt]

for i = 1:length(pairedCoord)

angleBackCalculated(i,1:2) = pairedCoord(i, 7:8);

angleBackCalculated(i,3) = atan2d(pairedCoord(i,1), dBoardMin\_in);

angleBackCalculated(i,4) = atan2d(pairedCoord(i,2)\*cosd(angleBackCalculated(i,3)), dBoardMin\_in);

angleBackCalculated(i,5) = atan2d(pairedCoord(i,3), dBoardMin\_in);

angleBackCalculated(i,6) = atan2d(pairedCoord(i,4)\*cosd(angleBackCalculated(i,5)) ,dBoardMin\_in);

angleBackCalculated(i,7) = atan2d(pairedCoord(i,5), dBoardMin\_in);

angleBackCalculated(i,8) = atan2d(pairedCoord(i,6)\*cosd(angleBackCalculated(i,7)), dBoardMin\_in);

end

%% Calc Percent Percent Error of back calculated angle vs command angle

for i = 1:length(angleBackCalculated)

percentErrorAngular(i,1) = abs((angleBackCalculated(i,3)-angleBackCalculated(i,1))/ angleBackCalculated(i,1))\*100;

percentErrorAngular(i,2) = abs((angleBackCalculated(i,4)-angleBackCalculated(i,2))/ angleBackCalculated(i,2))\*100;

percentErrorAngular(i,3) = abs((angleBackCalculated(i,5)-angleBackCalculated(i,1))/ angleBackCalculated(i,1))\*100;

percentErrorAngular(i,4) = abs((angleBackCalculated(i,6)-angleBackCalculated(i,2))/ angleBackCalculated(i,2))\*100;

percentErrorAngular(i,5) = abs((angleBackCalculated(i,7)-angleBackCalculated(i,1))/ angleBackCalculated(i,1))\*100;

percentErrorAngular(i,6) = abs((angleBackCalculated(i,8)-angleBackCalculated(i,2))/ angleBackCalculated(i,2))\*100;

end

% figure();

% image(myPhoto);

% axis image;

% hold on;

% plot(centroids(:,1),centroids(:,2), 'b\*')

% hold off

%