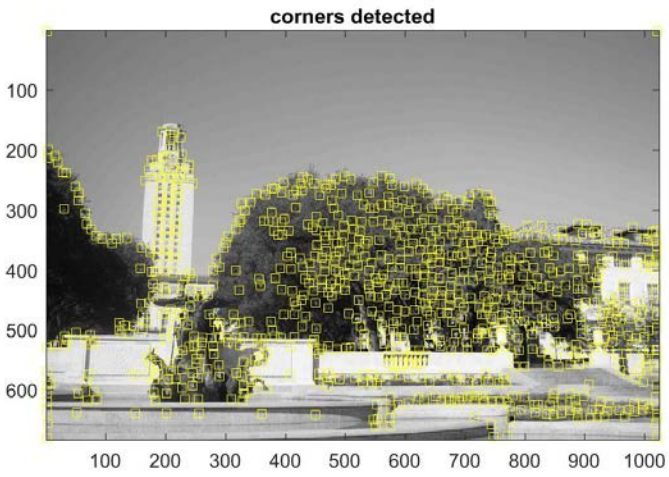
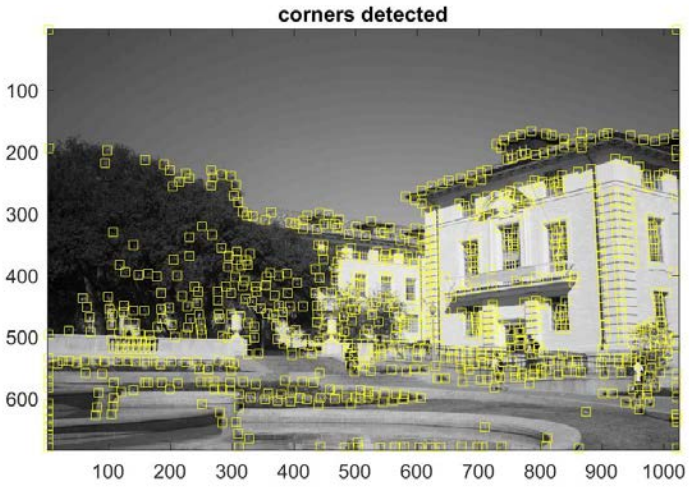
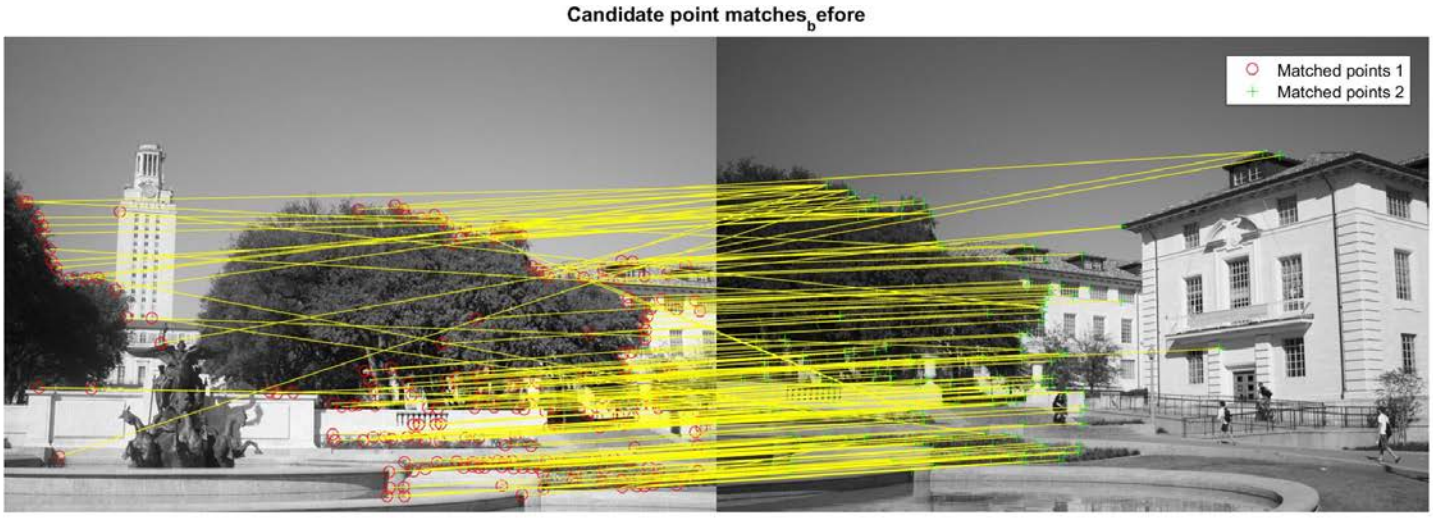
Stitching pairs of images

- After applying Harris detector on both images, the results are shown below.

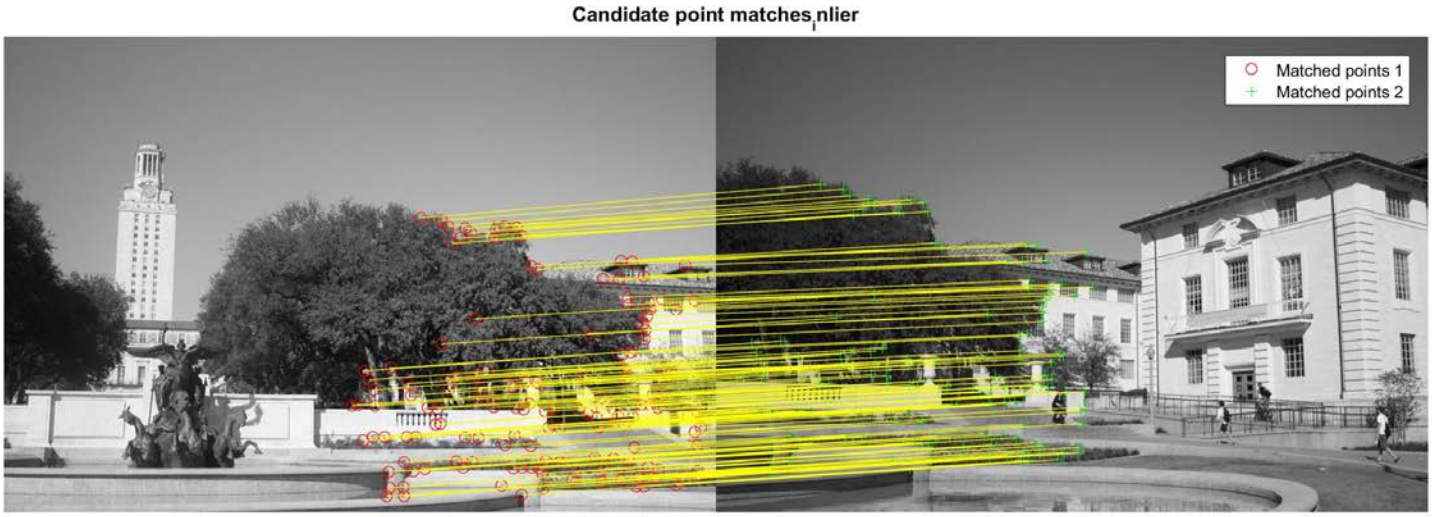




- After selecting top 200 descriptor pairs, the results are shown below.



- After applying RANSAC, the image below is the pairs of inliers



  - Top 200 pairs were selected at the windows of 35*35 as an import to my RANSAC function. After 1000 integrations, the optimized projective matrix is

    M =
    ```
    1.3424   -0.0392 -615.8533
    0.1499    1.3149 -180.8405
    0.0002    0.0002    1.0000
    ```

  - The number of inliers is 145 in both images.
  - The number of outliers is 55 in both images.
  - The maximum inliers rate is 86%.
  - The average residual of inliers is 2.8746 pixels.

- After stitching two images, the combined image is blurred at some pixels. Since the threshold I set for inliers is 10 pixels, and the average residual distance after RANSAC is 2.8746, when randomly select 4 pairs of inliers to form the panorama, there are might be shifted key points.

- Code for RANSAC

```
function
[M_opti,inlierPercent_Max,ProR_opt,MatchPointsL_opt,average_residual]
= getM(numIteration,MatchPointsL,MatchPointsR,inlierThreshold)

n1 = size(MatchPointsL,1);
inlierPercent_Max = 0;
M_opti = zeros(3,3);

for i = 1:numIteration

     m1 = randsample(n1,4); %Randomly generate row number of match
points
    %4 points randomly generated from left image
    pL1 = MatchPointsL(m1(1),:);
    pL2 = MatchPointsL(m1(2),:);
    pL3 = MatchPointsL(m1(3),:);
    pL4 = MatchPointsL(m1(4),:);

    %4 points randomly generated from right image
    pR1 = MatchPointsR(m1(1),:);
    pR2 = MatchPointsR(m1(2),:);
    pR3 = MatchPointsR(m1(3),:);
    pR4 = MatchPointsR(m1(4),:);

    %get C matirx
    c1 = [pL1(1);0;pL2(1);0;pL3(1);0;pL4(1);0];
    c2 = [pL1(2);0;pL2(2);0;pL3(2);0;pL4(2);0];
    c3 = [1;0;1;0;1;0;1;0];
    c4 = [0;pL1(1);0;pL2(1);0;pL3(1);0;pL4(1)];
    c5 = [0;pL1(2);0;pL2(2);0;pL3(2);0;pL4(2)];
    c6 = [0;1;0;1;0;1;0;1];
    c7 = [-pL1(1)*pR1(1);-pL1(1)*pR1(2);-pL2(1)*pR2(1);-
pL2(1)*pR2(2);-pL3(1)*pR3(1);-pL3(1)*pR3(2);-pL4(1)*pR4(1);-
pL4(1)*pR4(2)];
    c8 = [-pL1(2)*pR1(1);-pL1(2)*pR1(2);-pL2(2)*pR2(1);-
pL2(2)*pR2(2);-pL3(2)*pR3(1);-pL3(2)*pR3(2);-pL4(2)*pR4(1);-
pL4(2)*pR4(2)];

    C = [c1 c2 c3 c4 c5 c6 c7 c8];

    pointsR = [pR1';pR2';pR3';pR4'];
    %sudo inverse get M (3*3)
    M = pinv(C)*pointsR;
    M = reshape([M;1],3,3)';

    %get Projected right image 2*200
    pointL = zeros(3,n1);
    pointL = [MatchPointsL';ones(1,n1)];
    pointsR_pro = M*pointL;
```

```
    pointsR_pro1 = zeros(2,n1);
    pointsR_pro1 =
[pointsR_pro(1,:)./pointsR_pro(3,:);pointsR_pro(2,:)./pointsR_pro(3,:)
]'; %200*2

    diff_x = pointsR_pro1(:,1)-MatchPointsR(:,1);
    diff_y = pointsR_pro1(:,2)-MatchPointsR(:,2);

    residuals = sqrt(diff_x.*diff_x+diff_y.*diff_y);

    %set threshold = 10 pixels to check percentage of inliers
    count = 0;
%     inliers = zeros(n1,2);
    for j =1:n1
        if residuals(j)< inlierThreshold
            count = count +1;
%             inliers (j,:) = pointsR_pro1(j,:);
        end
    end

    inlierPercent = count./n1;

    if inlierPercent_Max <inlierPercent
        inlierPercent_Max = inlierPercent;
        M_opti = M;

    end


end

%get optimized projective 100 points
pointLL = zeros(3,n1);
pointLL = [MatchPointsL';ones(1,n1)];
ProR_opt = M_opti*pointLL;
ProR_opt =
[ProR_opt(1,:)./ProR_opt(3,:);ProR_opt(2,:)./ProR_opt(3,:)]';

error_x = ProR_opt(:,1)-MatchPointsR(:,1);
error_y = ProR_opt(:,2)-MatchPointsR(:,2);

distance = sqrt(error_x.*error_x + error_y.*error_y);
MatchPointsL_opt = MatchPointsL;
for k =1:n1
    if distance(k)> inlierThreshold
        ProR_opt (k,:) = [0,0];
        MatchPointsL_opt(k,:)= [0,0];
        distance (k) = 0;
    end
end

sum = 0;
```

```
num = 0;
for n = 1:size(distance)
    if distance(n) ~=0
        sum = sum + distance(n);
        num = num+1;
    end
end
average_residual = sum./num;
```

References

Dey, S. (2018, Feb 28). *Implementing Lucas-Kanade Optical Flow algorithm in Python.* Retrieved from Data Science Central: https://www.datasciencecentral.com/profiles/blogs/implementing-lucas-kanade-optical-flow-algorithm-in-python

Matlab. (2015). Retrieved from http://home.deib.polimi.it/boracchi/teaching/IAS/Stitching/stitch.html

Urtasun, R. (2013, Jan 29). Retrieved from Computer Vision: Cameras: https://www.cs.toronto.edu/~urtasun/courses/CV/lecture07.pdf