Location Mapping with 360 Photos

Indiana University Purdue University Indianapolis

Jaskarn Judge and Veerpartap Singh

Location Mapping with 360 Photos

Abstract

With recent technological advancement of modern science people are now expecting to be able to use GPS location tracking anywhere they are. While these demands might not seem to far out, GPS tracking is not able to pinpoint accurate location while you are indoors. With this project, the aim is to implement a method to use 360 photos to create a database of a building to use for location mapping. With the use of DFT, FFT, SURF, and SHIFT we were able to train using the 360 photos to be able to cluster images based on building layout. By this implementation the user would be able to take a picture of their location and the system would be able to pinpoint them on a layout of the building.
Keywords: GPS, 360, DFT, FFT, SURF, SHIFT, Location

Location Mapping with 360 Photos

Introduction

The use of GPS tracking is increasing everyday. It has been easier and easier to take out your phone and access apps such as Google Maps and Apple maps for all your direction needs. The problem now is when you lose GPS tracking due to limitation of signal not being able to go through buildings. At this point those apps can't do much and base there tracking from offline data of how much a user has moved. When you are looking on Google Maps when you lose GPS signal you see the pinpoint location circle get bigger. As you are not able to be pinpointed to an exact location. Now if you are in a building that you have not visited before or not know very well you probably have to ask for directions. Now what could happen is by the time you get help you end up on the wrong side of the building or there person might not even know them self where you are trying to go. With this problem in our head we try to solve it using few new methods which we learned

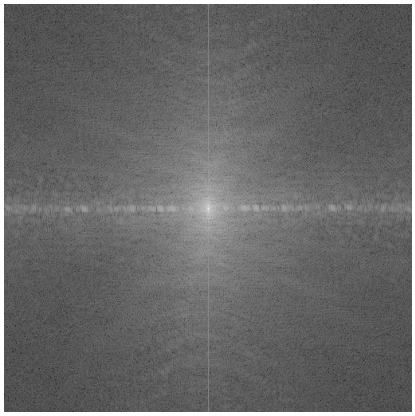
The objective of the project is to create a tool, which can detect where the user is standing within a building. The way this would work is the user takes a photo of where they are standing, it can be a picture of whatever is by them(ex: door, hall, etc). That picture would then be sent to the machine which processes it and the user would get a map with a location of where they are at. Then what the user can do is find there way around using that map or they can put in a room number/name and it would load up a route for them to take. The final objective would be to make all this happen in real time in the palm of anyone's hand through an app.

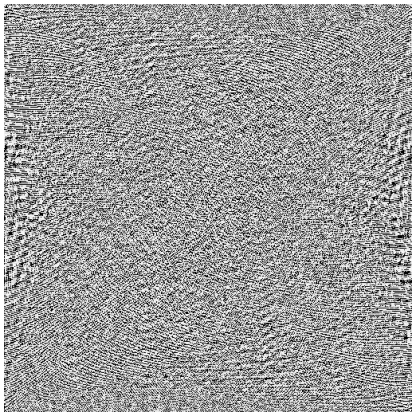
This research is important as more and more people are moving into the technology world. There is a limit on what other technologies such as GPS can do for us. With the help of tools like this people could find there way without every feeling lost. Business or people could use a tool like this to help people navigate around. One example that comes to the mind is the store IKEA. If a tool like this could get going company like IKEA can use it to help people navigate around there hige stores without the person feeling trapped. With more advancements Airports can use technology like this to help a passenger full navigate the airport without every looking at the map.

There are some new technologies coming out from Google which uses AR to show direction while walking. This has not been fully tested yet. Other technologies that we have seen are the use of Unity to map out a building and make a navigation out of it. With this there are certain starting points and it just lays out your destination on a map.

Method and Implementation

We tried many methods to achieve our goal. Many different codes were used trying to implement the location mapping with 360 photos. First, we went around the IT building and took 360 pictures with Samsung 360 Gear. The Samsung 360 Gear is a camera made by Samsung in 2017 and it was ahead of its time, an innovation that shaped the 360 community. It has 2 lenses that take an image simultaneously, which then the Samsung software stitches them together forming a seamless 360 image. We gave that software multiple tries to stitch our images, but they all resulted in failure. I tired multiple time to install the software but kept getting an error. The error was "could not complete installation due to visual redistribution package 2013." After a couple of day of head scratching and yelling at our computers Veer finally got it installed. We took multiple images within a parameter of building slowly covering the whole building. We then clustered the imaged based off of location and similarities within that location. Despite getting the software installed we couldn't get the images to stitch for some odd reason. We even reached out to Samsung, but they didn't have an answer for why the images were taking so long to stitch, it took about 30-40 minutes per image. When we tried to stitch images manually we were not getting a proper image that we needed. We needed an image that concave out to help get edges. When all else failed we just ended up letting the images run over night and the next morning they were fully stitched and ready to proceed to the next step. Once all the images were in clusters and separated, we began to run them through DFT code using MATLAB. Our original code was really slow on the full-size image. We let it run for about 2-3 hours to compile the DTF spectrum, but the computer ended up crashing. We tried to run it with a smaller image size. We compressed the image to 500px and the code completed the DFT in 1-1/2 hours, which is extremely long to even try to test with the rest of the images and it isn't optimal for our use which should be able to map the image to a location. Below are the output images of the code we ran. They look very noisy and are not very meaningful. One of the images is a DTF, the other we have no clue. Seems like it is a phase shift but we are not fully sure.



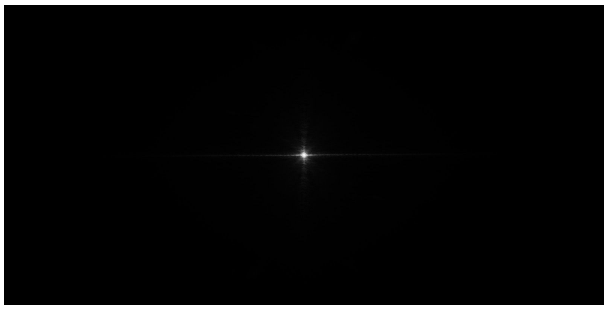


Code used in the first implementation of our image processing.

```
% program to find the 2D dft
clear all;
[filename, pathname, filterindex] = uigetfile( ...
{  '*.jpg','JPEG (*.jpg)'; ...
   '*.bmp','Windows Bitmap (*.bmp)'; ...
   '*.fig','Figures (*.fig)'; ...
   '*.*',  'All Files (*.*)'}, ...
   'Choose image(s) to be processed', ...
   'MultiSelect', 'off');
if filterindex==0, end
filename=cellstr(filename);
im2= imread(horzcat(pathname,char(filename)));
im1=rgb2gray(im2);
im3=im2double(im1);
[n,m]=size(im3);
c1=0;
h = waitbar(0,'Calculating DFT please wait.....');
k=1;l=1;
for l=0:1:m-1
    for k=0:1:n-1
        for x=0:1:n-1
            for y=0:1:m-1
                a=x+1;b=y+1;
                c= im3(a,b) * exp(-1i*2*pi*(k*x/n + l*y/m));
                c1=c1+c;
            end
        end
        aa=l+1;bb=k+1;
        im(bb,aa)=c1;
        c1=0;
    end
    waitbar(l / m);
end
ims = im*255;
close(h)
imshow(ims);title('dft plot');
% figure
d=ifft2(im);
figure
imshow(log(abs(ims)),[-1 5]); colormap(jet); colorbar;title('absolute value of dft plot');
```

Location Mapping with 360 Photos

After struggling with the old code, it was time to rewrite the code to better optimize the image processing for faster compilation times, but the result was not so promising. After some trial and error, we got a something running but the output was mostly black with a bright small center. The output had a lot of lost details and could not be used. The image on the top is the original image and the bottom one is the output. It is very dark and had little to none details. To get the output I had to put in some normalization which could be the reason the image is mostly dark. I tried to take out the normalization but that resulted in a completely white image. We compressed our images down to 3000x1500x as well as 2000x1000 but results where not any better,



The code used to get the output.

```
clear all;

for k = 1:18

    img = sprintf("%d.jpg", k);
    img_dft = sprintf("%d_dft_test5.jpg", k);
    I = imread(img);
    I = rgb2gray(I);
    [r , c] = size(I);
    for i = 1:r
            X(i,:) = fft(I(i,:));
    end
    for j = 1:c
            Y(:,j) = fft(X(:,j));
    end
    M = Y;
    M = fftshift(M);
    Ab = abs(M);
    Ab = (Ab - min(min(Ab)))./(max(max(Ab))).*255;
    imwrite(Ab,img_dft);
    clearvars -except k;
end
```

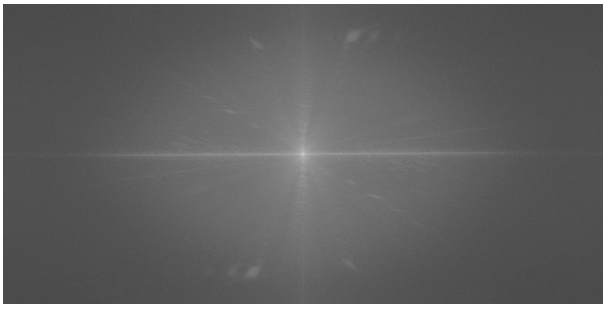
This code basically clears all previous variables and runs a loop from 1 to 18. I was processing 18 images at a time without clusters to see what kind of results i was getting from the DTF. I had variables set for the

input image and the output so it is easier to modify. The image is then read and converted to grayscale to run the fourier transformation. Then the size of the image is determined. Then the image is run in the fft built in function from matlab with the x and the y coordinates of the image based on the size. The the fft shift is applied and finally normalized to get the output. Normalization is required because of the imwrite function of matlab it is restricted between 0 and 1 anything above those values will result in either a white or black image.

With that we need to get better results for the DTF and we need to improve the code or start over. We decided to re write the code without looping through the pixels one by one but instead just read and process the image as a whole. When we did that we got better and more consistent results. We actually got a proper DTF output with the full image size compared to the first implementation of the code we did where we had to shrink the image size down to 500px and it still took a really long time to process. With the new code the time to process the full size image was cut down to 30-40 seconds. I will be showing the same image results for this code that was used in the last one. I was really happy with the output but we still needed to make sure it is useful. Even though the result looked good does not mean it is meaningful to our end goal.

The code used to get the output.

```
clear all;

for k = 1:4

    img = sprintf("Cluster/Cluster A/%d.jpg", k);
    img_dft = sprintf("Cluster/Cluster A/%d_dft.jpg", k);
    I = imread(img);
    I = rgb2gray(I);
    F = fft2(I);
    S = abs(F);
    Fcenter = fftshift(F);
    S2 = log(1+abs(Fcenter));
    S2=(S2-min(S2(:)))./max(S2(:));
    imwrite(S2,img_dft);
end
```



Location Mapping with 360 Photos

This code start the same way as the previous one but it was implemented with the clusters instead of a sample of 18 images. Same as last time it converts the image to grayscale to be used with the built in fft function from matlab but the only difference is it runs the fft on the whole image instead of by pixels. We needed to normalize the results because of the way imwrite function behaves in matlab it disregards any value greater than 1 or lower than 0 resulting in a white or black image.

After the completion of the DTF we needed to compare the clusters with each other to see if they have a definable value that we can use to determine the relative location with each other compared to other clusters. For that we used a basic correlation function that is built into matlab. This code results in a table comparing the dtf images within the cluster and their correlation.

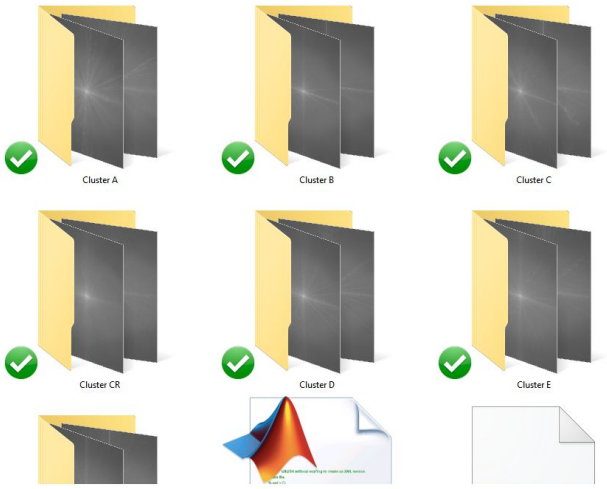
The code used for the correlation

```
clear all

for i = 1:4
   for k = 1:4
      img_a = sprintf("Cluster/Cluster A/%d_dft.jpg",i);
      img_b = sprintf("Cluster/Cluster A/%d_dft.jpg",k);
      a = imread(img_a);
      b = imread(img_b);
      c = corr2(a,b);
      cor{i,k} = c;
   end
end

fileID = fopen('Cluster/TableA.dat','w');
formatSpec = '%f %f %f %f\n';
[nrows,ncols] = size(cor);
for row = 1:nrows
   fprintf(fileID,formatSpec,cor{row,:});
end
fclose(fileID);
```

This is the result of the output table.

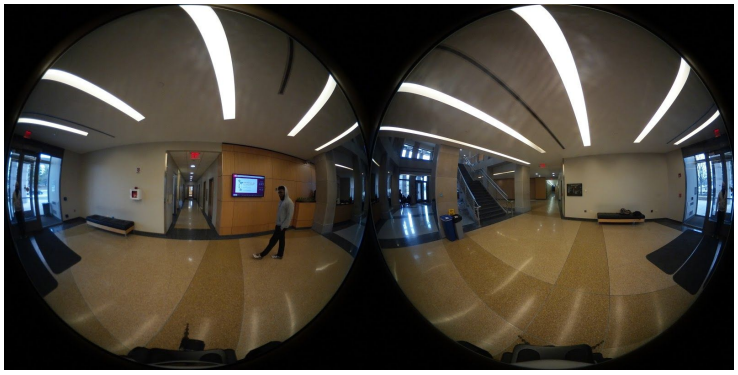
1.000000	0.778817	0.772254	0.773677
0.778817	1.000000	0.776421	0.775290
0.772254	0.776421	1.000000	0.77996
0.773677	0.775290	0.77996	1.000000

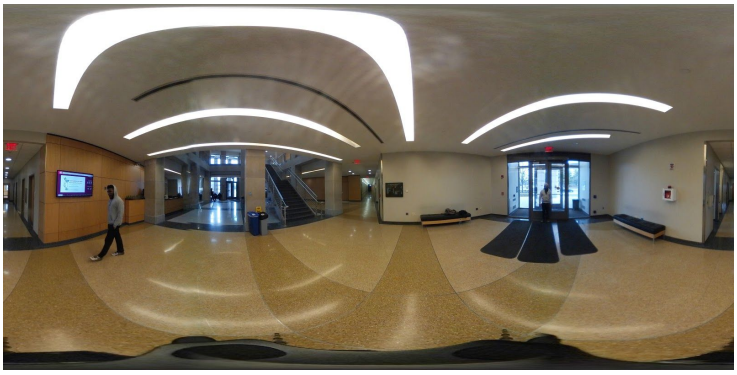
All of these tests were run on Windows 10 machines using matlab to help us process the images. It was easier to use a windows device rather than any other devices because we both have Windows desktops at home which run 24/7 so we could run tests over night and have the results in the morning while having more power than our laptops. Matlab is a very helpful tool that is great for image manipulation and many other things. It has a lot of built in function that we could use instead of writing our own which helped us prevent buffer overflow issues.



Implementation and Experiments

For the implementation we gathered all of the images we took and created cluster. After the clusters were created we ran them in the matlab code to process their DTF's so the machine can understand the images to create a general understanding of the local location of the clusters. With that information the computer can determine how meaningful the image data that are close to each other is compared to the other clusters. As for some of the images we have taken with the

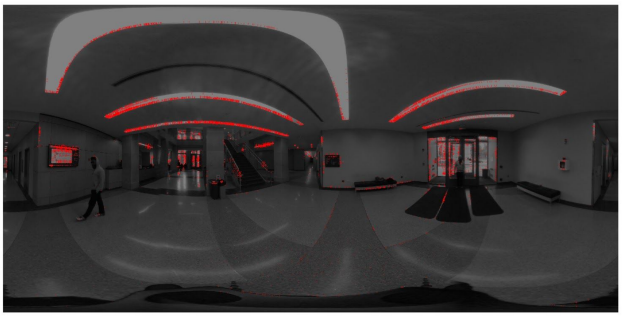




camera they were first 2 circular images which were expanded into a panoramic 360 view so we are able to view them in a meaningful manner.

Some major problem we are facing are not knowing if the data output from the DTF is meaningful or not. The numbers look very similar to each other and each cluster. We are still doing more test to determine if it is possible for the computer to recognize each cluster as a separate location and if it can match an uploaded image to its respective location. Since we were not sure if the data was good we also experimented with SIFT to match locations. We found some previous code on SIFT as there was nothing on the Matlab website and found that it was not a built in function within Matlab. An alternative we found was SURF. SURF was used because the professor who implemented SIFT patented it and the code isn't easily accessible whereas SURF is a built in function in matlab.

```
clear all; close all;
I = imread('lena.jpg');
I = vl_impattern('lena') ;
I = single(rgb2gray(I)) ;
[f,d] = vl_sift(I) ;
perm = randperm(size(f,2)) ;
sel = perm(1:50) ;
h1 = vl_plotframe(f(:,sel)) ;
h2 = vl_plotframe(f(:,sel)) ;
set(h1,'color','k','linewidth',3) ;
set(h2,'color','y','linewidth',2) ;
h3 = vl_plotsiftdescriptor(d(:,sel),f(:,sel)) ;
set(h3,'color','g') ;
```



Once we got some limits set our images came out much better. The main difference between the pictures is that the total amount of keypoints was reduced from few thousand to the top 100. With this all the key points where not on top of each other as the picture is 6000x4000.

```
clear;
clc;
image = imread('images/testimage.jpg');
image = rgb2gray(image);
image = double(image);
keyPoints = SIFT(image,9,5,1.3);
image = SIFTKeypointVisualizer(image,keyPoints);
imshow(uint8(image))
```

As we tested picture matching with different images of same area and random areas, we noticed this tool would not be useful with the 360 pictures we are using. Results showed many parts matching with lot of different photos but there was not enough points to show different locations. For example most of the keypoints selected by SURF in the picture above with green circles ended up either matching with random points on other images of none at all with images in same area. As we are learning more about this tool and setting more parameters the results seem to get better. As time was limited for when this tool started being used, some more testing with normal pictures will have to wait.

Conclusion

In this project we took 360 pictures of the IT building with a 360° camera. We took about 3-4 images in small sections to help identify locations easier also to help break them into clusters. The reason for having a lot of images is that if one image fails to locate a area of the build the others will have different perspective therefore be able to match. After the pictures were taken they were grouped into clusters based on their location and similarities. All of the 360 image were then converted into 360 panoramic image to make it easier to manipulate with matlab. This was a very difficult process because the given samsung software failed to provide the results we were after. Then when we did get to a decent output there was a painstaking wait time. We ended up letting the software run over night to process all of the images. Once all of the images were converted to the panoramic format the next step wast to prep them for the location matching. After a series of trial and error we managed to process them with the Discrete Fourier Transformation or DTF for short. The DTF outputs gave us some number which we don't know are meaningful or not. But we kept going and slowly we started to make tables matching imaged within cluster. After matching imaged within cluster we tried to match cluster with each other and the data was too similar to define a difference between them. That forced us to shift from DFT to a different approach. We were given a idea of SIFT to match significant points within a image to a different image to find similarities. Doing some research into SIFT we realized it was not possible to implement because it was patented by the professor who created it and was not easy to use with matlab. Then we turned our heads to SURF which is similar to SIFT but not as reliable. SURF was a built in function in matlab and could be implemented with ease. Since started to implement SURF we will be doing more testing and we plan on continue to work on this project even after the end of this class. We are confident that we will be able to make SURF work with our end goal and be able to match location with a simple click of a button. Only thing we have left to implement is the matching of the images. Because of the difficulties we faced in the beginning of the project and the complication with DFT we don't have a fully working project but we

haven't given up yet. We are going to continue working on this project this summer to have a fully working project on our resume.

Appendix

Programs used for this assignments were: Matlab R2018A and Samsung Wear App. All the codes are runnable through Matlab as is, for some of the SURF codes you might need Matlab Computer Vision Toolbox.