MIN-VIS-2016

ASSIGNMENT 10 – GROUP4

Group Members:

- Sandio Fongang R
- Armaan Rustami

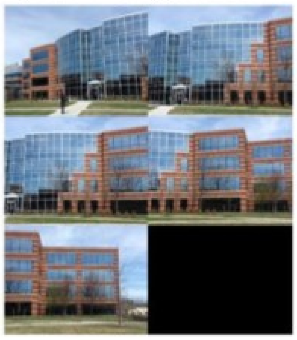
MINES-VIS Assignment 11 Feature based Panoramic Image Stitching.

Feature detection and matching are powerful techniques used in many computer vision applications such as image registration, tracking, and object detection. In this assignment , feature based techniques should be used to automatically stitch together a set of images. The procedure for image stitching is an extension of feature based image registration.
Instead of registering a single pair of images, multiple image pairs are successively registered relative to each other to form a panorama.

To create the panorama, first start with registering successive image pairs by using the following procedure:

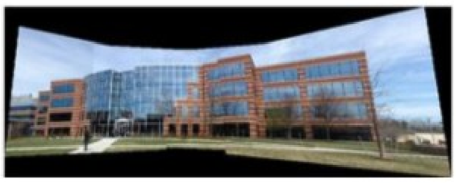
1. $Detect\ and\ match\ features\ between\ I(n)\ and\ I(n-1)$
2. $Estimate\ the\ geometric\ transformation\ T(n), that\ maps\ I(n)\ to\ I(n-1)$
3. $Compute\ the\ transformation\ that\ maps\ I(n)\ into\ the\ panorama\ image\ as\ T(1) * T(2)\ \dots.$
$* T(n-1) * T(n)$

Also create an initial empty image ("panorama") into which all images are mapped.
Depicted: the images to be stitched. See also "buildings.rar"on SharePoint.



OpenCV has high level class such as Stitcher you might use his methods as long you can show the steps as described above in the procedure. It's recommend to study this class to become more familiar with the theory. See also Stitcher pipeline.

The following image is an example of the result you might get:

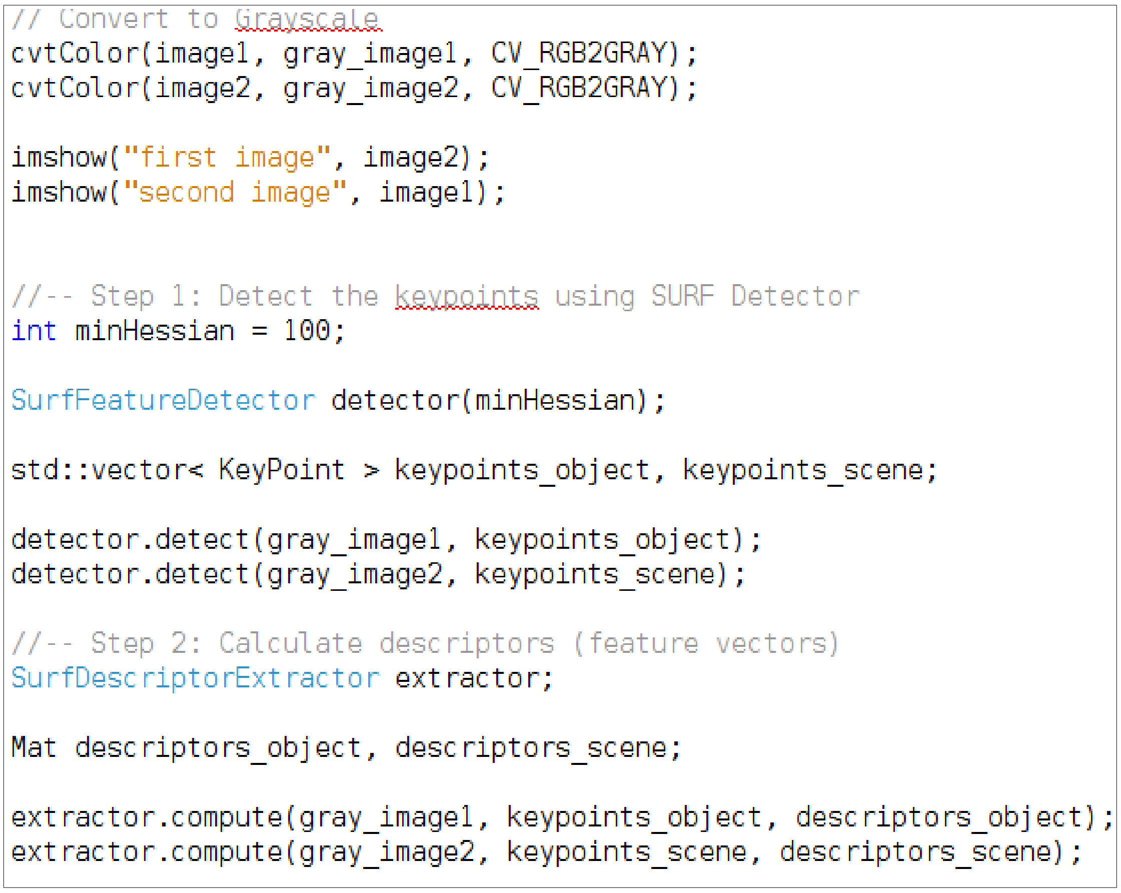


2. Solution

a. **Feature based Panoramic Image Stitching:**

In order to stitch images base on the matches and make Panoramic Image from it we tried to make the images into 3 blocks, which means each block is stitched of two images,, here are some Steps we took:.

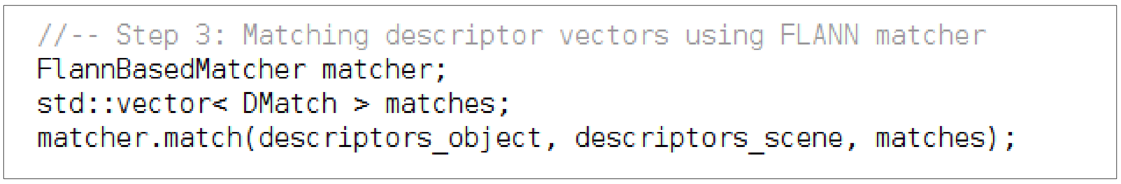
- Load the images and convert
- Detect keypoints using SurfFeatureDetector on Grayscale images based on specific number of key Point . we use the concept of “Detect and match features between I(n) and I(n − 1)” and Calcuate and Compute Descriptor using keyPoints .*Figure1*



```
// Convert to Grayscale
cvtColor(image1, gray_image1, CV_RGB2GRAY);
cvtColor(image2, gray_image2, CV_RGB2GRAY);

imshow("first image", image2);
imshow("second image", image1);


//-- Step 1: Detect the keypoints using SURF Detector
int minHessian = 100;

SurfFeatureDetector detector(minHessian);

std::vector< KeyPoint > keypoints_object, keypoints_scene;

detector.detect(gray_image1, keypoints_object);
detector.detect(gray_image2, keypoints_scene);

//-- Step 2: Calculate descriptors (feature vectors)
SurfDescriptorExtractor extractor;

Mat descriptors_object, descriptors_scene;

extractor.compute(gray_image1, keypoints_object, descriptors_object);
extractor.compute(gray_image2, keypoints_scene, descriptors_scene);
```

Figure 1

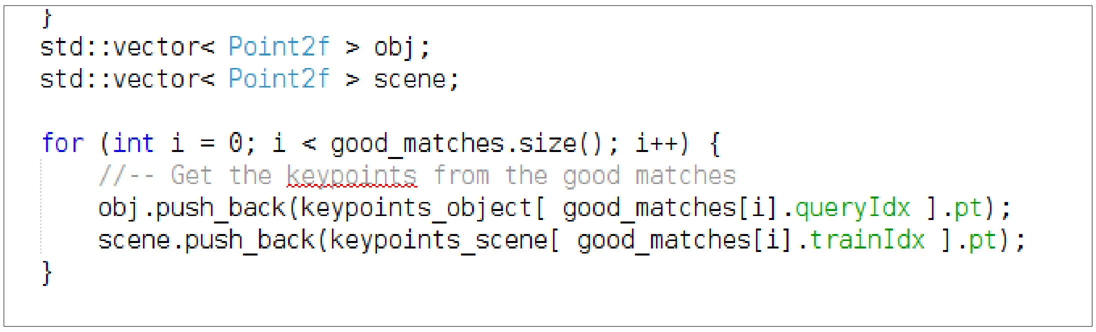
- After getting the descriptors we try to get matches using FlannBasedMatcher OpenCV function.



```
//-- Step 3: Matching descriptor vectors using FLANN matcher
FlannBasedMatcher matcher;
std::vector< DMatch > matches;
matcher.match(descriptors_object, descriptors_scene, matches);
```

Figure 2

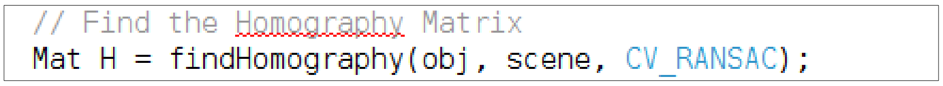
- Get keypoint from the good Matches.



```
}
std::vector< Point2f > obj;
std::vector< Point2f > scene;

for (int i = 0; i < good_matches.size(); i++) {
    //-- Get the keypoints from the good matches
    obj.push_back(keypoints_object[ good_matches[i].queryIdx ].pt);
    scene.push_back(keypoints_scene[ good_matches[i].trainIdx ].pt);
}
```

Figure 3

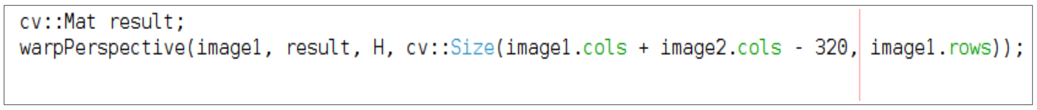
- Used the function *findHomography* to find the transform between matched keypoints



```
// Find the Homography Matrix
Mat H = findHomography(obj, scene, CV_RANSAC);
```

Figure 4

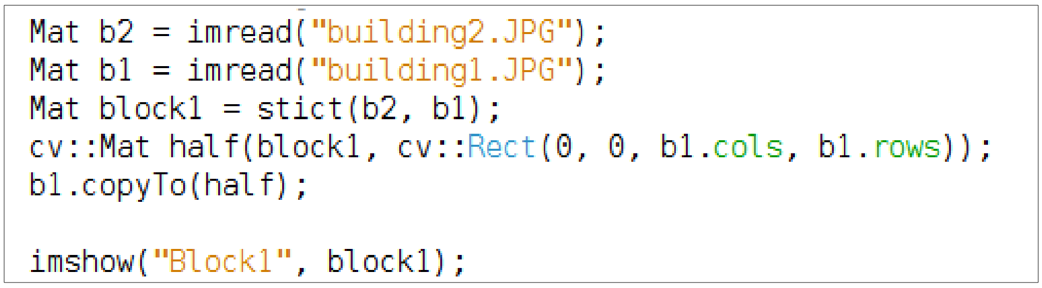
- After all we used WrapPerspective OpenCV function to transform the source image using the findHomography matrix result and use sum of first image and second image columns as Width and rows size of right image as Height of WrapPerspective Size.



```
cv::Mat result;
warpPerspective(image1, result, H, cv::Size(image1.cols + image2.cols - 320, image1.rows));
```

Figure 5

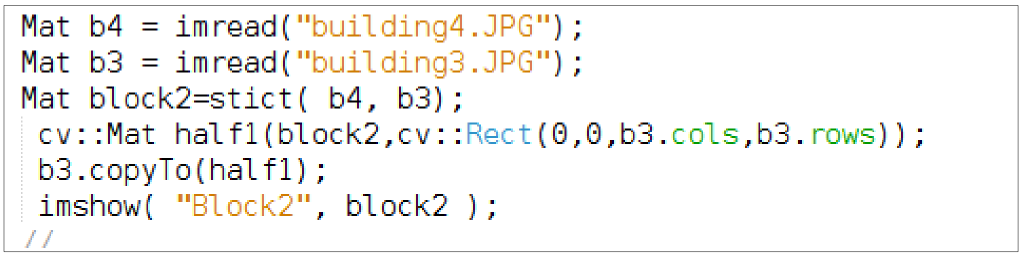
- Create Mat variable assigning the result of WrapPerspective with rectangle using Height and width size of left image for the rectangle and copy the leftImage to the created Mat, where it makes the WrapPerspective restult stitched with left image and make them as one image where we called it as Block in this assignment



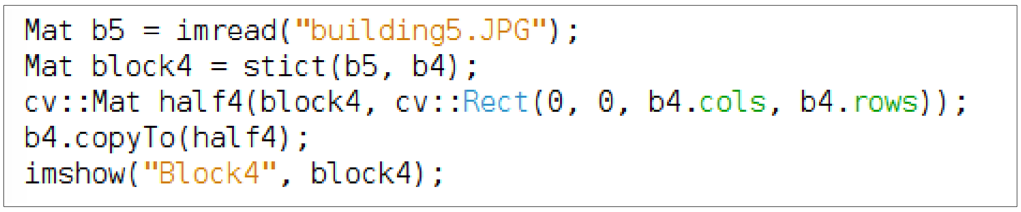
```
Mat b2 = imread("building2.JPG");
Mat b1 = imread("building1.JPG");
Mat block1 = stict(b2, b1);
cv::Mat half(block1, cv::Rect(0, 0, b1.cols, b1.rows));
b1.copyTo(half);

imshow("Block1", block1);
```

Figure 6

- So the same we applied the mentioned above methods for other images also to get one block from each two images.



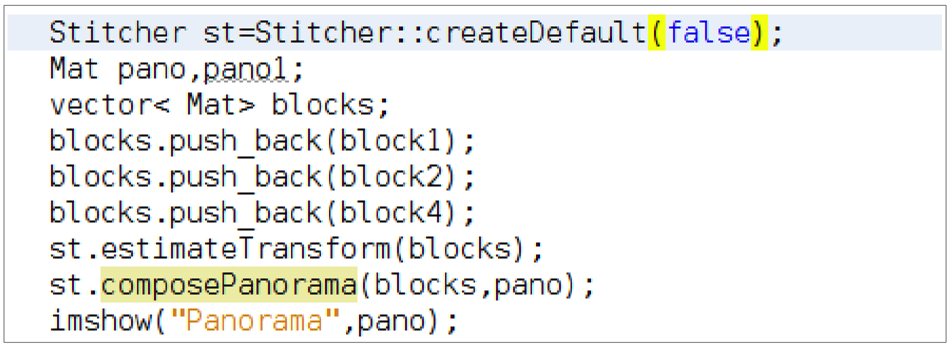
```
Mat b4 = imread("building4.JPG");
Mat b3 = imread("building3.JPG");
Mat block2=stict( b4, b3);
 cv::Mat half1(block2,cv::Rect(0,0,b3.cols,b3.rows));
 b3.copyTo(half1);
 imshow( "Block2", block2 );
//
```



```
Mat b5 = imread("building5.JPG");
Mat block4 = stict(b5, b4);
cv::Mat half4(block4, cv::Rect(0, 0, b4.cols, b4.rows));
b4.copyTo(half4);
imshow("Block4", block4);
```

Figure 7

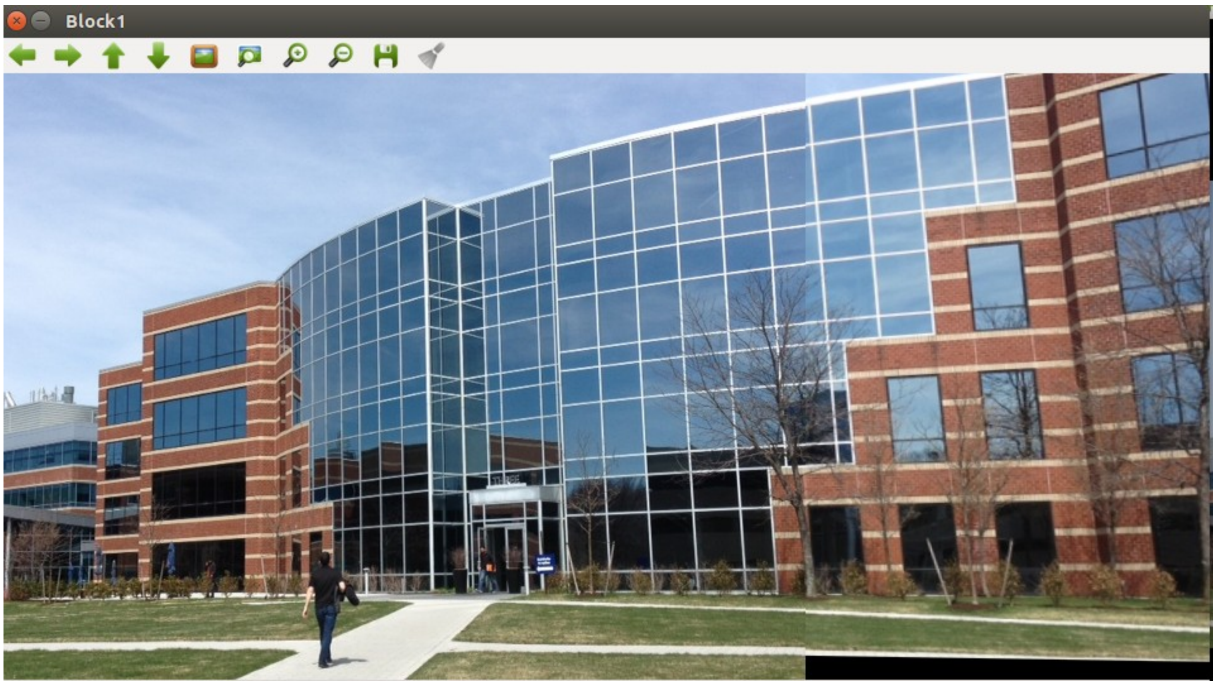
- Finally to get Panorama Image we used estimateTransform which the one of the Stitcher Functions, is used to match the given images and to estimate rotations And we used composePoanorama function which is used to compose the given images into the final pano under the assumption that the image transformations were estimated before.

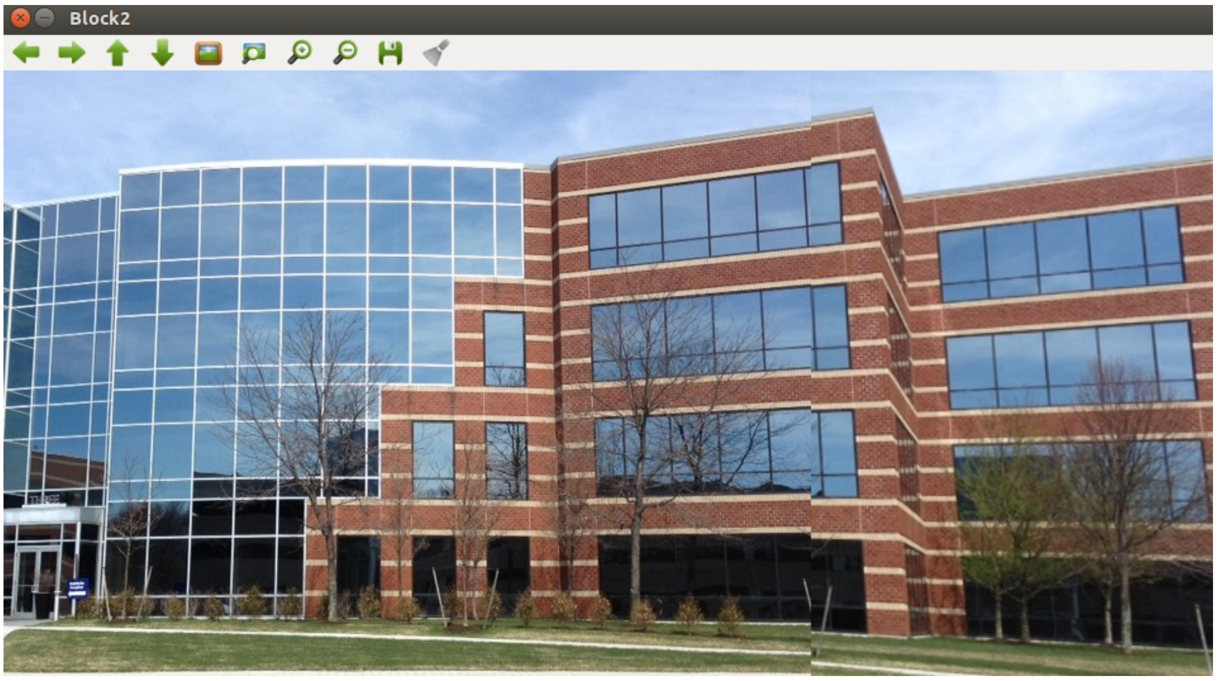


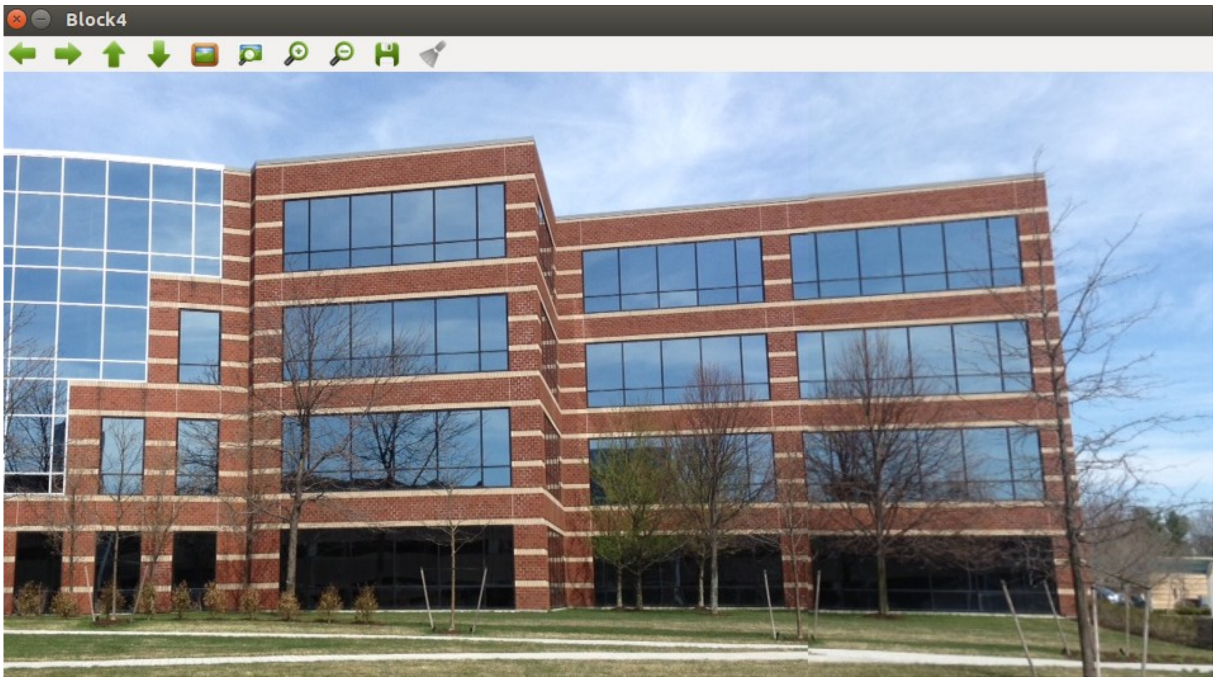
```
Stitcher st=Stitcher::createDefault(false);
Mat pano,pano1;
vector< Mat> blocks;
blocks.push_back(block1);
blocks.push_back(block2);
blocks.push_back(block4);
st.estimateTransform(blocks);
st.composePanorama(blocks,pano);
imshow("Panorama",pano);
```

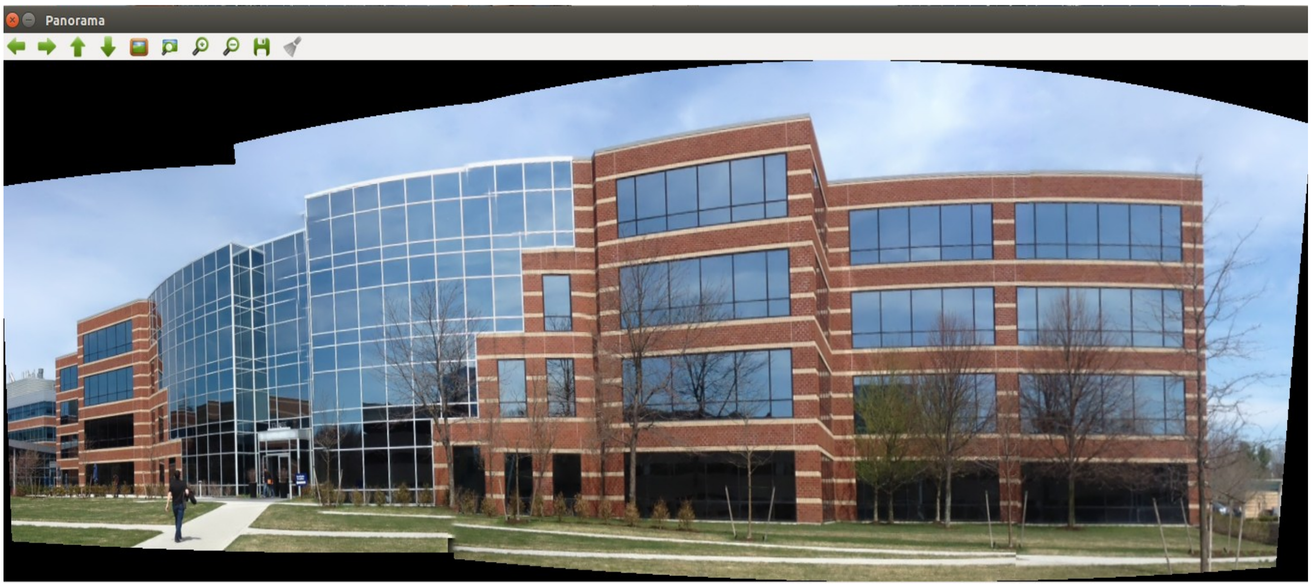
Figure 8

Output:






Block4


Panorama