In [23]:

```
import cv2
import numpy as np


def importData(fileName, imageDirectory):
    '''
    :param fileName: Name of the pose data file in string form e.g. "da
tasets/imageData.txt"
    :param imageDirectory: Name of the directory where images arer stor
ed in string form e.g. "datasets/images/"
    :return: dataMatrix: A NumPy ndArray contaning all of the pose dat
a. Each row stores 6 floats containing pose information in XYZYPR form
        allImages: A Python List of NumPy ndArrays containing images.
    '''

    allImages = [] #list of cv::Mat aimghes
    dataMatrix = np.genfromtxt(fileName,delimiter=",",usecols=range(1,7
),dtype=float) #read numerical data
    fileNameMatrix = np.genfromtxt(fileName,delimiter=",",usecols=[0],d
type=str) #read filen name strings
    print(fileNameMatrix)
    for i in range(0,fileNameMatrix.shape[0]): #read images
        allImages.append(cv2.imread(imageDirectory+fileNameMatrix[i]))
    return allImages,dataMatrix

def display(title, image):
    '''
    OpenCV machinery for showing an image until the user presses a key.
    :param title: Window title in string form
    :param image: ndArray containing image to show
    :return:
    '''

    cv2.namedWindow(title,cv2.WINDOW_NORMAL)
    cv2.resizeWindow(title,1920,1080)
    cv2.imshow(title,image)
```

```
    cv2.waitKey(400)
    cv2.destroyWindow(title)

def drawMatches(img1, kp1, img2, kp2, matches):
    """
    Makes an image with matched features denoted.
    drawMatches() is missing in OpenCV 2. This boilerplate implementati
on taken from http://stackoverflow.com/questions/20259025/module-object
-has-no-attribute-drawmatches-opencv-python
    """

    # Create a new output image that concatenates the two images togeth
er
    # (a.k.a) a montage
    rows1 = img1.shape[0]
    cols1 = img1.shape[1]
    rows2 = img2.shape[0]
    cols2 = img2.shape[1]

    out = np.zeros((max([rows1,rows2]),cols1+cols2,3), dtype='uint8')

    # Place the first image to the left
    out[:rows1,:cols1] = np.dstack([img1, img1, img1])

    # Place the next image to the right of it
    out[:rows2,cols1:] = np.dstack([img2, img2, img2])

    # For each pair of points we have between both images
    # draw circles, then connect a line between them
    for m in matches:

        # Get the matching keypoints for each of the images
        img1_idx = m.queryIdx
        img2_idx = m.trainIdx

        # x - columns
        # y - rows
        (x1,y1) = kp1[img1_idx].pt
        (x2,y2) = kp2[img2_idx].pt
```

```
        # Draw a small circle at both co-ordinates
        radius = 8
        thickness = 3
        color = (255,0,0) #blue
        cv2.circle(out, (int(x1),int(y1)), radius, color, thickness)
        cv2.circle(out, (int(x2)+cols1,int(y2)), radius, color, thickne
ss)

        # Draw a line in between the two points
        cv2.line(out, (int(x1),int(y1)), (int(x2)+cols1,int(y2)), color
, thickness)

    # Also return the image if you'd like a copy
    return out
```

In [24]:

```
import numpy as np
import cv2
import math as m

def computeUnRotMatrix(pose):
    '''
    See http://planning.cs.uiuc.edu/node102.html. Undoes the rotation o
f the craft relative to the world frame.
    :param pose: A 1x6 NumPy ndArray containing pose information in [X,
Y,Z,Y,P,R] format
    :return: A 3x3 rotation matrix that removes perspective distortion
 from the image to which it is applied.
    '''
    a = pose[3]*np.pi/180 #alpha
    b = pose[4]*np.pi/180 #beta
    g = pose[5]*np.pi/180 #gamma
    #Compute R matrix according to source.
    Rz = np.array(([m.cos(a), -1*m.sin(a),    0],
                   [m.sin(a),    m.cos(a),    0],
                   [       0,           0,     1]))

    Ry = np.array(([ m.cos(b),           0,     m.sin(b)],
                   [        0,           1,           0],
                   [-1*m.sin(b),           0,     m.cos(b)]))
```

```
    Rx = np.array([        1,           0,           0],
                   [        0,    m.cos(g),  -1*m.sin(g)],
                   [        0,    m.sin(g),     m.cos(g)]))
    Ryx = np.dot(Rx,Ry)
    R = np.dot(Rz,Ryx) #Care to perform rotations in roll, pitch, yaw o
rder.
    R[0,2] = 0
    R[1,2] = 0
    R[2,2] = 1
    Rtrans = R.transpose()
    InvR = np.linalg.inv(Rtrans)
    #Return inverse of R matrix so that when applied, the transformatio
n undoes R.
    return InvR

def warpPerspectiveWithPadding(image,transformation):
    '''
    When we warp an image, its corners may be outside of the bounds of
 the original image. This function creates a new image that ensures thi
s won't happen.
    :param image: ndArray image
    :param transformation: 3x3 ndArray representing perspective trransf
ormation
    :param kp: keypoints associated with image
    :return: transformed image
    '''

    height = image.shape[0]
    width = image.shape[1]
    corners = np.float32([[0,0],[0,height],[width,height],[width,0]]).r
eshape(-1,1,2) #original corner locations

    warpedCorners = cv2.perspectiveTransform(corners, transformation) #
warped corner locations
    [xMin, yMin] = np.int32(warpedCorners.min(axis=0).ravel() - 0.5) #n
ew dimensions
    [xMax, yMax] = np.int32(warpedCorners.max(axis=0).ravel() + 0.5)
    translation = np.array(([1,0,-1*xMin],[0,1,-1*yMin],[0,0,1])) #must
translate image so that all of it is visible
```

```
    fullTransformation = np.dot(translation,transformation) #compose warp and translation in correct order
    result = cv2.warpPerspective(image, fullTransformation, (xMax-xMin, yMax-yMin))
    return result
```

In [25]:

```
import cv2
import numpy as np
import copy

class Combiner:
    def __init__(self,imageList_,dataMatrix_):
        '''
        :param imageList_: List of all images in dataset.
        :param dataMatrix_: Matrix with all pose data in dataset.
        :return:
        '''
        self.imageList = []
        self.dataMatrix = dataMatrix_
        detector = cv2.ORB()
        for i in range(0,len(imageList_)):
            image = imageList_[i][::2,::2,:] #downsample the image to speed things up. 4000x3000 is huge!
            M = gm.computeUnRotMatrix(self.dataMatrix[i,:])
            #Perform a perspective transformation based on pose information.
            #Ideally, this will mnake each image look as if it's viewed from the top.
            #We assume the ground plane is perfectly flat.
            correctedImage = gm.warpPerspectiveWithPadding(image,M)
            self.imageList.append(correctedImage) #store only corrected images to use in combination
        self.resultImage = self.imageList[0]
    def createMosaic(self):
        for i in range(1,len(self.imageList)):
            self.combine(i)
        return self.resultImage

    def combine(self, index2):
```

```
        '''
        :param index2: index of self.imageList and self.kpList to combi
ne with self.referenceImage and self.referenceKeypoints
        :return: combination of reference image and image at index 2
        '''

        #Attempt to combine one pair of images at each step. Assume the
order in which the images are given is the best order.
        #This intorduces drift!
        image1 = copy.copy(self.imageList[index2 - 1])
        image2 = copy.copy(self.imageList[index2])

        '''
        Descriptor computation and matching.
        Idea: Align the images by aligning features.
        '''
        detector = cv2.SURF(500) #SURF showed best results
        detector.extended = True
        gray1 = cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
        ret1, mask1 = cv2.threshold(gray1,1,255,cv2.THRESH_BINARY)
        kp1, descriptors1 = detector.detectAndCompute(gray1,mask1) #kp
 = keypoints

        gray2 = cv2.cvtColor(image2,cv2.COLOR_BGR2GRAY)
        ret2, mask2 = cv2.threshold(gray2,1,255,cv2.THRESH_BINARY)
        kp2, descriptors2 = detector.detectAndCompute(gray2,mask2)

        #Visualize matching procedure.
        keypoints1Im = cv2.drawKeypoints(image1,kp1,color=(0,0,255))
        util.display("KEYPOINTS",keypoints1Im)
        keypoints2Im = cv2.drawKeypoints(image2,kp2,color=(0,0,255))
        util.display("KEYPOINTS",keypoints2Im)

        matcher = cv2.BFMatcher() #use brute force matching
        matches = matcher.knnMatch(descriptors2,descriptors1, k=2) #fin
d pairs of nearest matches
        #prune bad matches
        good = []
        for m,n in matches:
            if m.distance < 0.55*n.distance:
```

```
                good.append(m)
        matches = copy.copy(good)

        #Visualize matches
        matchDrawing = util.drawMatches(gray2,kp2,gray1,kp1,matches)
        util.display("matches",matchDrawing)
        path='/content/drive/MyDrive/finalResult.png'

        #NumPy syntax for extracting location data from match data stru
cture in matrix form
        src_pts = np.float32([ kp2[m.queryIdx].pt for m in matches ]).r
eshape(-1,1,2)
        dst_pts = np.float32([ kp1[m.trainIdx].pt for m in matches ]).r
eshape(-1,1,2)

        '''
        Compute Affine Transform
        Idea: Because we corrected for camera orientation, an affine tr
ansformation *should* be enough to align the images
        '''
        A = cv2.estimateRigidTransform(src_pts,dst_pts,fullAffine=False
) #false because we only want 5 DOF. we removed 3 DOF when we unrotated
        if A == None: #RANSAC sometimes fails in estimateRigidTransform
(). If so, try full homography. OpenCV RANSAC implementation for homogr
aphy is more robust.
            HomogResult = cv2.findHomography(src_pts,dst_pts,method=cv2
.RANSAC)
            H = HomogResult[0]

        final=cv2.imread(path)
        '''
        Compute 4 Image Corners Locations
        Idea: Same process as warpPerspectiveWithPadding() excewpt we h
ave to consider the sizes of two images. Might be cleaner as a functio
n.
        '''
        height1,width1 = image1.shape[:2]
        height2,width2 = image2.shape[:2]
        corners1 = np.float32(([0,0],[0,height1],[width1,height1],[widt
h1,0]))
```

```
        corners2 = np.float32(([0,0],[0,height2],[width2,height2],[widt
h2,0]))
        warpedCorners2 = np.zeros((4,2))
        for i in range(0,4):
            cornerX = corners2[i,0]
            cornerY = corners2[i,1]
            if A != None: #check if we're working with affine transform
or perspective transform
                warpedCorners2[i,0] = A[0,0]*cornerX + A[0,1]*cornerY +
A[0,2]
                warpedCorners2[i,1] = A[1,0]*cornerX + A[1,1]*cornerY +
A[1,2]
            else:
                warpedCorners2[i,0] = (H[0,0]*cornerX + H[0,1]*cornerY
+ H[0,2])/(H[2,0]*cornerX + H[2,1]*cornerY + H[2,2])
                warpedCorners2[i,1] = (H[1,0]*cornerX + H[1,1]*cornerY
+ H[1,2])/(H[2,0]*cornerX + H[2,1]*cornerY + H[2,2])
        allCorners = np.concatenate((corners1, warpedCorners2), axis=0)
        [xMin, yMin] = np.int32(allCorners.min(axis=0).ravel() - 0.5)
        [xMax, yMax] = np.int32(allCorners.max(axis=0).ravel() + 0.5)

        '''Compute Image Alignment and Keypoint Alignment'''
        translation = np.float32(([1,0,-1*xMin],[0,1,-1*yMin],[0,0,1]))
        warpedResImg = cv2.warpPerspective(self.resultImage, translatio
n, (xMax-xMin, yMax-yMin))
        if A == None:
            fullTransformation = np.dot(translation,H) #again, images m
ust be translated to be 100% visible in new canvas
            warpedImage2 = cv2.warpPerspective(image2, fullTransformati
on, (xMax-xMin, yMax-yMin))
        else:
            warpedImageTemp = cv2.warpPerspective(image2, translation,
(xMax-xMin, yMax-yMin))
            warpedImage2 = cv2.warpAffine(warpedImageTemp, A, (xMax-xMi
n, yMax-yMin))
        self.imageList[index2] = copy.copy(warpedImage2) #crucial: upda
te old images for future feature extractions

        resGray = cv2.cvtColor(self.resultImage,cv2.COLOR_BGR2GRAY)
        warpedResGray = cv2.warpPerspective(resGray, translation, (xMax
```

```
-xMin, yMax-yMin))

        '''Compute Mask for Image Combination'''
        ret, mask1 = cv2.threshold(warpedResGray,1,255,cv2.THRESH_BINAR
Y_INV)
        mask3 = np.float32(mask1)/255

        #apply mask
        warpedImage2[:,:,0] = warpedImage2[:,:,0]*mask3
        warpedImage2[:,:,1] = warpedImage2[:,:,1]*mask3
        warpedImage2[:,:,2] = warpedImage2[:,:,2]*mask3

        result = warpedResImg + warpedImage2
        #visualize and save result
        self.resultImage = result
        util.display("result",result)
        cv2.imwrite("results/intermediateResult"+str(index2)+".png",res
ult)
        return result
```

In [35]:
```
import matplotlib.pyplot as plt
result=cv2.imread('/content/drive/MyDrive/results/finalResult.png')
plt.imshow(result)
```

Out[35]:
```
<matplotlib.image.AxesImage at 0x7f526ce7d410>
```