

Complete the claimed points and sections below.

Total Points Claimed [60] / 250

Core

- 1. Stitch two keyframes [20] / 20
- 2. Panorama using five key frames [15] / 15
- 3. Map the video to the reference plane [15] / 15
- 4. Create background panorama [0] / 15
- 5. Create background movie [0] / 10
- 6. Create foreground movie [0] / 15
- 7. Quality of results and report [10] / 10

B&W

- 8. Insert unexpected object [0] / 15
- 9. Process your own video [0] / 20
- 10. Smooth blending [0] / 30
- 11. Improved fg/bg videos [0] / 40
- 12. Generate a wide video [0] / 10
- 13. Remove camera shake [0] / 20
- 14. Make streets more crowded [0] / 15

1. Stitch two key frames

Image frame 270:



Image frame 450:



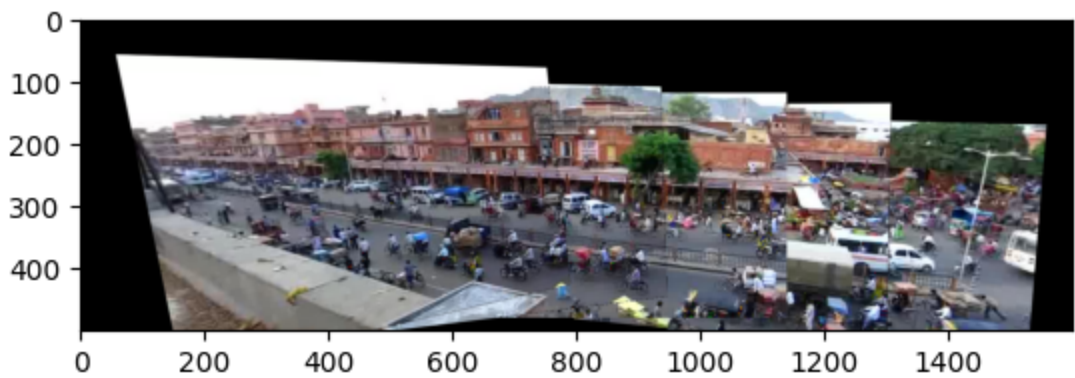
3x3 homography matrix normalized so that the largest value is 1:

```
[ [-4.88611548e-03 -2.52306298e-04 1.00000000e+00]
  [-4.33954263e-05 -4.67004011e-03 6.79006356e-02]
  [-1.63938229e-06 -1.84504112e-07 -4.08208908e-03]]
```

```
[ [ 1.00000000e+00  5.53208568e-02 -2.05219375e+02]
  [ 7.05586449e-03  9.63019973e-01 -1.46687656e+01]
  [ 3.51022884e-04  6.09244563e-05  8.26075654e-01]]
```

2. Panorama using five key frames

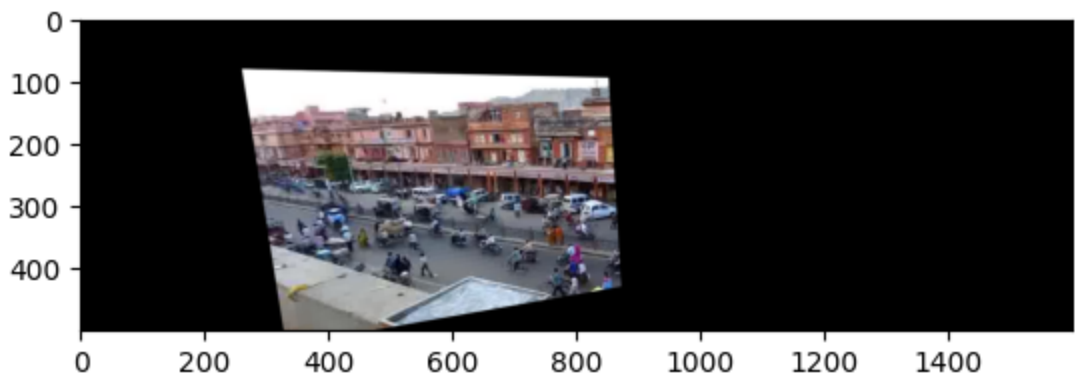
My panoramic image:



3. Map the video to the reference plane

Link to my video: <https://youtu.be/IIX-UfeDO0E>

Frame 200 of my video:



**Briefly explain how you solved for the transformation between each frame and the reference frame:**

I first identified the key frame closest to each of the 900 frames. Using this, I computed the homography between each frame and its closest key frame. To determine the transformation between these frames and the reference frame (computed in part 2), I multiplied the homography matrix for each frame and its closest key frame with the transformation matrix for the key frame and the reference frame using np.dot. This allowed me to map the transformations from individual frames to the reference frame and produce the final video.

7. Quality of results / report

Nothing extra to include (scoring: 0=poor 5=average 10=great).

## **Acknowledgments / Attribution**

All images were provided by the professor/class and all code was written by me

## Project #5: Video Stitching and Processing

## ✓ CS445: Computational Photography - Spring 2020

---

### ✓ Setup

```
from google.colab import drive
drive.mount('/content/drive')
```

↗ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive")

```
# modify to where you store your project data including utils
datadir = "/content/drive/My Drive/VideoStitching/mp5/"
```

```
utilfn = datadir + "utils.py"
!cp "$utilfn" .
imagesfn = datadir + "images"
!cp -r "$imagesfn" .
```

```
!pip uninstall opencv-python -y
# downgrade OpenCV a bit to use SIFT
!pip install opencv-contrib-python==4.5.5.64
!pip install ffmpeg-python # for converting to video
```

```
import ffmpeg
import cv2
import numpy as np
import os
from numpy.linalg import svd, inv
import utils
%matplotlib inline
from matplotlib import pyplot as plt
```

↗ **WARNING: Skipping opencv-python as it is not installed.**

```
Collecting opencv-contrib-python==4.5.5.64
  Downloading opencv_contrib_python-4.5.5.64-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (66.7MB)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-contrib-python==4.5.5.64)
Downloading opencv_contrib_python-4.5.5.64-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (66.7/66.7 MB 12.0 MB/s eta 0:00:00)
Installing collected packages: opencv-contrib-python
  Attempting uninstall: opencv-contrib-python
    Found existing installation: opencv-contrib-python 4.10.0.84
    Uninstalling opencv-contrib-python-4.10.0.84:
      Successfully uninstalled opencv-contrib-python-4.10.0.84
Successfully installed opencv-contrib-python-4.5.5.64
Requirement already satisfied: ffmpeg-python in /usr/local/lib/python3.10/dist-packages (0.2.0)
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (from ffmpeg-python==0.2.0)
```

### ✓ Part I: Stitch two key frames

This involves:

1. compute homography H between two frames;
2. project each frame onto the same surface;
3. blend the surfaces.

Check that your homography is correct by plotting four points that form a square in frame 270 and their projections in each image.

```
def score_projection(pt1, pt2):
    """
    Score corresponding to the number of inliers for RANSAC
    Input: pt1 and pt2 are 2xN arrays of N points such that pt1[:, i] and pt2[:,i] should be close in l
    Outputs: score (scalar count of inliers) and inliers (1xN logical array)
    """

    # Initialize outputs
    inliers = np.zeros(pt1.shape[1], dtype=bool)
    score = 0

    # Compute diff and select inliers where threshold < 1
    inliers = (np.linalg.norm(pt1 - pt2, axis=0)) < 1
    score = np.sum(inliers)

    return score, inliers


def auto_homography(Ia,Ib, homography_func=None,normalization_func=None):
    """
    Computes a homography that maps points from Ia to Ib

    Input: Ia and Ib are images
    Output: H is the homography

    """
    if Ia.dtype == 'float32' and Ib.dtype == 'float32':
        Ia = (Ia*255).astype(np.uint8)
        Ib = (Ib*255).astype(np.uint8)

    Ia_gray = cv2.cvtColor(Ia,cv2.COLOR_BGR2GRAY)
    Ib_gray = cv2.cvtColor(Ib,cv2.COLOR_BGR2GRAY)

    # Initiate SIFT detector
    sift = cv2.xfeatures2d.SIFT_create()

    # find the keypoints and descriptors with SIFT
    kp_a, des_a = sift.detectAndCompute(Ia_gray,None)
    kp_b, des_b = sift.detectAndCompute(Ib_gray,None)

    # BFMatcher with default params
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des_a,des_b, k=2)

    # Apply ratio test
    good = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good.append(m)

    numMatches = int(len(good))
```

```

matches = good

# Xa and Xb are 3xN matrices that contain homogeneous coordinates for the N
# matching points for each image
Xa = np.ones((3,numMatches))
Xb = np.ones((3,numMatches))

for idx, match_i in enumerate(matches):
    Xa[:,idx][0:2] = kp_a[match_i.queryIdx].pt
    Xb[:,idx][0:2] = kp_b[match_i.trainIdx].pt

## RANSAC
niter = 1000
best_score = 0
n_to_sample = 4 # Put the correct number of points here

for t in range(niter):
    # estimate homography
    subset = np.random.choice(numMatches, n_to_sample, replace=False)
    pts1 = Xa[:,subset]
    pts2 = Xb[:,subset]

    H_t = homography_func(pts1, pts2, normalization_func) # edit helper code below (computeHomog

    # score homography
    Xb_ = np.dot(H_t, Xa) # project points from first image to second using H

    score_t, inliers_t = score_projection(Xb[:,2,:]/Xb[2,:], Xb_[:,2,:]/Xb_[2,:])

    if score_t > best_score:
        best_score = score_t
        H = H_t
        in_idx = inliers_t

print('best score: {:.02f}'.format(best_score))

# Optionally, you may want to re-estimate H based on inliers

return H

def computeHomography(pts1, pts2,normalization_func=None):
    """
    Compute homography that maps from pts1 to pts2 using SVD. Normalization is optional.

    Input: pts1 and pts2 are 3xN matrices for N points in homogeneous
    coordinates.

    Output: H is a 3x3 matrix, such that pts2~H*pts1
    """
    # Creat svd array
    svdArr = np.zeros((pts1.shape[1] * 2, 9))
    for i in range(pts1.shape[1]):
        # Normalize points
        p1Col = pts1[:, i] / pts1[2, i]
        p2Col = pts2[:, i] / pts2[2, i]
        # Populate array appropriately to solve for H
        svdArr[i*2] = [-p1Col[0], -p1Col[1], -p1Col[2], 0, 0, 0, p1Col[0]*p2Col[0], p1Col[1]*p2Col[0],
        svdArr[i*2 + 1] = [0, 0, 0, -p1Col[0], -p1Col[1], -p1Col[2], p1Col[0]*p2Col[1], p1Col[1]*p2Col

```

```
# Return last row reshaped of V transpose
U, S, Vh = np.linalg.svd(svdArr)
H = Vh[-1, :].reshape((3, 3))
return H

# images location
im1 = './images/input/frames/f0270.jpg'
im2 = './images/input/frames/f0450.jpg'

# Load an color image in grayscale
im1 = cv2.imread(im1)
im2 = cv2.imread(im2)

H = auto_homography(im1,im2, computeHomography)
print(H/H.max())

# plot the frames here
box_pts = np.array([[300, 400, 400, 300, 300], [100, 100, 200, 200, 100], [1, 1, 1, 1, 1]])
plt.figure()
plt.imshow(im1[:, :, [2, 1, 0]])
plt.plot(box_pts[0, :], box_pts[1, :], 'r-')

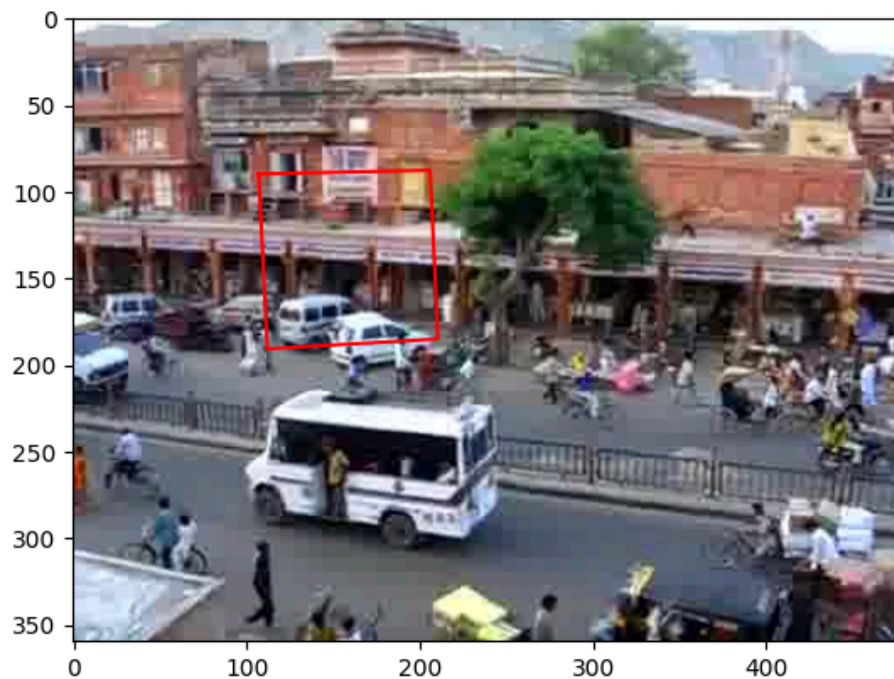
# T0 D0: project points into im2 and display the projected lines on im2
box_pts2 = np.dot(H, box_pts)
box_pts2 = box_pts2 / box_pts2[2, :]

plt.figure()
plt.imshow(im2[:, :, [2, 1, 0]])
plt.plot(box_pts2[0, :], box_pts2[1, :], 'r-')
```

```

best score: 154.000000
[[-4.88611548e-03 -2.52306298e-04  1.00000000e+00]
 [-4.33954263e-05 -4.67004011e-03  6.79006356e-02]
 [-1.63938229e-06 -1.84504112e-07 -4.08208908e-03]]
[<matplotlib.lines.Line2D at 0x78cb3893cbb0>]

```



```

projectedWidth = 1600
projectedHeight = 500
Tr = np.array([[1, 0, 660], [0, 1, 120], [0, 0, 1]])

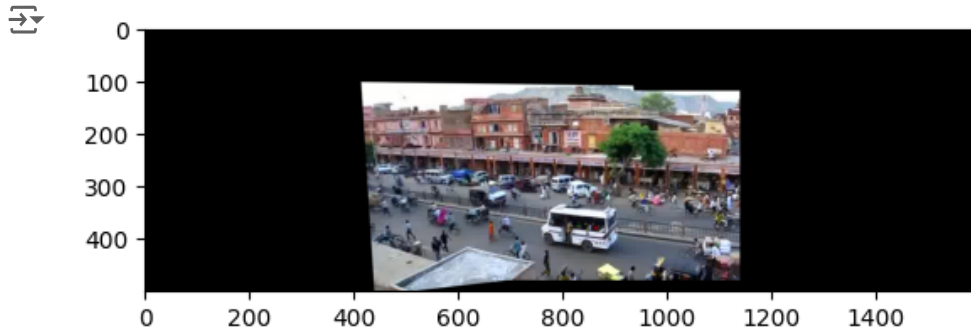
# T0 D0: warp and blend the two images
projIm1 = cv2.warpPerspective(im1, np.dot(Tr, H), (projectedWidth, projectedHeight))
projIm2 = cv2.warpPerspective(im2, np.dot(Tr, np.eye(3)), (projectedWidth, projectedHeight))
blendedImg = utils.blendImages(projIm1, projIm2)

plt.imshow(blendedImg[:, :, [2, 1, 0]])

```



```
plt.plot();
```



## ▼ Part II: Panorama using five key frames

Produce a panorama by mapping five key frames [90, 270, 450, 630, 810] onto the same reference frame 450.

```
key_frames_idx = np.array([90, 270, 450, 630, 810])-1

frames = np.zeros((len(key_frames_idx), im1.shape[0], im1.shape[1], im1.shape[2]), dtype='uint8')
for n in range(len(key_frames_idx)):
    frames[n] = cv2.imread("./images/input/frames/f0{num}.jpg".format(num=str(key_frames_idx[n]+1).zfill(3)))

# Final img
blendedImg = np.zeros((projectedHeight, projectedWidth, 3))
# Layer 270, 450, 630 as before:
H90 = auto_homography(frames[0], frames[1], computeHomography)
H270 = auto_homography(frames[1], frames[2], computeHomography)
H630 = auto_homography(frames[3], frames[2], computeHomography)
H810 = auto_homography(frames[4], frames[3], computeHomography)

# Paste blended 1st image by multiplying H270 and H90
H90Img = cv2.warpPerspective(frames[0], np.dot(Tr, np.dot(H270, H90)), (projectedWidth, projectedHeight))
blendedImg = utils.blendImages(H90Img, blendedImg)

# Paste blended 2nd image
H270Img = cv2.warpPerspective(frames[1], np.dot(Tr, H270), (projectedWidth, projectedHeight))
blendedImg = utils.blendImages(H270Img, blendedImg)

# Paste 3rd image
H450Img = cv2.warpPerspective(frames[2], np.dot(Tr, np.eye(3)), (projectedWidth, projectedHeight))
blendedImg = utils.blendImages(H450Img, blendedImg)

# Paste blended 4th image
H630Img = cv2.warpPerspective(frames[3], np.dot(Tr, H630), (projectedWidth, projectedHeight))
blendedImg = utils.blendImages(H630Img, blendedImg)

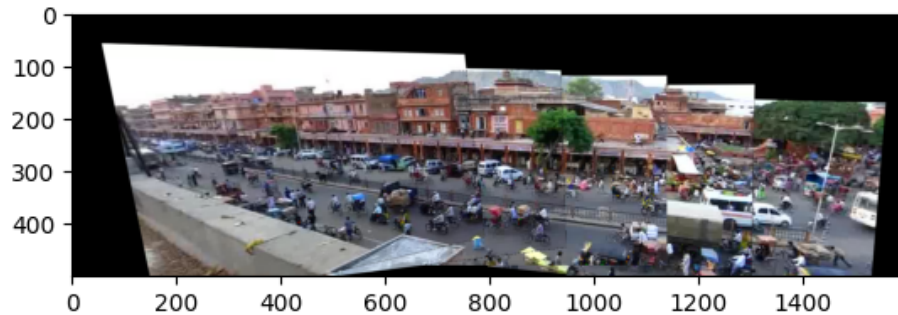
# Paste blended 5th image by multiplying H810 and H630
H810Img = cv2.warpPerspective(frames[4], np.dot(Tr, np.dot(H810, H630)), (projectedWidth, projectedHeight))
blendedImg = utils.blendImages(H810Img, blendedImg)

plt.imshow(blendedImg[:, :, [2, 1, 0]])
plt.plot();
```

```

➡ best score: 215.000000
  best score: 152.000000
  best score: 153.000000
  best score: 99.000000

```



```
plt.imshow()
```

### ✓ Part 3: Map the video to the reference plane

Project each frame onto the reference frame (using same size panorama) to create a video that shows the portion of the panorama revealed by each frame

```

# read all the images
import os
dir_frames = 'images/input/frames'
filenames = []
filesinfo = os.scandir(dir_frames)

filenames = [f.path for f in filesinfo if f.name.endswith(".jpg")]
filenames.sort(key=lambda f: int(''.join(filter(str.isdigit, f))))

frameCount = len(filenames)
frameHeight, frameWidth, frameChannels = cv2.imread(filenames[0]).shape
frames = np.zeros((frameCount, frameHeight, frameWidth, frameChannels), dtype='uint8')

for idx, file_i in enumerate(filenames):
    frames[idx] = cv2.imread(file_i)

# Create video
allProjImgs = np.zeros((frameCount, projectedHeight, projectedWidth, frameChannels), dtype='uint8')
for i in range(frameCount):
    projIm = 0
    if i < 180:
        curH = auto_homography(frames[i], frames[89], computeHomography)
        if (i == 89):
            curH = np.eye(3)
        projIm = cv2.warpPerspective(frames[i], np.dot(Tr, np.dot(curH, np.dot(H270, H90)))), (projectedWidth, projectedHeight))
    elif 180 <= i < 360:
        curH = auto_homography(frames[i], frames[269], computeHomography)
        if (i == 269):
            curH = np.eye(3)
        projIm = cv2.warpPerspective(frames[i], np.dot(Tr, np.dot(curH, H270))), (projectedWidth, projectedHeight))
    elif 360 <= i < 540:
        curH = auto_homography(frames[i], frames[449], computeHomography)

```

```

    if (i == 449):
        curH = np.eye(3)
        projIm = cv2.warpPerspective(frames[i], np.dot(Tr, np.dot(curH, np.eye(3))), (projectedWidth, i
elif 540 <= i < 720:
    curH = auto_homography(frames[i], frames[629], computeHomography)
    if (i == 629):
        curH = np.eye(3)
        projIm = cv2.warpPerspective(frames[i], np.dot(Tr, np.dot(curH, H630)), (projectedWidth, proje
elif i >= 720:
    curH = auto_homography(frames[i], frames[809], computeHomography)
    if (i == 809):
        curH = np.eye(3)
        projIm = cv2.warpPerspective(frames[i], np.dot(Tr, np.dot(curH, np.dot(H810, H630))), (projecti
    allProjImgs[i] = projIm
# Save video to drive
utils.vidwrite_from_numpy(datadir + "myVideo.mp4", allProjImgs[:, :, :, [2, 1, 0]])

```

```

⇒ best score: 364.000000
best score: 371.000000
best score: 382.000000
best score: 366.000000
best score: 403.000000
best score: 388.000000
best score: 412.000000
best score: 395.000000
best score: 420.000000
best score: 419.000000
best score: 426.000000
best score: 406.000000
best score: 418.000000
best score: 412.000000
best score: 430.000000
best score: 419.000000
best score: 439.000000
best score: 437.000000
best score: 426.000000
best score: 415.000000
best score: 417.000000
best score: 428.000000
best score: 429.000000
best score: 432.000000
best score: 451.000000
best score: 421.000000
best score: 428.000000
best score: 439.000000
best score: 444.000000
best score: 440.000000
best score: 456.000000
best score: 452.000000
best score: 457.000000
best score: 458.000000
best score: 444.000000
best score: 456.000000
best score: 458.000000
best score: 447.000000
best score: 432.000000
best score: 472.000000
best score: 461.000000
best score: 469.000000
best score: 458.000000
best score: 457.000000
best score: 468.000000
best score: 421.000000
best score: 488.000000
best score: 472.000000

```

```


best score: 468.000000
best score: 473.000000
best score: 471.000000
best score: 465.000000
best score: 479.000000
best score: 477.000000
best score: 495.000000
best score: 454.000000
best score: 487.000000
best score: 515.000000

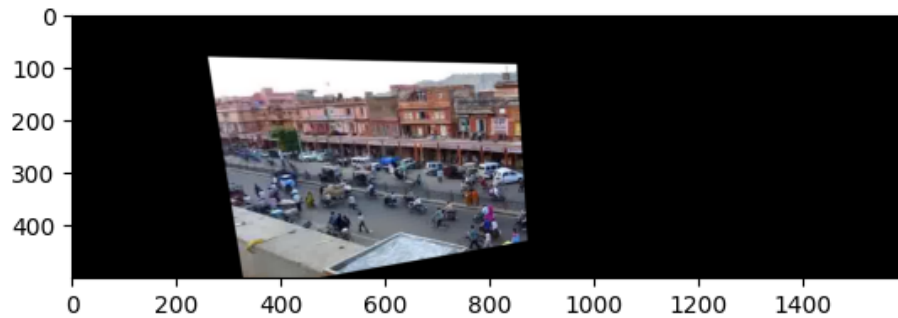
```

```

plt.figure()
plt.imshow(allProjImgs[199][:, :, [2, 1, 0]])

```

 <matplotlib.image.AxesImage at 0x78cb2cfcd4b0>



#### ✓ Part 4: Create background panorama

Create a background panorama based on the result from Part 3.

```
# T0 D0 part 4
```

#### ✓ Part 5: Create background movie

Generate a movie that looks like the input movie but shows only background pixels. For each frame of the movie, you need to estimate a projection from the panorama to that frame. Your solution can use the background image you created in Part 4 and the per-frame homographies you created in Part 3.

```
# T0 D0 part 5
```

#### ✓ Part 6: Create foreground movie

In the background video, moving objects are removed. In each frame, those pixels that are different enough than the background color are considered foreground. For each frame determine foreground pixels and generate a movie that