Comp Photography
Final Project

Robert Bobkoskie

Summer 2016

rbobkoskie3@gatech.edu

Having Fun with Video

The project was experimentation in Video Textures (P-11), Pyramid Blending (A-06) and Feature Detection and Matching. I utilized much of the code from A-11 and A-06, however, using the concepts from A-07 for feature detection did not work well for this application. I implemented feature detection and matching using template matching from opencv. I separated this project into two phases. For the first phase, I implemented feature detection, auto mask generation and blending. For the second phase, I enhanced mask generation by auto locating and sizing a mask and 'white' image based on features detected in a base image. The definition for 'white' image will be discussed in detail in this write-up.

Phase 1: https://goo.gl/photos/KEpwD9mZ3EBcycQk7

Phase 2**: https://goo.gl/photos/Hv9WaPvd4QrQJ9TZA**

The Goal of Your Project

My goal for the final project was to implement what we've learned in Assignments: 11, 06 and 07 to create a novel video. I was motivated by the material covered in the video lectures and the wonderful assignments. The lessons learned from the lecture and assignments provided a strong foundation, and gave me confidence that I could create a novel final project in the allocated timeframe.

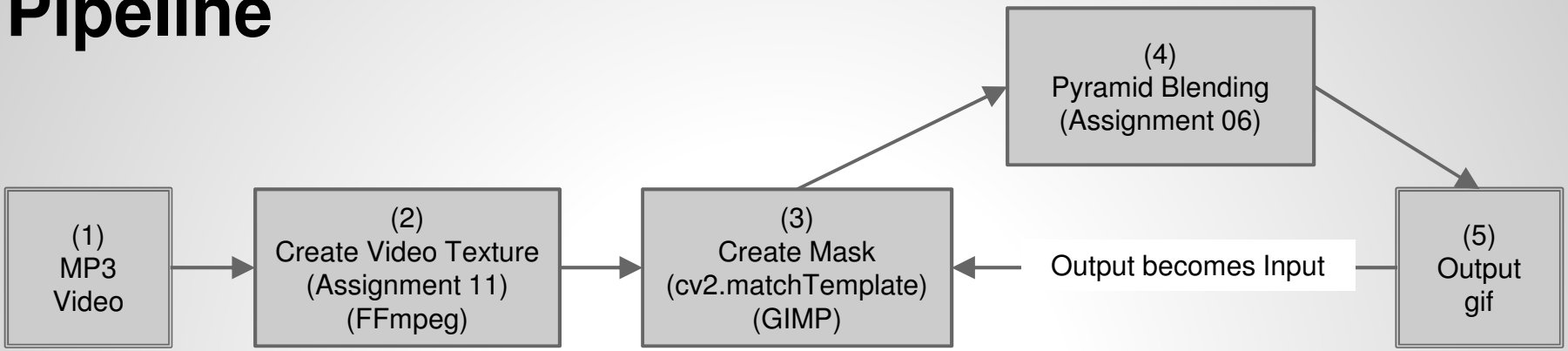
I broke the project into two phases. For Phase 1, I wanted to keep the scope succinct, to make certain that I could deliver something. For this phase, the goal was to replace a boring office scene that I used to create a video texture for assignment 11 with something more scenic. Some parts in this phase were implemented manually using GIMP. For Phase 2, I wanted to expand on the work done in Phase 1. My goal for Phase 2 was to code the manual parts of Phase 1. Since I was saving time by automating the manual processes, I also wanted to create a more complex video for Phase 2. Throughout this write-up, any references to video and gif have the same meaning.

Scope Changes

My final project was different from my original proposal. Essentially, when I submitted my proposal, I had not completed all the video lectures or assignments. For me, Assignment 11 was very interesting. Assignment 6 was also a lot of fun, so it seemed very natural to use what I enjoyed and learned to create a novel video.

My original interest was facial recognition or blur correction. I thought about applying blur correction to the images I generated from the Camera Obscura experiment. I did some further research, and determined that blur correction may not work well for this application, so I looked at facial recognition. I found some papers on Eigenfaces that seemed interesting and might have enough for the computation pipeline. It was about this time that I started Assignment 11, Video Textures. I really had a lot of fun with this assignment, and considered ways to expand the scope by incorporating material from previous assignments. I thought this might be a really fun and interesting Final Project.

Pipeline



(1) Start with an MP3 video and create a video texture using code from Assignment 11. The length of the gif was 52 frames.
(2) Identify other interesting gifs on the internet and use FFmpeg to break them into 52 frames.
(3) Create a mask. For Phase 1, I used both manual and automated processes. For the manual process, I used GIMP to generate a mask corresponding to the cartoon gif. For the automated process, I wrote a function based on opencv's **`matchTemplate`** method to identify a region in the frame for the mask. For Phase 2, I expanded on mask generation by implementing additional functionality from the **`matchTemplate`** method: **`max_val`** to conditionally identify a match, **`top_left`** to identify the location of the matched region. This enabled the auto generation of both the mask, and the corresponding 'white' image. The 'white' image is defined as the image that has the same shape as the mask, where the blended image corresponds to the 'white' area of the mask.
(4) Blend images with mask using code from Assignment 06.
(5) Generate output gif using GIMP. For both phases, I used the output to feed back into step (3) to generate and blend a novel gif as the final output. This feedback loop was a manual process. Given the timeframe for the project, and the number of times I expected to use the feedback loop, I found it faster to manually adjust the output to become input for the next cycle.

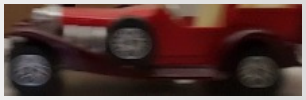
Results (1)

Results for Project, Video with annotations can be found here:
PHASE 1: https://goo.gl/photos/KEpwD9mZ3EBcycQk7
PASHE 2: https://goo.gl/photos/Hv9WaPvd4QrQJ9TZA

Code is on GitHub:
CODE: https://github.com/rbobkoskie3/CS6475-FP

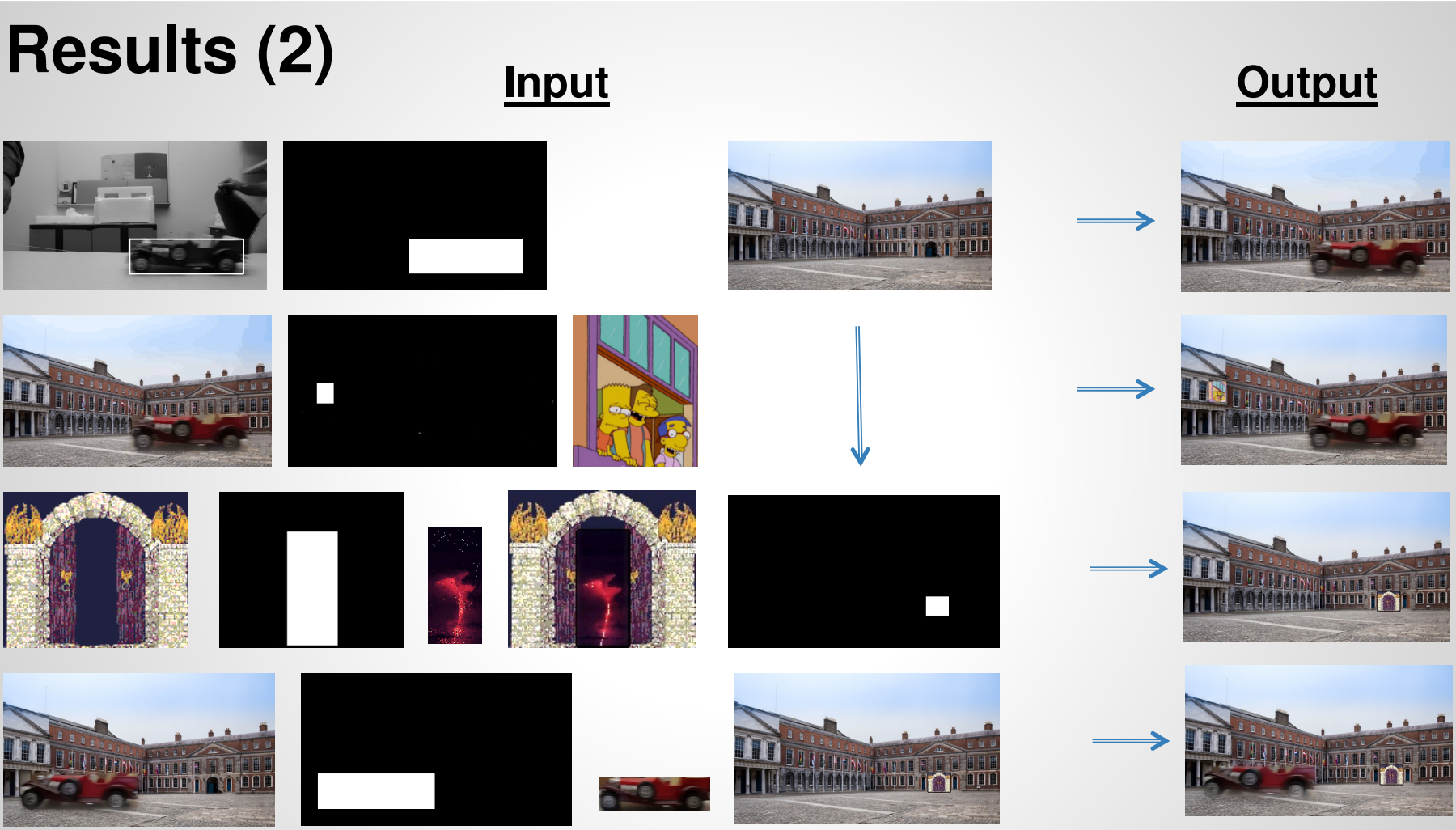
This page shows the templates used for identifying objects in the frames. The arched doorway template was used to match the doorway in frames (below left). The car template was used to match the car in frames (below right). The template for the street scene was used match the right side of the street scene. The black rectangle was used to match the dark space in the open doorway (below right). Results (2) depicts input images, masks and corresponding output.

TEMPLATES

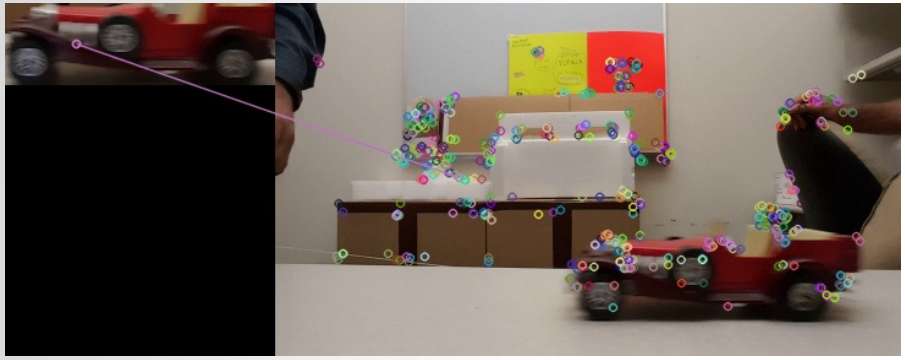


Results (2)

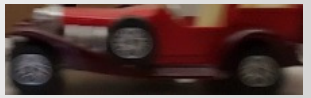


Computation: Project Development (1)

I started with an idea to replace the boring office scene used to create my video texture for Assignment 11. I initially planned to use cv2.VideoCapture to read and write video (**READ_VIDEO.py**), but had issues writing files to my windows machine. I realized I could use FFmpeg and GIMP to do the job. Feature detection was my next consideration, and I first attempted to employ the concepts: ORB, SIFT from Assignment 07 using **MATCHER.py** to identify objects in frames. However, I quickly hit a dead-end, as the matching was not ideal, and I was not able to reliably identify the location of the matched object.



From the opencv documentation, I determined the matchTemplate method might work really well to identify a region of interest in a frame.

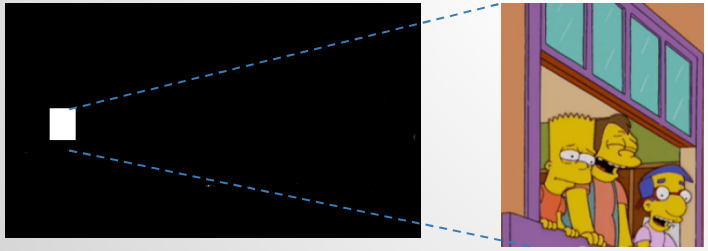


Computation: Project Development (2)

After I was successful in matching regions of interest, I proceeded to make a first video of the moving car with a new background scene. From **VIDEO.py**, I used slicing to create a mask with the rectangular matched shape of the car corresponding to the 'white' portion of the mask (255), all other pixels were black (0). This was applied for all frames, and was used to blend (**VIDEO.py**) with the street scene jpg to create a novel video. The gif was created using GIMP, open all frames as layers and export as a gif with a 60 ms delay between frames.

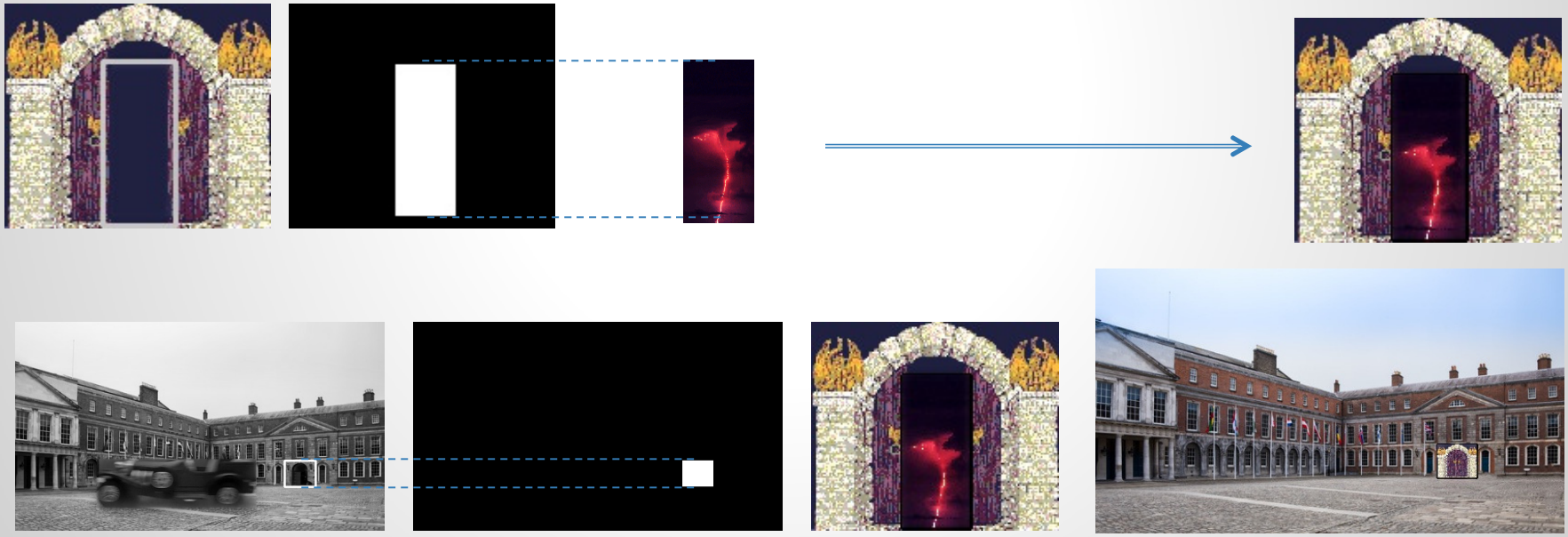
Computation: Project Development (3)

I was pleased with the novel car gif, but thought about blending additional gifs to make it more interesting. I considered replacing a window, or door in the street scene with another gif. I searched the net and found a cartoon gif that looked like it would work. I was already using GIMP to create novel gifs, so I used GIMP to create a mask for the cartoon gif. But first, I needed to break out the Cartoon gif into frames. For this, I used FFmpeg to create 52 frames (the same number of frames as the novel gif). I then opened a frame of the novel gif and a frame of the cartoon gif as layers in GIMP, and resized and moved the layer corresponding to the cartoon gif to a location over a window in the novel gif. I created the mask by coloring in the area of the cartoon gif white and the remainder black. From GIMP, I was also able to determine the size of the 'white' area of the mask, and used this to resize all 52 frames of the cartoon gif with **RESIZE.py**. I ran this through **VIDEO.py** with `CreateMaskWithMotion = False` to blend the cartoon gif with the novel gif. This was my final output for Phase 1.



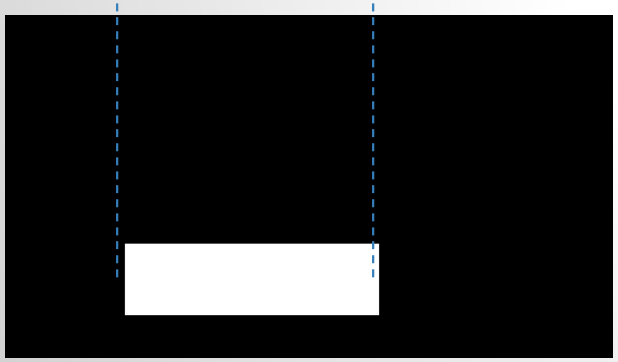
Computation: Project Development (4)

The final gif for Phase 1 turned out well, but I wanted to automate some of the manual steps for mask, and ‘white’ image generation. This was not trivial, as blending images necessitates that the image, mask and ‘white’ image are all the same shape. Creating a mask of np.zero’s with the same shape as an image is trivial, however, I needed to generate both a ‘white’ image, as well as the ‘white’ portion of the mask that corresponds to the shape and location of the ‘white’ image. I wrote a program, **AUTO_SIZE.py**, which expanded on **VIDEO.py**. I was able to provide multiple (gif) frames as input to blend into a novel video. Another issue was that I wanted the lightening to appear only in the open doorway. I needed to account for a match of the dark space in the open door, but still return a mask and ‘white’ images for the closed door. I used **AUTO_SIZE.py** extensively to produce the novel videos for Phase 2.



Computation: Project Development (5)

Two more runs through **AUTO_SIZE.py** created the final gif for Phase 2. The disappearing car used a street scene to mask out the left side of the image.



Computation: Code Explanation (1)

I provided a high level detail of project development in the previous five slides. I have also provided two additional slides, '**Code Development: Masking**', and '**Code Development: Blending**' that follow the Code Explanation slides. These additional slides provide some insight into a couple experiments I ran to validate my functions for masking and blending. I will try to keep this code explanation section focused on code that solved challenging problems. I hope that the snippets of code will highlight and provide some understanding about these problems.

From **VIDEO.py**: The **`CreateMask`** function identifies a '**template**' in an '**img**' and returns a '**mask**' based on the templates location in the image. Although I converted to a single channel, **`cv2.COLOR_RGB2GRAY`**, this was not necessary, I could have used three, and in fact did use three channels in **AUTO_SIZE.py**.

```
def CreateMask(img, template, idx):
    '''
    This function uses Template Matching Functions from opencv to create a mack for each frame:
    http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html
    '''
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    w, h = template.shape[::-1]
    method = eval('cv2.TM_SQDIFF_NORMED')
    res = cv2.matchTemplate(img,template,method)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc
    mask = np.zeros(shape=img.shape, dtype=np.float64)
    mask[top_left[1]:top_left[1] + h, top_left[0]:top_left[0] + w] = 255.0
    return mask
```

Computation: Code Explanation (2)

The other key function in **VIDEO.py** is the blending function. Essentially, this is an adaption of the pyramid blending from Assignment 06, see GitHub for details. Using **VIDEO.py**, I was able to identify the car in the office scene, generate a mask, and blend in the car with a street scene.

```
def BlendImagePair(img_1, img_2, mask):
    '''
    Adaption of Blending code from Assignment 06
    '''
```

To add in the cartoon gif, I needed to break out the frames. For this, I used Ffmpeg to create the same number of frames as the Video Texture of the car (52 frames) from the cartoon gif. I used GIMP (manual steps to create a mask and 'white' image). Next, using **RESIZE.py**, I resized all frames in the cartoon gif to the shape of the 'white' area in the mask. Finally, to create the final gif for Phase 1, I used **VIDEO.py** to blend the car in the street scene with the cartoon gif.

The fps value in Ffmpeg was dependent on the gif. The invariant in the equation was the number of frames. I would modify the fps variable in the Ffmpeg command to generate 52 frames.


```
ffmpeg -i C:\PYCODE\CS6475\IMAGES\20160629_100145.mp4 -vf fps=58 C:\PYCODE\CS6475\RESIZE\%03d.png
```

Computation: Code Explanation (3)

I wanted to automate the manual processes from Phase 1, so I developed **AUTO_SIZE.py**. For brevity, I did not include all the statements in the main function. I eliminated from this presentation, statements that initialized directories or filenames specific to my environment. Note that some commented out lines, are not comments, or broken code. For example, the lines defining the **`bld_mask`** variable are used to define masks for different blends. Blending two frames that are the same size needed to be accounted for differently than frames that were not the same size. I could have used a conditional, but, given the deadline for this project, found it more convenient to comment out, and not add more complexity.

```
def main():
    for idx, (inp_name, bld_name) in enumerate(zip(files_inp, files_bld)):
        img = cv2.imread(inp_name)       #FOR DYNAMIC SCENE
        #img = cv2.imread('street.jpg')  #FOR STATIC SCENE
        blend_img = cv2.imread(bld_name)
        #img = cv2.imread(img_name, cv2.IMREAD_GRAYSCALE)
        mask, top_left, bld_img = CreateMask(img, template, idx)
        cv2.imwrite(mypath_msk+str(idx).zfill(4)+'.jpg', mask) #FOR VERIFICATION
        bld_mask = np.zeros(shape=img.shape, dtype=np.float64)
        resize_img = ResizeImage(blend_img, template.shape)
        if top_left != (0, 0):
            #bld_mask[top_left[1]:top_left[1] + resize_img.shape[0], top_left[0]:top_left[0] + \
            #    resize_img.shape[1]] = resize_img
            bld_mask[top_left[1]:top_left[1] + resize_img.shape[0], top_left[0]:top_left[0] + \
                resize_img.shape[1]] = bld_img  #USE FOR FINAL BLEND
            #bld_mask[top_left[1]:top_left[1] + resize_img.shape[0] - 10, top_left[0]:top_left[0] + \
            #    resize_img.shape[1] - 10] = bld_img  #USE FOR FINAL BLEND - TESTING
        #final_Image = BlendImagePair(img, bld_mask, mask)
        final_Image = BlendImagePair(blend_img, bld_mask, mask)
```

Computation: Code Explanation (4)

AUTO_SIZE.py was similar to **VIDEO.py**. However, there are some important differences. First, **AUTO_SIZE.py** can handle three channels of color, and uses **`max_val`** from **`cv2.minMaxLoc`** to condition on creating a 'white' area in a mask. This was needed to account for matching the open doorway in the doorway gif. Using a single channel, without a degree of confidence in the match always returned a match somewhere in the frame. I wanted to have the lightening flash only in the open doorway, thus, added three channels, and a degree of confidence to the matched object.

```
def CreateMask(img, template, idx):
    '''
    This function uses Template Matching Functions from opencv to create a mack for each frame:
    http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html
    '''
    mask = np.zeros(shape=img.shape, dtype=np.float64)
    d, w, h = template.shape[::-1]
    top_left = (0, 0)
    method = eval('cv2.TM_CCORR_NORMED')
    res = cv2.matchTemplate(img,template,method)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
    if max_val >= 0.90: #Use 0.90 for most matches, 0.80 is needed to match the moving car in all frames
        if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
            top_left = min_loc
        else:
            top_left = max_loc
        OFFSET = 5
        #MAKE MASK AREA SLIGHTLY SMALLER FOR BETTER BLEND
            mask[top_left[1] + 2:top_left[1] + h - OFFSET, top_left[0] + OFFSET:top_left[0] + w - 2] = 255.0
            bld_img = img[top_left[1]:top_left[1] + h, top_left[0]:top_left[0] + w]
    return mask, top_left, bld_img
```

Code Development: Blending

The foundation of the final gif was the moving car. I needed to identify it and mask out everything else in the frame for all frames. I was not able to use a mask of the exact shape of the car (**fig 2**). My implementation required a rectangular shape (**fig 1**), thus there would be pixels from the template bleeding into the blend. To compensate for this, I experimented with using a smaller area for the 'white' mask. The snippet of code below shows the slice used for creating the 'white' mask.
See 'Blending experiments' video: https://goo.gl/photos/Hv9WaPvd4QrQJ9TZA

Figure 1: Rectangular Shape

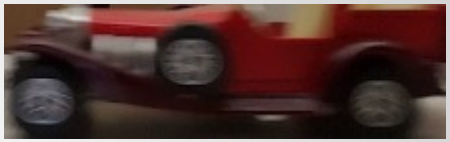


Figure 2: Exact Shape

```
#MAKE MASK AREA SLIGHTLY SMALLER FOR BETTER BLEND
OFFSET = 5
mask[top_left[1] + 2:top_left[1] + h - OFFSET, top_left[0] + OFFSET:top_left[0] + w - 2] = 255.0
```

Frame from final gif showing a blend using a 'white' mask cropped by 0 pixels in both [h, w]. **Notice edges**.

Frame from final gif showing a blend using a 'white' mask cropped by 40 pixels in both [h, w]. **Too much**.

Frame from final gif showing a blend using a 'white' mask cropped by 7 pixels in both [h, w]. **Better edges**.

Code Development: Masking

As I was implementing code to generate both a mask and second image based on the mask, I wanted to verify that I was slicing and constructing them correctly. To prove this, I modified some frames in the input video by marking locations on the frame with a white ‘X’. I marked locations that were expected to get masked out (on the buildings) as well as locations that were expected to appear in the final blended video (near the car). I confirmed that my code was slicing and generating the mask and ‘white; image correctly. As expected, only the ‘X’s near the car appeared in the output video.
See ‘X experiments’ video: https://goo.gl/photos/Hv9WaPvd4QrQJ9TZA

3 Images from X experiment **Video Input**



3 Images from X experiment **Video Output**



Details: What worked?

From **AUTO_SIZE.py**, **`CreateMask(img, template, idx)`**: Generating a mask with a slightly smaller 'white' area for the matched image worked well to improve the blending. This was necessary to compensate for not being able to match a non-rectangular shape.

```
#MAKE MASK AREA SLIGHTLY SMALLER FOR BETTER BLEND
OFFSET = 5
mask[top_left[1] + 2:top_left[1] + h - OFFSET, top_left[0] + OFFSET:top_left[0] + w - 2] = 255.0
```

Also from **AUTO_SIZE.py**, **`CreateMask(img, template, idx)`**: Conditioning on the degree of confidence of the matched object worked well in detecting just the open door in the doorway gif.

```
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
if max_val >= 0.90: #Use 0.90 for most matches, 0.80 is needed to match the moving car in all frames
    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc
```

If **`max_val`** `>= 0.90` is not used, an 'invalid' area in the frame is detected.

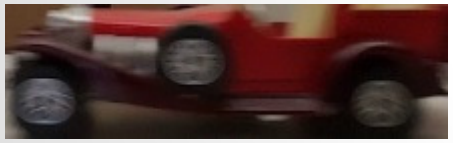
Using **`max_val`** `>= 0.90`, only the open door is matched

Details: What did not work? Why?

I was not able to get **ORB** and **SIFT** feature detection and matching to work well for this application. I was also not able to get **cv2.VideoWriter** to work in my windows environment. I didn't spend too much time debugging and answering the question 'why?', as I found alternative solutions that sot the job done.

Matching a non rectangular template did not work. I was able to compensate for this by adjusting the 'white' area in the mask, see 'What worked' for details.

Rectangular Shape was used for template



Could not match an exact Shape for template

Not related to code, but spell checker in PowerPoint is not working for me. I noticed that 'implimnet' passes, I think it should be 'implement'. I'm not sure if others have had this issue. My apologies if I did not catch all the spelling errors.

Details: What would you do differently?

I would like to match the exact shape of a template. This would have made for a cleaner blend. Since I was able to work through this issue, I did not focus on it, but chose to use my time working other areas of the project.

Resources

PROJECT
Assignment 06: Blending Code
Assignment 07 Feature Detection and Matching (concepts but no code
Assignment 11: Video Textures

GIFS
http://giphy.com/search/open-a-window
http://bestanimations.com/Nature/Storms/Storms.html
http://www.animatedimages.org/cat-doors-135.htm

OPENCV
http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html
http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html
http://docs.opencv.org/3.1.0/dd/d43/tutorial_py_video_display.html

OTHER
http://www.pyimagesearch.com/2014/01/20/basic-image-manipulations-in-python-and-opencv-resizing-scaling-rotating-and-cropping/

Your Code

Code is on GitHub [VIDEO.py , AUTO_SIZE.py , RESIZE.py :: MATCHER.py , READ_VIDEO.py]:
https://github.com/rbobkoskie3/CS6475-FP

VIDEO.py and RESIZE.py were used for Phase 1.
VIDEO.py and AUTO_SIZE.py were used for Phase 2.
MATCHER.py and READ_VIDEO.py were developed, but not used.

THANKS!

I would like to thank the instructors for this class. I appreciate the feedback provided on piazza. I had a very positive experience learning the material, and eventually with the autograder (bonnie).

I would like to thank my classmates. The peer reviews and posts on Piazza were a big help.

Last, but not least, I would like to thank my family. I had to postpone some vacation time and work weekends to achieve my goals for this class.