University of Pennsylvania, Philadelphia, Pennsylvania
CIS 581: Computer Vision, Fall 2020

Final Project: 2D Mapping and Detection from Video Footage

Srisa Changolkar, Celestina Saven, Vinay Senthil, Kevin Sun

Final Draft

Date Submitted ..... December 20, 2020
Instructor ..... Dr. Jianbo Shi

Contents

1 Abstract

In this project, we utilized the methods taught in CIS 581 to map the frames of a video to a single 2D plane upon which object detection tasks could be performed. After testing a variety of mapping methods between adjacent frames of a video, we found that utilizing the optical flow as a feature map between frames led to the best stitching outcome, resulting in a combined image of all frames that was almost entirely of jittering and misalignment. Finally, we performed transfer learning on a region-based convolutional neural network on Pytorch, trained on images we annotated, to carry out object detection tasks on the resulting image.

2 Introduction

With the ever rising usage of video technology in social media, via platforms like YouTube, Snapchat, and TikTok, videos are increasingly replacing static images as a mainstream method for conveying ideas, knowledge, and information. However, extracting information from these videos requires watching the video and processing it frame by frame. If it were somehow possible to stitch the frames of the video into one large, continuous image, it would be a lot easier to scan through the singular image to search for features of interest.

We discussed image stitching in lecture and developed a decent understanding of it through Project 4; the context of our use in class compared the features of two images in order to accurately stitch them together. While this method is sufficient for images, with video footage we can take things one step further. Given a video of continuous panning, we can stitch together the frames to produce a flat version of some 3D space or object. For example, drone footage nowadays is very popular to capture landscapes, such as cities, solar panel farms, and other large expanses of land. We could stitch together such footage obtained from drones to generate 2D maps of landscapes and cityscapes. Additionally, one may simply have many several static images (video frames) and want to stitch them together, something that existing panoramas cannot do. In order to perform object detection and other tasks more easily, it is important to be able to stitch the video frames together to obtain a single large image of the space, which is then a simpler format to work with.

As an example use case, applying object detection to stitched images of cities could allow us to analyze whether or not people are following social distancing guidelines. We could also create maps and use object detection to identify roads and trails in remote areas to create literal maps of an area. Other applications include scanning large documents and stitching together panoramas.

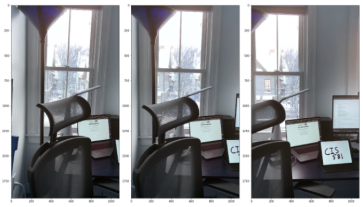
3 Mapping

3.1 Methods and Implementation

To accomplish transforming video footage, whether drone footage or camera panoramas, into one big image, many techniques we have covered in class (and some we haven't) are used. The main techniques include feature matching, RANSAC, warping, and combining images with the minimum-cost path for the image stitching segment, as well as Lucas-Kanade OpenCV for the optical flow segment. The main libraries used in this part are numpy, scipy, matplotlib, opencv, pillow, and scikit-image.

3.1.1 Method 1: SIFT

We first explored various stitching algorithms in an attempt to seamlessly stitch static video frames together. As a test, we used the following three frames (obtained from a video of the room of one of our group members):



We started with extending our image stitching from CIS581 Project 4 to stitch together multiple images into one long, continuous image. The steps involved in this approach are listed below:

1. Use SIFT feature points and match between them to find corresponding pairs between the two images. Use RANSAC to reject outliers. These techniques get a pretty good estimation of the homography between two images (given 4 pairs of corresponding points/features), and this transformation will enable us to stitch the two images together.
2. Compute the size of the new image that will hold the stitched result.
3. Paste img1 and img2 to largeImg1 and largeImg2. For each pixel in the largeImg2, u.se inverse warping (to not leave holes) and interpolation to get values from img2.
4. Put the two intermediate results largeImg1 and largeImg2 together. Take the averages of the two images in an overlapping area to make it look better.

However, it quickly became clear that the resulting image (shown in the next section) was distorted by excessive jittering. Due to this issue, before enhancing video stitching with optical flow, we first researched more advanced methods to get better stitching results for the frames of the video.

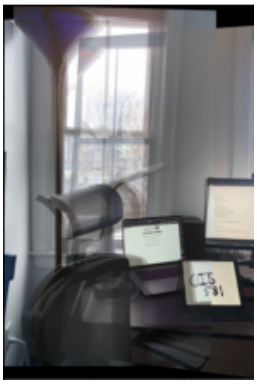
3.1.2 Method 2: ORB and MCP

The optimized workflow, with more advanced techniques includes:

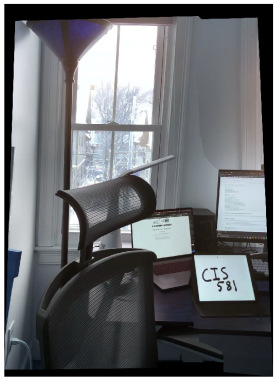
1. **Pre-Processing**: this included resizing, converting all images to grayscale because many feature detectors do not work on color images, and cropping to feature regions.
2. **Feature Detection and Matching**: Similar to 581, to estimate a transformation that relates these images, we defined one as the target image to remain anchored as the others were warped. However, this time, instead of using SIFT to perform feature detection and matching, we utilized Oriented Fast and rotated Brief (ORB) features, in conjunction with RANSAC to reject the false matches. ORB performs feature matching by computing the intensity weighted centroid of a patch with located corner at center. The direction of the vector from this corner point to centroid gives the orientation, and moments are computed to improve the rotation invariance [6]. This rotational invariance is what allows ORB to perform accurately despite the minute jittering in film frames (i.e. due to a shaky hand when filming), making it a better choice over SIFT in our use case.
3. **Warping:** This simply involved finding the shape of the output image to contain the stitched images along with applying the transformations to warp the images into place (using inverse mapping as in class).
4. **Combining Images:** This step was the most different from the techniques learned in 581. Simply summing or averaging warped images, tracking how many images overlap to create each point, and normalizing the result was not good enough, as this resulted in a blurry image with lines at image boundaries. Thus, we stitched images along a minimum-cost path (MCP): instead of blending pixels in an overlap region, find a vertical path through the difference image which stays as close to 0 as possible. We use that to build a mask, defining a transition between images, resulting in a seamless result [5]. Due to the difficulty of implementing this process, we used a reference guide [8].

3.2 Mapping Results:

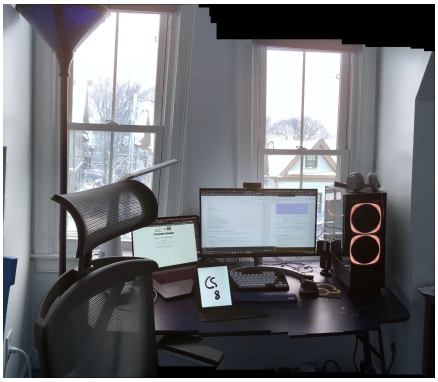
While we performed the Mapping algorithm on several videos (refer to attached files), we show the results for the room below. As we can see using ORB with MCP produces significantly better results than SIFT: Finally, to solve the issue of video frames being unaligned due to shaky video or drone footage, we used Lucas Kanade Optical Flow to feed feature points to the improved stitching algorithm. We also support for stitching in different directions, for example, a drone flying from left to right (no rotation) or a drone flying from bottom to top (rotated to get a horizontal stitched image). We loop through the frames of the video, and stitch each new frame onto the previous accumulated stitched previous frames. The final output is one singular, combined image.



SIFT



ORB + MCP



Result After Optical Flow (8 Frames)

4 Object Recognition

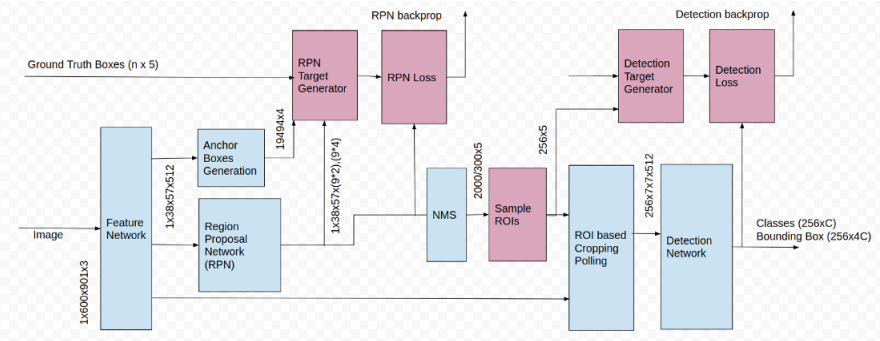
Given the stitched image compiled from a video, we can use object detection to identify a number of things, from recognizing roads and people on drone footage to identifying text. We chose to use our at-home footage to identify desk-top objects.

4.1 Methods and Implementation

While we originally were using a CNN structured similarly to the ones we used in class, we quickly found that the complexity of object detection tasks required a more advanced structure. Research led us to select transfer learning as a viable path forward. Transfer learning is a machine learning technique where a model developed for a task is reused as the starting point for a model on a second task. Usually the pre-trained model [2] has been trained on a large set of images and the set of classes in the second data set is much smaller. Thus, we started with a pretrained model and updated all of the model's parameters for our new task, in essence retraining the whole model. The outline of steps is as follows:

1. Initialize the pre-trained model.
2. Reshape the final layer(s) to have the same number of outputs as the number of classes in the new data set
3. Define for the optimization algorithm which parameters we want to update during training
4. Retrain the model

In our case, we updated the box predictor layer of a pretrained Faster-RCNN model from Pytorch. Faster-RCNN is one of the best models for object detection, utilizing components called region-proposal networks, which take an image as input and output a set of rectangular-bounded object proposals [7]. A schematic for the Faster R-CNN is shown below [3]:



To create our smaller dataset of desk images, our team found around 50 'desk' images on the internet and annotated them with 4 classes, 'monitor', 'keyboard', 'desktop', and 'plant'. To do so, we used an annotation tool [1], in which we labeled and drew bounding boxes for each object in an image. We compiled all this data into a CSV. Finally, we used this CSV to create a Pytorch compatible Dataloader. The loss for our R-CNN consisted of two components: classification loss and localization loss. While the former was determined whether the RPN's predicted label for an object was correct (calculated using cross-entropy loss), the latter sought to determine the accuracy of bounding for a given prediction. This involved taking the Euclidean distance between predicted and actual coordinate bounds [7].

4.2 Results

Our results on the combined desk image is shown below. Although we were able to successfully detect the

Testing Result

Training Sample

monitor on our stitched result, we were unable to do so for the keyboard and desktop, likely due to the fact that the former was partially hidden and the latter is a unique design (compared to our training samples).

5 Conclusion & Future Considerations

Although we were overall successful in achieving our intended aim of consolidating video footage into a singular 2D plane upon which object recognition tasks could be performed well, there are several, we originally had much more ambitious plans of also reconstructing video in 3D which turned out to be far too ambitious to achieve. Future extensions include implementing this, potentially by applying a depth determination algorithm to project the 2D image into its respective 3D components (see reference) [4].

Additionally, while we were able to identify the monitor in our final image correctly, our object detection model was unable to do so for the desktop, showing that there is significant room for improvement. We have brainstormed the following potential improvements and optimizations for our models.

Mapping:

- For something gridlike (e.g. solar panels), apply homography to edge/line detection and do line straightening i.e. (Sobel y-direction filter, canny edge detection, calculate Hough lines)
- Instead of stitching each new frame onto the stitched of the previous frames, use the first as the base and keep track of and build up homographies for each succesive frame. Then stitch them all at once at the end. Check this reference [9].
- Too much space in model robust, but this can be fixed by auto cropping.
- More preprocessing such as the vertical line rotation.
- Keypoints stop getting set on the panels in later frames.

Object Detection:

- Utilize a larger training sample set with greater variety (i.e. 'non-desk' images): 50 images was not enough for accurate detection. Maybe train on the COCO dataset.
- Train a R-CNN from scratch (something we did not have time to repeatedly do within the span of this project) and compare results to those obtained via transfer learning.
- Grayscale images before training or prediction.
- Duplicate training instances with various transformations

We found the skills we learned in 581 to be extremely useful when completing this project. We hope to continue exploring computer vision in the future, and want to thank the TAs and the professor for a great semester!

Bibliography

[1] URL `https://www.makesense.ai/`.

[2] Source code for torchvision.models.detection.faster_rcnn. URL `https://pytorch.org/docs/stable/_modules/torchvision/models/detection/faster_rcnn.html`.

[3] S. Goswami. A deeper look at how faster-rcnn works, Jul 2018. URL `https://whatdhack.medium.com/a-deeper-look-at-how-faster-rcnn-works-84081284e1cd`.

[4] Mapillary. mapillary/opensfm. URL `https://github.com/mapillary/OpenSfM`.

[5] S. P. and S. Dinesh. Panoramic image creation. *IOSR Journal of Electronics and Communication Engineering*, 2017. doi: 10.9790/2834.

[6] S. A. Perera. A comparison of sift , surf and orb, Aug 2018. URL `https://medium.com/@shehan.a.perera/a-comparison-of-sift-surf-and-orb-333d64bcaaea`.

[7] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017. doi: 10.1109/tpami.2016.2577031.

[8] Scikit-Image. scikit-image/skimage-tutorials. URL `https://github.com/scikit-image/skimage-tutorials/blob/master/lectures/solutions/adv3_panorama-stitching-solution.ipynb`.

[9] user622194, memecsmemecs5, LukeLuke3, and B. T. Tymchenko. Stitch multiple images using opencv (python), Aug 1963. URL `https://stackoverflow.com/questions/24563173/stitch-multiple-images-using-opencv-python`.

Data Sources:
`https://pixabay.com/videos/from-the-air-from-above-9798/`
`https://pixabay.com/videos/port-yachts-water-sea-boat-marina-33014/`
`https://www.reddit.com/r/Minimal_Setups/`
`https://images.google.com/`