Video Tracking

charles dunn

January 27, 2017

1 Problem

The goal of this project was to create an algorithm that tracks a rectangle in a video file. The inputs are a video with moderate unsteadiness and a rectangle specified by (x, y) pixel coordinates from the top left corner of the first frame.

In the sample videos provided, and perhaps unsurprisingly considering Uru's product, the rectangle is on a flat, blank wall. There are some discernible objects in view, like a TV and a door.

2 Approach

Typical object tracking relies on obvious features on the object, such as text or sharp corners. This direct tracking method of clear features is a luxury in our case. There may be an approach using speckle patterns on the wall. I decided the video quality and lighting were much too poor to use this approach.

I therefore settled on the idea of tracking the rectangular section of wall by measuring the movement of other objects in the scene. Ideally, we would only track objects that are close to the rectangle in 3D space, but it turns out the video is surprisingly devoid of features. I attempted to use a variety of typical feature extraction methods, including SIFT, SURF, FAST, and BRISK. A typical image correspondence problem could be solved by using RANSAC with a sparse feature set. None of the standard feature extraction methods were providing enough features to robustly track the scene, however. Even after lowing default thresholds significantly, stable solutions were not found. An additional problem was the processing time required by feature extraction and RANSAC. The benefit is a generation of a full homography matrix, representing shifts, scales, rotations, and skews of our original rectangle in 3D space. I even had dreams of potentially applying a Kalman filter to the homography matrix produced by feature correspondence in order to smooth the final output. In the end, I tried a variety of methods to improve the performance with this method including filtering the homography matrix elements and/or rectangular coordinates over time.

I then went back to the drawing board and decided dense correspondence would be a better approach. This initially seemed to be much more computationally intensive that sparse features. The basic approach is to simply find the x and y offsets between each adjacent frame. There are a few different metrics commonly used, typically the sum of squared differences (SSD/L2) or sum of absolute differences (SAD/L1). I opted to use the sum of squared differences on the gradient of the frames.

I used the gradient for two reason. First, I was worried that the large, flat areas of the video would dominate an offset search, and that shadows in the video would lead to large errors. The gradient, however, denotes edges and corners where tracking is less ambiguous. Calculating the gradient also helped with finding angle offsets later.

Using the gradient images, I hoped to find a minimum in the SSD between the previous frame and a shifted version of the current frame. Calculating the SSD for integer offsets within a 32x32 pixel region was extremely slow, as you might expect. I therefore made the assumption of convexity within this small region. This is effectively equivalent to assuming only small shifts between frames, or input video without extreme shaking or motion.

With the assumption of convexity for the region of small offsets, I search the SSD space over each integer pixel offset by a numerical gradient descent of sorts. I initialize my search by the offset of the previous frame (employing the assumption that the camera motion is not entirely independent from frame to frame), and search a small region around my initial point. I then find the minimum in my SSD function, and again search in a small region around this new point. I check if the SSD has already been calculated to avoid repeated computation. I limit the offset to be ± 32 pixels in any one direction.