Venkatakrishnan Ranganathan
CV – MP7

Template-matching based Target Tracking

Introduction:

Template matching is a popular technique used in computer vision for target tracking applications, where the goal is to locate a specific object or target in each scene or video. The technique involves comparing a pre-defined template image with a larger search space, looking for the best match based on a certain similarity measure. The objective is to estimate the location and orientation of the target in each frame of a video stream, allowing for its trajectory and movement to be tracked over time.

There are several methods for calculating the similarity between the template and the search space, including Sum of Squared Differences (SSD), Cross-Correlation (CC), and Normalized Cross-Correlation (NCC). SSD measures the difference between the intensity values of each pixel in the template and the corresponding pixel in the search space, summing them up to obtain a single similarity score. CC calculates the correlation between the intensity values of the template and the search space, giving higher scores to regions with high correlation. NCC is similar to CC, but normalizes the intensity values to account for differences in brightness and contrast.

In addition to selecting a suitable similarity measure, the performance of template matching can be improved by using a local search window. This technique involves sliding the template over small regions of the search space, rather than comparing it to the entire search space at once. By performing the template matching on smaller sub-regions, it is possible to reduce the computational complexity of the algorithm and improve its accuracy, especially in cases where the target is moving rapidly or undergoing significant changes in appearance.

Python Solution:

Python script for creating a video from a sequence of images, where the images have bounding box around a target object. The video is created by iteratively adding each image to the video container object. The script includes functions for calculating the normalized cross-correlation score (NCC), cross-correlation score (CC), and sum of squared differences (SSD) between two images.

The script begins by importing the necessary libraries, which include av, PIL, os, and numpy.
The first function, **target_tracking_ncc_g(img1, img2)**, calculates the NCC score between two grayscale images. The function first converts the input images into NumPy arrays and calculates the mean and standard deviation of each image. The images are then normalized and the NCC score is computed.

The second function, **target_tracking_cc_g(img1, img2)**, calculates the CC score between two grayscale images. The function also converts the input images into NumPy arrays, calculates the mean of each image, subtracts the mean from each image, and computes the CC score.

Venkatakrishnan Ranganathan
CV – MP7

The third function, **target_tracking_ncc(img1, img2)**, calculates the NCC score between two color images. The function first converts the input images into NumPy arrays and calculates the mean and standard deviation for each color channel. The images are then normalized and the NCC score is computed for each channel. The function returns the average NCC score across all channels.

The fourth function, **target_tracking_cc(img1, img2)**, calculates the CC score between two color images. The function first converts the input images into NumPy arrays and computes the CC score for each color channel. The function returns the average CC score across all channels.
The fifth function, **target_tracking_ssd(template, image)**, calculates the SSD between a template image and an input image. The function converts the input images into NumPy arrays, calculates the sum of squared differences between the template and the image, and returns the total sum of squared differences.

The sixth function, **target_tracking_ssd_g(template, image)**, calculates the SSD between two grayscale images. The function converts the input images into NumPy arrays, calculates the sum of squared differences between the template and the image, and returns the total sum of squared differences.

The last function, **images_to_video(image_folder_path, video_path, fps, comp_type='ssd')**, creates a video from a sequence of images. The function takes four input parameters: the path to the folder containing the images, the path and name of the output video file, the frame rate of the video, and the type of comparison to use when tracking the target object. The function loads the first image from the folder to get the size of the images, creates a video container object, and adds a video stream to the container. The function then iterates over all the images in the folder, loads each image as a PIL image, converts it to RGB mode, and aligns it to the target object. The aligned image is then added to the video stream in the container. The function uses one of the four comparison methods to track the target object: normalized cross-correlation for grayscale images, cross-correlation for grayscale images, normalized cross-correlation for color images, and cross-correlation for color images. The comparison method is specified using the **comp_type** parameter, which defaults to SSD.

Analysis:

The time complexity for template matching per frame with a sliding window of 30x30 can be calculated as follows:

Let's say we have an image of size MxN and a template of size PxQ. First, we need to slide the 30x30 window over the image. This can be done in O((M-30+1) x (N-30+1)) time complexity. For each position of the sliding window, we need to compute the sum of squared differences (SSD) between the template and the image pixels covered by the 30x30 window. The time complexity for computing SSD is O(PxQ), since we need to compute the difference between each pixel in the template and the corresponding pixel in the image, and then sum up the squared differences.

Therefore, the total time complexity for template matching with a sliding window of 30x30 is O((M-30+1) x (N-30+1) x PxQ). Note that this is just the time complexity for the brute-force method of template matching.

There are more advanced algorithms like Fast Fourier Transform (FFT) based template matching, which can reduce the time complexity significantly.

Results:

The results of the template-matching based target tracking solution indicate that there is scope for improving the performance of the system. One potential area of improvement is optimizing the size of the local search window used during template matching. The current implementation uses a fixed size local search window of 30 x 30, which may not be optimal for all tracking scenarios. By experimenting with different window sizes and shapes, it may be possible to improve the accuracy of the tracking system.

Another interesting finding is that color SSD (Sum of Squared Differences) performs better than grayscale SSD, indicating that color information can be useful for tracking targets. However, when it comes to grayscale CC (Cross Correlation), it outperformed the color CC. This suggests that while color information can be useful in some cases, it may not always be necessary for achieving high accuracy in tracking.

When it comes to NCC (Normalized Cross Correlation), grayscale NCC performed better than color NCC, as well as both grayscale and color SSD and CC. This is an important finding as NCC is known to be more robust to changes in illumination and contrast, making it a promising approach for target tracking in diverse environments.

Finally, there is a need for improving the overall time complexity of the system. The current implementation requires a significant amount of computation for each frame, which may not be feasible for real-time applications. By optimizing the algorithms used for template matching and other tasks, it may be possible to reduce the overall computational requirements and improve the efficiency of the tracking system.

Venkatakrishnan Ranganathan
CV – MP7

Extra Credit

Even when the girl's face is partially blocked by another person, the tracking algorithm still successfully tracks her face. This is because the algorithm always selects the highest normalized cross-correlation value (without setting a threshold) within the search window for each frame. Even though the similarity is decreased due to the occlusion, the algorithm can still identify the girl's face. However, this approach has a disadvantage in that it always produces a bounding box without considering the accuracy of the match. The video file attached to this submission, ***result_video_ncc.mp4***, displays the results of this algorithm.