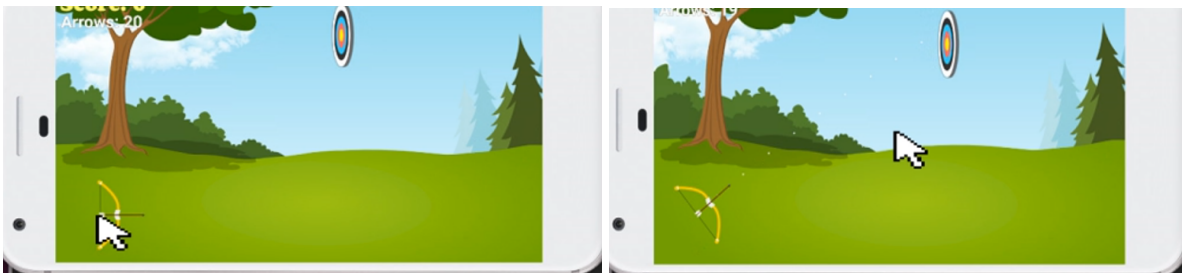
Motion Detection in Videos

Goal:

The input is a video consisting of multiple frames. My goal is to detect the objects moved constantly, significantly, and in different directions in the video and will output figures of the objects detected.

Example:

Input: a video



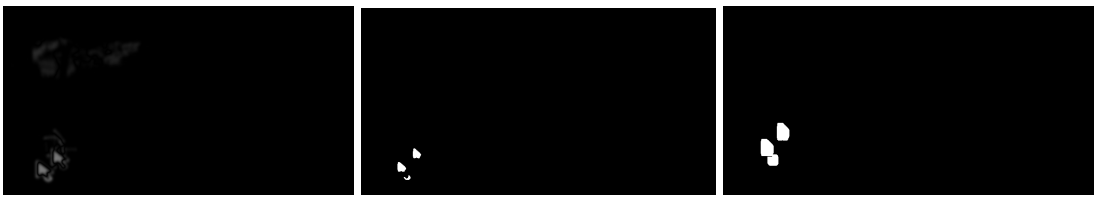
(1) Example frame 1 *(2) Example frame 2*

In the input video, the cursor and the target moved constantly. The players control the cursor to shoot arrows to the target. If the arrow hits the target, the target will move to the next place.

In the pictures shown above, we can see that the cursor moved from left to right significantly and we know it will move to other places later. The moving direction of the cursor will not be left only. Thus, our algorithm will detect the cursor and capture it. However, the cloud behind the tree moves subtly and in a single direction, left. Thus, our algorithm will consider the cloud as a natural moving object and will not output the picture of the cloud.

Basic Idea:

The basic assumption behind motion detection is to assume the first frame is a static background. Then, we do matrix subtraction between the first and later frames. If nothing moves, we will have a null matrix after subtraction. However, if something moves, we will have a block of non-zero numbers representing changing pixels in a later frame. Then, we set a certain threshold to eliminate trivial change but keep parts changed significantly. In the end, we thresholded the figure to increase the area of white part (moving objects) in the picture so that we can find the contour of this object.



(3) Pictures after subtraction (4) Thresholded picture (5) Dilated picture

In the end, I set a threshold to eliminate all the gray parts and dilated the picture to find contours and ignore the parts which are too small. Finally we successfully get positions of the moving objects in the first frame.

(11) Remove the gray part (12) Dilation

Diffculties:

Change of background:

In some game videos, the background is not static. For example, in candy-crush-like games, eventually, the player will eliminate all the candies occured in the first frame. Thus, when we do figure subtraction with the first frame in this scenario, the computer will think that everything is moving.

(4) First Frame

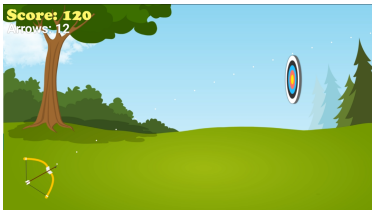
(5) Later Frames

Duplicates:

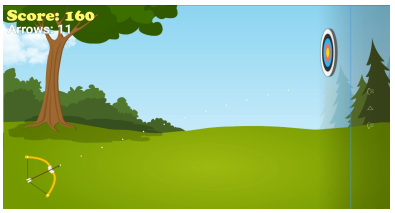
Since we did figure subtraction for all the frames with the first frame and rey to find the contour from it, we will inevitably capture the same objects again and again.

Meaningless captured:

Suppose Figure (6) is the first frame, and Figure (7) is one of the later frames. The target occured in the first frame and moved a little bit in the later frames. From the subtraction result, we can see that there are two targets shown, which means that our computre will think that there are two moving objects and will store them. Eventually, we will have two figures: one is the target, one is part of the sky.

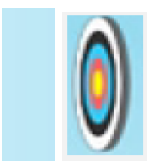


(6) First Frame



(7) Later Frames

(8) Subtraction

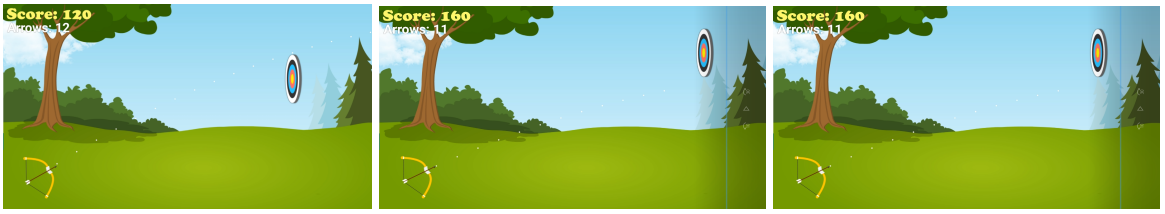


(9) outputs

Solution:

Change of background:

To solve this problem, we will not set a golden first frame and let it be a static background. Instead, we will change the background after a certain time period (represented by frames) determined by users. In other words, if users set the time period to be eight frames, the program will set the first frame to be the frame that came in eight frames later. By doing this, if the background changed completely, our program will still function properly.



(10) Old First Frame *(11) Frame After X Frames* *(12) New First Frame*

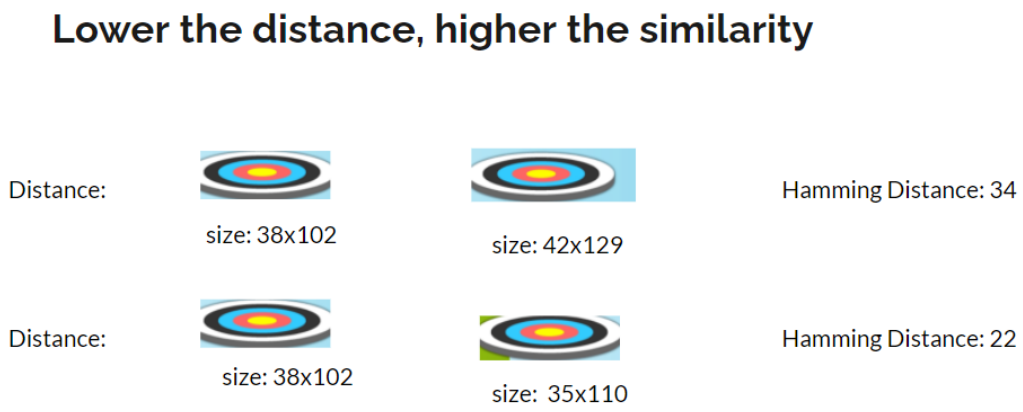
Duplicates:

To remove duplicates from captured objects, we need to calculate the similarity of figures and remove the figures that are too similar to other figures. A straightforward way to calculate the figure similarity is to calculate the pixel similarity. We will first transform two input figures into the same size and calculate the Euclidean distance as error between them. In the end, we use (1 - error) / figure size, we can get a number that represents the similarity. We will remove the figures that are too similar to each other.

Although this method works well mostly, it is severely influenced by the size of the input figure and the background color.

(13) Problem of previous method

Thus, an alternative method is to detect the feature point of the object in the figure. In the example arrow shooting games, the bow and the target are objects having distinct features; instead of comparing each pixel, we can compare the feature point detected. I decided to use AZAKE feature detection which is an efficient and also stable algorithm. Then, I use a BF matcher that will compare all the feature points detected in the first input figure with all the feature points detected in the second input figure. We can sum the distances among these points and divide them by the total number of feature points. Finally, we can get the distance between objects detected in two figures.



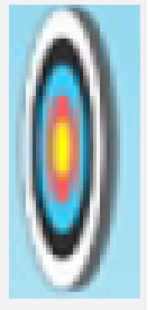
(14) Problem solved

Meaningless captured:

To remove the unwanted capture, the most straightforward way is to detect the feature point in the captured figure. If the feature points cannot be detected or the number of detected points is lower than the expectation, we can know this captured figure does not have wanted objects or just a blank figure.

Final Output: the moving cursor and the target

(15) Moving cursor



(16) Target