Virtual TV - capture a video of a frame to act as a TV's bounding frames and inlay a video to fit this frame

Abstract

This is a project that we have come up named as "VIRTUAL -TV". So a virtual-tv means a virtual set in which a television studio which allows the background to be manipulated in real time.
A virtual set will permit you to alter the background or setting with digital graphics your imagination is the limit. Some of the benefits of Vitual-tv is that it helps in visualisation, reducing the cost, creates a safe environment for learning , etc

Objectives

Our main objective is to play a video in a moving or video background.

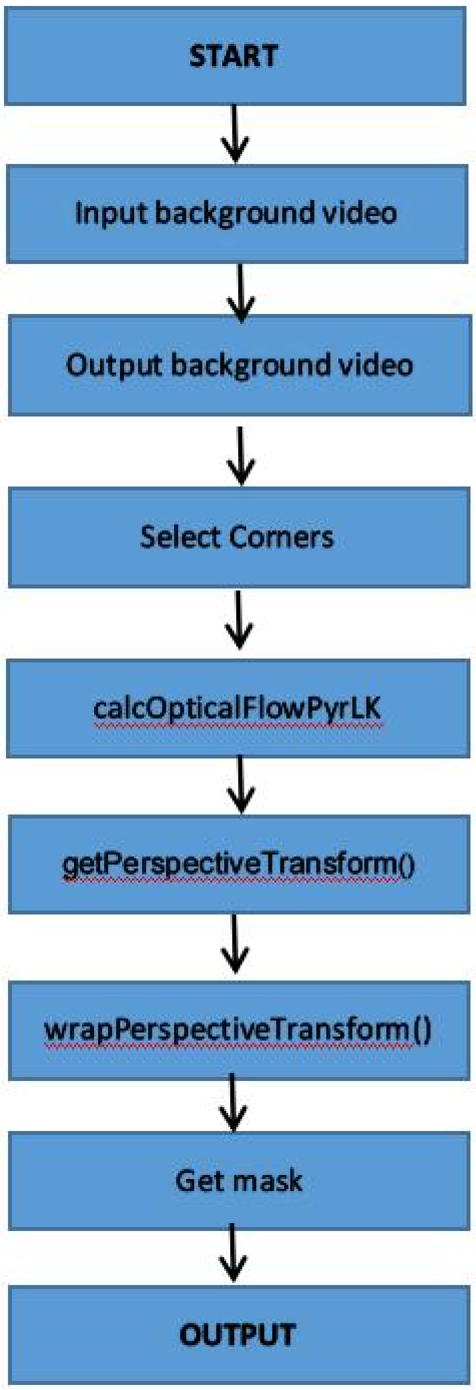
Assumptions

For doing this all we want moving background and a video to play inside that background . We have used python language for coding and used Open CV which is a huge open-source library for image processing .
In this project in the moving background there should be a frame and the frame should be with a dimension of four vertices. The video given as the input to play inside the frame should track the given points on the frame and should be within that frame.

System Architecture

For developing this library we have used Open CV and the the language we took for coding is python since it has simplesyntax , helps in implementing new features and extensive libraries. For doing this any language can be used but it should have support of Open CV.

BLOCK DIAGRAM



Description

1) Start: Beginning of the task

2) Input Background Video: Giving a video as input which contains the frame.

3) Input Capture Video: The input video that should be placed inside the frame.

4) Select Corners: Selecting the 4 corners in which the video should be placed.
5) calcOpticalFlowPyrLK(): Calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids.
using the calcOpticalFlowPyrLK function which takes the old frame (note: not the tv frame) and the current frame (note: not the tv frame), and the old positions of the corners then the function calculates the new position of the corners.

6) getPerspectiveTransform(): Getting the perspective transform using the getPerspectiveTransform function arguments passed are the four corners of the second video stored in pt1 variable and the new corners variable which contains the newly updated corners of the tv frame.

7) wrapPerspectiveTransform():warpPerspective function is used to apply the previously found matrix on the second video. We also make sure that the size of the resultant image is same as the size of the first video.

8) Get mask: once we get the mask we apply this mask for each frame (note: not the tv frame) of the first video (which is defined by the frame vairable) i.e we apply the mask in every iteration for the frame in the first video.

Image Processing Modules-

Optical flow: Optical flow is a task of per-pixel motion estimation between two consecutive frames in one video. Basically, the Optical Flow task implies the calculation of the shift vector for pixel as an object displacement difference between two neighboring images. The main idea of Optical Flow is to estimate the object's displacement vector caused by it's motion or camera movements.

Lucas-Kanade Algorithm-

The Lucas-Kanade method is commonly used to calculate the Optical Flow for a sparse feature set. The main idea of this method based on a local motion constancy assumption, where nearby pixels have the same displacement direction. This assumption helps to get the approximated solution for the equation with two variables.

The common approach in practice is to use a multi-scaling trick. We need to create a so-called image pyramid, where every next image will be bigger than the previous one by some scaling factor (for example scale factor is 2). In terms of the fixed-size window, the abrupt movement on the small-sized image is more noticeable rather than on the large one. The founded displacement vectors in the small images will be used on the next larger pyramid stages to achieve better results.

As we mentioned before, Dense Optical Flow algorithms calculate the motion vector for the sparse feature set, so the common approach here is to use the Shi-Tomasi corner detector. It is used to find corners in the image and then calculate the corners' motion vector between two consecutive frames.

OpenCV has the implementation of Pyramid Lucas & Kanade with Shi-Tomasi algorithm

Contribution Of Each Member of the team:

1. GAUTAM KRISHNA(AM.EN.U4CSE20325)

    Worked on Get mask function: by creating a mask which can be used to get a polygonal region enclosed by the four corners.

2. SRILAKSHMI S R (AM.EN.U4CSE20269)

    Worked on Optical Flow: the old frame (note: not the tv frame) and the current frame (note: not the tv frame), and the old positions of the corners then the function calculates the new position of the corners.

2. KARTHIK PRASAD (AM.EN.U4CSE20240)

    Worked on Selecting Corners : Setting the frame in the background and aslo worked on setMouseCallback function.