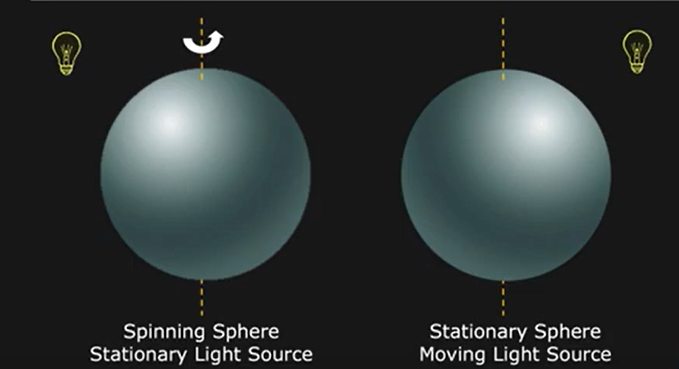
**AGV Task 1**

**By Uddipto Mandal (24IM10065)**

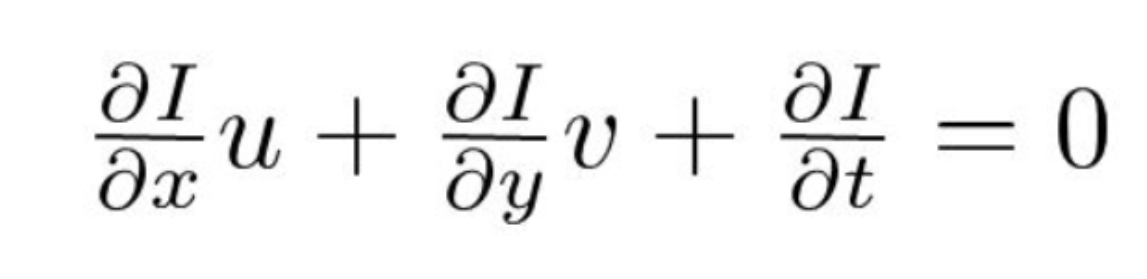
**Sources:**

* <https://www.youtube.com/playlist?list=PL2zRqk16wsdoYzrWStffqBAoUY8XdvatV>
* <https://www.youtube.com/watch?v=5AUypv5BNbI>
* <https://ppms.cit.cmu.edu/media/project_files/OpticalFlow_IV2020_final.pdf>
* <https://nanonets.com/blog/optical-flow/>

**Subtask 1**

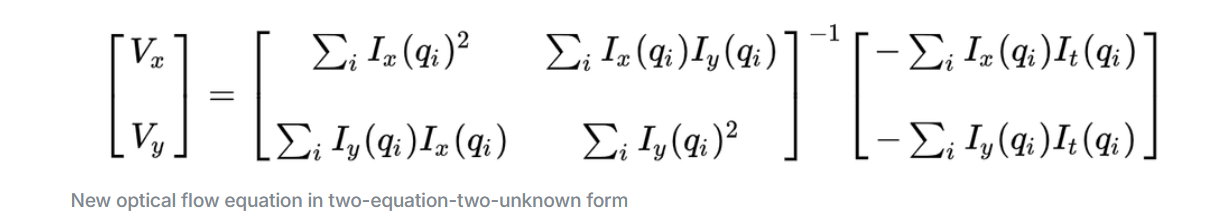
Here we had to implement Lucas Kanade sparse optical flow algorithm, on clips of the video of GTA San Andreas. When a body moves in 3d world, its motion vector (or velocity vector) is called the motion flow. When we capture his movement through a camera, and see it as different frames on 2d, the motion of the projection of the body on the 2d plane is called optical flow. For many cases optical flow and motion flow may be same, but in many they are different. Specifically: when the body is moving in a direction perpendicular to the camera plane, then that motion cannot be tracked by the camera, and does not show up in the optical flow. This is called the **aperture or pinhole problem**. Also, optical flow is calculated based of the intensities of a point. If the intensity of a point changes with time, but the body does not move, optical flow will think it was moving. An example of this is given below. 

Here in 1st case, optical flow is 0 while motion flow is there, and in 2nd case, optical flow is there while motion flow is 0.



This is the optical flow equation, which we get when we assume the intensity of a certain point as it moves across the different frames in the video is constant, and applying taylor series expansion, and assuming it to be linear. This is an under-determined equation. From here there a number of algorithms we can use to find the optical flow, such as Lucas Kanade,Horn Schunk, Farneback’s, or Deep learning based methods, but we use the Lucas Kanade sparse optical flow method.

Lucas Kanade method assumes the optical flow in a small window to be same. Then the equation becomes overdetermined, and we use Lease square fitting to solve it. After some rearrangements we get the equation below.



This is how we calculate optical flow for a point(x,y), where qi are the points in a window around the point (x,y). Ix, Iy are the partial derivatives wrt x and y, while It is the time derivative. For

**My code:**

In my code, I have looped through all the frames in the video, made them grayscale, calling prevgr = img in previous frame, and framegr as img in current frame. I have also initialized prevgr for the first frame. Then I have used the inbuilt function goodFeaturesToTrack for shi tomasi corner detection, on which points I will apply sparse optical flow. I have marked these points by red in the frame. For each point, I have applied lucas Kanade algorithm, assuming window size of 3x3. I have used 5x5 sobel operator for calculating the Ix and Iy gradients. I had tried with using 3x3 sobel and using definition of derivative on 2x2 blocks also but 5x5 sobel seemed to give the best results.

‘arr1’ is the window of the previous frame, while ‘arr2’ is the window in the current frame. Then I convolved with the sobel kernel to get the derivatives in x and y direction for each point in the window. I also calculated the time derivative for the points. I appened the [Ix, Iy] to a matrix A, and [-It] to B. Hence Av = B, where v is the optical flow.

A is a 9x2 matrix. To get v, we need it to first be an invertible matrix. Hence the steps we use is:

* Av = B
* (A.T)Av = (A.T)B
* v = inv[(A.T)A](A.T)(B)

I have used a try catch block there as A.T times A may have cases where it becomes non invertible, so it handles the error.

The optical flows that I got were in the range of 0.00xxxx which will only be considered as 0. So I have scaled all the optical flows by a factor of 75 which is giving a decent flow. It is not as continuous lines as it should have been and I am unable to figure out how to make it continuous lines as the problem statement wanted, but this is the best fix I could figure out. The inbuild function of lucas Kanade gets fully continuous lines throughout the frames but I was not able to fix the code to give such continuous lines. The points are being tracked well though, as we can see by replacing the good points to track by following a single point throughout the frames.

Points\_and\_vectors holds 2 points in each element, and in each frame we need to draw a line between all of these pairs. I have drawn those points in green. Then is the waitkey(5) meaning there is 5 millisecond wait for each frame, and we can also exit the output video at any time by pressing q. Then is releasing video capture and closing all windows.

**Subtask 2**

**Optical flow based controller**

I was not able to implement this subtask.

I have gone through the paper mentioned, (<https://ppms.cit.cmu.edu/media/project_files/OpticalFlow_IV2020_final.pdf>) and understood a lot of it and a high level, although I am unclear on why that method works in most of the formulae given.