Car Tracking Algorithm-v3

Algorithm 1 tracking

```
Input Parameters:
imagespath:(List of paths to all images we want to perform tracking on.)
boundingbox:(A box around car at first frame. This box will update as we move to next images.)
IntervalSize: (Interval Size to calculate forward and backward flow error.)
GeomatricThreshold: (Threshold to filter points on base of geomatric distance.)
ForwardZeroFlowThreshold: (Threshold to filter points on base of zero flow while moving forward.)
BackwardZeroFlowThreshold: (Threshold to filter points on base of zero flow while moving backward.)
DescriptorThreshold: (Threshold to filter points on base of descriptor matching.)
flowinboxmethod: (method to computer flow in box from one frame to other .)
ransacmethod: (method to compute ransac vector for flow vectors..)
maxcorners: (specfiy how many corner you want to find at max.)
qualitylevel: (threshold to get or ignore corner in corner detector)
mindistance: (Minimum possible Euclidean distance between the returned corners.)
useHarrisDetectorboolean: (boolean to check whether to use harris detector or not)
k: (free parameter in harris corner detector.)
Output:
featurespointineveryframe: (The list of list of good points which we track in all images

My Algorithm:
$featurespointineveryframe = []$
$ImageCounter \leftarrow 0$
while $ImageCounter < Len(imagespath)$ do
    $images, grayimages \leftarrow getframes(imagespath, ImageCounter, IntervalSize)$
    $pointsat1stframe \leftarrow Initialization(BoundingBox, grayimages[0], method =' gCorners', maxcorners, qualitylevel, mindistance, useHarrisDetectorboolean, k)$
    $pointsatkframe, filteredpointsat1stframe \leftarrow getforwardbackwardflow(grayimages, pointsat1stframe, pointsat1stframe, ForwardZeroFlowThreshold,' forward')$
    $pointsat1stframebackward, filteredpointsat1stframe \leftarrow getforwardbackwardflow(Images, pointsatkframe, filteredpointsat1stframe, BackwardZeroFlowThreshold,' backward')$
    $GoodPoints \leftarrow applyallfilters(filteredpointsat1stframe, pointsatkframe, pointsat1stframebackwar, grayimages, GeomatricThreshold, DescriptorThreshold, method =' ssd')$
    $featurespointineveryframe.append \leftarrow GoodPoints$
    $Flow, FlowPoints \leftarrow CalculateBoxFlow(grayimages[ImageCounter], grayimages[ImageCounter], GoodPoints, flowinboxmethod, ransacmethod)$
    $BoundingBox \leftarrow UpdateFlowvector$
    if $ImageCounter == len(AllImages) - 2$ then
        $featurespointineveryframe.append \leftarrow FlowPoints$
    $ImageCounter + +$
return $featurespointineveryframe$
```

Algorithm 2 getframes

```
Input Parameters:
imagespath: (This is the list of images path in same order as images in video stream.)
firstimageindex: (starting index to read images from path given in list of images path.)
intervalsize: (Specifies how many frames to read)
images: ( queue of numpy arrays and each index specfiy a rgb image.)
grayimages: (queue of numpy arrays and each index specfiy a image in gray scale.)
Output:
images: (queue of numpy array and each index specify a rgb image.)
grayimages: (queue of numpy array and each index specify a gray image.)
Status:(1 for successful and -1 in case of no frame found)

My Algorithm:

if $firstimageindex$ or intervalsize $< 0$ then return $images, grayimages, status \leftarrow$ -1
else
    $a \leftarrow$ firstimageindex + intervalsize
    if $a > len(imagespath) - 1$ then
        $a \leftarrow len(imagespath) - 1$
    if $images.empty$ then
        while $firstimageindex < a$ do
            img $\leftarrow$ read(imagespath[firstimageindex])
            $grayimg \leftarrow convertimgtograyscale$
            $images.enqueue \leftarrow img$
            $grayimages.enqueue \leftarrow grayimg$
            firstimageindex++
    else
        $images \leftarrow images.dequeue$
        $grayimages \leftarrow grayimages.dequeue$
        img $\leftarrow$ read(imagespath[a-1])
        $grayimg \leftarrow convertimgtograyscale$
        $images.enqueue \leftarrow img$
        $grayimages.enqueue \leftarrow grayimg$
    if $len(images) == 0$ then return $images, grayimages, status \leftarrow$ -1
    else
        return $images, grayimages, status \leftarrow$ 1
```

Algorithm 3 initialization

```
Input Parameters:
boundingbox: (This contains the starting and ending index of box.[xmin,ymin,xmax,ymax].)
grayimage: (Image of which we want to initilize points.)
method: (Method which specify, how to get Image Points)
maxcorners: (specfiy how many corner you want to find at max.)
qualitylevel: (threshold to get or ignore corner in corner detector)
mindistance: (Minimum possible Euclidean distance between the returned corners.)
useHarrisDetectorboolean: (boolean to check whether to use harris detector or not)
k: (free parameter in harris corner detector.)
Output:
Points: (List of points in range of given box)
Status:(1 for successful and -1 in case of no points found)

My Algorithm:

$Status = -1$
if $Method == "grid"$ then
    Points ← points between(xmin,ymin) and (xmax,ymax)
    return $Points, status$ ← 1
if $Method == "gCorners"$ then
    $patch = im[ymin : ymax, xmin : xmax]$
    $corners \leftarrow goodFeaturesToTrack(patch, maxcorners, qualitylevel, mindistance, useHarrisDetector =$
    $useHarrisDetectorboolean, k)$
    $corners \leftarrow corners.reshape((-1, 2))$
    $corners[:, 0], corners[:, 1] \leftarrow corners[:, 0] + xmin, corners[:, 1] + ymin$
    return $Points \leftarrow corners, status \leftarrow$ 1
```

Algorithm 4 getforwardbackwardflow

```
Input Parameters:
grayimages: (queue of numpy array of gray Images)
$featurepointsin1stimage : (Thislistcontainsthefeaturespointsonfirstimage, ofwhichwewanttocompute$
$filteredfeaturepointsin1stimage : (Thislistcontainsthefeaturespointsonfirstimage, ofwhichwewanttoc$
thresholdzeroflow: (This value specify the theshold value to filter background points from object points)
direction: (this specifiy the direction to compute flow on given images)
Output:
$trackedpointsinkimage : (givenfeaturepointsatlastframeofgivenimages.)$
$trackedpointsin1stimage : (Givenfeaturespoint, whichwewanttotrackongivensetofimages)$
Status:(1 for successful and -1 in case of no points found after zero flow filter)

My Algorithm:

$Status = -1$
$listofgrayimages \leftarrow list(grayimages.queue)$
if $direction ==' Forward'$ then
    $a \leftarrow 0$
    while $a < len(listofgrayImages)$ do:
        $next \leftarrow a++$
        $currentImage - Gray \leftarrow listofgrayimages[a]$
        $nextImage - Gray \leftarrow listofgrayimages[next]$
        $flowpoints, st, err \leftarrow cv2.calcOpticalFlowPyrLK(currentImage - Gray, nextImage-Gray, featurepointsin1stimage.astype(np.float32), None, ** lk_params)$
        $a++$
        if $a \leftarrow (len(listofgrayimages) - 2)$ then
            $trackedpointsinkimage, trackedpointsin1stimage \leftarrow CalculateZeroFlow(flowpoints, featurepointsin1stimage, thresholdzeroflow)$
else
    $a \leftarrow len(listofgrayImages) - 1$
    while $a > 0$ do:
        $next \leftarrow a--$
        $currentimage - Gray \leftarrow listofgrayimages[a]$
        $nextImage - Gray \leftarrow listofgrayimages[next]$
        $flowpoints, st, err \leftarrow cv2.calcOpticalFlowPyrLK(currentImage - Gray, nextImage-Gray, featurepointsin1stimage.astype(np.float32), None, ** lk_params)$
        $a--$
        if $a \leftarrow 1$ then
            $trackedpointsinkimage, trackedpointsin1stimage \leftarrow CalculateZeroFlow(flowpoints, filteredfeaturepointsin1stimage, thresholdzeroflow)$
if $len(trackedpointsin1stimage) == 0$ then return $trackedpointsinkimage, trackedpointsin1stimage, Status \leftarrow$ -1
else
    return $trackedpointsinkimage, trackedpointsin1stimage, Status \leftarrow 1$
```

Algorithm 5 CalculateZeroFlow

```
***Input Parameters:***
*pointsinKimage:(Points at the last frame after optical flow as an numpy array.)*
*pointsin1stimage:(Given Points at the first image as an numpy array)*
*threshold:(Threshold to filter points on base of zero flow)*
***Output:***
*filteredpointsinKimage: (Last Image point after zero flow filter)*
*filteredpointsin1stimage: (First Image point after zero flow filter)*

***My Algorithm:***

$filteredpointsinKimage = []$
$filteredpointsin1stimage = []$
$difference \leftarrow np.linalg.norm(pointsin1stimage - pointsinkimage, 2, 1)$
$tempdiffernece = difference < threshold$
$filteredpointsinKimage \leftarrow pointsinKimage[tempdiffernece]$
$filteredpointsin1stimage \leftarrow pointsin1stimage[tempdiffernece]$
**return** $filteredpointsinKimage, filteredpointsin1stimage$
```

Algorithm 6 applyallfilters

```
Input Parameters:
pointsin1stimage:(Features Points at start of image)
pointsinkimageforward:(Given Points tracked to given interval size.)
pointsin1stimagebackward:(Given forward Points tracked back to first image.)
grayimages:(Set of images of given interval size, on which we want to track points.)
GeomatricThreshold:(Threshold to filter points on base of geomatric distance.)
DescriptorThreshold:(Threshold to filter points on base of descriptor SSD or Dot Product)
Output:
filteredpoints: (Return Good features points which pass criteria of filters. )
Status: (Return -1 if no good feature point found otherwise 1.)

My Algorithm:

$varGoodPoints = []$
$varForwardPoints = []$
$filteredpoints = []$
$Status \leftarrow -1$
$geomatricdisplacement \leftarrow np.linalg.norm(pointsin1stimage - pointsin1stimagebackward, 2, 1)$
$tempgeomatricdisplacement = geomatricdisplacement < GeomatricThreshold$
$i \leftarrow 0$
$varGoodPoints.append \leftarrow pointsin1stimage[tempgeomatricdisplacement]$
$varForwardPoints.append \leftarrow pointsinkimageforward[tempgeomatricdisplacement]$
$orb \leftarrow cv2.ORB_{c}reate()$
$FirstImageGray \leftarrow grayimages[First]$
$LastImageGray \leftarrow grayimages[Last]$
$keyPoints_{G}ood \leftarrow KeyPoints(varGoodPoints)$
$keyPoints_{F}arword \leftarrow KeyPoints(varFarwordPoints)$
$Goodkp, GoodptsDes \leftarrow orb(FirstGrayImage, keyPoints_{G}ood)$
$Farwordkp, FarwordptsDes \leftarrow orb(LastGrayImage, keyPoints_{F}arword)$
$t = 0$
$gdpt_{d}es \leftarrow NormalizedGoodptsDes$
$forward_{d}es \leftarrow NormalizedFarwordptsDes$
$ssd_{d}istance \leftarrow np.linalg.norm(forward_{d}es - gdpt_{d}es, 2, 1)$
$tempssd_{d}istance \leftarrow ssd_{d}istance < DescriptorThreshold$
$FinalGoodPoints \leftarrow varGoodPoints[tempssd_{d}istance]$
if $len(FinalGoodPoints) == 0$ then return $FinalGoodPoints, Status \leftarrow$ -1
else
    return $FinalGoodPoints, Status \leftarrow 1$
```

Algorithm 7 CalculateBoxFlow

```
***Input Parameters:***
*firstimage:(Current Image from which we want to calculate flow)*
*nextimage: (Very next image to current image)*
*goodpoints: (Features points on current image to compute flow as numpy array)*
*method: (Method to compute box flow.)*
*ransacmethod: (Method to compute flow vector through ransac.)*
***Output:***
*Flow: (This contain the change in x and y coordinates, which specifies the change in flow of box.)*
*ForwardPoints: (This list contain flow of points from current image to next image)*

***My Algorithm:***

$ForwardPoints, st, err \leftarrow cv2.calcOpticalFlowPyrLK(currentImage, nextImage, FirstImageFeatureP$
    $lk_params)$
$flow \leftarrow (forwardpoints - goodpoints)$
**if** $method ==' median'$ **then**
    $flow \leftarrow np.sort(flow)$
    $flowinbox = np.median(flow)$
**if** $method ==' ransac'$ **then**
    $flowinbox = getRansac(flow, threshold, ransacmethod)$
***return*** $flowinbox, forwardpoints$
```

Algorithm 8 GetRansac

```
***Input Parameters:***
*flow:(Flow of good points, we get as output of calculate flow method)*
*Threshold:(Threshold to filter inlairs from outlairs)*
*Method:(method to get ransac. One option is to return best hypothesis and second is to return medium of*
***Output:***
*FlowPoint: (Return a single point which has the most number of inlairs.)*

***My Algorithm:***

$Inlairs = []$
$NormalizedFlowPoints \leftarrow Normalized\ to 1$
$samplesize \leftarrow 20$
if $len(NormalizedFlowPoints) - 1 < samplesize$ then
    $samplesize = len(NormalizedFlowPoints) - 1$
$randomsample = random.sample(range(0, len(NormalizedFlowPoints) - 1), samplesize)$
while $t < samplesize$ do
    $tempInlairs = []$
    $i \leftarrow randomsample[t]$
    $hypothesis \leftarrow NormalizedFlowPoints[i]$
    $product \leftarrow np.dot(NormalizedFlowPoints, hypothesis)$
    $tempproduct \leftarrow product > threshold$
    $tempInlairs \leftarrow product[tempproduct]$
    *loop:till len(product)*
    for $j \leftarrow inproduct$ do
        if $j >= threshold$ then
            $tempInlairs.append \leftarrow j$
    if $len(tempInlairs) >= len(Inlairs)$ then
        $Inlairs \leftarrow tempInlairs[]$
        if $method ==' besthypothesis'$ then
            $FlowPoint \leftarrow flow[i]$
        if $method ==' median'$ then
            $flow \leftarrow np.sort(inlairs)$
            $FlowPoint = np.median(flow)$
return $FlowPoint$
```