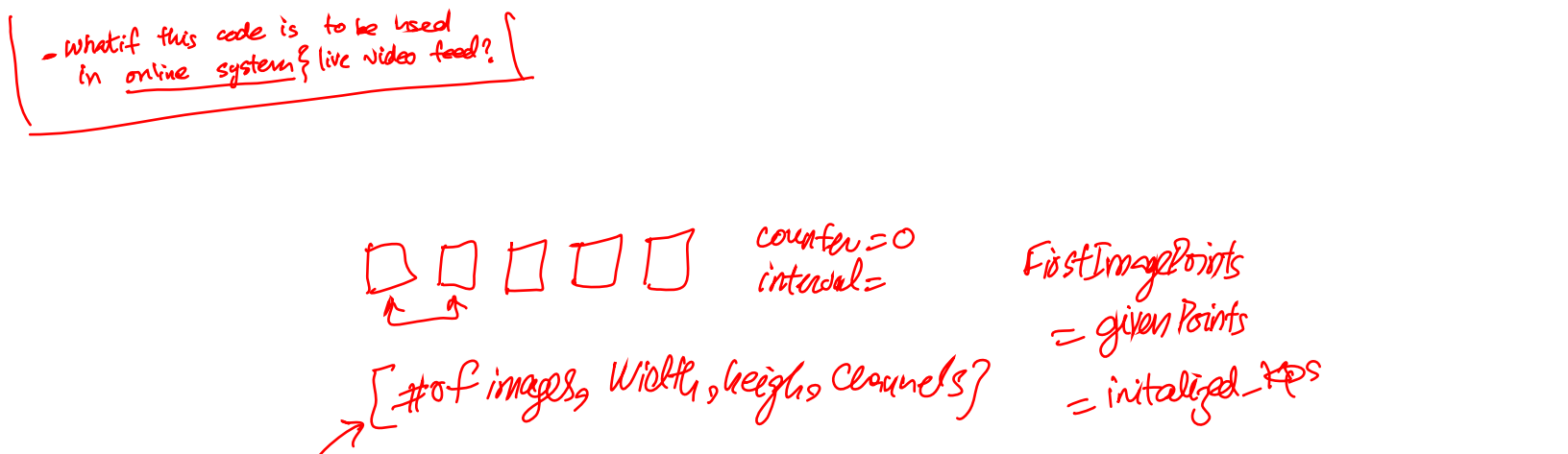
{Google Convention Guide on python programming [PEP-8]}

Car Tracking Algorithm

{Can this code 'track' multiple objects?

{Can this code work on Reve system by only changing 'get_frames' method?



Algorithm 1 Tracking

Input Parameters:
AllImages:(List of names of all images we want to perform tracking on.)
BoundingBox:(A box around car.)
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.)
Output:
TrackedGoodPoints: (The list of list of good points which we track

My Algorithm:
$FinalGoodPoints = []$
$ImageCounter \leftarrow 0$
loop:on len(AllImages)-1
$Images \leftarrow GetFrames(AllImages, ImageCounter, IntervalSize)$
$FirstImagePoints \leftarrow GetPoints(BoundingBox, Images[0], method = 'gCorners')$
$OpticalFlowForward, TrackedFirstImagePoints \leftarrow GetForwardFlow(Images, FirstImagePoints, ForwardZeroFlowThreshold)$
$OpticalFlowBackward, TrackedFirstImagePoints \leftarrow GetBackwardFlow(Images, OpticalFlowForward, TrackedFirstImagePoints, BackwardZeroFlowThre$
$GoodPoints \leftarrow GetGoodPoints(TrackedFirstImagePoints, OpticalFlowForward, OpticalFlowBackwar$
$ssd')$
$TrackedGoodPts \leftarrow TrackGoodPoints(Images, GoodPoints)$
$FinalGoodPoints.append \leftarrow TrackedGoodPts$
$Flow, FlowPoints \leftarrow CalculateFlow(Images[ImageCounter], Images[ImageCounter], GoodPoints)$
$RansacFlow = GetRansac(Flow)$
$BoundingBox \leftarrow UpdatebyRansacFlowvector$
if $ImageCounter == len(AllImages) - 2$ **then**
    $GoodPointsList \leftarrow FlowPoints$
    $FinalGoodPoints.append \leftarrow GoodPointsList$
$ImageCounter++$
endloop:on len(AllImages)
return $FinalGoodPoints$

Compute time = # of images × K × O(Image Read time)

Ideal time = # of images × O(Image Read time)

Error handling

Algorithm 2 GetFrames

time based

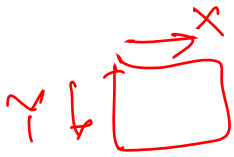
1: ***Input Parameters:***
2: *ListofImagesName: (List contains the path to every image in ascending order)*
3: *FirstImageIndex: (starting index to read images from list.)*
4: *IntervalSize: (Specifies how many frames to read)*
5: ***Output:***
6: *Images: (List of Images.)*
7: *Status:(1 for successful and -1 in case of no frame found)*
8: ***My Algorithm:***

error('-ve indices')

9: **if** $FirstImageIndex$ *or* $IntervalSize < 0$ **then return** $Images, status \leftarrow$ *-1*
10: **else**
11: $a \leftarrow FirstImageIndex + IntervalSize$
12: *loop*:
13: **if** $FirstImageIndex < a$ **then**
14: **if** $FirstImageIndex > len(ListofImagesName)$ **then return**
15: **else**
16: $Img \leftarrow read(ListofImagesName[FirstImageIndex])$
17: $Images.append \leftarrow Img$
18: $FirstImageIndex{+}{+}$
19: *Endofloop*:
20: **if** $len(Images) == 0$ **then return** $Images, status \leftarrow$ *-1*
21: **else**
22: ***return*** $Images, status \leftarrow 1$

K O(Image Read time)

{intervalsize} × image size



Algorithm 3 GetPoints (Initialization)

Input Parameters:
BoundingBox: (This contains the starting and ending index of box.[xmin,ymin,xmax,ymax].)
Image: (Image of which we want to get points.)
Method: (Method which specify, how to get Image Points)
Output:
Points: (List of points in range of given box)
Status:(1 for successful and -1 in case of no points found)

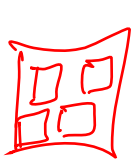
My Algorithm:

$Status = -1$
if $Method == "grid"$ **then** } Resolution
    $Points \leftarrow$ *points between(xmin,ymin) and (xmax,ymax)*
    return $Points, status \leftarrow 1$
if $Method == "gCorners"$ **then**
    $patch = im[ymin : ymax, xmin : xmax]$
    $corners \leftarrow goodFeaturesToTrack(patch, 1000, 0.0001, 1, useHarrisDetector = True, k = 0.001)$
    $corners \leftarrow corners.reshape((-1, 2))$
    $corners[:, 0], corners[:, 1] \leftarrow corners[:, 0] + xmin, corners[:, 1] + ymin$
    return $Points \leftarrow corners, status \leftarrow 1$

[numpy arrays]

zero flow till 'K' frames

K ≠ # of frames for PB tracking



Single call for multi-objects

Algorithm 4 GetForwardFlow

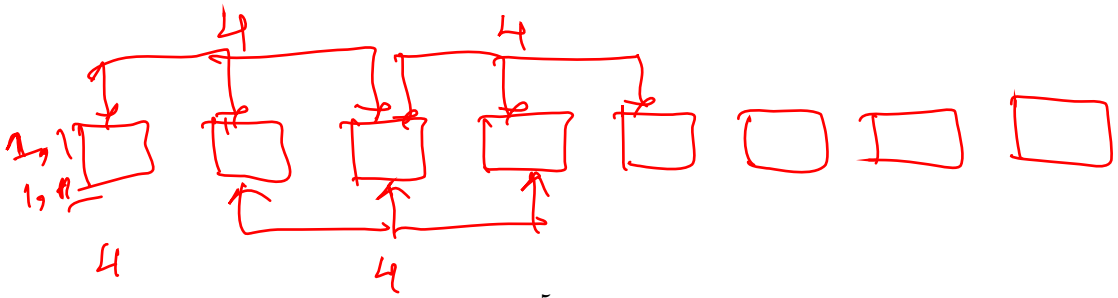
Input Parameters:
Images: (List of Images on which we want to compute forward flow.)
FirstImageFeaturePoints: (This list contains the features points on first image, from where we want to cor
ThresholdZeroFlow: (This value specify the theshold value to filter background points from object points)
Output:
FlowPointsAtLastImage: (Flow points at the First frame.)
FirstImageFeaturePoints:(Given features point, which we want to track at the first image)
Status:(1 for successful and -1 in case of no points found after zero flow filter)

My Algorithm:

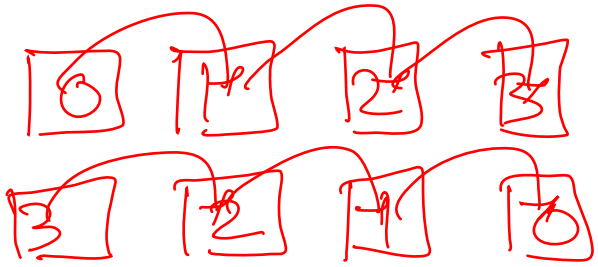
color2gray conversion = O(One image conversion time) x # of images

ideal time

$Status = -1$
$a \leftarrow 0$
loop: till len(Images):
$a < len(Images)$:
$next \leftarrow a++$
$currentImage{-}Gray \leftarrow cv2.cvtColor(Images[a], cv2.COLOR_BGR2GRAY)$
$nextImage{-}Gray \leftarrow cv2.cvtColor(Images[next], cv2.COLOR_BGR2GRAY)$
$ForwardPoints, st, err \leftarrow cv2.calcOpticalFlowPyrLK(currentImage - Gray, nextImage{-}Gray, FirstImageFeaturePoints.astype(np.float32), None, **lk_params)$
if $a \leftarrow (len(Images) - 2)$ **then**
    $ForwardPoints, GivenFirstImagePoints \leftarrow CalculateZeroFlow(ForwardPoints, GivenFirstImagePoints, ThresholdZeroFlow)$
endofloop
if $len(ForwardPoints) == 0$ **then return** $ForwardPoints, FirstImageFeaturePoints, Status \leftarrow$ *-1*
else
    return $ForwardPoints, FirstImageFeaturePoints, Status \leftarrow 1$



interval = 3



Isn't this same as ForwardFlow

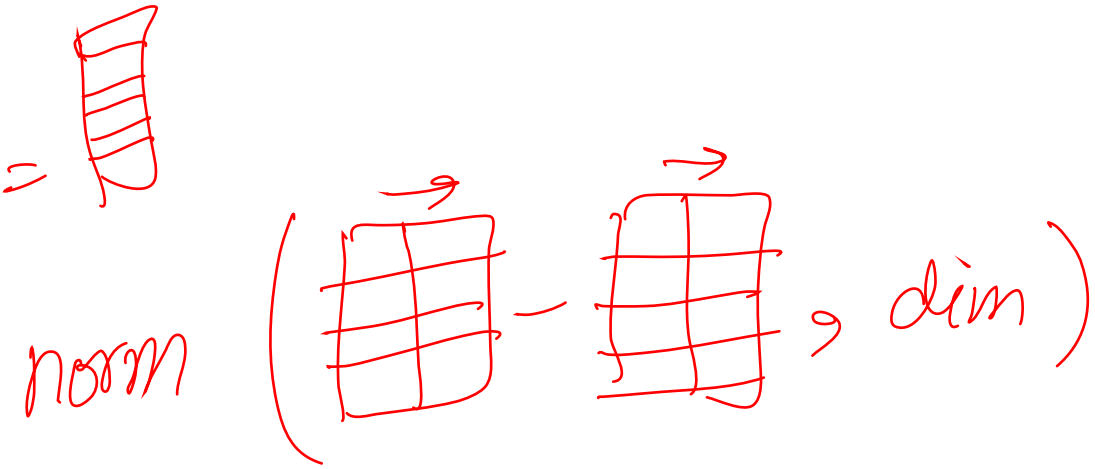
Algorithm 5 GetBackwardFlow

Input Parameters:
Images: (List of Images on which we want to compute Backward flow.)
FirstImageFeaturePoints: (This list contains the features points on last image, from where we want to con
ThresholdZeroFlow: (This value specify the theshold value to filter background points from object points)
Output:
FlowPointsAtLastImage: (Flow points at the last frame. We get from ForwardFlow method)
FirstImageFeaturePoints:(Given features point, which we want to track at the first image)
Status:(1 for successful and -1 in case of no points found after zero flow filter)

My Algorithm:

$Status = -1$
$a \leftarrow len(Images) - 1$
loop: till len(Images):
$a > 0$:
$next \leftarrow a - -$
$currentImage{-}Gray \leftarrow cv2.cvtColor(Images[a], cv2.COLOR_BGR2GRAY)$
$nextImage{-}Gray \leftarrow cv2.cvtColor(Images[next], cv2.COLOR_BGR2GRAY)$
$ForwardPoints, st, err \leftarrow cv2.calcOpticalFlowPyrLK(currentImage - Gray, nextImage{-}Gray, FirstImageFeaturePoints.astype(np.float32), None, ** lk_params)$
if $a \leftarrow 1$ **then**
    $ForwardPoints, FirstImageFeaturePoints \leftarrow CalculateZeroFlow(ForwardPoints, GivenFirstImagePoints, ThresholdZeroFlow)$
endofloop
if $len(ForwardPoints) == 0$ **then return** $ForwardPoints, FirstImageFeaturePoints, Status \leftarrow$ *-1*
else
    return $ForwardPoints, FirstImageFeaturePoints, Status \leftarrow 1$

tracked_points_in_K init_points_in_1



Algorithm 6 CalculateZeroFlow

Input Parameters:
LastImagePoints:(Points at the last frame after optical flow.)
FirstImagePoints:(Given Points at the first image)
Threshold:(Threshold to filter points on base of zero flow)
Output:
FilteredLastImagePoints: (Last Image point after zero flow filter)
FilteredFirstImagePoints: (First Image point after zero flow filter)

My Algorithm:

vectorize

$FilteredLastImagePoints = []$
$FilteredFirstImagePoints = []$
$t \leftarrow 0$
loop: on len(FirstImagePoints)
if $np.linalg.norm(LastImagePoints[a] - FirstImagePoints[a]) > Threshold$ **then**
    $FilteredLastImagePoints.append \leftarrow LastImagePoints[t]$
    $FilteredFirstImagePoints.append \leftarrow FirstImagePoints[t]$
$t++$
endloop: on len(FirstImagePoints)
return $FilteredLastImagePoints, FilteredFirstImagePoints$

Keypoints

Algorithm 7 GetGoodPoints (Apply all filters)

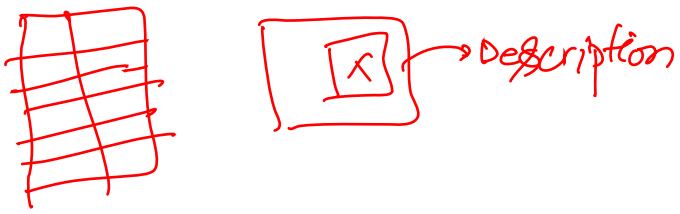
Input Parameters:
FirstImagePoints:(Features Points at start of image)
ForwardPoints:(Given Points tracked to given interval size.)
BackwardPoints:(Given forward Points tracked back to first image.)
Images:(Set of images of given interval size, on which we want to track points.)
GeomatricThreshold:(Threshold to filter points on base of geomatric distance.)
DescriptorThreshold:(Threshold to filter points on base of descriptor SSD or Dot Product)
Method:(Specfiy the method to use to compare descriptor)
Output:
GoodPoints: (Return Good features points which pass criteria of filters.)
Status: (Return -1 if no good feature point found otherwise 1.)

My Algorithm:

$varGoodPoints = []$
$varForwardPoints = []$
$FinalGoodPoints = []$
$Status \leftarrow -1$
loop:till len(BackwardPoints)
$i \leftarrow 0$
$dist \leftarrow np.linalg.norm(BackwardPoints[i] - FirstImagePoints[i])$
if $dist <= GeomatricThreshold$ **then**
    $varGoodPoints.append \leftarrow FirstImagePoints[i]$
    $varForwardPoints.append \leftarrow ForwardPoints[i]$
    $i++$
Endloop:len(BackwardPoints)
$orb \leftarrow cv2.ORB_create()$
$FirstImageGray \leftarrow GrayImageofImages[First]$
$LastImageGray \leftarrow GrayImageofImages[Last]$
$keyPoints_Good \leftarrow KeyPoints(varGoodPoints)$
$keyPoints_Farword \leftarrow KeyPoints(varFarwordPoints)$
$Goodkp, GoodptsDes \leftarrow orb(FirstGrayImage, keyPoints_Good)$
$Farwordkp, FarwordptsDes \leftarrow orb(LastGrayImage, keyPoints_Farword)$
if $Method ==$ "$DotProduct$" **then**
    loop:len(varGoodPoints)
    $t = 0$
    $gdpt_des \leftarrow NormalizedGoodptsDes$
    $forward_des \leftarrow NormalizedFarwordptsDes$
    $product \leftarrow dot(forward_des, gdpt_des)$
    if $product <$ "$DescriptorThreshold$" **then**
        $FinalGoodPoints.append \leftarrow varGoodPoints[t]$
        $t++$
        Endloop:len(varGoodPoints)
    if $len(FinalGoodPoints) == 0$ **then return** $FinalGoodPoints, Status \leftarrow -1$
    else
        return $FinalGoodPoints, Status \leftarrow 1$

Vectorize

this method is not mathematically correct



```
if Method == "SSD" then
    loop:len(varGoodPoints)
    t = 0
    gdpt_des ← NormalizedGoodptsDes
    forward_des ← NormalizedFarwordptsDes
    ssd_distance ← np.linalg.norm(forward_des − gdpt_des)
    if ssd_distance < "DescriptorThreshold" then
        FinalGoodPoints.append ← varGoodPoints[t]
        t + +
        Endloop:len(varGoodPoints)s
    if len(FinalGoodPoints) == 0 then return FinalGoodPoints, Status ← -1
    else
        return FinalGoodPoints, Status ← 1
```

Algorithm 8 TrackGoodPoints

Input Parameters:
Images: (List of Images on which we want to compute forward flow.)
GoodPoints: (This list contains the good features points on first image, from where we want to compute fo
Output:
*TrackedPoints: (List of good points tracked in given images. The size of list is equal to 2*size of images b*

My Algorithm:

$TrackedGoodPoints = []$
$GoodPointsList \leftarrow$Reshape goodpoints in 2 by lenght list. At zero index all x-coordinate and at 1 all y-coordinate of good points.
$TrackedGoodPoints.append \leftarrow (GoodPointsList[0])$
~~$TrackedGoodPoints.append \leftarrow (GoodPoints[1])$~~
$a \leftarrow 0$
loop: till len(Images):
$a < len(Images)$:
$next \leftarrow a + +$
$currentImage{-}Gray \leftarrow cv2.cvtColor(Images[a], cv2.COLOR_BGR2GRAY)$
$nextImage{-}Gray \leftarrow cv2.cvtColor(Images[next], cv2.COLOR_BGR2GRAY)$
$ForwardPoints, st, err \leftarrow cv2.calcOpticalFlowPyrLK(currentImage - Gray, nextImage{-}Gray, FirstImageFeaturePoints.astype(np.float32), None, ** lk_params)$
$GoodPointsList \leftarrow ForwardPoints$
$TrackedGoodPoints.append \leftarrow (GoodPointsList[0])$
$TrackedGoodPoints.append \leftarrow (GoodPoints[1])$
endofloop
return $TrackedGoodPoints$

Algorithm 9 CalculateFlow

1: ***Input Parameters:***
2: *CurrentImage:(Current Image from which we want to calculate flow)*
3: *NextImage: (Very next image to current image)*
4: *GoodPoints: (Features points on current image to computer flow.)*
5: ***Output:***
6: *Flow: (This list contain difference of flow of points from current image to next image)*
7: *ForwardPoints: (This list contain flow of points from current image to next image)*

8: ***My Algorithm:***
9: $currentImage{-}Gray \leftarrow cv2.cvtColor(CurrentImage, cv2.COLOR_BGR2GRAY)$
10: $nextImage{-}Gray \leftarrow cv2.cvtColor(NextImage, cv2.COLOR_BGR2GRAY)$
11: $ForwardPoints, st, err \leftarrow cv2.calcOpticalFlowPyrLK(currentImage - Gray, nextImage{-}Gray, FirstImageFeaturePoints.astype(np.float32), None, ** lk_params)$
12: $a \leftarrow 0$
13: *loop: till len(ForwardPoints)*
14: $x \leftarrow FlowPoints[a][0] - GoodPoints[a][0]$
15: $y \leftarrow FlowPoints[a][1] - GoodPoints[a][1]$
16: $Flow.append \leftarrow (x, y)$
17: *endofloop:*
18: ***return*** $Flow, ForwardPoints$

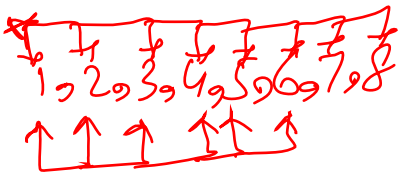
Median

(whether to get flow of best hypothesis or median flow of all inliers)

Algorithm 10 GetRansac

Input Parameters:
Flow:(Flow of good points, we get as output of calculate flow method)
Threshold:(Threshold to filter inlairs from outlairs)
Output:
FlowPoint: (Return a single point which has the most number of inlairs.)

My Algorithm:




```
Inlairs = []
NormalizedFlowPoints ← Normalizedto1
loop: till len(NormalizedFlowPoints)
tempInlairs = []
i ← NormalizedFlowPoints
product ← np.dot(NormalizedFlowPoints, i)
loop:till len(product)
j ← product
if j >= threshold then
    tempInlairs.append ← j
endloop:Product
if len(tempInlairs) >= len(Inlairs) then
    Inlairs ← tempInlairs[]
    FlowPoint ← Flow[i]
endofloop:NormalizedFlowPoints
return FlowPoint
```

$\frac{\#\text{ofpoints}}{\#\text{ofhypothesis}} \times \#\text{of points}$

$8 \times 8 \sim 64$

$1000 \times 1000 \sim 10^6$

100×1000

① Random Sample
② Evaluate hypothesis' distance
  2a compute
  2b compare
③ Check if this is the best hypothesis