# Lab 5
# Image Sequence Processing

Jonathan Ananda Nusantara
jan265

In this lab, the techniques of manipulating multi-frame image files will be explored. The first experiment will be about a tracking program that allows user to follow a set of image features through different time slices of the image (temporal sequence of image frames). In the second experiment, we will explore two types of temporal image filtering: temporal mean filtering and temporal median filtering. In the last experiment, we will explore a change detection program, which is utilizing threshold between a temporal sequence of image frames, which will help detect changes between image slices.

We will mainly be focusing in these 3 images. The first image file will be the hand.vs image. This image includes 15 time slices, showing two people shaking their hands. The image is as shown below:

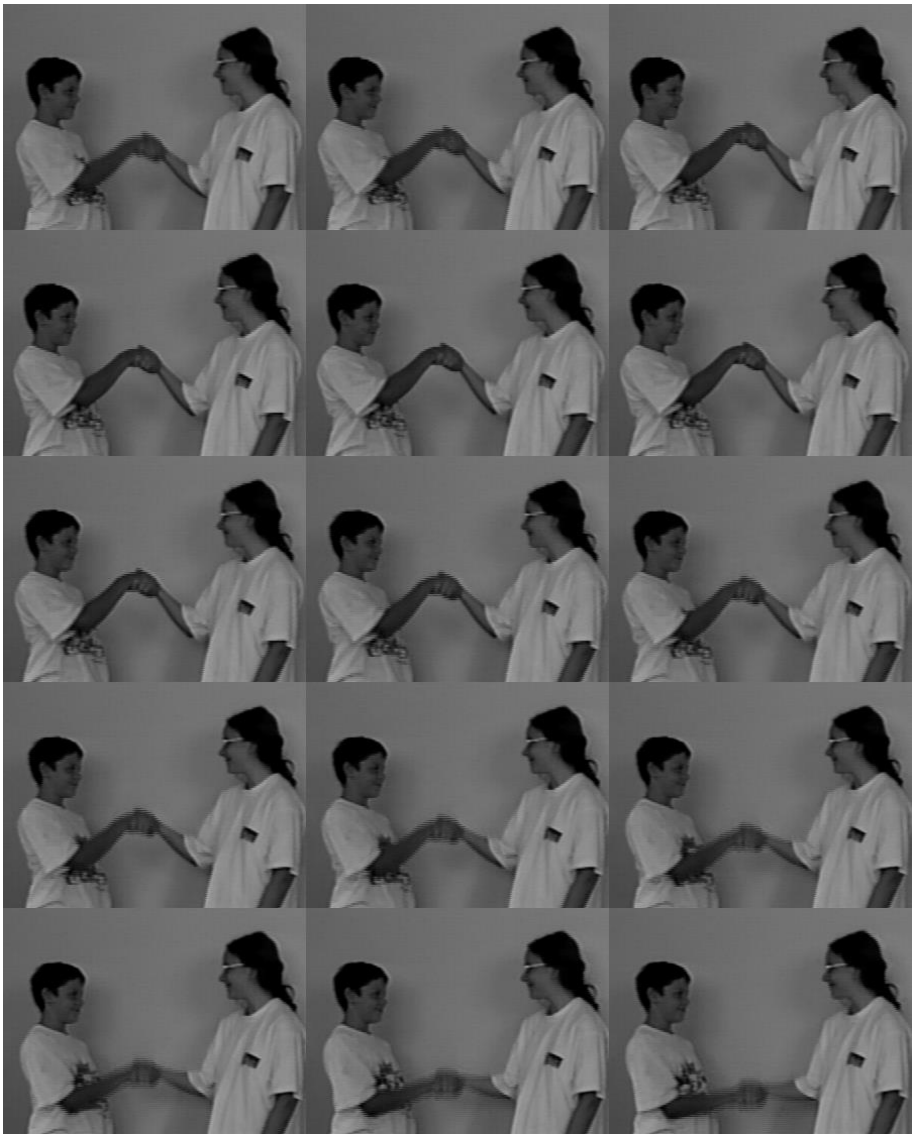

Figure 1 – 15 frames of hand.vs image.

The second image file is the taxi.vs. It is an image file with 21 image slices, showing an intersection with 3 vehicles moving as we go through the time slices:

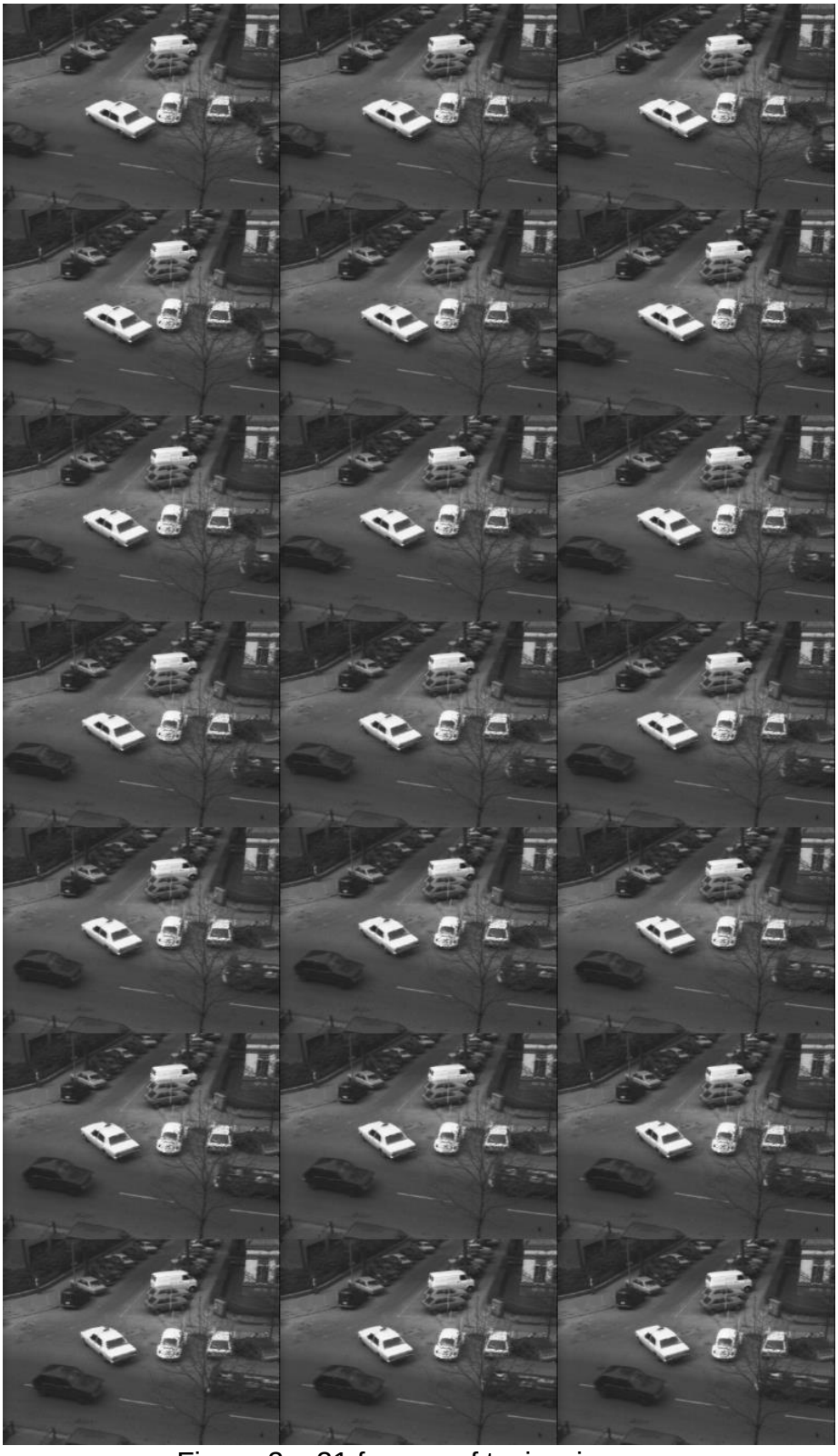

Figure 2 – 21 frames of taxi.vs image.

The last image file that will be discussed is the lb5.vs image file. It is an image file with 7 image slices showing mainly of a woman, but some image slices are shifted, added salt and pepper noise, or replaced with an image of a bear:

Figure 3 – 7 frames of lb5.vs image.

## Feature Tracking:

The first section is to explore the tracking program, which allows us to track an image feature through different image slices in the sequence. This can be done using the VisionX command “vtrack”, which can track a set of patches through a sequence of image. The first image to be tracked is the hand.vs image file. The command below is executed:

vtrack if=hand.vs pf=loc1 h=10 v=10 of=trk.vs gf=trk.g

The pf input to the command is a text file that has the x and y coordinates of the image feature/patch that needs to be tracked, as well as the number of patches to be tracked. The h and v input is the horizontal and vertical pixel size of search size, where the image patch will then be searched within this boundary from the specified coordinates. The command provides two outputs: of which is annotated image with the labeled tracked image patches and gf which is the vector output file that contains the trajectory of the image patches or features. The first image slice of the annotated hand.vs output is as below:

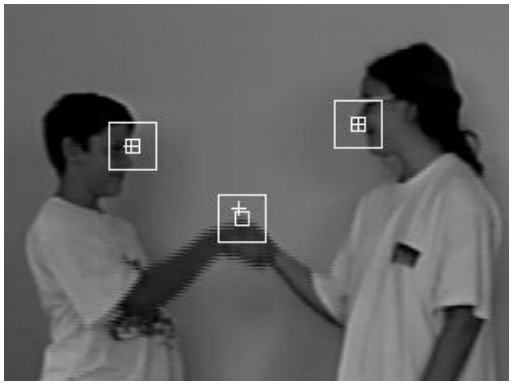

Figure 4 – First image slice of trk.vs, which is the hand.vs with tracked features.

From the Figure 4, we can see a total of 3 features tracked: the top of the hands of the two people, the tip of the nose of the person on the right, and the tip of the face (eye-level) of the person on the left. The location of the feature is marked with the + sign. The location of the feature in the previous image slice is marked with the smaller square. The search size defined by the user is marked by the larger square. The image slices that are produced from the vtrack command with hand.vs, which is named trk.vs, will now be displayed as a sequence in a single image. The vtile command is used to combine the image slices in sequence in a single frame. The following command is executed:

vtile trk.vs n=3,4 of=trkt.vs

The n input defines the number of columns and rows, where in this example we will have 3 columns and 4 rows. The output tile of image sequence, trkt.vs, is displayed below:

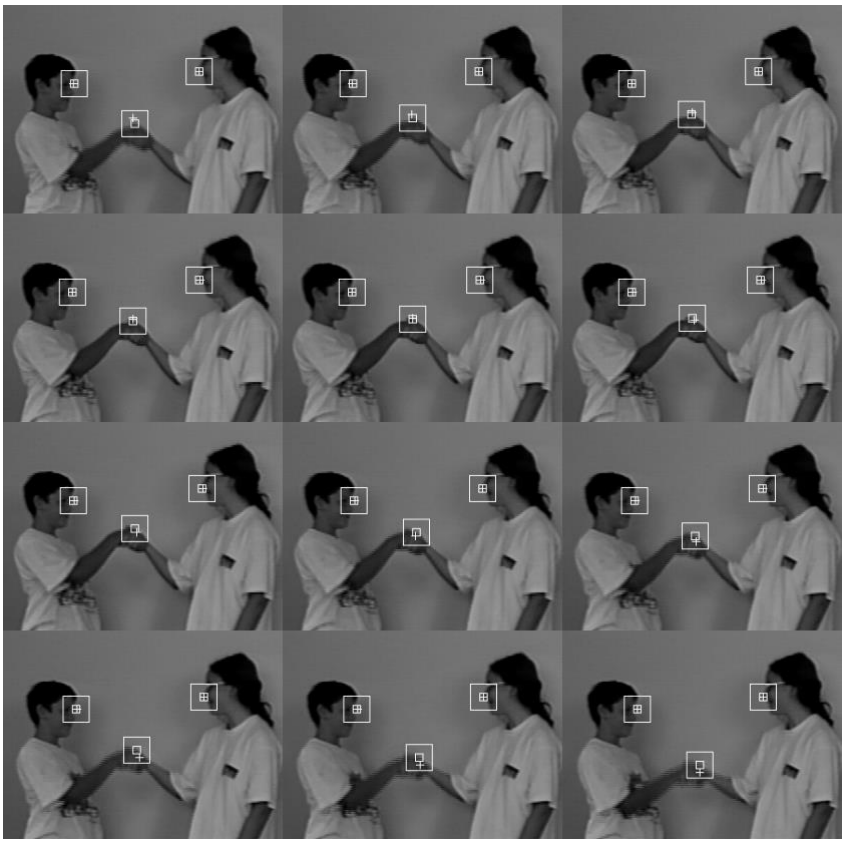

Figure 5 – trkt.vs, the sequence of image slices of trk.vs.

From Figure 5, we can see how out of the three tracked features, only the hand is moving significantly. All of the features were able to be tracked well. The trajectory file trk.g was also explored. It can be viewed by either of these two ways below:

vpr trk.g

Tools ->Vx File Content

This will display a text file showing a table with x, y, and z axis, where z is the axis for image or time slices. The trk.g is displayed as below:

```
[Bounding Box   ]: [6]
        0       299         0       223         0       14
[Object ID.     ]: 2
[3-D vector     ]: [45]
       141       96         0       139       102        1
       138      106         2       139       108        3
       139      110         4       139       111        5
       141      109         6       143       105        7
       142      101         8       143        95        9
       146       87        10       147        79       11
       147       71        12       148        61       13
       148       51        14
[Object ID.     ]: 1
[3-D vector     ]: [45]
       210      152         0       210       152        1
       210      152         2       210       152        3
       211      152         4       212       152        5
       213      152         6       214       152        7
       214      152         8       215       152        9
       215      152        10       215       152       11
       215      152        12       214       152       13
       212      152        14
[Object ID.     ]: 0
[3-D vector     ]: [45]
        76      139         0        75       139        1
        74      139         2        74       139        3
        74      139         4        74       139        5
        75      139         6        76       139        7
        77      139         8        78       139        9
        79      139        10        80       139       11
        80      139        12        80       139       13
        80      140        14
```

The object ID shows the different features that were tracked. The table has a weird format where they are divided into two different columns of x, y, and z. So, the table can be seen to be displayed in this sequence: x, y, z. In object 1 and 2, we can see that x and y barely changes, which reflects the image slices where the head of the person barely moved. On the other hand, object 0 is the hands of the two people, and we can see that in the image slices the hands moved significantly. Last but not least, the trajectory data can be displayed as a plot using the command below:

v3d trk.g cf=v3.in

A screenshot of the plot is shown below:

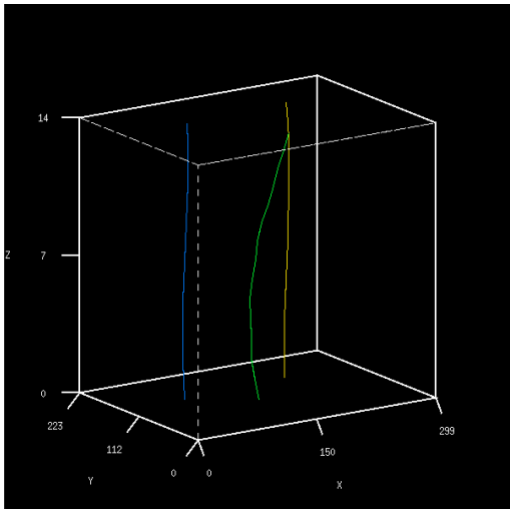

Figure 6 – Trajectory plot of the three features in trk.vs.

From the trajectory plot above, we can see how the object marked by the color blue and yellow barely moved while the green line movement can be clearly seen.

The tracking program is then again tested on the taxi.vs image shown in Figure 2. A total of 9 features were tracked and their original coordinates in the first image slice is saved in a text file named loc2. The command below is executed:

vtrack if=taxi.vs pf=loc2 h=4 v=4 sz=4 of=taxi-trk.vs gf=taxi-trk.g

The features will be tracked with the search size of 4x4 and square correlation patch matrix of 4x4. The first slice of the output image is as below:


Figure 7 – First image slice of taxi-trk.vs

In the image slice above, we can see the 9 locations of the tracked features. These features are chosen to see how well the program can track both moving and non-moving features. On the black car on the bottom left, the right headlight and windshield are tracked. On the white taxi in the middle, the rear left side-window, rear left wheel, and taxi sign on the roof are tracked. On the car on the bottom right, the bumper is tracked. The three vehicles above were moving through the image slices. Three tracked features are non-moving and they are the feature that are located above the location of the taxi in the image. The program was able to track all of the features well throughout the different image slices. In order to best visualize the result, the vtile command is again used and the first 15 image slices of the sequence is displayed below:

Figure 8 – The sequence of image slices of taxi-trk.vs.

From Figure 8, we can see how all of the features were able to be tracked well. This tracking performance was able to be achieved after tweaking the parameters to control the size of the square correlation patch and the search size. The trajectory of each of the feature can also be analyzed by looking at the plot displayed by the v3d command:

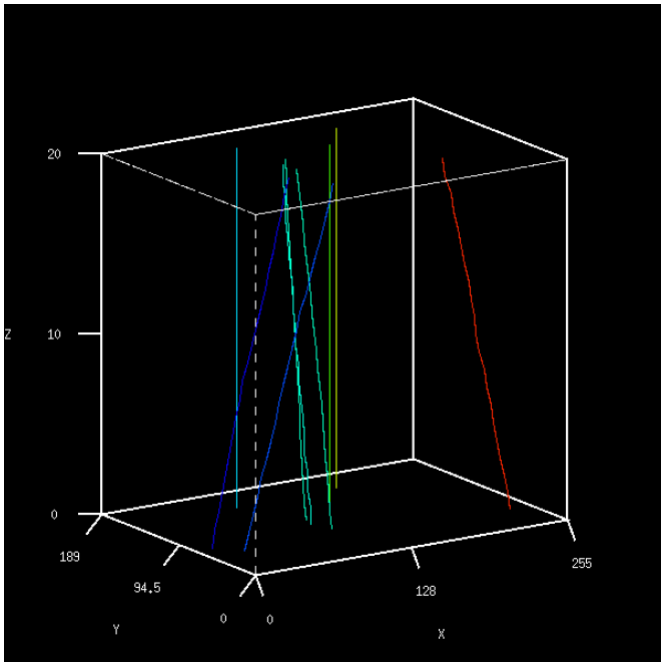

Figure 9 – Trajectory plot of the nine features tracked in taxi-trk.vs.

The plot in Figure 9 shows the different movements of the features in the x and y coordinates through the sequence of image slices in axis z.

From the results shown in this section using two different images: hand.vs and taxi.vs, we have learned how a feature tracking program like the vtrack command can effectively be used to tracked the different features chosen by the user on a sequence of image slices. There needs to be tweaking of some parameters, such as the search size in the image, but when the right parameters are passed to the commands a great feature tracking performance can be achieved. This type of program would be great to be used to track a feature through image frames, such as a video.

## Temporal Domain Filtering:

The next section will discuss how a temporal mean and median filter may be used on an image sequence. In order to further understand how the filter works, the vssump program is used. This program will perform the temporal mean operation on the image sequence. The program is first used on the hand.vs image sequence. The command below is first executed:

vssump hand.vs n=3 of=hand.m

The vssump will accept an input of an image sequence. The user will also input the value of how many frames should the mean operation be performed on, which is denoted by the variable n. For example, we are using n=3 in the example above. This means that each of three image frames will be analyzed at once and the mean of each pixel will be calculated across the sequence of 3 image frames. The mean value of each pixel will be used for the output of 1 image frame. So the program will first find the mean in image slice 1, 2, and 3, and output an image slice with the mean pixel values of the three image slices. Next, it performs the same operation on the set of image slice 2, 3, and 4, then the set of 3, 4, and 5, and so on until the end of the input image slice. The output image of vssump will have less number of image slices. The resulting image with n=3 after combining the image slices with vtile is as below:

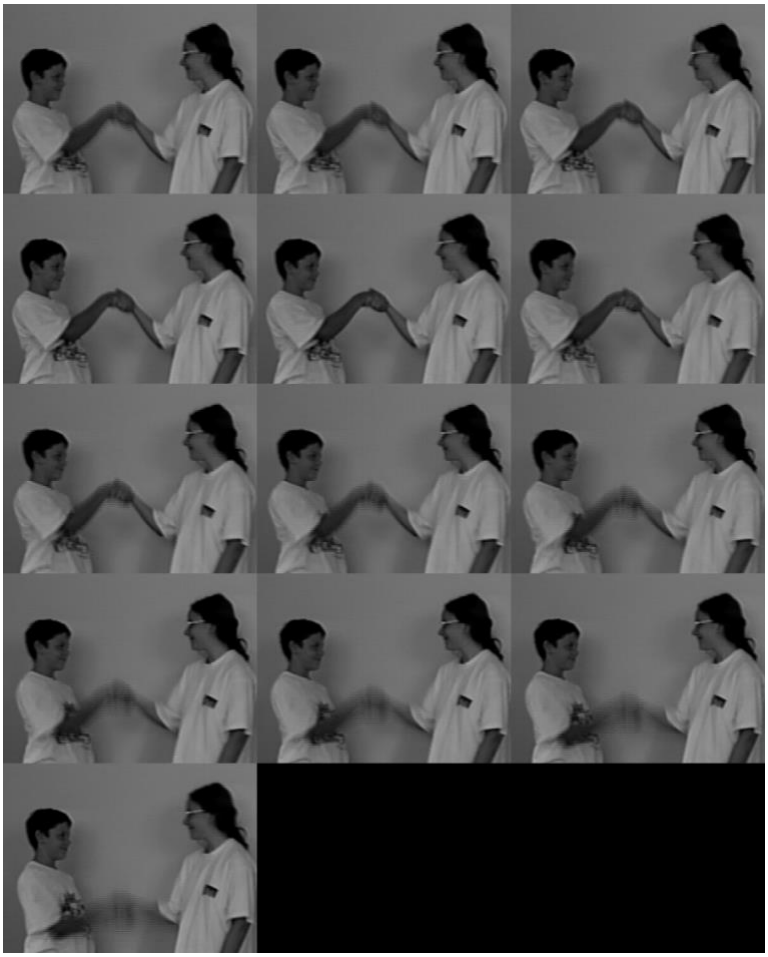

Figure 10 – image slices of temporal mean filtered image of hand.vs with window of 3 frames.

From the image above, due to the mean filtering that was done on the image, we can see that the moving features (high frequency) are more blurred out due to the mean filter that was performed. This can be clearly seen when the image above is compared to the original image in Figure 1. This shows that temporal mean filtering can be used to blur out or attenuate features that are not of interest, where in the case of using a mean operation means features of high frequencies.

Next, the temporal mean operation is also performed on a larger window. The program is experimented with n=5, where the mean across 5 image frames will be calculated. The following command was executed:

vssump hand.vs n=5 of=hand.m5

The output of the command above after combining the image slices with vtile is as below:

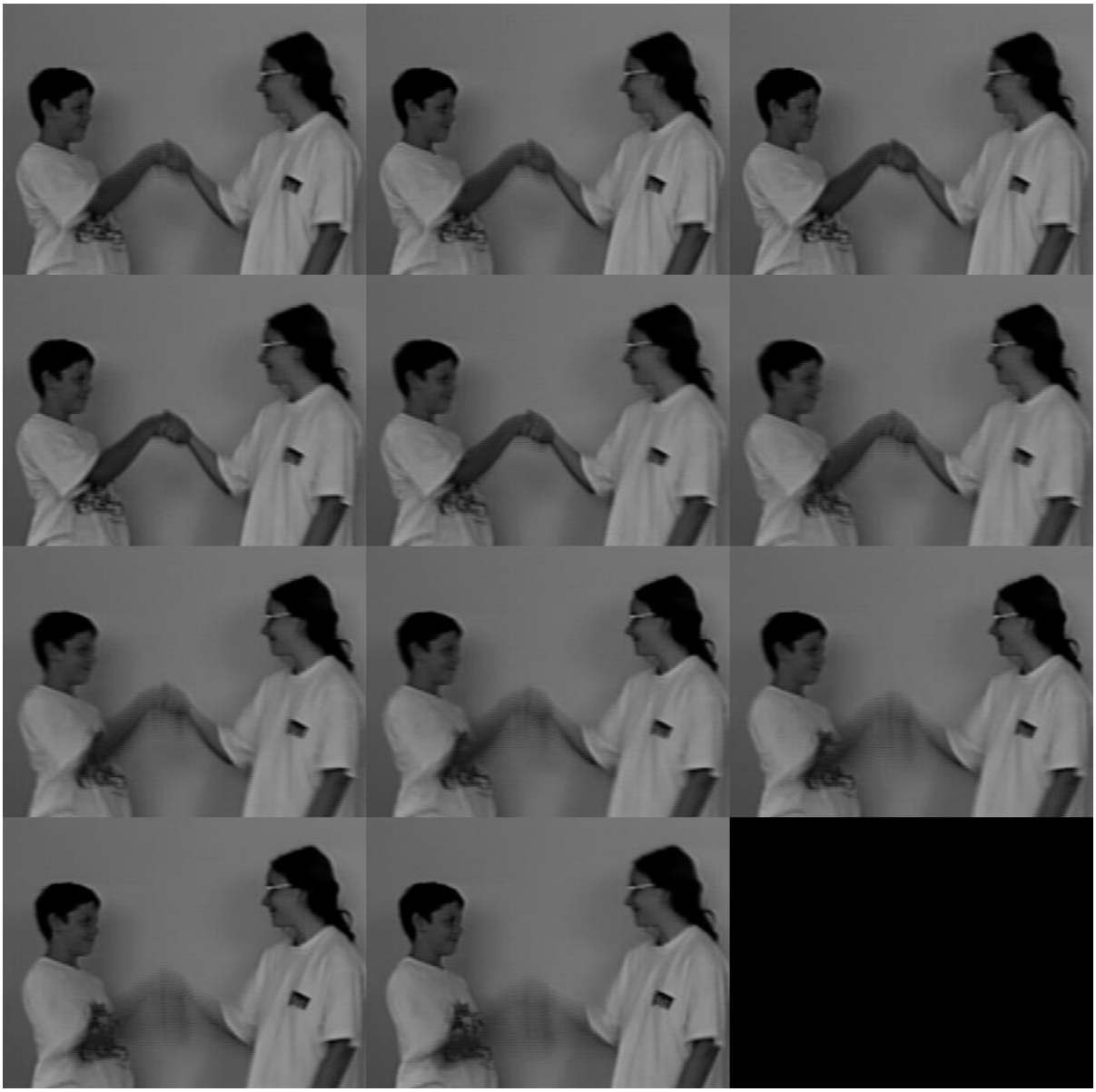

Figure 11 – image slices of temporal mean filtered image of hand.vs with window of 5 frames.

From the image above, we can see how the high frequencies features in the image slices are even more blurred out when the mean filter window is 5, compared to when using the mean filter window of 3. This means that a larger window will remove more noise, where in this case is the high frequency. The main difference to using a mean window of 3 is that image slices are now analyzed in sets of 5 and the mean is calculated from there. The wider averaging window helps in attenuating more noise.

One thing to note is that there is a bug in the original vssump, where the first image slice of the output would be flawed and is identical to the second slice of the image. As a result, the first image slice should be ignored. Since other programs created in this lab is based on vssump, they will all have this same flaw.

Next, a median filtering program is created based on vssump. It will perform median operation on 3 image slices sequence. The full code vsmed is displayed below:

```
#!/usr/bin/env python
""" vsmed Compute local 1x1x1 median using the buffer method """
import sys
from numpy  import *
from v4 import vx
from vxbuffer import *

of=' '
vxif=' '
clist = vx.vxparse(sys.argv,  "if= of= -v  - ")
exec (clist )

if 'OPT' in locals():
   print ("vsmed 1x1xn local temporal median filter")
   print ("if= input file")
   print ("of= output file")
   print ("[-v] verbose mode for very small images")
   exit(0)

optv = 'OPTv' in locals()

invx = vxIbuf(vxif, 3);
outvx = vxObuf( of );
im = invx.i
imr = empty( im[0].shape, dtype=im.dtype);
while invx.read():
    im = invx.i

    for t in range(im.shape[0] ):
        print(t)
        for y in range(im.shape[1]):
            for x in range(im.shape[2]):
                med = 0
                list_med = []
                for t in range (3):
                    list_med.append(im[t][y][x]) # append 3 data to list
                if max(list_med) == list_med[0]: # If index 0 is biggest
                    if list_med[1] > list_med[2]: # index 1 is second highest
                        imr[y][x] = list_med[1]
                    else: # index 2 is second highest or (1 and 2) is equal
                        imr[y][x] = list_med[2]
                elif max(list_med) == list_med[1]: # If index 1 is biggest
                    if list_med[0] > list_med[2]: # index 0 is second highest
                        imr[y][x] = list_med[0]
                    else: # index 2 is second highest or (0 and 2) is equal
                        imr[y][x] = list_med[2]
                else: # If index 2 is biggest
                    if list_med[0] > list_med[1]: # index 0 is second highest
                        imr[y][x] = list_med[0]
                    else: # index 1 is second highest or (0 and 1) is equal
                        imr[y][x] = list_med[1]


```



Figure 12 – The code of vsmed for temporal median filtering.

The main change in the vsmed code compared to vssump is in the main nested for loop that iterates through each pixel. Instead of calculating the average of each pixel across the frames, it will now choose the median pixel value across the frame. Another change is that the program will not accept the “n” input as the user will have no option of choosing the number of frames where median will be calculated from. That value will have a default of 3 frames. This program is then tested on the hand.vs image slices. The following command was executed:

vsmed hand.vs of=hand-median.m

The output of the temporal median filtering on the hand.vs after combining the image slices with vtile is as below:

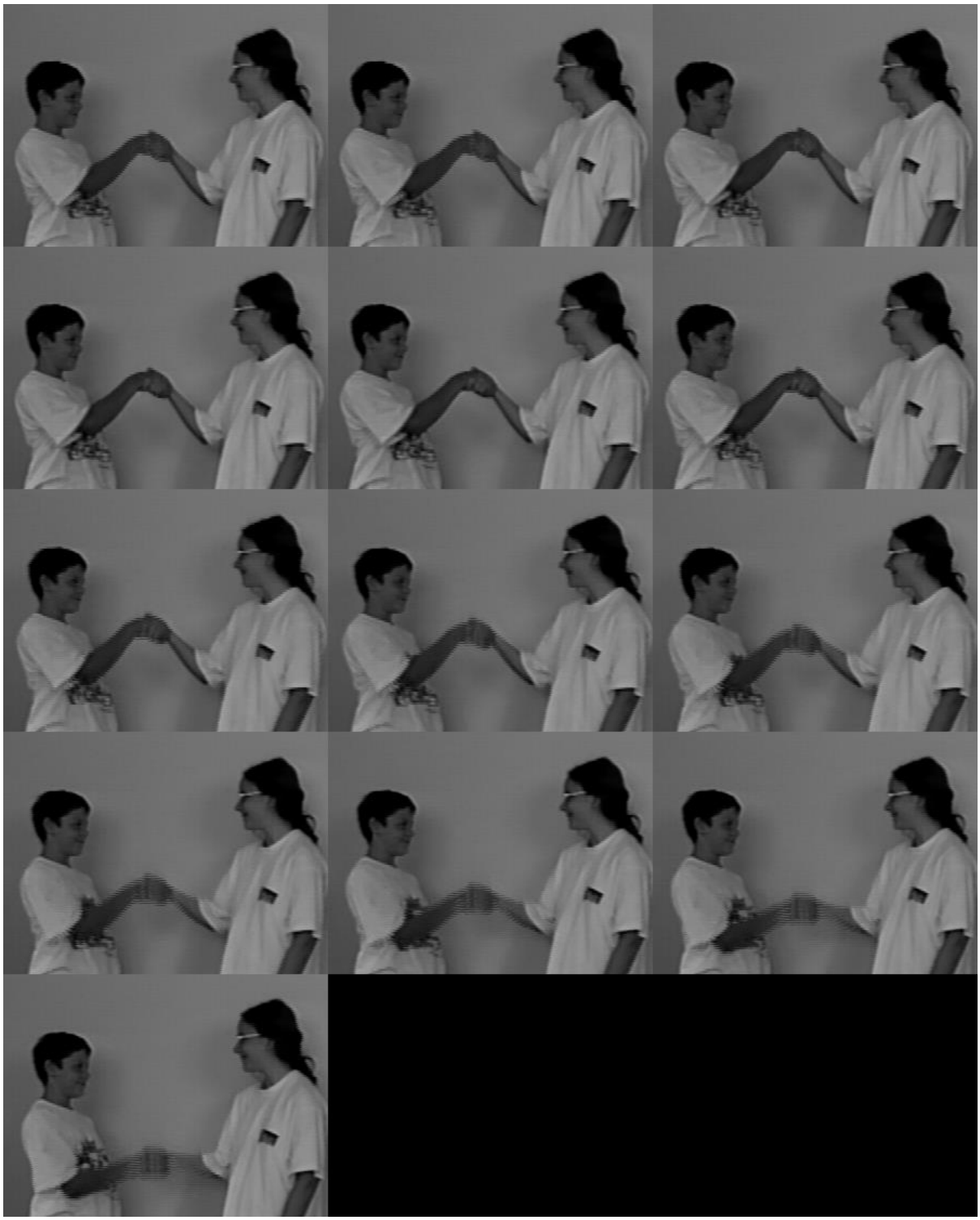

Figure 13 – image slices of temporal median filtered image of hand.vs with window of 3 frames.

From the image above, we can see how the output image is different to the results of the temporal mean filtering. Although the moving hands were still blurred out, the features of the hands are more visible. This is mainly because the edges of the features or hands are more

preserved, and so you can see the moving hands more clearly. The reason is that in median filtering, the value of the pixel came from one of the 3 pixel values from each frame, instead of coming up with a new value (which is what mean filtering is doing by calculating the mean), and so the pixel value is more realistic.

The same process of temporal mean filtering with windows of 3 and 5, as well as median filtering, were performed on the lb5.vs image shown in Figure 3. The mean filtering with window of 3 is first performed and the result after combining the image slices with vtile is as below:

Figure 14 – image slices of temporal mean filtered image of lb5.vs with window of 3 frames.

The result has a total of 5 image slices. The first image should be ignored as it is flawed. In the second image, since the windowing frame contains the salt and pepper and bear image, the output image slice has the two components, where we can see a blurred salt and pepper and a bear in the image. In the third and fourth image slice, the salt and pepper no longer exist, but the woman is now blurred due to the shifted image being included inside the mean filtering window of 3. In the last image slice, the bear no longer exists, but the woman is even more blurred. All of these are expected based on the temporal mean filter that was used.

Next, the temporal mean filtering with window of 5 is performed and the result is after combining the image slices with vtile as below:

Figure 15 – image slices of temporal mean filtered image of lb5.vs with window of 5 frames.

The result has a total of 3 image slices. The first image should be ignored as it is flawed. In the second image, it should be the output of the frames that includes the salt and pepper noise, the bear, and the shifted image. So, the mean pixel values of them should include all of these components. This is confirmed by the second slice of the image we could see the features salt and pepper and the bear, and that the woman in the image is a bit blurred due to the shifted image. In the third slice, we can see how the woman is very blurred due to the shifted image and that it does not have salt and pepper, as the salt and pepper image slice is not a part of the five image slices analyzed for this image slice output.

Finally, the median filtering is performed on lb5.vs and the result after combining the image slices with vtile is as below:

Figure 16 – image slices of temporal median filtered image of lb5.vs with window of 3 frames.

The result has a total of 5 image slices. The first image should be ignored as it is flawed. One of the major differences compared to the mean image is that in the mean filtered image, we can see the salt and pepper in the first two image slices across the whole image. In this image, since the pixel value is based on an existing pixel value, the salt and pepper pixel values are only chosen as a median pixel value in some pixels. As a result, the salt and pepper only appears in some sections of the image. Additionally, the bear could barely be seen in the median filtered image, which is for the similar reasons where the bear's pixel value was not chosen as the median pixel value in most pixels. Lastly, since median operation is used, the woman in the last 2 image slices are less blurred as the pixel values used are of an existing pixel value from one of the slices.

From the experiment using the lb5.vs, where the image slices has a lot of different components like salt and pepper, bear, and shifted image, it provides a better visualization and understanding on how the temporal mean and median filtering works, as well as how the size of the window affects the output. Looking at the results from mean filter of window=3 against the median filter with window=3, the result from median filter seemed to be better visually. This is because the resulting image have more details and are less blur, because the output pixels are based on existing pixels of one of the image slices.

## Change Detection:

The final section in this lab is about change detection. A program called vsdif will be written, which will be based from vssump. This program will be a temporal binary difference filter program, where the difference of pixel values of each pixel will be calculated between each pixel for each of two consecutive frames in the entire image slices and a threshold will be used to determine whether the pixel of the output image slice will be of a value of 255 if the difference is above the threshold, or 128 if the difference is below the threshold. This process will be performed throughout the entire image slices sequence of the input image. The user will be able to pass their own threshold value by passing to the input “th”.

The program is first written and is as shown below: The program is first written and is as shown below:



```
 1  #!/usr/bin/env python
 2  """ vsdif Compute local 1x1x1 thresholded difference using the buffer method """
 3  import sys
 4  from numpy  import *
 5  from v4 import vx
 6  from vxbuffer import *
 7
 8  of=' '
 9  vxif=' '
10  clist = vx.vxparse(sys.argv,  "if= of= th= -v  - ")
11  exec (clist )
12
13  if 'OPT' in locals():
14     print ("vsdif 1x1xn local thresholded difference filter")
15     print ("if= input file")
16     print ("of= output file")
17     print ("th= threshold value")
18     print ("[-v] verbose mode for very small images")
19     exit(0)
20
21  optv = 'OPTv' in locals()
22  threshold = int(th)
23
24  invx = vxIbuf(vxif, 2);
25  outvx = vxObuf( of );
26  im = invx.i
27  imr = empty( im[0].shape, dtype=im.dtype);
28  while invx.read():
29      im = invx.i
30
31      for y in range(im.shape[1] ):
32          for x in range(im.shape[2]):
33              new_val = 0
34              if abs(int(im[0][y][x]) - int(im[1][y][x])) > threshold:
35                  new_val = 255
36              else:
37                  new_val = 128
38              imr[y][x] = new_val
39
40      if optv:
41         print (imr)
42      outvx.add(imr)
43
44  outvx.close()
45
```

Figure 17 – The python program vsdif.

The program will call for a threshold value input from the user and will no longer require a “n” number of frames input. One of the main changes in this program compared to the vssump is on line 24, where the input parameter of the number of frames is now set to 2 instead of taking an input from the user. This is because we will always be comparing a sequence of 2 frames and detecting the pixel value changes. Another major change is in the main for loop, where it is now not iterating through the t axis because it would only be performed once. Also, it will now be calculating the absolute difference between the same pixel in two consecutive image slices, and based on the value compared to the threshold, the output pixel will either be 255 or 128.

The next step is to test this program with the image sequence taxi.vs. In this image sequence, there are a total of three moving features, which are the three vehicles in the image. With this program, it would help us to detect and visualize the three moving features. This is because as the features move, there will be a change in the values of the pixels affected by the features’ movement. These changes in values will be detected by the program and will be clearly represented based on whether the changes are below or above the chosen threshold.

FIrstly, the program is run using the threshold of 20. The following command was executed:

vsdif if=taxi.vs th=20 of=taxi-dif-20.vx

The resulting image slices are then combined into a single tiled image using vtile command. The tiled output is as below:

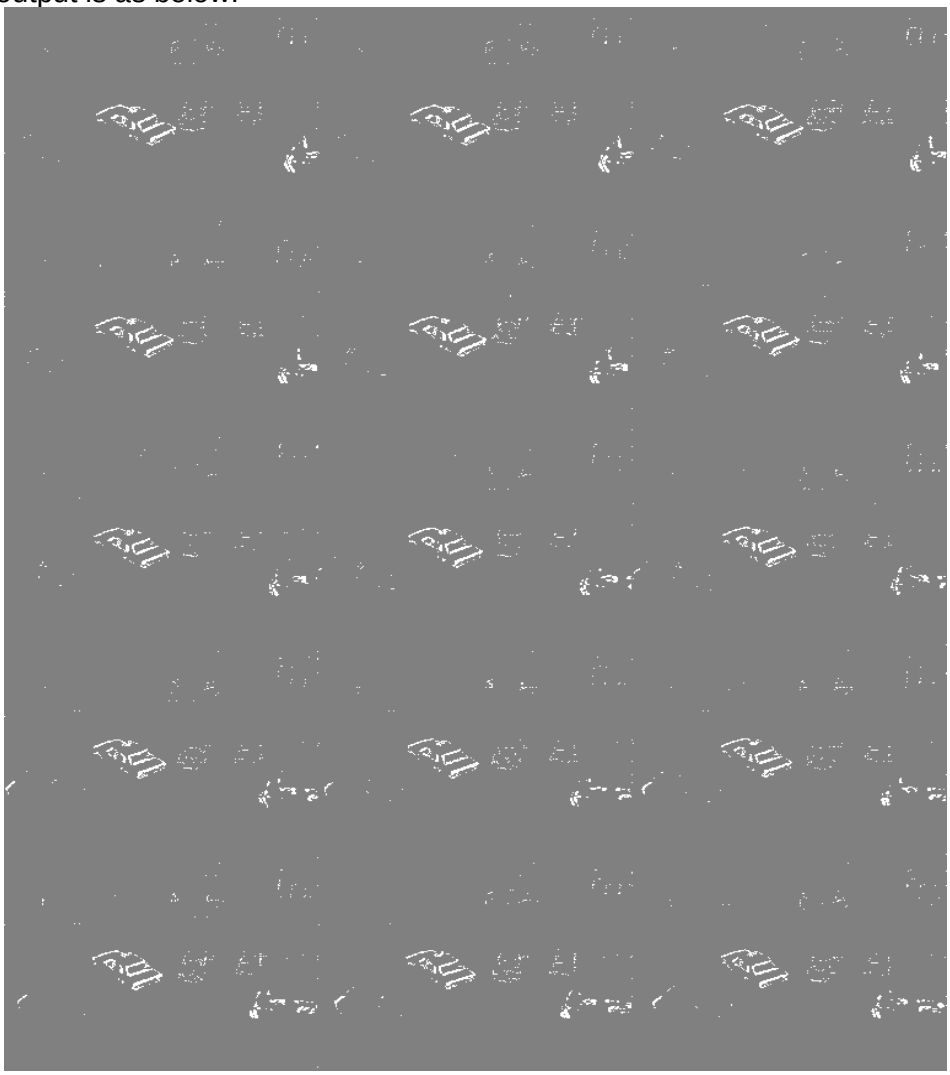

Figure 18 – Tiled image of vsdif output of taxi.vs with th=20.

In the image tile above, we can clearly see the moving taxi in the middle of the image. However, we can barely see the movement of the vehicle on the bottom left and bottom right. This is because these two vehicles have a pixel value (or intensity) similar to the road. So, the changes of pixel values as their location changes through the images slices is not significant enough to pass the high threshold of 20. This is different to the taxi, which is very contrast to the intensity of the road, and so the changes in the taxi's location was detected well by the program.

Next, the program is then tested with a much lower threshold of 5 to see if the two vehicles will be detected well this time. The tiled image of the output is found to be as below:

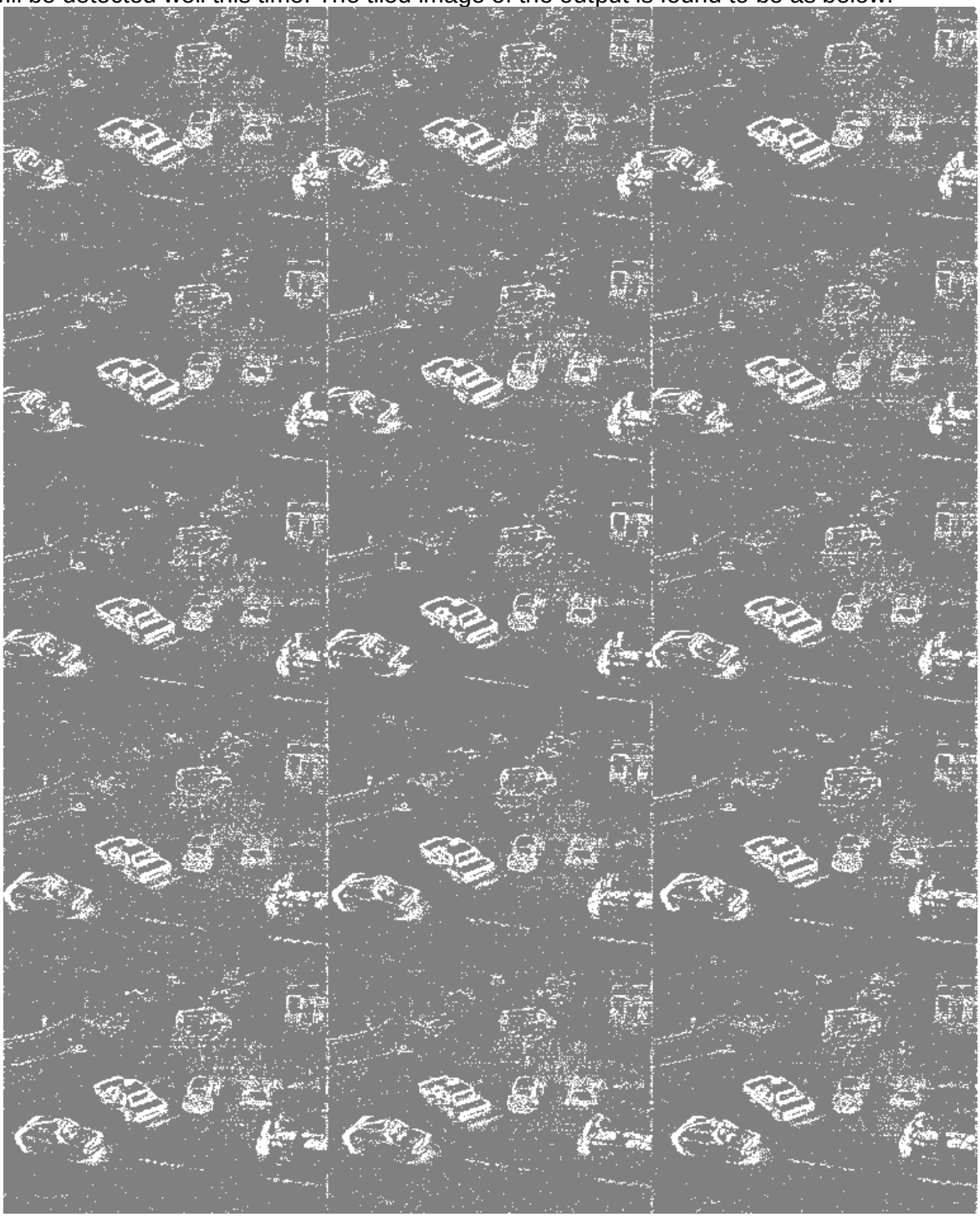

Figure 19 – Tiled image of vsdif output of taxi.vs with th=5.

In this image, we can now clearly see the bottom-left and bottom-right vehicles moving. However, there are now too much noise (white dots) from pixels that does not actually mean anything. Most of them are just still features that are not moving. This means that the threshold is too low.

Now that we have learned that the threshold should be somewhere between 5 and 20, more experimentations were performed. In the end, threshold 12 was found to be the best threshold that would provide the best visualization. The tiled image output is as below:

Figure 20 – Tiled image of vsdif output of taxi.vs with th=12.

Using this threshold value, we can see how the movement of the two vehicles on the bottom can be identified well. At the same time, there is not much noise throughout the image, which makes it look nice and clear. Through the experimentation, threshold that is a bit lower like th=10 also works well without giving too much noise.

From this section, we have learned how a change detection program like vsdif can be used to identify moving features through an image sequence. This is done by detecting the changes of pixel values caused by the movement of those features through image slices. The effect of the threshold is on how sensitive the program is to the changes in pixel values. If the threshold is low, then the program will be sensitive and small change in pixel values will be categorized/output as a high (255) pixel value in the output. On the other hand, if the threshold is high, then the program is less sensitive to changes in pixel values and the high values in the output will be those pixels with huge changes in pixel value. Using this program, one would need to tweak the threshold parameters to get the best performance out of the program, where the features can be identified without getting much noise.