Video QR code analysis

The analysis is done using one Multiplexed QR code. This process will be scaled for the number of QR code present in a video. Each image will be processed for noise and distortions.

```
!pip install svglib
!pip install svgwrite



from cgi import test
from ctypes import sizeof
from numpy import char
import svgwrite
from svgwrite import cm, mm
import svglib
import os
from svglib.svglib import svg2rlg
from reportlab.graphics import renderPM
import cv2
import zlib
from google.colab.patches import cv2_imshow
import time
from timeit import default_timer as timer
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: svglib in /usr/local/lib/python3.8/dist-packages (1.4.1)
Requirement already satisfied: lxml in /usr/local/lib/python3.8/dist-packages (from svglib) (4.9.2)
Requirement already satisfied: reportlab in /usr/local/lib/python3.8/dist-packages (from svglib) (3.6.12)
Requirement already satisfied: cssselect2>=0.2.0 in /usr/local/lib/python3.8/dist-packages (from svglib) (0.7.0)
Requirement already satisfied: tinycss2>=0.6.0 in /usr/local/lib/python3.8/dist-packages (from svglib) (1.2.1)
Requirement already satisfied: webencodings in /usr/local/lib/python3.8/dist-packages (from cssselect2>=0.2.0->svglib) (0.5.1)
Requirement already satisfied: pillow>=9.0.0 in /usr/local/lib/python3.8/dist-packages (from reportlab->svglib) (9.3.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: svgwrite in /usr/local/lib/python3.8/dist-packages (1.4.3)
```


Data being used for Testing purposes

```
file = open("data.txt","w")

file.write("When Mr. Bilbo Baggins of Bag End announced that he would shortly be celebrating his eleventy-first birthday with a party of
```

```
file.close()
```

Generating QR

Designing Algo for displaying QR code in SVG

> The reason we are desiging our own QR code algorithim is for numerous reasons:

1. Different Compression algorithim can be used (Huffman)
2. Remove redundant checks that are only necessary for pritable QR codes
3. Change size of pixels to best suit for our edge detection algo
4. Can scale QR code to as large or small as we require

```
def basic_shapes(lst,n,color,name="test.svg" ):
   # Drawing grid
    dwg = svgwrite.Drawing(filename=name, debug=True,size=(1450,1450))
    dwg.add(dwg.rect(insert=(0, 0), size=('100%', '100%'), rx=None, ry=None, fill='rgb(0,0,0)'))

    hlines = dwg.add(dwg.g(id='hlines', stroke='white'))
    for y in range(n-1):
        hlines.add(dwg.line(start=(2*int(1400/50)-16, (2+y)*int(1400/50)-16), end=(n*int(1400/50)-16, -16+(2+y)*int(1400/50))))
    vlines = dwg.add(dwg.g(id='vline', stroke='white'))
    for x in range(n-1):
        vlines.add(dwg.line(start=((2+x)*int(1400/50)-16, 2*int(1400/50)-16), end=((2+x)*int(1400/50)-16, -16+n*int(1400/50))))


    shapes = dwg.add(dwg.g(id='shapes', fill='red'))

    #Drawing points
    for i in range(len(lst)):
        for j in range(len(lst[1])):

            if(lst[i][j]):
                shapes.add(dwg.rect(insert=((j*int(1400/50)+42), (i*int(1400/50)+42)), size=(25, 25),fill=color, stroke_width=0))
    dwg.save()
```

Generating filter

```
def makefilter(n):
   # Drawing grid
    name="Filter.svg"
    color = "white"
    dwg = svgwrite.Drawing(filename=name, debug=True,size=(1450,1450))
    dwg.add(dwg.rect(insert=(0, 0), size=('100%', '100%'), rx=None, ry=None, fill='rgb(0,0,0)'))

    shapes = dwg.add(dwg.g(id='shapes', fill='red'))
    lst =[]
    lst2=[]
    for i in range(n-2):
      for j in range(n-2):
          lst2.append(1)
      lst.append(lst2)
      lst2=[]
    #Drawing points
    for i in range(len(lst)):
        for j in range(len(lst[1])):

            if(lst[i][j]):
                shapes.add(dwg.rect(insert=((j*int(1400/50)+50), (i*int(1400/50)+50)), size=(10, 10),fill=color, stroke_width=0))


    dwg.save()
    print("Filter made")
```

Designing Algo for svg to png

> The reason for SVG is that we require a scalable QR code that then can be converted in to png for the video.

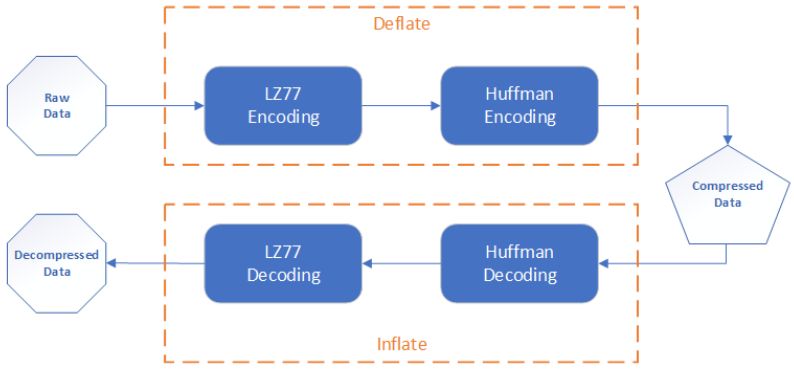
```
def make():

    # Conversion
    drawing = svg2rlg('File1.svg')
    renderPM.drawToFile(drawing, 'File1.png', fmt='PNG')
    drawing1 = svg2rlg('File2.svg')
    renderPM.drawToFile(drawing1, 'File2.png', fmt='PNG')
    drawing2 = svg2rlg('File3.svg')
    renderPM.drawToFile(drawing2, 'File3.png', fmt='PNG')
    drawing2 = svg2rlg('Filter.svg')
    renderPM.drawToFile(drawing2, 'Filter.png', fmt='PNG')
    #Multiplexing
    img1 = cv2.imread('File1.png')
    img2 = cv2.imread('File2.png')
    img3 = cv2.imread('File3.png')
    cv2.waitKey(0)
    dst1= cv2.add(img1,img2)
    dst = cv2.addWeighted(img1,0.9,img2,0.8,0)
    dst = cv2.addWeighted(dst,0.9,img3,0.7,0)
    cv2.imwrite('ff'+'.png', dst)
```

Compressing Data

2.1. Deflate

Deflate method was originally defined by Phil Katz in PKWARE's archiving tool PKZIP 2.x. It is a combination of the **LZ77 algorithm** and **Huffman encoding**.

The following figure illustrates the deflate and inflate process from a high level.




```
def compress(data):
  data = bytes(data,"utf-8")
  t = zlib.compress(data)
  x= list(t)
  bit_list=[]
  for i in x:
      bit_list.append(format(i,'08b'))

  bit_stream= ''.join(bit_list)

  print("Bit stream: "+  bit_stream)
  print("Size before compression: " + str(len(data)*8))
  print("Size after compression: " + str(len(bit_stream)))
  return (bit_stream)

x=compress("When Mr. Bilbo Baggins of Bag End announced that he would shortly be celebrating his eleventy-first birthday with a party of
```

```
Bit stream: 0111100010011100011001010101100001001101100100111101101110110110000100101111110000101011110010000100100110111011010101
Size before compression: 36152
Size after compression: 17368
```

Main function

```
def main():
    color="blue"
    name= "File1.svg"
    n=50
    f = open("data.txt", "r")
    test_str = f.read(1800)
    lst =[]
    lst2=[]

    val = compress(test_str) # compressing data
    val =''.join(format(ord(i), '08b') for i in test_str)
    print(test_str)
   ## Getting a matrix for displaying svg    ----------- Blue QR
    for i in range(n-2):
        for j in range(n-2):
            index = (i*(n-2))+j
            if ( index < len(val)):
                lst.append(int(val[index]))
            else:
                lst.append(0)
        lst2.append(lst)
        lst=[]
    basic_shapes(lst2,n,color,name)
    print("for blue the data is: "+val[0:index])
    print("for red the data is: "+val[index:index+index])
    print("for green the data is: "+val[index+index:index+index+index])

    val =val[index::]

    ## Getting a matrix for displaying svg    ----------- Red QR
    lst2=[]
    lst=[]
    color = "red"
    name = "File2.svg"
    print(len(val))
    if (len(val) > index): # check if there is more data to add in QR code

        for i in range(n-2):
            for j in range(n-2):
                index = (i*(n-2))+j
                if ( index < len(val)):
                    lst.append(int(val[index]))
                else:
                    lst.append(0)
            lst2.append(lst)
            lst=[]
        basic_shapes(lst2,n,color,name)
    val =val[index::]


    ## Getting a matrix for displaying svg    ----------- Green QR
    lst2=[]
    lst=[]
    color = "green"
    name = "File3.svg"

    if (len(val) > index): # check if there is more data to add in QR code

        for i in range(n-2):
            for j in range(n-2):
                index = (i*(n-2))+j
                if ( index < len(val)):
                    lst.append(int(val[index]))
                else:
                    lst.append(0)
            lst2.append(lst)
            lst=[]
        basic_shapes(lst2,n,color,name)

        makefilter(n)
```

```
startencoding = timer()
main()
make()
endencoding = timer()

print(endencoding - startencoding)
```

```
Bit stream: 0111100010011100011001010101010111001011011011101110001100111000000100001111110010010101110111011010011011001000000000
Size before compression: 14400
```

```
Size after compression: 7512
When Mr. Bilbo Baggins of Bag End announced that he would shortly be celebrating his eleventy-first birthday with a party of specia
for blue the data is: 0101011101101000011001010110111000100000010011010111001000101110001000000100001001101001011011000110001001101
for red the data is: 0011001010111011001100101011100100010000001110011011010010110111001100011011001010010000001101000011010010111 0
for green the data is: 0101110011001000000110111001101111011101000010000001100101011011100110111101110101011001110110100000100000001
12097
Filter made
3.0303624279999894
```

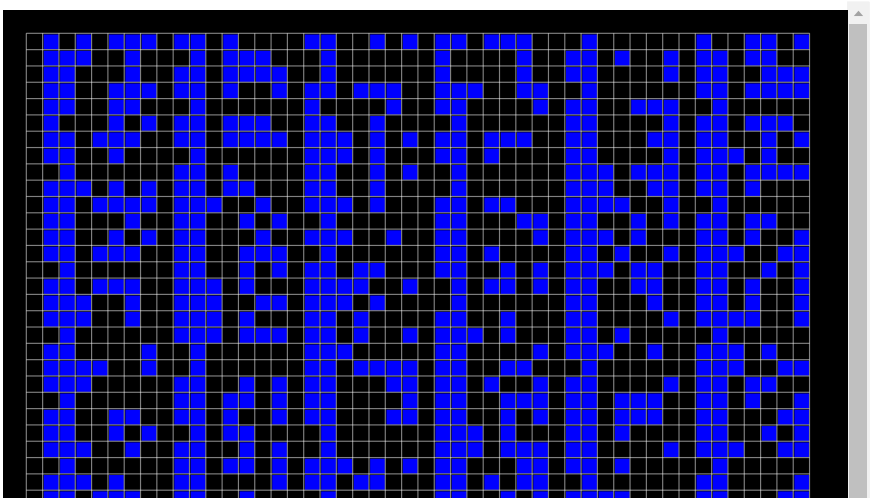
```
!pip install CairoSVG
from cairosvg import svg2png
svg_code = open("File1.svg", 'rt').read()
svg2png(bytestring=svg_code,write_to='output.png')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting CairoSVG
  Downloading CairoSVG-2.5.2-py3-none-any.whl (45 kB)
     |████████████████████████████████| 45 kB 2.2 MB/s
Collecting cairocffi
  Downloading cairocffi-1.4.0.tar.gz (69 kB)
     |████████████████████████████████| 69 kB 4.1 MB/s
Requirement already satisfied: pillow in /usr/local/lib/python3.8/dist-packages (from CairoSVG) (9.3.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.8/dist-packages (from CairoSVG) (0.7.1)
Requirement already satisfied: cssselect2 in /usr/local/lib/python3.8/dist-packages (from CairoSVG) (0.7.0)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.8/dist-packages (from CairoSVG) (1.2.1)
Requirement already satisfied: cffi>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from cairocffi->CairoSVG) (1.15.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-packages (from cffi>=1.1.0->cairocffi->CairoSVG) (2.21)
Requirement already satisfied: webencodings in /usr/local/lib/python3.8/dist-packages (from cssselect2->CairoSVG) (0.5.1)
Building wheels for collected packages: cairocffi
  Building wheel for cairocffi (setup.py) ... done
  Created wheel for cairocffi: filename=cairocffi-1.4.0-py3-none-any.whl size=88775 sha256=f45462faebb65c99e9496faca5a6548e9fb41e58
  Stored in directory: /root/.cache/pip/wheels/01/a9/c0/5c05f9dd73c21f9a7716690642823cdba55594d17a9bd69daf
Successfully built cairocffi
Installing collected packages: cairocffi, CairoSVG
Successfully installed CairoSVG-2.5.2 cairocffi-1.4.0
```

```
def display(filename):
  img = cv2.imread(filename)
  cv2_imshow(img)
```

Showing color QR code

```
display("File1.png")
display("File2.png")
display("File3.png")
```



Showing multiplex QR code


```
display("ff.png")
```

Showing filter

```
display("Filter.png")
```

Designing a degradation function

> This function is only for our theoretical results. we will first model some common degradation and see if our algorithim is able to detect the data at reciever end.

Adding noise

> We are adding Gaussian noise as that is a more probable model to be added when we do the experimental tests. This is because the screens have uniform noise.

```
!pip install wand
!apt-get install libmagickwand-dev
from wand.image import Image

# Read image using Image() function
with Image(filename ="ff.png") as img:
```

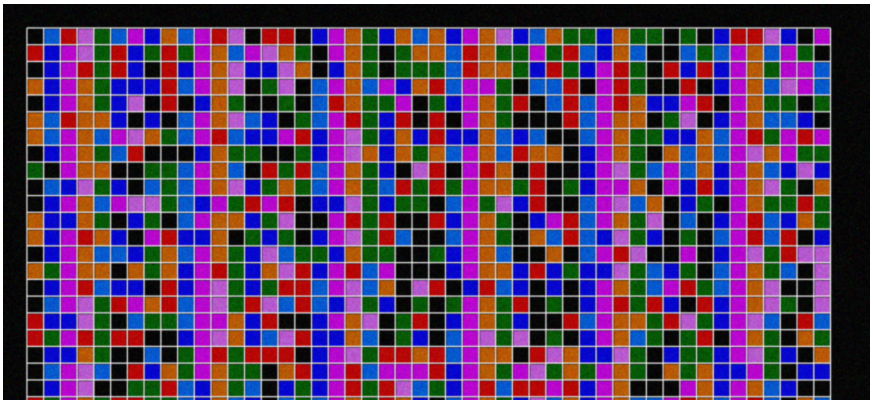
```
    # Generate noise image using spread() function
    img.noise("gaussian", attenuate = 0.9)
    img.save(filename ="noisy.png")

display("noisy.png")
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheel
Collecting wand
  Downloading Wand-0.6.10-py2.py3-none-any.whl (142 kB)
     |████████████████████████████████| 142 kB 5.1 MB/s
Installing collected packages: wand
Successfully installed wand-0.6.10
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libnvidia-common-460
Use 'apt autoremove' to remove it.
The following additional packages will be installed:
  fonts-droid-fallback fonts-noto-mono ghostscript gir1.2-freedesktop
  gir1.2-gdkpixbuf-2.0 gir1.2-rsvg-2.0 gsfonts imagemagick-6-common
  libcairo-script-interpreter2 libcairo2-dev libcupsfilters1 libcupsimage2
  libdjvulibre-dev libdjvulibre-text libdjvulibre21 libgdk-pixbuf2.0-dev
  libgs9 libgs9-common libijs-0.35 libjbig2dec0 liblcms2-dev liblqr-1-0
  liblqr-1-0-dev libmagickcore-6-arch-config libmagickcore-6-headers
  libmagickcore-6.q16-3 libmagickcore-6.q16-3-extra libmagickcore-6.q16-dev
  libmagickwand-6-headers libmagickwand-6.q16-3 libmagickwand-6.q16-dev
  libpixman-1-dev librsvg2-dev libwmf-dev libwmf0.2-7 libxcb-shm0-dev
  poppler-data
Suggested packages:
  fonts-noto ghostscript-x libcairo2-doc inkscape libjxr-tools librsvg2-doc
  libwmf-doc libwmf0.2-7-gtk poppler-utils fonts-japanese-mincho
  | fonts-ipafont-mincho fonts-japanese-gothic | fonts-ipafont-gothic
  fonts-arphic-ukai fonts-arphic-uming fonts-nanum
The following NEW packages will be installed:
  fonts-droid-fallback fonts-noto-mono ghostscript gir1.2-freedesktop
  gir1.2-gdkpixbuf-2.0 gir1.2-rsvg-2.0 gsfonts imagemagick-6-common
  libcairo-script-interpreter2 libcairo2-dev libcupsfilters1 libcupsimage2
  libdjvulibre-dev libdjvulibre-text libdjvulibre21 libgdk-pixbuf2.0-dev
  libgs9 libgs9-common libijs-0.35 libjbig2dec0 liblcms2-dev liblqr-1-0
  liblqr-1-0-dev libmagickcore-6-arch-config libmagickcore-6-headers
  libmagickcore-6.q16-3 libmagickcore-6.q16-3-extra libmagickcore-6.q16-dev
  libmagickwand-6-headers libmagickwand-6.q16-3 libmagickwand-6.q16-dev
  libmagickwand-dev libpixman-1-dev librsvg2-dev libwmf-dev libwmf0.2-7
  libxcb-shm0-dev poppler-data
0 upgraded, 38 newly installed, 0 to remove and 20 not upgraded.
Need to get 31.0 MB of archives.
After this operation, 91.0 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/main amd64 fonts-droid-fallback all
Get:2 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 imagemagick-6-comm
Get:3 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libmagickcore-6-a
Get:4 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libmagickcore-6-h
Get:5 http://archive.ubuntu.com/ubuntu bionic/main amd64 liblqr-1-0 amd64 0.4.2-2.
Get:6 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libmagickcore-6.q
Get:7 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libmagickwand-6-h
Get:8 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libmagickwand-6.q
Get:9 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libdjvulibre-text
Get:10 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libdjvulibre21 a
Get:11 http://archive.ubuntu.com/ubuntu bionic/main amd64 libwmf0.2-7 amd64 0.2.8.
Get:12 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libmagickcore-6.
Get:13 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libdjvulibre-dev
Get:14 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 liblcms2-dev amd
```

Reciever end processing

Gausian smoothing

```
from PIL import Image, ImageFilter
# Opening the image
# (R prefixed to string in order to deal with '\' in paths)
image = Image.open("noisy.png")
# Blurring image by sending the ImageFilter.
# GaussianBlur predefined kernel argument
image = image.filter(ImageFilter.GaussianBlur(radius=1))
# Displaying the image
image.save("test.png")
display("test.png")
```



Demux

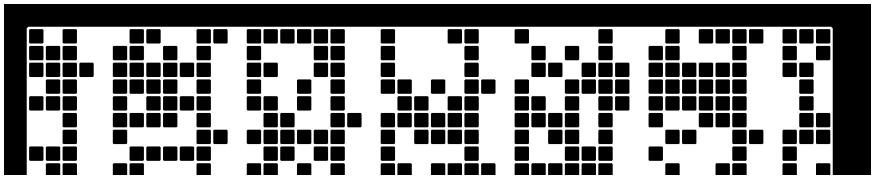

```
img1 = cv2.imread('test.png')
imgb = cv2.imread('test.png')
imgr = cv2.imread('test.png')
img2 = cv2.imread('File3.png')

img1[:,:,0]=0
img1[:,:,2]=0

imgb[:,:,2]=0
imgb[:,:,1]=0

imgr[:,:,0]=0
imgr[:,:,1]=0


img3 = cv2.hconcat([img1,img2])

cv2.imwrite("greenR.png" , img1)
cv2.imwrite("blueR.png" , imgb)
cv2.imwrite("redR.png" , imgr)

cv2_imshow(img3)
```

Right image = orignal

Left image = recieved image

Otsu thresholding

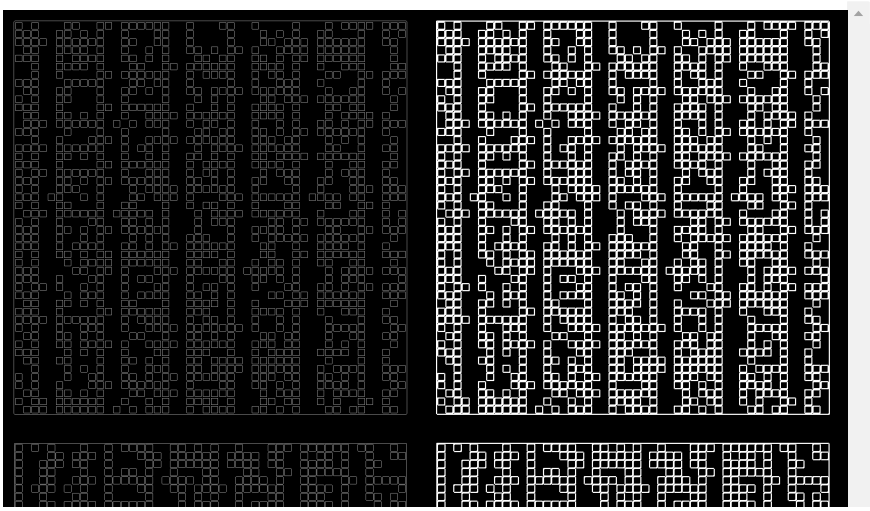
```
# Otsu's thresholding after Gaussian filtering
img = cv2.imread('greenR.png')
img1 = cv2.imread('blueR.png')
img3 = cv2.imread('redR.png')

img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img2 = img2.astype('uint8')
blur = cv2.GaussianBlur(img2,(5,5),0)

img11 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img11 = img11.astype('uint8')
blur1 = cv2.GaussianBlur(img11,(5,5),0)

img33 = cv2.cvtColor(img3, cv2.COLOR_BGR2GRAY)
img33 = img33.astype('uint8')
blur3 = cv2.GaussianBlur(img33,(5,5),0)



ret3,th3 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
ret2,th2 = cv2.threshold(blur1,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
ret1,th1 = cv2.threshold(blur3,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

cv2.imwrite("thresGreen.png" , th3)
cv2.imwrite("thresblue.png" , th2)
cv2.imwrite("threshred.png" , th1)
cv2_imshow(th3)
```



Edge detection using laplacien

> Use in the instance of tranlation of image due to the movement of camera while capturing the video. This will take care of any jitter or movemet by isolating the gride and using it for image registration.

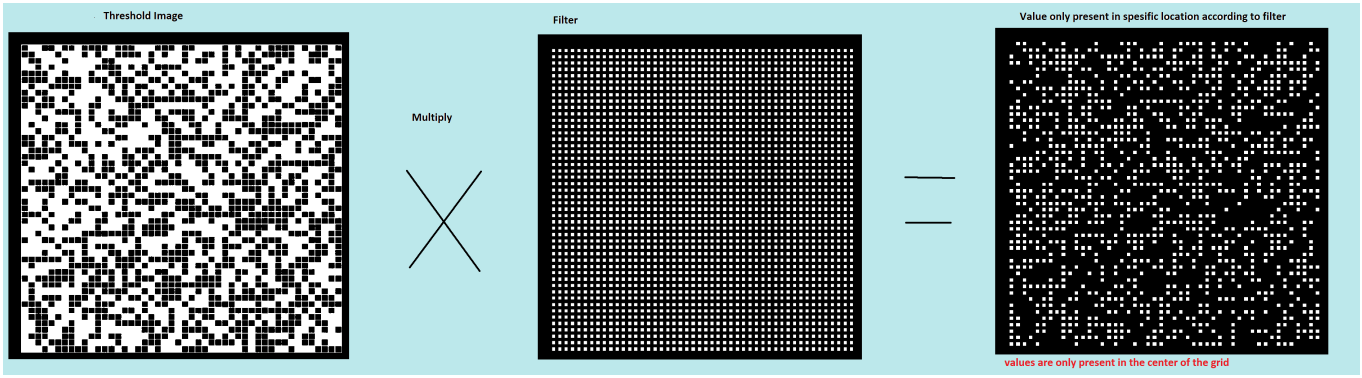
```
img = cv2.imread('thresGreen.png',0)
laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)
mag = abs(sobelx+sobely)

img3 = cv2.hconcat([laplacian,mag])
cv2_imshow(img3)
cv2.imwrite("edgeGreen.png",mag)


img = cv2.imread('thresblue.png',0)
laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)
mag = abs(sobelx+sobely)

img3 = cv2.hconcat([laplacian,mag])
cv2_imshow(img3)
cv2.imwrite("edgeblue.png",mag)


img = cv2.imread('threshred.png',0)
laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)
mag = abs(sobelx+sobely)

img3 = cv2.hconcat([laplacian,mag])
cv2_imshow(img3)
cv2.imwrite("edgered.png",mag)
```



Left: Laplacien RightL sobel Magnitude

Getting Values from QR

> To extract the binary information we are using a filter to center all values, this will remove any noise or distortions present in the image that havent been filtered out in the presvious stages. This process is robust as it takes grid of values that represnt each bit in the final bit stream hence in the next step when we extract each square, we will be taking a mean, this would ensure each bit is accurate. As we decrease the grid size the total area decreases hence our ability to do error correction would decrease but this would only matter in extream noisy situations, at that point we can anyways keep the grid size as large as the one we have right now.




```
img1 = cv2.imread('thresGreen.png')
img2 = cv2.imread('Filter.png')

dst= cv2.multiply(img1,img2,0.2)
dst2= cv2.addWeighted(dst,0.9,img2,0.7,0)
cv2.imwrite('green_extract.png', dst)
cv2_imshow(dst)
print("filter superimposed on image showing zeros and ones")
cv2_imshow(dst2)


img1 = cv2.imread('thresblue.png')
img2 = cv2.imread('Filter.png')

dst= cv2.multiply(img1,img2,0.2)
dst2= cv2.addWeighted(dst,0.9,img2,0.7,0)
cv2.imwrite('blue_extract.png', dst)
cv2_imshow(dst)
print("filter superimposed on image showing zeros and ones")
cv2_imshow(dst2)



img1 = cv2.imread('threshred.png')
img2 = cv2.imread('Filter.png')

dst= cv2.multiply(img1,img2,0.2)
```

```
dst2= cv2.addWeighted(dst,0.9,img2,0.7,0)
cv2.imwrite('red_extract.png', dst)
cv2_imshow(dst)
print("filter superimposed on image showing zeros and ones")
cv2_imshow(dst2)
```

```
import pandas as pd
import numpy as np
img = cv2.imread('Filter.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img = img.astype('uint8')

df = pd.DataFrame(img)
# isolating the first bit using the filter as a reference,
# first bit starts from 64 and there are 13X13 bits of info representing one bit

pd.set_option('max_colwidth', 400)
pd.describe_option('max_colwidth')
df
```

```
display.max_colwidth : int or None
    The maximum width in characters of a column in the repr of
    a pandas data structure. When the column overflows, a "..."
    placeholder is embedded in the output. A 'None' value means unlimited.
    [default: 50] [currently: 400]
```

	0	1	2	3	4	5	6	7	8	9	...	1440	1441	1442	1443	1444	1445	1446
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1445	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1446	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1447	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0

```
import pandas as pd
img = cv2.imread('blue_extract.png')
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
out = cv2.normalize(img2.astype('int32'), None, 0.0, 1.0, cv2.NORM_MINMAX)
size1 = len(out[:]) # same size since square
print("size of matrix: " + str(size1))
temp=[]
data=[]
for i in range(64,size1,28):
  for j in range(64,size1,28):
    sum_filter = out[i+1:i+15,j+1:j+15]
    value = max(map(lambda x: max(x), sum_filter))
    temp.append(value)
  data.append(temp)
  temp=[]
print(data)

index=0;
data_temp=[]
data_lst=[]
k=0
i=0
j=0
for k in range(49):
  if k == 47:
    continue
  for i in range(6):
    for j in range(8):
      index= (i*8)+j
      if (index == 46):
        data_temp.append(data[k][47])
      elif (index == 47):
        data_temp.append(data[k][48])
      else:

        data_temp.append(data[k][index])
    data_lst.append(data_temp)
    data_temp=[]

print(data_lst)

final_data_blue=[]
for i in range(49):
  final_data_blue.append(int(''.join(map(lambda x: str(x), data_lst[i])),2))


img = cv2.imread('red_extract.png')
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
out = cv2.normalize(img2.astype('int32'), None, 0.0, 1.0, cv2.NORM_MINMAX)
size1 = len(out[:]) # same size since square
print("size of matrix: " + str(size1))
temp=[]
data=[]
for i in range(64,size1,28):
  for j in range(64,size1,28):
    sum_filter = out[i+1:i+15,j+1:j+15]
    value = max(map(lambda x: max(x), sum_filter))
    temp.append(value)
  data.append(temp)
  temp=[]
```

```
print(data)

index=0;
data_temp=[]
data_lst=[]
k=0
i=0
j=0
for k in range(49):
  if k == 47:
    continue
  for i in range(6):
    for j in range(8):
      index= (i*8)+j
      if (index == 46):
        data_temp.append(data[k][47])
      elif (index == 47):
        data_temp.append(data[k][48])
      else:

        data_temp.append(data[k][index])
    data_lst.append(data_temp)
    data_temp=[]

print(data_lst)

final_data_red=[]
for i in range(49):
  final_data_red.append(int(''.join(map(lambda x: str(x), data_lst[i])),2))


img = cv2.imread('green_extract.png')
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
out = cv2.normalize(img2.astype('int32'), None, 0.0, 1.0, cv2.NORM_MINMAX)
size1 = len(out[:]) # same size since square
print("size of matrix: " + str(size1))
temp=[]
data=[]
for i in range(64,size1,28):
  for j in range(64,size1,28):
    sum_filter = out[i+1:i+15,j+1:j+15]
    value = max(map(lambda x: max(x), sum_filter))
    temp.append(value)
  data.append(temp)
  temp=[]
print(data)

index=0;
data_temp=[]
data_lst=[]
k=0
i=0
j=0
for k in range(49):
  if k == 47:
    continue
  for i in range(6):
    for j in range(8):
      index= (i*8)+j
      if (index == 46):
        data_temp.append(data[k][47])
      elif (index == 47):
        data_temp.append(data[k][48])
      else:

        data_temp.append(data[k][index])
    data_lst.append(data_temp)
    data_temp=[]

print(data_lst)

final_data_green=[]
for i in range(49):
  final_data_green.append(int(''.join(map(lambda x: str(x), data_lst[i])),2))

final_data = final_data_blue+ final_data_red+final_data_green
print(final_data)

string=""
for i in final_data_green:
  string=string+chr(i)
print(string)
```

```
size of matrix: 1450
[[1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0,
[[1, 1, 1, 0, 0, 1, 0, 0], [0, 1, 0, 1, 1, 1, 0, 0], [0, 1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 1, 0, 0], [1, 1, 0, 1, 0, 0, 1, 0],
size of matrix: 1450
[[0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
[[0, 1, 1, 0, 1, 0, 0, 1], [0, 1, 1, 0, 1, 1, 1, 0], [0, 1, 1, 0, 0, 0, 1, 1], [0, 1, 1, 0, 0, 1, 0, 1], [0, 0, 1, 0, 0, 0, 0, 0],
size of matrix: 1450
[[0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1,
[[0, 0, 1, 1, 0, 0, 1, 0], [1, 0, 1, 1, 0, 1, 1, 1], [0, 0, 1, 1, 0, 1, 1, 1], [1, 0, 1, 1, 1, 0, 1, 0], [1, 0, 1, 1, 0, 0, 1, 1],
[228, 92, 64, 132, 210, 216, 196, 222, 64, 132, 194, 204, 206, 210, 220, 230, 64, 220, 204, 64, 132, 194, 206, 64, 138, 220, 200, 6
2·7º³´ 37¹ 00¶²  842¹2 8º¹ 0¶8· 44¹ 897¶7´3²² ;4³
```

```
import cv2
import numpy as np
import math

from PIL import Image as im

startdecoding = timer()

# Load the image using OpenCV
img = cv2.imread("green_extract.png",cv2.IMREAD_GRAYSCALE)

kernel =  [[1 for i in range(10)] for j in range (10)]
kernel = np.array(kernel,dtype="int")

output = cv2.morphologyEx(np.array(img,dtype="uint8"), cv2.MORPH_HITMISS,kernel)

cv2.imwrite('outputgreen'+'.png', output)


lstgreen = []
for i in range(55,28*48+55,28):
    lst2 = []
    for j in range(55,28*49+55,28):
        if output[i][j]:
            lst2.append(1)
        else: lst2.append(0)
    lstgreen.append(lst2)




img = cv2.imread("red_extract.png",cv2.IMREAD_GRAYSCALE)

kernel =  [[1 for i in range(10)] for j in range (10)]
kernel = np.array(kernel,dtype="int")

output = cv2.morphologyEx(np.array(img,dtype="uint8"), cv2.MORPH_HITMISS,kernel)

cv2.imwrite('outputred'+'.png', output)


lstred = []
for i in range(55,28*48+55,28):
    lst2 = []
    for j in range(55,28*49+55,28):
        if output[i][j]:
            lst2.append(1)
        else: lst2.append(0)
    lstred.append(lst2)




img = cv2.imread("blue_extract.png",cv2.IMREAD_GRAYSCALE)

kernel =  [[1 for i in range(10)] for j in range (10)]
kernel = np.array(kernel,dtype="int")

output = cv2.morphologyEx(np.array(img,dtype="uint8"), cv2.MORPH_HITMISS,kernel)

cv2.imwrite('outputblue'+'.png', output)


lst = []
for i in range(55,28*48+55,28):
    lst2 = []
    for j in range(55,28*49+55,28):
        if output[i][j]:
```