# detection

July 18, 2019

## 1 import

```
In [1]: import cv2
        print(cv2.__version__)
        import matplotlib
        import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        import os
        from numba import jit
```

```
4.1.0
```

## 2 File selection

```
In [2]: path = "/Users/oliviermanette/Desktop/trailer detection challenge/data/P473_Arizona_Day
        os.chdir(path)
```

```
In [3]: pwd
```

```
Out[3]: '/Users/oliviermanette/Desktop/trailer detection challenge/data/P473_Arizona_Day_Aspha
```

```
In [4]: ls
```

```
P473_Arizona_Day_Asphalt_Close_To_Sunset_dry_Nominal_8300lx.avi*
P473_Arizona_Day_Asphalt_Close_To_Sunset_dry_Nominal_8300lx.dat_GT.csv*
```

```
In [5]: #fileName='W420_ES_Hi_Snow_Slush_Asphalt_28klux.avi'
        fileName='P473_Arizona_Day_Asphalt_Close_To_Sunset_dry_Nominal_8300lx.avi'
```

## 3 Test Video Loop

```
In [6]: cap = cv2.VideoCapture(fileName)  # load the video
        while (cap.isOpened()):  # play the video by reading frame by frame
            ret, frame = cap.read()
```

```
        if ret == True:
            # optional: do some image processing here
            cv2.imshow('frame', frame)
            # show the video
            if cv2.waitKey(1) & 0xFF == ord('q'):
                #if 0xFF == ord('q'):
                break
        else:
            break
    cap.release()
    cv2.destroyAllWindows()
```

# 4 Variables globales

Pour des raisons de lisibilité du code, l'ensemble des variables locales seront précédés du préfixe 'l' afin de les différencier des variables globales qui n'ont pas de préfixe. ## Type de données de position

```
In [81]: posType = np.dtype([('x', 'u1'), ('y', 'u2')])
```

## 4.1 Type de données de Neurones

```
In [7]: NeuronType = np.dtype([('longueur', 'u1'), ('angle', 'f4'), ('weight', 'f4'),
                               ('precision', 'f4'), ('xPos', 'u1'), ('yPos', 'u2'),
                               ('group', 'u1'), ('layer', 'u1')])
```

## 4.2 Taille des champs récepteurs neuronaux

```
In [80]: tailleField = 7
```

# 5 Fonctions

## 5.1 Calcul d'un neurone champ moyen

A partir d'une liste de neurones, il retourne le neurone moyen

```
In [82]: def getAvgFieldNeuron(lNeuronList, typeList=NeuronType):
             lNeurons = np.zeros(1, dtype=typeList)
             lpNeurons = pd.DataFrame(lNeurons)
             lpNeurons['longueur'] = int(lNeuronList.longueur[0:1])
             lpNeurons['angle'] = float(
                 np.sum((lNeuronList.angle * lNeuronList.weight) / np.sum(lNeuronList.weight))
             lpNeurons['weight'] = float(
                 np.sum((lNeuronList.weight * lNeuronList.weight) / np.sum(lNeuronList.weight)
             lpNeurons['precision'] = float(
                 np.sum((lNeuronList.precision * lNeuronList.weight) / np.sum(lNeuronList.weig
             lpNeurons['xPos'] = int(
                 np.sum((lNeuronList.xPos * lNeuronList.weight) / np.sum(lNeuronList.weight)))
```

```
            lpNeurons['yPos'] = int(
                np.sum((lNeuronList.yPos * lNeuronList.weight) / np.sum(lNeuronList.weight)))
            return lpNeurons
```

### 5.2 Matrice des directions

Afin de faciliter le calcul des angles des pixels, une matrice de poids est générée afin d'appliquer à chaque pixel centré sur un champs récepteur un poids correspondant à l'angle d'une ligne passant par ce centre. Voici comment les angles sont représentés IMAGE

```
In [9]: @jit(nopython=True, parallel=True)
        def fillAngleMat(lSize):
            lOutput = np.zeros((lSize, lSize))
            lOffset = int(np.floor(lSize / 2))
            for lX in range(0, lSize):
                for lY in range(0, lSize):
                    if (lX - lOffset) == 0:
                        lOutput[lX, lY] = 90
                    else:
                        lOutput[lX, lY] = np.around(
                            np.arctan((lY - lOffset) / (lOffset - lX)) / pi * 180, 2)
            lOutput[lOffset, lOffset] = 0
            return lOutput
```

### 5.3 Fonction d'activation des neurones

Chaque neurone retourne une valeur comprise entre 0 et 255 qui reflète son niveau d'activation. Cette activation reflète le niveau de confiance que le neurone a sur le lien existant entre sa fonction de base et les pixels reçus dans son champs récepteur. Plus les pixels sont organisés de façon à former une ligne avec l'angle correspondant à la fonction de base du neurone et plus ce dernier sera activé. Comme on ne souhaite pas obtenir une activation de valeur infinie, on utilise donc une fonction sigmoide qui s'applique à l'écart-type des angles supposés.

```
In [10]: @jit(nopython=True, parallel=True)
         def sigmoidActivationFctN1(activationVector):
             lDenom = (1 + np.exp(0.1 * (np.abs(np.std(activationVector)) - 30)))
             return 255 / lDenom
```

### 5.4 Création d'une liste de neurones à champs récepteurs

```
In [11]: #@jit(nopython=True, parallel=True)
         def getNeuronActivationList(idxX, idxY, size, frameE, nbPixelPts, layer=1):
             #commencer par créer le tableau de neurones
             lNeuronType = np.dtype([('longueur', 'u1'), ('angle', 'f4'),
                                     ('weight', 'f4'), ('precision', 'f4'),
                                     ('xPos', 'u1'), ('yPos', 'u2'), ('group', 'u1')])

             lCriterion = nbPixelPts > size
```

```
nbNeurons = sum(lCriterion)
lNeurons = np.zeros(nbNeurons, dtype=lNeuronType)
lpNeurons = pd.DataFrame(lNeurons)
lpNeurons['longueur'] = size
lpNeurons['layer'] = layer

offsetField = int(np.floor(size / 2))
lAngleMat = fillAngleMat(size)

newX = idxX[lCriterion]
newY = idxY[lCriterion]
print("size :" + str(len(newX)))

print("newX")
print(np.min(newX))
print(np.max(newX))
print("newY")
print(np.min(newY))
print(np.max(newY))
print()
pos = 0
lnPos = 0
for lintX in newX:
    lintY = newY[pos]

    lNeuronFieldFrame = frameE[
        int(lintX - offsetField):int(lintX + offsetField + 1),
        int(lintY - offsetField):int(lintY + offsetField + 1)] / 255

    tmp = np.multiply(lAngleMat, lNeuronFieldFrame)

    lNeuronFieldValues = tmp[np.nonzero(tmp)]
    if (lNeuronFieldValues.size > 0):
        lpNeurons.loc[pos, ['angle']] = np.mean(lNeuronFieldValues)
        lpNeurons.loc[pos, ['weight']] = sigmoidActivationFctN1(
            lNeuronFieldValues)
        lpNeurons.loc[pos, ['precision']] = np.std(lNeuronFieldValues)
        lpNeurons.loc[pos, ['xPos']] = lintX
        lpNeurons.loc[pos, ['yPos']] = lintY

        lnPos += 1
    else:
        True  #print ("error it shouldn't be zero")
    pos += 1
print("nb de positions couvertes : " + str(lnPos) + " sur " + str(pos))

return lpNeurons
#return idxY
```

### 5.5 Nombre de pixels actifs dans chaque champs récepteur

A partir des coordonnées des centres supposés de chaque champs récepteurs et de la taille du champs récepteur, recherche sur la frame bitmap passée en paramètres, retourne un tableau contenant le nombre de pixels allumés à l'intérieur de chacun de ces champs.

```
In [12]: @jit(nopython=True, parallel=True)
         def nbPixelField(tableX, tableY, frameEdge, lintTailleField=3):
             idx = 0
             results = np.zeros(tableX.size)
             rayon = np.floor(lintTailleField / 2)
             tailleMaxX = frameEdge.shape[0]
             tailleMaxY = frameEdge.shape[1]
             halfX = tailleMaxX / 3

             for posX in tableX:
                 posY = tableY[idx]
                 if posX > halfX and posX >= rayon and (posX + rayon) < tailleMaxX:
                     results[idx] = np.sum(
                         frameEdge[int(posX - rayon):int(posX + rayon + 1),
                                   int(posY - rayon):int(posY + rayon + 1)] / 255)
                 idx += 1  #tailleField
             return results
```

```
In [13]: #@jit(nopython=True, parallel=True)
         def getNonZero(img):
             return np.where(img != [0])
```

### 5.6 Retourne les coordonnées du centre d'un champs récepteur neuronal

```
In [14]: def getNFCoordinate(lNeurone):
             try:
                 lintDist = int(np.floor(lNeurone.longueur / 2))
             except:
                 P1 = (0, 0)
                 P2 = (0, 0)
                 return (P1, P2)
             if np.abs(lNeurone.angle / 180 * np.pi) < 45:
                 lAlpha = lNeurone.angle / 180 * pi
                 lintX1 = np.around(lNeurone.xPos - lintDist * np.tan(lAlpha))
                 lintY1 = lNeurone.yPos + lintDist
                 lintX2 = np.around(lNeurone.xPos + lintDist * np.tan(lAlpha))
                 lintY2 = lNeurone.yPos - lintDist
             else:
                 lAlpha = 90 - lNeurone.angle / 180 * pi
                 lintY1 = np.around(lNeurone.yPos - lintDist * np.tan(lAlpha))
                 lintX1 = lNeurone.xPos - lintDist
                 lintY2 = np.around(lNeurone.yPos + lintDist * np.tan(lAlpha))
                 lintX1 = lNeurone.xPos + lintDist
```

```
            P1 = (int(lintY1), int(lintX1))
            P2 = (int(lintY2), int(lintX2))
            return P1, P2
```

### 5.7 Calcule la distance entre deux points

```
In [15]: def getDistance(x1, y1, x2, y2):
             return np.sqrt(np.power(np.abs(x1 - x2), 2) + np.power(np.abs(y1 - y2), 2))
```

### 5.8 Retourne les neurones les plus proches d'un point

```
In [16]: def closestFieldNeurons(neuronList, posX, posY, distance):
             return neuronList[(neuronList.xPos > posX - distance)
                               & (neuronList.xPos < posX + distance) &
                               (neuronList.yPos > posY - distance) &
                               (neuronList.yPos < posY + distance)]
```

### 5.9 Ajoute les fonctions de base des neurones à champs récepteur sur un bitmap

```
In [17]: def drawFieldNeurons(lNeuronList, lBitmap):
             for index, lNeuron in lNeuronList.iterrows():
                 #for lNeuron in lNeuronList:
                 lCoord = getNFCoordinate(lNeuron)
                 #print(lNeuron)
                 try:
                     cv2.line(lBitmap, lCoord[0], lCoord[1], (int(
                         lNeuron.weight), int(lNeuron.weight), int(lNeuron.weight)), 3)
                 except:
                     True
             return lBitmap
```

### 5.10 Find neuronal groups

Un groupe neuronal est un ensemble de neurone dont les champs récepteurs sont complémentaires les uns des autres. Pour faire partie d'un champs récepteur, deux conditions doivent être réunies. (A compléter) ### Translation Retourne les coordonnées d'un point translaté d'une certaine distance avec un certain angle. Cette fonction demande un angle, une distance et les coordonnées d'un point de départ. Il retourne ensuite les coordonnées après translation.

```
In [18]: #@jit(nopython=True, parallel=True)
         def moveCoordDeg(angle, startX, startY, distance):
             tipX = startX + distance * np.sin(angle / 180 * pi)
             tipY = startY - distance * np.cos(angle / 180 * pi)
             return tipX, tipY
```

Effectue le même calcul que la fonction moveCoordDeg mais prend comme paramètre un neurone. Il effectue la translation en prenant comme point de départ le centre du champs récepteur et effectue un déplacement de la taille de ce champs dans la direction de la fonction de base.

```
In [19]: def getNextPosition(neuroneMoyen):
             return moveCoordDeg(float(neuroneMoyen.angle), int(neuroneMoyen.xPos),
                                 int(neuroneMoyen.yPos), int(neuroneMoyen.longueur))
```

### 5.10.1 Calcul des groupes à partir d'une liste de neurones à champs récepteurs

```
In [20]: def findGroups(neuronList):
             # Sélection d'un nouveau numéro de Groupe (GroupID)
             lintCurrentGroupID = 0
             lintNbGroups = 0
             lIndex = 0

             # liste des neurones sans groupe
             lNoGroupList = neuronList[neuronList.group == 0]

             while lNoGroupList.shape[0] > 0:

                 #Sélection d'un neurone dans la liste (ceux sans groupID ou groupID=0)
                 lMoyenNeuron = lNoGroupList.iloc[0]
                 lIndex = lNoGroupList.head().index.values[0]

                 while True:
                     #Assignation d'un nouveau numéro de GroupID en cours
                     lintNbGroups += 1
                     lintCurrentGroupID += 1
                     if neuronList[neuronList.group ==
                                   lintCurrentGroupID].shape[0] == 0:
                         break

                 neuronList.loc[lIndex, ['group']] = lintCurrentGroupID

                 #déplacement
                 lnPos = getNextPosition(lMoyenNeuron)

                 #recherche de neurones proches
                 lClosestNeurons = closestFieldNeurons(
                     neuronList, lnPos[0], lnPos[1],
                     int(np.floor(lMoyenNeuron.longueur / 2)))

                 #Oui ==> retour étape 1
                 while lClosestNeurons.shape[0] != 0:
                     #recherche des groupID dans cette sous-sélection
                     if lClosestNeurons[lClosestNeurons.group > 0].shape[0] == 0:
                         #Non => Assigner à tous les neurones de la sous-sélection le groupID
                         for lintIdx in lClosestNeurons.head().index.values:
                             neuronList.loc[lintIdx, ['group']] = lintCurrentGroupID
                     else:
                         #Oui
```

```
                #Récupération de la liste de tous les groupID utilisés
                #Sélection du groupID le plus petit (en comparant aussi avec le group
                lintPreviousGroupID = lintCurrentGroupID
                lintCurrentGroupID = np.min(
                    lClosestNeurons[lClosestNeurons.group > 0].group)
                #Assigner à tous les neurones de la sous-sélection ce nouveau groupID
                for lintIdx in lClosestNeurons.head().index.values:
                    neuronList.loc[lintIdx, ['group']] = lintCurrentGroupID
                    #remplacer dans la liste globale, pour chaque groupID présent dan
                    for lintGroupID in lClosestNeurons[
                            lClosestNeurons.group > 0].group:
                        neuronList.loc[neuronList.group == lintGroupID,
                                       'group'] = lintCurrentGroupID
                if lintPreviousGroupID == lintCurrentGroupID:
                    #si tous les neurones
                    if lClosestNeurons[lClosestNeurons.group >
                                       0].shape[0] == lClosestNeurons[
                                           lClosestNeurons.group ==
                                           lintPreviousGroupID].shape[0]:
                        break  # sortie de la boucle while
            #Calcul du neurone Field moyen
            lMoyenNeuron = getAvgFieldNeuron(lClosestNeurons)
            #déplacement
            lnPos = getNextPosition(lMoyenNeuron)

            #recherche de neurones proches
            lClosestNeurons = closestFieldNeurons(
                neuronList, lnPos[0], lnPos[1],
                int(np.floor(lMoyenNeuron.longueur / 2)))

        lNoGroupList = neuronList[neuronList.group == 0]
    return neuronList
```

# 6 Video Loop

In [21]:
```
kernelSize=21   # Kernel Bluring size

# Edge Detection Parameter
parameter1=20
parameter2=40
intApertureSize=1

#cap = cv2.VideoCapture(0)
cap = cv2.VideoCapture(fileName)
lCounter=0
while(cap.isOpened()):
    # Capture frame-by-frame
```

```
        ret, frame = cap.read()
        if ret==True:
            # Our operations on the frame come here
            if lCounter==1:
                frame = cv2.GaussianBlur(frame, (kernelSize,kernelSize), 0, 0)
                frame = cv2.Canny(frame,parameter1,parameter2,intApertureSize)  # Canny e
                lCounter = 0
            lCounter += 1;
            #frame = cv2.Laplacian(frame,cv2.CV_64F) # Laplacian edge detection
            #frame = cv2.Sobel(frame,cv2.CV_64F,1,0,ksize=kernelSize) # X-direction Sobel
            #frame = cv2.Sobel(frame,cv2.CV_64F,0,1,ksize=kernelSize) # Y-direction Sobel
            #frame[0:10,10]=255
            indices = np.where(frame != [0])
            # Display the resulting frame
            cv2.imshow('Canny',frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):  # press q to quit
                break
        else:
            break
    # When everything done, release the capture
    cap.release()
    cv2.destroyAllWindows()
```

## 7 Sandbox

```
In [22]: frame.shape
```

```
Out[22]: (800, 1280)
```

```
In [23]: frame.max()
```

```
Out[23]: 255
```

```
In [24]: indices = np.where(frame != [0])
```

```
In [25]: tata =  getNonZero(frame)
```

```
In [26]: coordinates = zip(indices[0], indices[1])
```

```
In [27]: indices[1].size
```

```
Out[27]: 9373
```

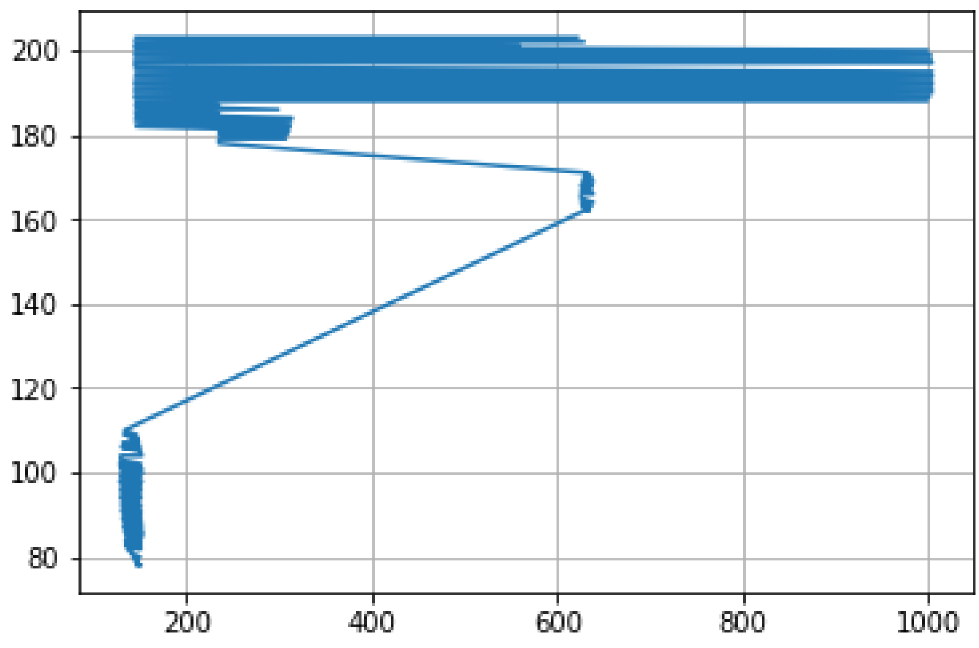
```
In [28]: indices[1][0:100]
```

```
Out[28]: array([147, 150, 151, 145, 146, 148, 142, 143, 144, 149, 150, 140, 141,
                136, 137, 138, 139, 151, 134, 135, 151, 134, 151, 134, 153, 133,
                153, 132, 133, 152, 132, 152, 131, 151, 131, 151, 130, 151, 130,
                149, 151, 130, 150, 129, 151, 129, 151, 129, 151, 129, 151, 128,
```

```
        152, 129, 151, 129, 151, 153, 129, 152, 128, 151, 129, 129, 151,
        152, 150, 151, 130, 131, 149, 150, 131, 143, 144, 145, 149, 139,
        142, 148, 132, 133, 134, 135, 136, 137, 138, 140, 141, 146, 133,
        629, 630, 631, 632, 633, 634, 635, 627, 628])
```

```
In [29]: fig, ax = plt.subplots()
         #s = pow(0.75,t)
         ax.plot(indices[1][0:300],indices[0][0:300])
         #ax.set(xlabel='time (s)', ylabel='voltage (mV)',
         #       title='About as simple as it gets, folks')
         ax.grid()

         #fig.savefig("test.png")
         plt.show()
```




```
In [30]: print(str(indices[0][0:30])+','+str(indices[1][0:30]))
```

```
[78 78 78 79 79 79 80 80 80 80 80 81 81 82 82 82 82 82 83 83 83 84 84 85
 85 86 86 87 87 87],[147 150 151 145 146 148 142 143 144 149 150 140 141 136 137 138 139 151
 134 135 151 134 151 134 153 133 153 132 133 152]
```

```
In [31]: tailleField = 3;#must be odd
         nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField);
         toto = nbPixelsAll > tailleField
         sum(toto)
```

Out[31]: 999

In [32]: nbPixelsAll

Out[32]: array([0., 0., 0., ..., 3., 3., 2.])

In [33]:
```
from numpy import pi
5-4*np.tan(np.arctan(-3/1))
```

Out[33]: 17.0

In [34]:
```
13-4*np.tan(np.arctan(-2/3))
```

Out[34]: 15.666666666666666

In [35]:
```
angleMat = fillAngleMat(7)
```

In [36]:
```
np.around(angleMat)
```

Out[36]:
```
array([[-45., -34., -18.,   0.,  18.,  34.,  45.],
       [-56., -45., -27.,   0.,  27.,  45.,  56.],
       [-72., -63., -45.,   0.,  45.,  63.,  72.],
       [ 90.,  90.,  90.,   0.,  90.,  90.,  90.],
       [ 72.,  63.,  45.,  -0., -45., -63., -72.],
       [ 56.,  45.,  27.,  -0., -27., -45., -56.],
       [ 45.,  34.,  18.,  -0., -18., -34., -45.]])
```

In [37]:
```
posX=indices[0][257]
posY=indices[1][257]
titi = frame[int(posX - tailleField):int(posX + tailleField + 1),
                               int(posY - tailleField):int(posY + tailleField + 1)]/255
```

In [38]:
```
test = np.multiply(angleMat,titi)
test2 = test[np.nonzero(test)]
np.mean(test2)
```

Out[38]: 2.179999999999999

In [39]:
```
test
```

Out[39]:
```
array([[ -0.  ,  -0.  ,  -0.  ,   0.  ,   0.  ,   0.  ,   0.  ],
       [-56.31,  -0.  ,  -0.  ,   0.  ,   0.  ,   0.  ,   0.  ],
       [-71.57, -63.43, -45.  ,   0.  ,   0.  ,   0.  ,  71.57],
       [  0.  ,   0.  ,   0.  ,   0.  ,  90.  ,  90.  ,   0.  ],
       [  0.  ,   0.  ,   0.  ,  -0.  ,  -0.  ,  -0.  ,  -0.  ],
       [  0.  ,   0.  ,   0.  ,  -0.  ,  -0.  ,  -0.  ,  -0.  ],
       [  0.  ,   0.  ,   0.  ,  -0.  ,  -0.  ,  -0.  ,  -0.  ]])
```

In [40]:
```
titi
```

```
Out[40]: array([[0., 0., 0., 0., 0., 0., 0.],
                [1., 0., 0., 0., 0., 0., 0.],
                [1., 1., 1., 0., 0., 0., 1.],
                [0., 0., 0., 1., 1., 1., 0.],
                [0., 0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0., 0.]])
```

```
In [41]: 255/(1+np.exp(0.1*(np.abs(np.std(test2))-30)))
```

```
Out[41]: 4.018509838601157
```

```
In [42]: sigmoidActivationFctN1(test2)
```

```
Out[42]: 4.018509838601157
```

```
In [43]: np.std(test2)
```

```
Out[43]: 71.3446804504122
```

```
In [44]: neuronList = np.zeros(1144,dtype=NeuronType)
```

```
In [45]: neuronList
```

```
Out[45]: array([(0, 0., 0., 0., 0, 0, 0, 0), (0, 0., 0., 0., 0, 0, 0, 0),
                (0, 0., 0., 0., 0, 0, 0, 0), ..., (0, 0., 0., 0., 0, 0, 0, 0),
                (0, 0., 0., 0., 0, 0, 0, 0), (0, 0., 0., 0., 0, 0, 0, 0)],
               dtype=[('longueur', 'u1'), ('angle', '<f4'), ('weight', '<f4'), ('precision', '<
```

```
In [46]: neuronList.size
```

```
Out[46]: 1144
```

```
In [47]: neuronList[0].shape
```

```
Out[47]: ()
```

```
In [48]: pNeurons = pd.DataFrame(neuronList)
```

```
In [49]: pNeurons.head()
```

Out[49]:

| | longueur | angle | weight | precision | xPos | yPos | group | layer |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0.0 | 0.0 | 0.0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0.0 | 0.0 | 0.0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0.0 | 0.0 | 0.0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0.0 | 0.0 | 0.0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0.0 | 0.0 | 0.0 | 0 | 0 | 0 | 0 |

```
In [50]: pNeurons['longueur'].head()
```

```
Out[50]: 0    0
         1    0
         2    0
         3    0
         4    0
         Name: longueur, dtype: uint8
```

```
In [51]: pNeurons.loc[1:3,['angle','weight']]
```

```
Out[51]:    angle  weight
         1    0.0     0.0
         2    0.0     0.0
         3    0.0     0.0
```

```
In [52]: pNeurons.loc[1,['angle']]=28.34
```

```
In [53]: pNeurons['longueur'] = tailleField
```

```
In [54]: pNeurons.head()
```

```
Out[54]:    longueur  angle  weight  precision  xPos  yPos  group  layer
         0         3   0.00     0.0        0.0     0     0      0      0
         1         3  28.34     0.0        0.0     0     0      0      0
         2         3   0.00     0.0        0.0     0     0      0      0
         3         3   0.00     0.0        0.0     0     0      0      0
         4         3   0.00     0.0        0.0     0     0      0      0
```

```
In [55]: nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField);
         lCriterion = nbPixelsAll > tailleField
         print(lCriterion.shape)
         print(len(indices[0]))
         print(np.sum(lCriterion))
         resultIndicesCriterion0 = indices[0][lCriterion]
         print(len(resultIndicesCriterion0))
         resultIndicesCriterion1 = indices[1][lCriterion]
         print(len(resultIndicesCriterion1))
         print("result0 :")
         print(np.min(resultIndicesCriterion0))
         print(np.max(resultIndicesCriterion0))
         print("result1 :")
         print(np.min(resultIndicesCriterion1))
         print(np.max(resultIndicesCriterion1))
```

```
(9373,)
9373
1515
1515
1515
result0 :
```

```
401
780
result1 :
56
1230
```

In [56]:
```
print(frame[(778-3):(778+3+1),(1246-3):(1246+3+1)])
```

```
[[0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]]
```

In [57]:
```
tailleField = 7
indices = np.where(frame != [0])
nbPixelsAll = nbPixelField(indices[0], indices[1], frame, tailleField);
print("indice 0")
print(np.min(indices[0]))
print(np.max(indices[0]))
print("indice 1")
print(np.min(indices[1]))
print(np.max(indices[1]))
titi = getNeuronActivationList(indices[0], indices[1], tailleField, frame,
                               nbPixelsAll)
```

```
indice 0
78
784
indice 1
17
1249
size :1515
newX
401
780
newY
56
1230

nb de positions couvertes : 1515 sur 1515
```

In [58]:
```
titi.describe()
```

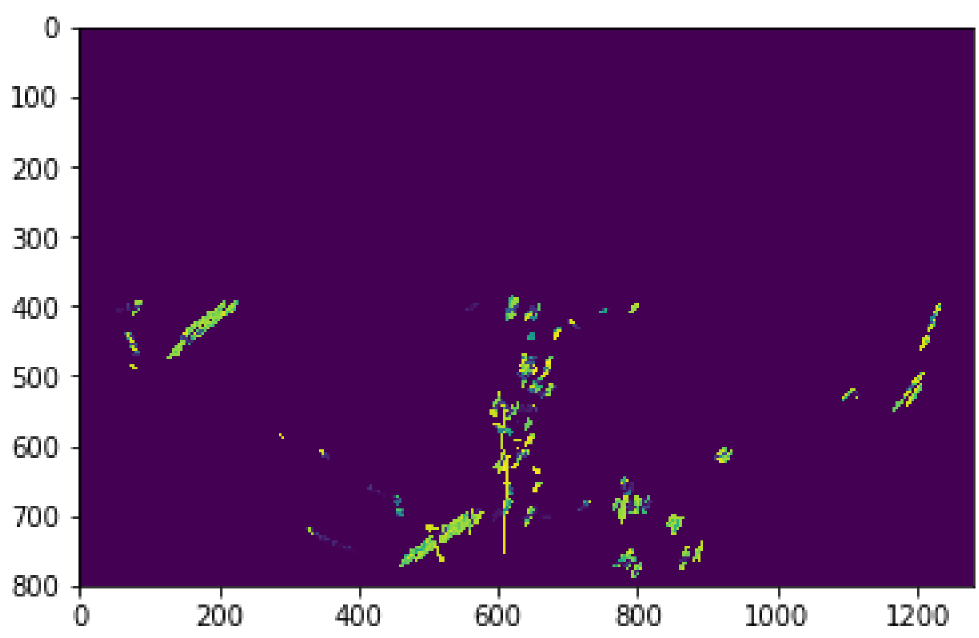
Out[58]:

|       | longueur | angle | weight | precision | xPos |
|---|---|---|---|---|---|
| count | 1515.0 | 1515.000000 | 1515.000000 | 1515.000000 | 1515.000000 |
| mean | 7.0 | 29.927622 | 134.548584 | 30.581026 | 578.699010 |
| std | 0.0 | 33.965427 | 86.172958 | 19.807312 | 123.133523 |
| min | 7.0 | -55.677143 | 1.950733 | 0.000000 | 401.000000 |
| 25% | 7.0 | 3.071667 | 33.061687 | 14.824060 | 457.000000 |
| 50% | 7.0 | 42.367142 | 179.661392 | 21.309187 | 553.000000 |
| 75% | 7.0 | 56.951166 | 209.146645 | 49.040255 | 698.000000 |
| max | 7.0 | 90.000000 | 242.906403 | 78.653786 | 780.000000 |

|       | yPos | group | layer |
|---|---|---|---|
| count | 1515.000000 | 1515.0 | 1515.0 |
| mean | 593.918152 | 0.0 | 1.0 |
| std | 286.638927 | 0.0 | 0.0 |
| min | 56.000000 | 0.0 | 1.0 |
| 25% | 460.000000 | 0.0 | 1.0 |
| 50% | 623.000000 | 0.0 | 1.0 |
| 75% | 723.000000 | 0.0 | 1.0 |
| max | 1230.000000 | 0.0 | 1.0 |

In [59]:

```
testBitmap = np.zeros(frame.shape)
testBitmap = drawFieldNeurons(titi,testBitmap)
imgplot = plt.imshow(testBitmap)
```

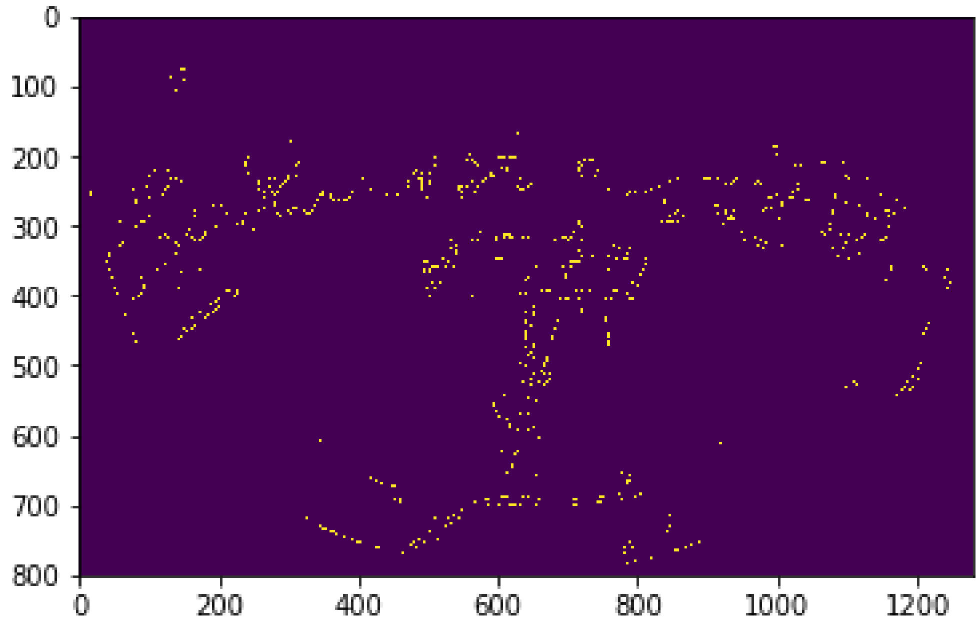



In [60]:

```
while(1):
    cv2.imshow('Canny',frame)
```

```
        if cv2.waitKey(1) & 0xFF == ord('q'):  # press q to quit
                break
```

In [61]: `imgplot = plt.imshow(frame)`



In [62]: `np.max(indices[0])`

Out[62]: 784

In [63]: `np.max(nbPixelsAll)`

Out[63]: 14.0

In [64]: `titi[0:4]`

Out[64]:

| | longueur | angle | weight | precision | xPos | yPos | group | layer |
|---|---|---|---|---|---|---|---|---|
| 0 | 7 | -14.110000 | 21.038452 | 54.088051 | 401 | 72 | 0 | 1 |
| 1 | 7 | -19.285715 | 42.362766 | 46.133179 | 401 | 73 | 0 | 1 |
| 2 | 7 | 58.060001 | 211.724838 | 14.122912 | 401 | 85 | 0 | 1 |
| 3 | 7 | 57.513332 | 210.000854 | 14.595321 | 401 | 86 | 0 | 1 |

In [65]: `moveCoordDeg(45,10,10,5)`

Out[65]: (13.535533905932738, 6.464466094067262)

In [66]: `findGroups(titi)`

Out[66]:

| | longueur | angle | weight | precision | xPos | yPos | group | layer |
|---|---|---|---|---|---|---|---|---|
| 0 | 7 | -14.110000 | 21.038452 | 54.088051 | 401 | 72 | 1 | 1 |
| 1 | 7 | -19.285715 | 42.362766 | 46.133179 | 401 | 73 | 1 | 1 |
| 2 | 7 | 58.060001 | 211.724838 | 14.122912 | 401 | 85 | 3 | 1 |
| 3 | 7 | 57.513332 | 210.000854 | 14.595321 | 401 | 86 | 4 | 1 |
| 4 | 7 | 53.608570 | 165.218674 | 23.901072 | 401 | 205 | 5 | 1 |
| 5 | 7 | 59.561111 | 198.477539 | 17.439621 | 401 | 206 | 6 | 1 |
| 6 | 7 | 64.285713 | 187.353317 | 19.813021 | 401 | 207 | 7 | 1 |
| 7 | 7 | 55.756248 | 226.026016 | 9.457481 | 401 | 223 | 8 | 1 |
| 8 | 7 | 20.302856 | 16.944448 | 56.425640 | 401 | 562 | 9 | 1 |
| 9 | 7 | 15.490000 | 13.673920 | 58.706589 | 401 | 563 | 10 | 1 |
| 10 | 7 | 40.392502 | 229.608414 | 7.980424 | 401 | 797 | 11 | 1 |
| 11 | 7 | -16.356667 | 28.694563 | 50.651779 | 402 | 73 | 1 | 1 |
| 12 | 7 | -34.639999 | 34.410900 | 48.579285 | 402 | 74 | 13 | 1 |
| 13 | 7 | 67.006248 | 208.213287 | 15.070362 | 402 | 83 | 14 | 1 |
| 14 | 7 | 63.304443 | 201.135040 | 16.825037 | 402 | 84 | 15 | 1 |
| 15 | 7 | 59.571251 | 178.042679 | 21.612276 | 402 | 85 | 16 | 1 |
| 16 | 7 | 54.208889 | 197.949036 | 17.559353 | 402 | 204 | 17 | 1 |
| 17 | 7 | 58.310001 | 208.170868 | 15.081461 | 402 | 205 | 18 | 1 |
| 18 | 7 | 65.448570 | 161.844803 | 24.476290 | 402 | 220 | 19 | 1 |
| 19 | 7 | 67.006248 | 192.758163 | 18.695911 | 402 | 221 | 20 | 1 |
| 20 | 7 | 67.006248 | 192.758163 | 18.695911 | 402 | 222 | 21 | 1 |
| 21 | 7 | 65.448570 | 161.844803 | 24.476290 | 402 | 223 | 22 | 1 |
| 22 | 7 | 51.428570 | 24.898960 | 52.236923 | 402 | 559 | 10 | 1 |
| 23 | 7 | 39.375000 | 9.533246 | 62.483761 | 402 | 560 | 23 | 1 |
| 24 | 7 | 22.500000 | 7.009454 | 65.661308 | 402 | 561 | 24 | 1 |
| 25 | 7 | 54.061428 | 21.936510 | 53.631584 | 402 | 566 | 25 | 1 |
| 26 | 7 | 57.857143 | 47.196831 | 44.822647 | 402 | 624 | 26 | 1 |
| 27 | 7 | 54.061428 | 21.936510 | 53.631584 | 402 | 625 | 27 | 1 |
| 28 | 7 | 55.224285 | 28.032522 | 50.914413 | 402 | 626 | 28 | 1 |
| 29 | 7 | 39.384998 | 216.244278 | 12.808699 | 402 | 796 | 29 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1485 | 7 | 53.562729 | 169.146408 | 23.218788 | 764 | 775 | 673 | 1 |
| 1486 | 7 | 48.795715 | 196.049194 | 17.983376 | 765 | 465 | 650 | 1 |
| 1487 | 7 | 51.428570 | 188.733917 | 19.533401 | 765 | 466 | 650 | 1 |
| 1488 | 7 | 30.556250 | 25.178288 | 52.113220 | 765 | 772 | 707 | 1 |
| 1489 | 7 | 33.342999 | 78.144646 | 38.167706 | 765 | 773 | 673 | 1 |
| 1490 | 7 | 55.571667 | 201.227356 | 16.803293 | 766 | 464 | 650 | 1 |
| 1491 | 7 | 50.615002 | 181.352921 | 20.988396 | 766 | 465 | 650 | 1 |
| 1492 | 7 | 36.181252 | 62.389664 | 41.272694 | 766 | 772 | 708 | 1 |
| 1493 | 7 | 16.875000 | 22.513260 | 53.347286 | 766 | 774 | 673 | 1 |
| 1494 | 7 | 61.884998 | 170.948883 | 22.900606 | 766 | 788 | 706 | 1 |
| 1495 | 7 | 65.901428 | 197.820389 | 17.588381 | 766 | 789 | 706 | 1 |
| 1496 | 7 | 55.853748 | 169.633820 | 23.133080 | 767 | 771 | 673 | 1 |
| 1497 | 7 | 52.445713 | 145.295685 | 27.190178 | 767 | 772 | 673 | 1 |
| 1498 | 7 | 6.428571 | 14.967848 | 57.748684 | 767 | 774 | 709 | 1 |
| 1499 | 7 | 56.250000 | 65.300484 | 40.664417 | 767 | 786 | 706 | 1 |
| 1500 | 7 | 70.571663 | 205.066269 | 15.873636 | 767 | 787 | 706 | 1 |

```
1501           7  68.856667  193.809845  18.471088   767   788    706       1
1502           7  26.711250    4.257048  70.758522   768   784    710       1
1503           7  54.342499   29.935907  50.173264   768   785    711       1
1504           7  41.928333   44.569134  45.521156   768   787    712       1
1505           7  65.901428  180.261963  21.195782   776   801    713       1
1506           7  54.342499  224.353333  10.093019   777   800    714       1
1507           7  59.472858  193.389725  18.561214   778   798    715       1
1508           7  57.663750  206.651169  15.474097   778   799    716       1
1509           7  55.360001   37.344028  47.627426   779   795    717       1
1510           7  61.875000   59.868324  41.815269   779   796    718       1
1511           7  72.330002  200.441238  16.987574   779   797    719       1
1512           7  64.285713  174.467499  22.269224   779   798    720       1
1513           7  68.534286  206.469620  15.520367   780   794    713       1
1514           7  20.448572   28.540234  50.712524   780   797    713       1

[1515 rows x 8 columns]
```

```
In [67]: titi[(titi.xPos>779-3)&(titi.xPos<779+3)&(titi.yPos>799-3)&(titi.yPos<799+3)]
```

```
Out[67]:       longueur      angle      weight  precision  xPos  yPos  group  layer
         1506         7  54.342499  224.353333  10.093019   777   800    714      1
         1507         7  59.472858  193.389725  18.561214   778   798    715      1
         1508         7  57.663750  206.651169  15.474097   778   799    716      1
         1511         7  72.330002  200.441238  16.987574   779   797    719      1
         1512         7  64.285713  174.467499  22.269224   779   798    720      1
         1514         7  20.448572   28.540234  50.712524   780   797    713      1
```

```
In [68]: titi.groupby('group').agg(['mean', 'count'])
```

```
Out[68]:      longueur           angle              weight            precision        \
                 mean count       mean count       mean count       mean count
         group
         1          7     3 -16.584127     3   30.698593     3  50.291004     3
         2          7     1  68.534286     1  206.469620     1  15.520367     1
         3          7     3  45.041550     3   86.994301     3  41.027340     3
         4          7     1  57.513332     1  210.000854     1  14.595321     1
         5          7    83  50.694473    83  189.651169    83  18.722481    83
         6          7     1  59.561111     1  198.477539     1  17.439621     1
         7          7     1  64.285713     1  187.353317     1  19.813021     1
         8          7    84  49.810238    84  176.023453    84  21.717052    84
         9          7     1  20.302856     1   16.944448     1  56.425640     1
         10         7     2  33.459286     2   19.286440     2  55.471756     2
         11         7     3  55.594643     3  216.942535     3  12.240467     3
         12         7     1  68.913750     1  210.158768     1  14.552647     1
         13         7     1 -34.639999     1   34.410900     1  48.579285     1
         14         7     1  67.006248     1  208.213287     1  15.070362     1
         15         7     1  63.304443     1  201.135040     1  16.825037     1
         16         7     1  59.571251     1  178.042679     1  21.612276     1
         17         7     1  54.208889     1  197.949036     1  17.559353     1
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 18 | 7 | 1 | 58.310001 | 1 | 208.170868 | 1 | 15.081461 | 1 |
| 19 | 7 | 1 | 65.448570 | 1 | 161.844803 | 1 | 24.476290 | 1 |
| 20 | 7 | 1 | 67.006248 | 1 | 192.758163 | 1 | 18.695911 | 1 |
| 21 | 7 | 1 | 67.006248 | 1 | 192.758163 | 1 | 18.695911 | 1 |
| 22 | 7 | 1 | 65.448570 | 1 | 161.844803 | 1 | 24.476290 | 1 |
| 23 | 7 | 1 | 39.375000 | 1 | 9.533246 | 1 | 62.483761 | 1 |
| 24 | 7 | 1 | 22.500000 | 1 | 7.009454 | 1 | 65.661308 | 1 |
| 25 | 7 | 1 | 54.061428 | 1 | 21.936510 | 1 | 53.631584 | 1 |
| 26 | 7 | 1 | 57.857143 | 1 | 47.196831 | 1 | 44.822647 | 1 |
| 27 | 7 | 1 | 54.061428 | 1 | 21.936510 | 1 | 53.631584 | 1 |
| 28 | 7 | 1 | 55.224285 | 1 | 28.032522 | 1 | 50.914413 | 1 |
| 29 | 7 | 1 | 39.384998 | 1 | 216.244278 | 1 | 12.808699 | 1 |
| 30 | 7 | 1 | 27.107500 | 1 | 222.248306 | 1 | 10.851595 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 691 | 7 | 1 | -32.373333 | 1 | 222.146988 | 1 | 10.887039 | 1 |
| 692 | 7 | 1 | 33.737999 | 1 | 115.601807 | 1 | 31.871830 | 1 |
| 693 | 7 | 1 | 64.285713 | 1 | 209.610550 | 1 | 14.700283 | 1 |
| 694 | 7 | 1 | 63.721428 | 1 | 215.068512 | 1 | 13.162086 | 1 |
| 695 | 7 | 1 | -30.658333 | 1 | 229.017227 | 1 | 8.236367 | 1 |
| 696 | 7 | 1 | -2.632857 | 1 | 21.514002 | 1 | 53.844181 | 1 |
| 697 | 7 | 1 | 55.224285 | 1 | 28.032522 | 1 | 50.914413 | 1 |
| 698 | 7 | 1 | 61.652859 | 1 | 47.678165 | 1 | 44.697987 | 1 |
| 699 | 7 | 1 | 79.428337 | 1 | 222.068848 | 1 | 10.914317 | 1 |
| 700 | 7 | 1 | 68.534286 | 1 | 188.829575 | 1 | 19.513887 | 1 |
| 701 | 7 | 1 | -33.730000 | 1 | 225.964462 | 1 | 9.481425 | 1 |
| 702 | 7 | 1 | -4.428333 | 1 | 61.198547 | 1 | 41.527107 | 1 |
| 703 | 7 | 1 | 77.142860 | 1 | 202.363434 | 1 | 16.533459 | 1 |
| 704 | 7 | 1 | 21.313334 | 1 | 18.056919 | 1 | 55.742908 | 1 |
| 705 | 7 | 1 | 5.314000 | 1 | 132.956497 | 1 | 29.143555 | 1 |
| 706 | 7 | 6 | 57.189545 | 6 | 166.328964 | 6 | 23.238663 | 6 |
| 707 | 7 | 1 | 30.556250 | 1 | 25.178288 | 1 | 52.113220 | 1 |
| 708 | 7 | 1 | 36.181252 | 1 | 62.389664 | 1 | 41.272694 | 1 |
| 709 | 7 | 1 | 6.428571 | 1 | 14.967848 | 1 | 57.748684 | 1 |
| 710 | 7 | 1 | 26.711250 | 1 | 4.257048 | 1 | 70.758522 | 1 |
| 711 | 7 | 1 | 54.342499 | 1 | 29.935907 | 1 | 50.173264 | 1 |
| 712 | 7 | 1 | 41.928333 | 1 | 44.569134 | 1 | 45.521156 | 1 |
| 713 | 7 | 3 | 51.628098 | 3 | 138.423935 | 3 | 29.142891 | 3 |
| 714 | 7 | 1 | 54.342499 | 1 | 224.353333 | 1 | 10.093019 | 1 |
| 715 | 7 | 1 | 59.472858 | 1 | 193.389725 | 1 | 18.561214 | 1 |
| 716 | 7 | 1 | 57.663750 | 1 | 206.651169 | 1 | 15.474097 | 1 |
| 717 | 7 | 1 | 55.360001 | 1 | 37.344028 | 1 | 47.627426 | 1 |
| 718 | 7 | 1 | 61.875000 | 1 | 59.868324 | 1 | 41.815269 | 1 |
| 719 | 7 | 1 | 72.330002 | 1 | 200.441238 | 1 | 16.987574 | 1 |
| 720 | 7 | 1 | 64.285713 | 1 | 174.467499 | 1 | 22.269224 | 1 |

```
             xPos                yPos       layer
             mean count          mean count  mean count
group
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 401.333333 | 3 | 72.666667 | 3 | 1 | 3 |
| 2 | 403.000000 | 1 | 81.000000 | 1 | 1 | 1 |
| 3 | 403.000000 | 3 | 81.333333 | 3 | 1 | 3 |
| 4 | 401.000000 | 1 | 86.000000 | 1 | 1 | 1 |
| 5 | 423.614458 | 83 | 174.493976 | 83 | 1 | 83 |
| 6 | 401.000000 | 1 | 206.000000 | 1 | 1 | 1 |
| 7 | 401.000000 | 1 | 207.000000 | 1 | 1 | 1 |
| 8 | 427.428571 | 84 | 186.321429 | 84 | 1 | 84 |
| 9 | 401.000000 | 1 | 562.000000 | 1 | 1 | 1 |
| 10 | 401.500000 | 2 | 561.000000 | 2 | 1 | 2 |
| 11 | 403.000000 | 3 | 794.666667 | 3 | 1 | 3 |
| 12 | 403.000000 | 1 | 82.000000 | 1 | 1 | 1 |
| 13 | 402.000000 | 1 | 74.000000 | 1 | 1 | 1 |
| 14 | 402.000000 | 1 | 83.000000 | 1 | 1 | 1 |
| 15 | 402.000000 | 1 | 84.000000 | 1 | 1 | 1 |
| 16 | 402.000000 | 1 | 85.000000 | 1 | 1 | 1 |
| 17 | 402.000000 | 1 | 204.000000 | 1 | 1 | 1 |
| 18 | 402.000000 | 1 | 205.000000 | 1 | 1 | 1 |
| 19 | 402.000000 | 1 | 220.000000 | 1 | 1 | 1 |
| 20 | 402.000000 | 1 | 221.000000 | 1 | 1 | 1 |
| 21 | 402.000000 | 1 | 222.000000 | 1 | 1 | 1 |
| 22 | 402.000000 | 1 | 223.000000 | 1 | 1 | 1 |
| 23 | 402.000000 | 1 | 560.000000 | 1 | 1 | 1 |
| 24 | 402.000000 | 1 | 561.000000 | 1 | 1 | 1 |
| 25 | 402.000000 | 1 | 566.000000 | 1 | 1 | 1 |
| 26 | 402.000000 | 1 | 624.000000 | 1 | 1 | 1 |
| 27 | 402.000000 | 1 | 625.000000 | 1 | 1 | 1 |
| 28 | 402.000000 | 1 | 626.000000 | 1 | 1 | 1 |
| 29 | 402.000000 | 1 | 796.000000 | 1 | 1 | 1 |
| 30 | 402.000000 | 1 | 1230.000000 | 1 | 1 | 1 |
| ... | ... | ... | ... | ... | ... | ... |
| 691 | 755.000000 | 1 | 516.000000 | 1 | 1 | 1 |
| 692 | 755.000000 | 1 | 787.000000 | 1 | 1 | 1 |
| 693 | 759.000000 | 1 | 870.000000 | 1 | 1 | 1 |
| 694 | 759.000000 | 1 | 871.000000 | 1 | 1 | 1 |
| 695 | 760.000000 | 1 | 518.000000 | 1 | 1 | 1 |
| 696 | 760.000000 | 1 | 792.000000 | 1 | 1 | 1 |
| 697 | 760.000000 | 1 | 866.000000 | 1 | 1 | 1 |
| 698 | 760.000000 | 1 | 867.000000 | 1 | 1 | 1 |
| 699 | 760.000000 | 1 | 868.000000 | 1 | 1 | 1 |
| 700 | 760.000000 | 1 | 869.000000 | 1 | 1 | 1 |
| 701 | 761.000000 | 1 | 518.000000 | 1 | 1 | 1 |
| 702 | 761.000000 | 1 | 793.000000 | 1 | 1 | 1 |
| 703 | 761.000000 | 1 | 865.000000 | 1 | 1 | 1 |
| 704 | 761.000000 | 1 | 868.000000 | 1 | 1 | 1 |
| 705 | 762.000000 | 1 | 793.000000 | 1 | 1 | 1 |
| 706 | 766.000000 | 6 | 788.500000 | 6 | 1 | 6 |
| 707 | 765.000000 | 1 | 772.000000 | 1 | 1 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 708 | 766.000000 | 1 | 772.000000 | 1 | 1 | 1 |
| 709 | 767.000000 | 1 | 774.000000 | 1 | 1 | 1 |
| 710 | 768.000000 | 1 | 784.000000 | 1 | 1 | 1 |
| 711 | 768.000000 | 1 | 785.000000 | 1 | 1 | 1 |
| 712 | 768.000000 | 1 | 787.000000 | 1 | 1 | 1 |
| 713 | 778.666667 | 3 | 797.333333 | 3 | 1 | 3 |
| 714 | 777.000000 | 1 | 800.000000 | 1 | 1 | 1 |
| 715 | 778.000000 | 1 | 798.000000 | 1 | 1 | 1 |
| 716 | 778.000000 | 1 | 799.000000 | 1 | 1 | 1 |
| 717 | 779.000000 | 1 | 795.000000 | 1 | 1 | 1 |
| 718 | 779.000000 | 1 | 796.000000 | 1 | 1 | 1 |
| 719 | 779.000000 | 1 | 797.000000 | 1 | 1 | 1 |
| 720 | 779.000000 | 1 | 798.000000 | 1 | 1 | 1 |

[719 rows x 14 columns]

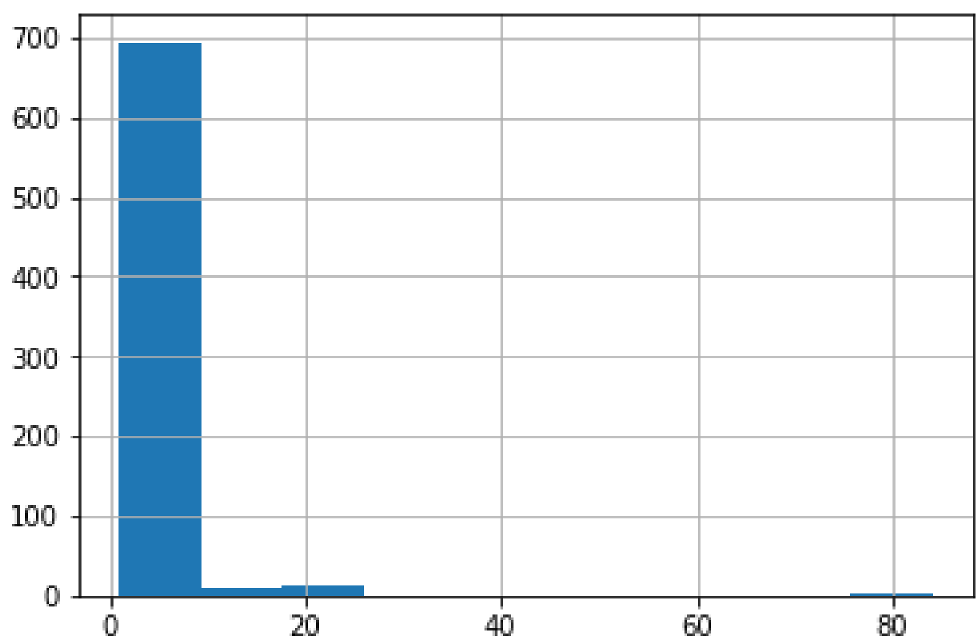
```
In [69]: closestFieldNeurons(titi, 779, 799, 3)
```

Out[69]:

| | longueur | angle | weight | precision | xPos | yPos | group | layer |
|---|---|---|---|---|---|---|---|---|
| 1506 | 7 | 54.342499 | 224.353333 | 10.093019 | 777 | 800 | 714 | 1 |
| 1507 | 7 | 59.472858 | 193.389725 | 18.561214 | 778 | 798 | 715 | 1 |
| 1508 | 7 | 57.663750 | 206.651169 | 15.474097 | 778 | 799 | 716 | 1 |
| 1511 | 7 | 72.330002 | 200.441238 | 16.987574 | 779 | 797 | 719 | 1 |
| 1512 | 7 | 64.285713 | 174.467499 | 22.269224 | 779 | 798 | 720 | 1 |
| 1514 | 7 | 20.448572 | 28.540234 | 50.712524 | 780 | 797 | 713 | 1 |

```
In [70]: titi.groupby('group').size().hist()

Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x124125cc0>
```

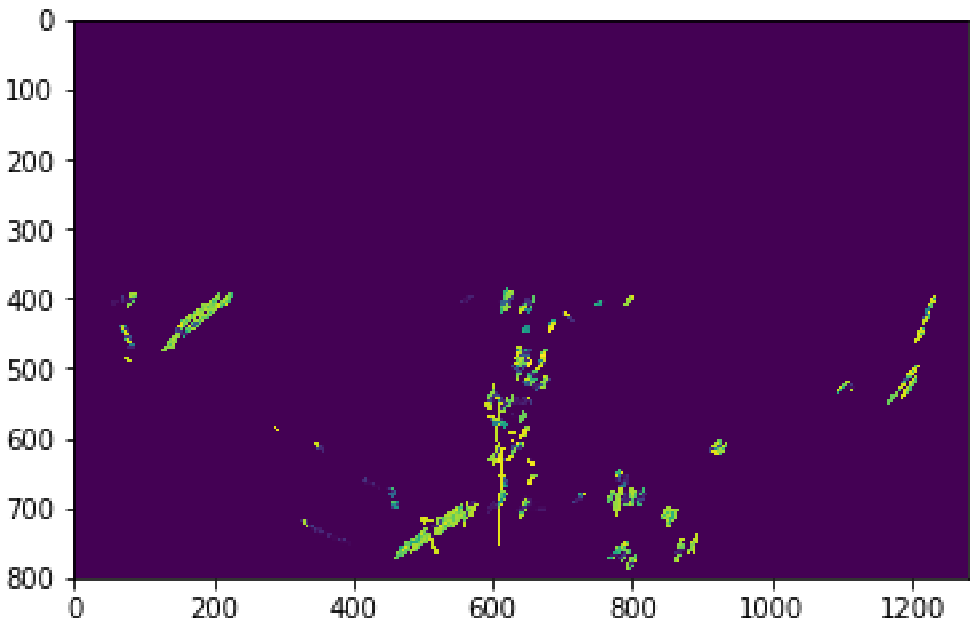


```
In [71]: resultGroup = titi.groupby('group').size()

In [72]: resultGroup[resultGroup>10]

Out[72]: group
         5      83
         8      84
         70     12
         170    17
         187    28
         204    13
         230    25
         244    18
         251    22
         254    18
         267    19
         285    19
         289    22
         380    11
         387    13
         417    19
         439    11
         545    21
         579    60
         637    22
         650    40
         651    25
         673    25
         dtype: int64

In [73]: testBitmap = np.zeros(frame.shape)
         testBitmap = drawFieldNeurons(titi,testBitmap)
         imgplot = plt.imshow(testBitmap)
```




In [78]:
```
lintI = 0
while (lintI < 10):
    cv2.imshow('Canny', testBitmap)
    if cv2.waitKey(1) & 0xFF == ord('q'):  # press q to quit
        break
    lintI += 1
```

In [76]:
```
titi.describe()
```

Out[76]:

| | longueur | angle | weight | precision | xPos | \ |
|---|---|---|---|---|---|---|
| count | 1515.0 | 1515.000000 | 1515.000000 | 1515.000000 | 1515.000000 | |
| mean | 7.0 | 29.927622 | 134.548584 | 30.581026 | 578.699010 | |
| std | 0.0 | 33.965427 | 86.172958 | 19.807312 | 123.133523 | |
| min | 7.0 | -55.677143 | 1.950733 | 0.000000 | 401.000000 | |
| 25% | 7.0 | 3.071667 | 33.061687 | 14.824060 | 457.000000 | |
| 50% | 7.0 | 42.367142 | 179.661392 | 21.309187 | 553.000000 | |
| 75% | 7.0 | 56.951166 | 209.146645 | 49.040255 | 698.000000 | |
| max | 7.0 | 90.000000 | 242.906403 | 78.653786 | 780.000000 | |

| | yPos | group | layer |
|---|---|---|---|
| count | 1515.000000 | 1515.000000 | 1515.0 |
| mean | 593.918152 | 339.048845 | 1.0 |
| std | 286.638927 | 223.527925 | 0.0 |
| min | 56.000000 | 1.000000 | 1.0 |
| 25% | 460.000000 | 164.500000 | 1.0 |
| 50% | 623.000000 | 317.000000 | 1.0 |

```
75%     723.000000  551.500000    1.0
max    1230.000000  720.000000    1.0
```

```
In [ ]:
```