# 40xvs20x_analysis

June 24, 2019

## 1 40x vs 20x Analysis

In [15]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from skimage import io
from skimage.util.shape import view_as_blocks
from scipy.spatial import cKDTree as KDTree
from sklearn.metrics import auc
import seaborn as sns
from tqdm import tqdm
```

In [2]:
```
# Levenshtein distance (https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/
def levenshtein(s1, s2):
    """ Function to compute Levenshtein distance between two strings.
        Returns the number of mismatches between the two strings.

        s1 : first string
        s2 : second string
    """
    if len(s1) < len(s2):
        return levenshtein(s2, s1)

    # len(s1) >= len(s2)
    if len(s2) == 0:
        return len(s1)

    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1 # j+1 instead of j since previous_row
            deletions = current_row[j] + 1       # than s2
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row
```

```
            return previous_row[-1]
```

In [16]:
```
def wrapCoords(a, tMatrix, offset):
    """ Apply affine tranformation to a set of pair of coordinates.
        Returns transformed set of pair of coordinates

        a : set of pairs of coords
        tMatrix : affine transformation matrix
        offset : transformation offset
    """
    return np.dot(tMatrix, a) + offset

def PointCloudReg(y0,x0,DistScale):
    """ Point cloud registration to map points x0 onto points y0 by iterative closest
        repeatedly finding the best y0 for each x0, and doing linear regression to fi

        that best maps x0 to y0.
        y0: target point cloud
        x0: source point cloud
        DistScale: any x0 whose nearest neighbor is further than this won't count
    """
    MaxIter = 10000
    Interactive = 0

    (nP,nD)=source.shape
    x=x0

    M0 = np.eye(nD)
    M = M0;
    k0 = KDTree(y0)

    # Find well isolated points
    d,idx = k0.query(y0,2)
    y = y0[d[:,1]>DistScale*2,:]

    k = KDTree(y)
    idx = np.zeros((nP,1))
    for i in range(MaxIter):
        LastNeighbor = idx
        xM = np.dot(x,M)
        d,idx = k.query(xM,distance_upper_bound=DistScale)
        idx = idx[~(d==np.inf)]
        nMatches = len(idx);
        M = np.linalg.lstsq(x[d!=np.inf],y[idx,:])[0]

        Error = np.sqrt(np.mean(np.power(d[d!=np.inf],2)));
```

```
            if np.array_equal(LastNeighbor, idx):
                break

    return {'M':M, 'Error':Error, 'nMatches':nMatches}

def runROC(exp_df,unexp_df):
    ppv = []
    tnr = []
    tpr = []
    fpr = []
    n_bins = 100
    fp_minQ = unexp_df.Q.min()
    fp_maxQ = unexp_df.Q.max()
    tp_minQ = exp_df.Q.min()
    tp_maxQ = exp_df.Q.max()
    Q_min = np.amin([fp_minQ,tp_minQ])
    Q_max = np.amax([fp_maxQ,tp_maxQ])
    step = (Q_max-Q_min)/n_bins
    Q=Q_min
    for n in range(n_bins):
        tp = len(exp_df[exp_df.Q>=Q])
        fp = len(unexp_df[unexp_df.Q>=Q])
        tn = len(unexp_df[unexp_df.Q<Q])
        fn = len(exp_df[exp_df.Q<Q])
        ppv.append(tp/(tp+fp))
        tnr.append(tn/(tn+fp))
        tpr.append(tp/(tp+fn))
        fpr.append(fp/(fp+tn))
        Q = Q + step
    return fpr, tpr, auc(fpr, tpr)

# Find best quality threshold that miximize expected vs unexpected separation
def find_d1_param(barcodes_df, tagList_df):
    r = np.linspace(2,5,20) # 20 linearly spaced numbers
    auc_list=[]
    for v in tqdm(r):
        def T_quality(x):
            return np.clip(1-np.log(1+x)/v,0,1)
        barcodes_df["Q"]=barcodes_df.seq_quality_min*barcodes_df.general_stain_min.ap

        exp_df = barcodes_df[barcodes_df.letters.isin(tagList_df.Seq)]
        unexp_df = barcodes_df[~barcodes_df.letters.isin(tagList_df.Seq)]
        unexp_tagList = unexp_df.letters.unique()
        auc_list.append(runROC(exp_df,unexp_df)[-1])

    return r[np.argmax(auc_list)]
```

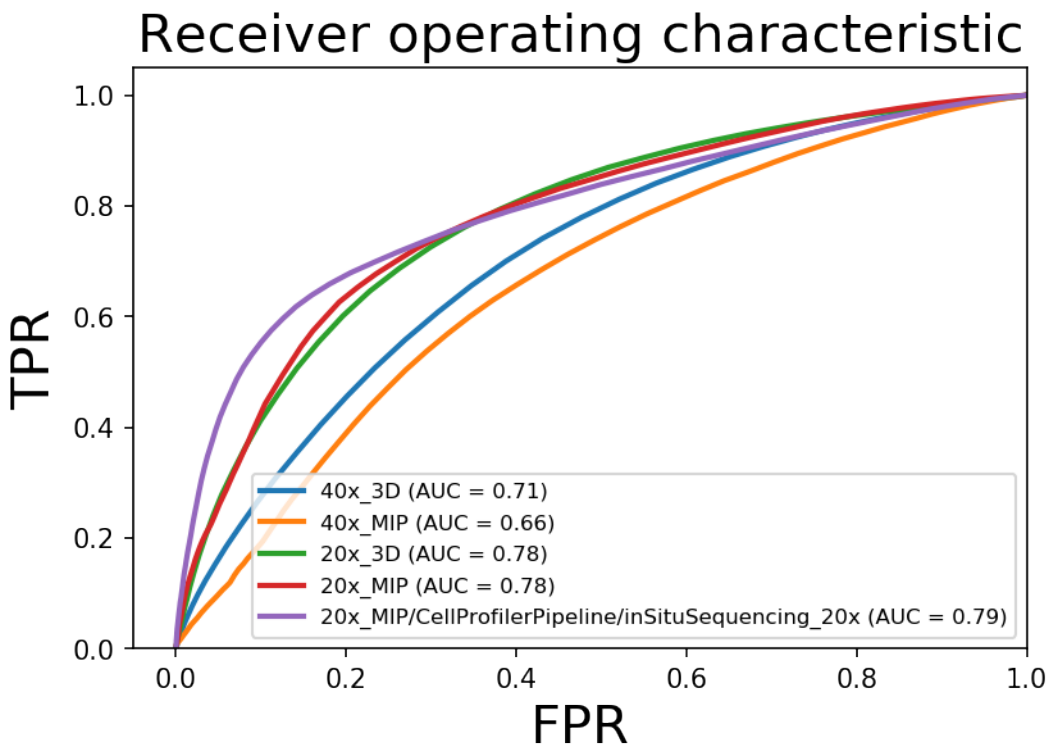
### 1.1 Unexpected vs Expected Comparison

ROC curvers and mismatch histograms are computed using as ground truth the codes in the taglist.

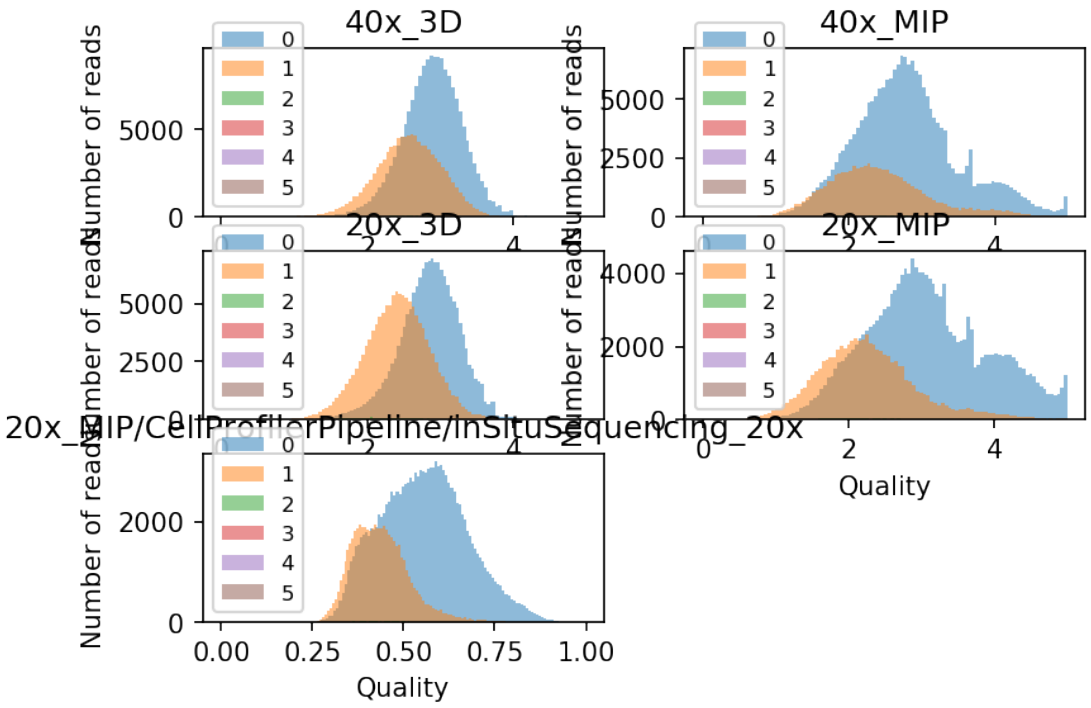
```
In [48]: tagList_df = pd.read_csv("../data/tagList_99-gene.csv", sep = ",", usecols = [0], hea

         def ROC(exp_df,unexp_df):
             ppv = []
             tnr = []
             tpr = []
             fpr = []
             n_bins = 100
             fp_minQ = unexp_df.seq_quality_min.min()
             fp_maxQ = unexp_df.seq_quality_min.max()
             tp_minQ = exp_df.seq_quality_min.min()
             tp_maxQ = exp_df.seq_quality_min.max()
             Q_min = np.amin([fp_minQ,tp_minQ])
             Q_max = np.amax([fp_maxQ,tp_maxQ])
             step = (Q_max-Q_min)/n_bins
             Q=Q_min
             for n in range(n_bins):
                 tp = len(exp_df[exp_df.seq_quality_min>=Q])
                 fp = len(unexp_df[unexp_df.seq_quality_min>=Q])
                 tn = len(unexp_df[unexp_df.seq_quality_min<Q])
                 fn = len(exp_df[exp_df.seq_quality_min<Q])
                 ppv.append(tp/(tp+fp))
                 tnr.append(tn/(tn+fp))
                 tpr.append(tp/(tp+fn))
                 fpr.append(fp/(fp+tn))
                 Q = Q + step
             return fpr, tpr, auc(fpr, tpr)

         # Plot ROC
         for i,dataset in enumerate(['40x_3D','40x_MIP','20x_3D','20x_MIP','20x_MIP/CellProfil
             barcodes_df = pd.read_csv("../data/results/170315_161220_hippo_4_1/"+dataset+"/ba

             if dataset != '20x_MIP/CellProfilerPipeline/inSituSequencing_20x':
                 d1 = find_d1_param(barcodes_df,tagList_df)
                 def T_quality(x):
                     return np.clip(1-np.log(1+x)/d1,0,1)
                 barcodes_df.seq_quality_min=barcodes_df.seq_quality_min*barcodes_df.general_s
             else:
                 DO_th=0.01
                 barcodes_df = barcodes_df.dropna()
                 barcodes_df=barcodes_df[barcodes_df.DO_intensity>=DO_th]
                 barcodes_df["Q"] = barcodes_df.apply(lambda row: np.min([row['seq_quality_1']
                 barcodes_df = barcodes_df.drop(['seq_quality_1','seq_quality_2', 'seq_quality
                 barcodes_df.columns = ['letters','global_X_pos','global_Y_pos','general_stain
```

```
        # Exclude homopolymers
        barcodes_df = barcodes_df[(barcodes_df.letters!="AAAAA") & (barcodes_df.letters!='

        exp_df = barcodes_df[barcodes_df.letters.isin(tagList_df.Seq)]
        unexp_df = barcodes_df[~barcodes_df.letters.isin(tagList_df.Seq)]

        fpr, tpr, AUC = ROC(exp_df,unexp_df)

        # plot
        lw = 2
        plt.plot(fpr, tpr,  lw=lw, label=str(dataset) + ' (AUC = %0.2f)' % AUC)
        plt.xlim([-0.05, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('FPR',fontsize=20)
        plt.ylabel('TPR', fontsize=20)
        plt.title('Receiver operating characteristic',fontsize=20)
        plt.legend(loc="lower right", fontsize=8)
```

```
100%|| 20/20 [01:03<00:00,  3.17s/it]
100%|| 20/20 [00:59<00:00,  2.96s/it]
100%|| 20/20 [01:03<00:00,  3.34s/it]
100%|| 20/20 [00:53<00:00,  2.67s/it]
```



```
In [51]: def evaluateLevenshteinDist(sequence,tagList):
             return np.min([levenshtein(sequence,x) for x in tagList])

         # Plot Histograms
         for i,dataset in enumerate(['40x_3D','40x_MIP','20x_3D','20x_MIP','20x_MIP/CellProfile
             barcodes_df = pd.read_csv("../data/results/170315_161220_hippo_4_1/"+dataset+"/ba

             if dataset != '20x_MIP/CellProfilerPipeline/inSituSequencing_20x':
                 d1 = find_d1_param(barcodes_df,tagList_df)
                 def T_quality(x):
                     return np.clip(1-np.log(1+x)/d1,0,1)
                 barcodes_df.seq_quality_min=barcodes_df.seq_quality_min*barcodes_df.general_st
             else:
                 DO_th=0.01
                 barcodes_df = barcodes_df.dropna()
                 barcodes_df=barcodes_df[barcodes_df.DO_intensity>=DO_th]
                 barcodes_df["Q"] = barcodes_df.apply(lambda row: np.min([row['seq_quality_1']
                 barcodes_df = barcodes_df.drop(['seq_quality_1','seq_quality_2', 'seq_quality_
                 barcodes_df.columns = ['letters','global_X_pos','global_Y_pos','general_stain_

             # Exclude homopolymers
             barcodes_df = barcodes_df[(barcodes_df.letters!="AAAAA") & (barcodes_df.letters!='


             def evaluateLevenshteinDist(sequence):
                 return np.min([levenshtein(sequence,x) for x in tagList_df.Seq])

             tqdm.pandas()
             barcodes_df['min_n_mismatch'] = barcodes_df.letters.progress_apply(evaluateLevensh

             plt.subplot(3,2,i+1)
             for i in range(len(barcodes_df.letters[0])+1):
                 plt.hist(np.array(barcodes_df[barcodes_df.min_n_mismatch==i].seq_quality_min)
             plt.legend(fontsize=8)
             plt.title(dataset)
             plt.xlabel("Quality")
             plt.ylabel("Number of reads")

100%|| 20/20 [01:03<00:00,  3.15s/it]
100%|| 293129/293129 [11:40<00:00, 418.45it/s]
100%|| 20/20 [00:55<00:00,  2.78s/it]
100%|| 274518/274518 [10:34<00:00, 432.56it/s]
100%|| 20/20 [00:57<00:00,  2.86s/it]
100%|| 286172/286172 [11:00<00:00, 433.24it/s]
100%|| 20/20 [00:47<00:00,  2.38s/it]
100%|| 229495/229495 [08:49<00:00, 433.52it/s]
100%|| 193269/193269 [07:27<00:00, 431.99it/s]
```




## 1.2 40x 3D Ground Truth

### 1.2.1 Align datasets

Align the coordinates of decoded reads to the 40x 3D results referenced as ground truth.

```
In [4]: # Load reference image
        img = np.amax(io.imread('../data/reg_img.tif'),axis=0)
        dataset_list = ['20x_3D','20x_MIP',"20x_MIP/CellProfilerPipeline/inSituSequencing_20x"]
```

```
/home/gapartel/miniconda3/lib/python3.7/site-packages/skimage/external/tifffile/tifffile.py:183
  warnings.warn("ome-xml: index out of range")
/home/gapartel/miniconda3/lib/python3.7/site-packages/skimage/external/tifffile/tifffile.py:257
  tile = decompress(tile)
/home/gapartel/miniconda3/lib/python3.7/site-packages/skimage/external/tifffile/tifffile.py:257
  warnings.warn("invalid tile data")
```

```
In [5]: for dataset in tqdm(dataset_list):
            print(dataset)
            barcodes_df = pd.read_csv("../data/results/170315_161220_hippo_4_1/"+dataset+"/barc
            target_df = pd.read_csv("../data/results/170315_161220_hippo_4_1/40x_3D/barcodes.cs
            tagList_df = pd.read_csv("../data/results/170315_161220_hippo_4_1/tagList_99-gene.c

            if dataset=="20x_MIP/CellProfilerPipeline/inSituSequencing_20x":
```

```
    DO_th=0.01
    barcodes_df = barcodes_df.dropna()
    barcodes_df=barcodes_df[barcodes_df.DO_intensity>=DO_th]
    barcodes_df["Q"] = barcodes_df.apply(lambda row: np.min([row['seq_quality_1'],
    barcodes_df = barcodes_df.drop(['seq_quality_1','seq_quality_2', 'seq_quality_3
    barcodes_df.columns = ['letters','global_X_pos','global_Y_pos','general_stain_m
else:
    d1 = find_d1_param(barcodes_df,tagList_df)

    def T_quality(x):
        return np.clip(1-np.log(1+x)/d1,0,1)

    barcodes_df.seq_quality_min=barcodes_df.seq_quality_min*barcodes_df.general_sta


# Remove Homopolymer
target_df = target_df[(target_df.letters!="AAAAA") & (target_df.letters!="CCCCC") &
barcodes_df = barcodes_df[(barcodes_df.letters!="AAAAA") & (barcodes_df.letters!="C

if dataset!='40x_MIP':
    # Crop exhiding part
    barcodes_df = barcodes_df[(barcodes_df.global_Y_pos < 6000)]
    tM = [1.997056969156388, -0.0049531684766723014, 0.005268331486827726, 1.997530
    tMatrix = np.array([[tM[3], tM[1]], [tM[2], tM[0]]])
    offset = np.array([[tM[5]], [tM[4]]])

    tCoords =  np.squeeze(np.array([wrapCoords(np.array([[barcodes_df.loc[x,'global
    barcodes_df["tX"] = tCoords[:,1]
    barcodes_df["tY"] = tCoords[:,0]
else:
    barcodes_df["tX"] = barcodes_df["global_X_pos"]
    barcodes_df["tY"] = barcodes_df["global_Y_pos"]
Q_th=0
source = np.squeeze(np.array([list(barcodes_df.loc[barcodes_df.seq_quality_min>Q_th
target = np.squeeze(np.array([[target_df.loc[:,'global_X_pos']],[target_df.loc[:,'g

patch_size=(1120,1376)
coord_max =np.array([12008,9960,1])
coord_min =np.array([1000,1000,1])
source = source[(source[:,0] <= coord_max[0]) & (source[:,1] <= coord_max[1]) & ((s
target = target[(target[:,0] <= coord_max[0]) & (target[:,1] <= coord_max[1]) & ((t

img_patches = view_as_blocks(img[coord_min[1]:coord_max[1],coord_min[0]:coord_max[0

for i in range(img_patches.shape[1]):
    for j in range(img_patches.shape[0]):
        barcodes_df_tmp = barcodes_df[(barcodes_df.tX>=coord_min[0]+i*patch_size[1]
        source_tmp = source[(source[:,0]>=coord_min[0]+i*patch_size[1]) & (source[
```

```
            target_tmp = target[(target[:,0]>=coord_min[0]+i*patch_size[1]) & (target[
            M=PointCloudReg(target_tmp,source_tmp,5)['M']

            tM = [M[0,0], M[0,1], M[1,0], M[1,1], M[2,0], M[2,1]]
            tMatrix = np.array([[tM[3], tM[1]], [tM[2], tM[0]]])
            offset = np.array([[tM[5]], [tM[4]]])

            tCoords =  np.squeeze(np.array([wrapCoords(np.array([[barcodes_df_tmp.loc[
            barcodes_df.loc[barcodes_df_tmp.index,"tX2"] = tCoords[:,1]
            barcodes_df.loc[barcodes_df_tmp.index,"tY2"] = tCoords[:,0]

#    plt.figure()
#    plt.imshow(img,cmap='gray',interpolation='None')
#    plt.plot(target_df['global_X_pos'],target_df['global_Y_pos'],'+',c='cyan')
#    plt.plot(barcodes_df['tX2'],barcodes_df['tY2'],'+',c='red')

    ### Barcodes agreement ###
    # Filter to common detection space
    barcodes_df = barcodes_df[(barcodes_df.tX2>=coord_min[0]) & (barcodes_df.tX2<=coord
    target_df = target_df[(target_df.global_X_pos>=coord_min[0]) & (target_df.global_X_

#    plt.figure()
#    plt.imshow(img,cmap='gray',interpolation='None')
#    plt.plot(target_df['global_X_pos'],target_df['global_Y_pos'],'+',c='cyan')
#    plt.plot(barcodes_df['tX2'],barcodes_df['tY2'],'+',c='red')

    target_df = target_df.reset_index(drop=True)
    barcodes_df = barcodes_df.reset_index(drop=True)

    d_th=3

    KDTree_df = KDTree(barcodes_df[['tX2','tY2']])
    KDTree_df_target = KDTree(target_df[['global_X_pos','global_Y_pos']])
    d, idx = KDTree_df.query(target_df[['global_X_pos','global_Y_pos']],distance_upper_

    hm_df = pd.DataFrame(data=np.zeros((len(target_df.letters.unique().tolist()+['NNNN
    for i, row in target_df.iterrows():
        i=int(i)
        if d[i]!=np.inf:
            hm_df.loc[row.letters,barcodes_df.loc[idx[i],'letters']] = hm_df.loc[row.le
            target_df.loc[i,'levenshtein_d'] = levenshtein(target_df.loc[i,'letters'],b
            target_df.loc[i,'d'] = d[i]
            target_df.loc[i,'idx'] = idx[i]
            barcodes_df.loc[idx[i],'levenshtein_d'] = levenshtein(target_df.loc[i,'let
            barcodes_df.loc[idx[i],'d'] = d[i]
            barcodes_df.loc[idx[i],'idx'] = i
        else:
            hm_df.loc[row.letters,'NNNN'] = hm_df.loc[row.letters,'NNNN'] + 1
```

```
                target_df.loc[i,'levenshtein_d'] = len(barcodes_df.letters[0]) + 1

            # Remove duplicate matches retriving only the closest
            idx = target_df[(target_df.idx.duplicated()) & (target_df.idx.notnull())].idx.uniqu
            for i in idx:
                # Based on min distance
                KDt1 = KDTree(target_df.loc[target_df.idx==i,'global_X_pos':'global_Y_pos'].val
                d,nn = KDt1.query(barcodes_df.loc[i,'tX2':'tY2'].values.reshape((1,2)))
                t1_idx = target_df.loc[target_df.idx==i,'global_X_pos':'global_Y_pos'].index.va
                t1_idx = np.delete(t1_idx, nn)
                target_df.loc[t1_idx,'levenshtein_d'] = len(barcodes_df.letters[0]) + 1

            barcodes_df.loc[barcodes_df.levenshtein_d.isnull(),'levenshtein_d'] = len(barcodes_

            # Correct one base mismatch that are expected
            barcodes_df.loc[(barcodes_df.levenshtein_d==1) & (barcodes_df.letters.isin(tagList_

            barcodes_df.to_pickle("../data/results/170315_161220_hippo_4_1/"+dataset+"/barcodes
            target_df.to_pickle("../data/results/170315_161220_hippo_4_1/"+dataset+"/target_df

  0%|          | 0/3 [00:00<?, ?it/s]

20x_3D



  0%|          | 0/20 [00:00<?, ?it/s]
  5%|         | 1/20 [00:02<00:55,  2.91s/it]
 10%|         | 2/20 [00:05<00:52,  2.89s/it]
 15%|        | 3/20 [00:08<00:50,  2.94s/it]
 20%|        | 4/20 [00:11<00:47,  2.95s/it]
 25%|       | 5/20 [00:14<00:43,  2.93s/it]
 30%|       | 6/20 [00:17<00:41,  2.99s/it]
 35%|      | 7/20 [00:20<00:38,  2.99s/it]
 40%|      | 8/20 [00:23<00:35,  2.96s/it]
 45%|     | 9/20 [00:26<00:32,  2.98s/it]
 50%|     | 10/20 [00:29<00:30,  3.01s/it]
 55%|    | 11/20 [00:32<00:27,  3.00s/it]
 60%|    | 12/20 [00:35<00:24,  3.00s/it]
 65%|   | 13/20 [00:38<00:20,  2.99s/it]
 70%|   | 14/20 [00:41<00:17,  2.96s/it]
 75%|  | 15/20 [00:44<00:14,  2.93s/it]
 80%|  | 16/20 [00:47<00:11,  2.95s/it]
 85%| | 17/20 [00:50<00:08,  2.97s/it]
 90%| | 18/20 [00:53<00:05,  2.95s/it]
 95%|| 19/20 [00:56<00:02,  2.95s/it]
100%|| 20/20 [00:59<00:00,  2.94s/it]/home/gapartel/miniconda3/lib/python3.7/site-packages/skim
  warn(RuntimeWarning("Cannot provide views on a non-contiguous input "
```

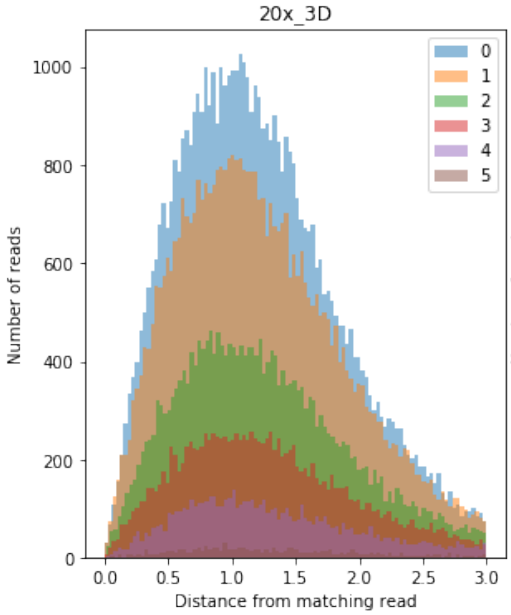
```
/home/gapartel/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:42: FutureWarning:
To use the future default and silence this warning we advise to pass `rcond=None`, to keep usi
 33%|      | 1/3 [16:51<33:42, 1011.01s/it]

20x_MIP



  0%|          | 0/20 [00:00<?, ?it/s]
  5%|         | 1/20 [00:02<00:47,  2.47s/it]
 10%|         | 2/20 [00:04<00:44,  2.46s/it]
 15%|        | 3/20 [00:07<00:41,  2.44s/it]
 20%|        | 4/20 [00:09<00:38,  2.42s/it]
 25%|       | 5/20 [00:12<00:36,  2.42s/it]
 30%|       | 6/20 [00:14<00:33,  2.40s/it]
 35%|      | 7/20 [00:16<00:31,  2.40s/it]
 40%|      | 8/20 [00:19<00:28,  2.38s/it]
 45%|     | 9/20 [00:21<00:26,  2.38s/it]
 50%|     | 10/20 [00:23<00:23,  2.38s/it]
 55%|    | 11/20 [00:26<00:21,  2.36s/it]
 60%|    | 12/20 [00:28<00:18,  2.36s/it]
 65%|   | 13/20 [00:31<00:16,  2.38s/it]
 70%|   | 14/20 [00:33<00:14,  2.37s/it]
 75%|  | 15/20 [00:35<00:11,  2.39s/it]
 80%|  | 16/20 [00:38<00:09,  2.38s/it]
 85%| | 17/20 [00:40<00:07,  2.37s/it]
 90%| | 18/20 [00:42<00:04,  2.36s/it]
 95%|| 19/20 [00:45<00:02,  2.35s/it]
 67%|   | 2/3 [31:25<16:10, 970.15s/it]

20x_MIP/CellProfilerPipeline/inSituSequencing_20x


100%|| 3/3 [45:19<00:00, 929.23s/it]
```

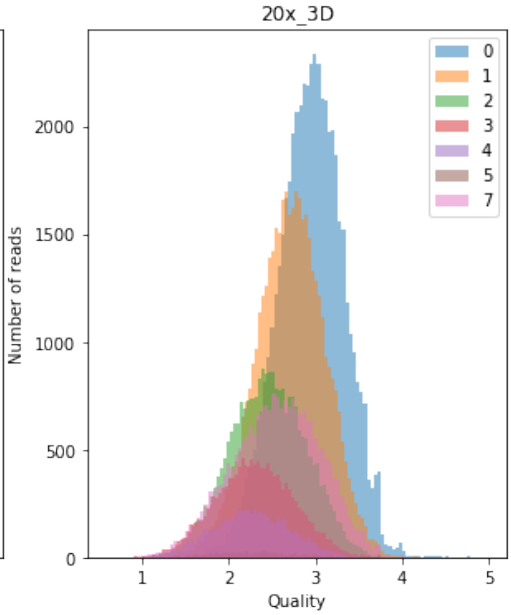
### 1.2.2 Plot Alignment Results

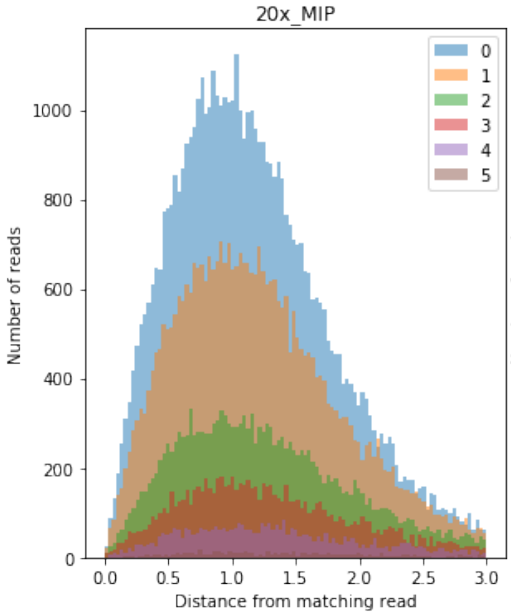
For each analysis result plot distances from matching reads and quality scores. Colors represent number of mismatches (7 stands for no match in ground truth).

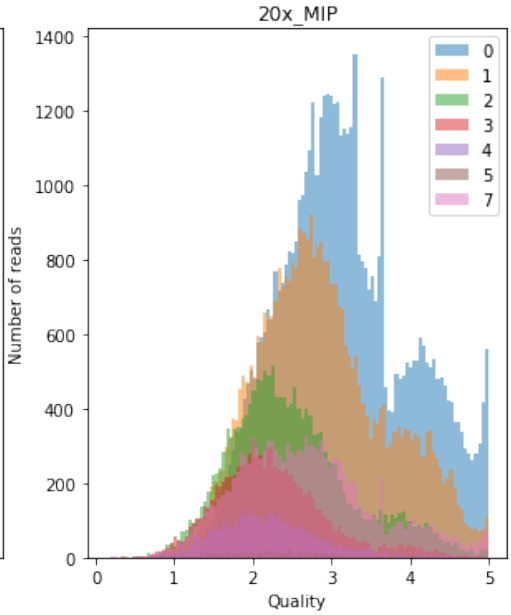
```
In [6]: plt.figure(figsize=(10,20))
        for p,df in enumerate(dataset_list):
            barcodes_df = pd.read_pickle("../data/results/170315_161220_hippo_4_1/"+df+"/barco
            target_df = pd.read_pickle("../data/results/170315_161220_hippo_4_1/"+df+"/target_

            plt.subplot(len(dataset_list),2,2*p+1)
            plt.rcParams["figure.dpi"] = 150
            for i in range(len(barcodes_df.letters[0])+1):
```

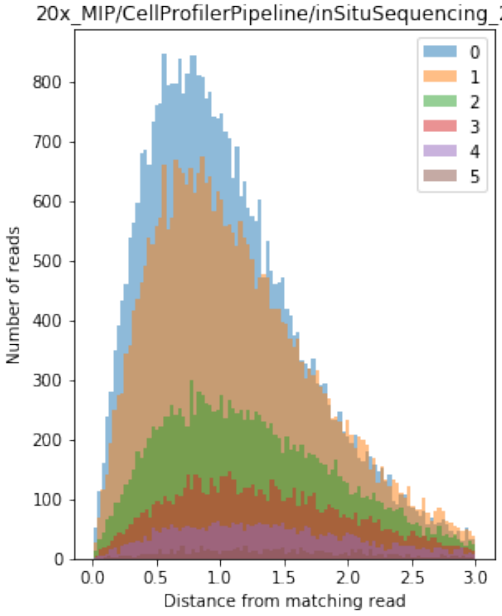
```
    plt.hist(np.array(target_df[(target_df.levenshtein_d==i) & (target_df.d.notnul
plt.legend()
plt.title(df)
plt.xlabel("Distance from matching read")
plt.ylabel("Number of reads")

plt.subplot(len(dataset_list),2,2*p+2)
plt.rcParams["figure.dpi"] = 150
for i in range(len(barcodes_df.letters[0])+1):
    plt.hist(np.array(barcodes_df.loc[target_df[target_df.levenshtein_d==i].idx].se
# Plot False Negative
plt.hist(np.array(barcodes_df[barcodes_df.levenshtein_d==7].seq_quality_min),bins=
plt.legend()
plt.title(df)
plt.xlabel("Quality")
plt.ylabel("Number of reads")
```

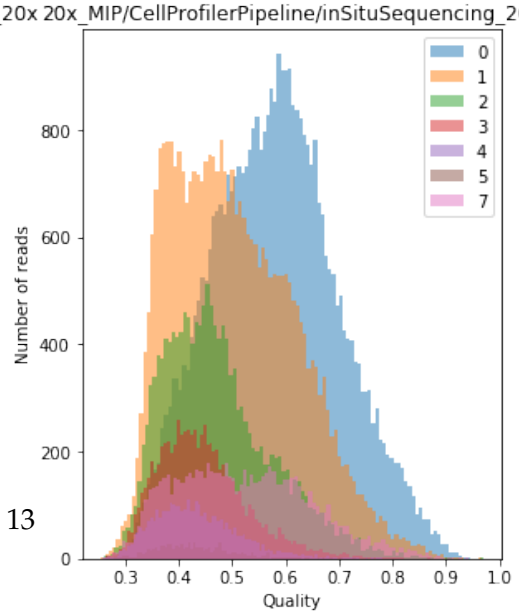

20x_3D
Number of reads
Distance from matching read
0
1
2
3
4
5


20x_3D
Number of reads
Quality
0
1
2
3
4
5
7


20x_MIP
Number of reads
Distance from matching read
0
1
2
3
4
5


20x_MIP
Number of reads
Quality
0
1
2
3
4
5
7


20x_MIP/CellProfilerPipeline/inSituSequencing_20x
Number of reads
Distance from matching read
0
1
2
3
4
5

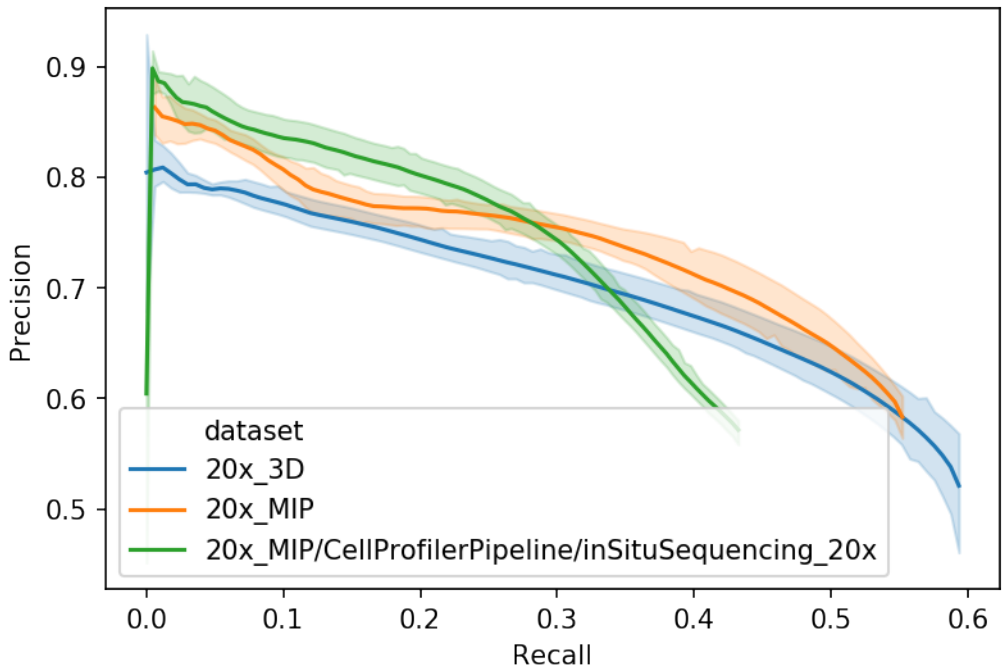

20x_MIP/CellProfilerPipeline/inSituSequencing_20x
Number of reads
Quality
0
1
2
3
4
5
7

### 1.2.3 Plot Precision VS Recall

In [10]:
```
from scipy.interpolate import interp1d

coord_max =np.array([12008,9960,1])
coord_min =np.array([1000,1000,1])

l_th=0
def PrecRecall(barcodes_df,target_df,l_th):
    exp_df = barcodes_df[barcodes_df.levenshtein_d<=l_th]
    unexp_df = barcodes_df[(barcodes_df.levenshtein_d>l_th)]

    p=[]
    r=[]
    q_th=[]
    n_bins = 100
    fp_minQ = unexp_df.seq_quality_min.min()
    fp_maxQ = unexp_df.seq_quality_min.max()
    tp_minQ = exp_df.seq_quality_min.min()
    tp_maxQ = exp_df.seq_quality_min.max()
    Q_min = np.amin([fp_minQ,tp_minQ])
    Q_max = np.amax([fp_maxQ,tp_maxQ])
    step = (Q_max-Q_min)/n_bins
    Q=Q_min
    for n in range(n_bins):
        tp = len(exp_df[(exp_df.seq_quality_min>=Q)])
        fp = len(unexp_df[unexp_df.seq_quality_min>=Q])
        tn = len(unexp_df[unexp_df.seq_quality_min<Q])
        fn = len(exp_df[exp_df.seq_quality_min<Q]) + len(target_df[(target_df.levensht
        precision = tp/(tp+fp)
        recall = tp/(tp+fn)
        p.append(precision)
        r.append(recall)
        q_th.append(Q)
        Q = Q + step
    return p,r,q_th

# Plot Precision-Recall curve
n=3
res_df = []
for df in dataset_list:
    bdf = pd.read_pickle("../data/results/170315_161220_hippo_4_1/"+df+"/barcodes_df.
    tdf = pd.read_pickle("../data/results/170315_161220_hippo_4_1/"+df+"/target_df.hd
    range_x = np.linspace(coord_min[0],coord_max[0],n).astype(np.uint)
    range_y = np.linspace(coord_min[1],coord_max[1],n).astype(np.uint)
```

```
            k=1
            res = []
            for i, c_i in enumerate(range_x[:-1]):
                for j,c_j in enumerate(range_y[:-1]):
                    bdf_tmp = bdf[(bdf.tX2>=range_x[i]) & (bdf.tX2<=range_x[i+1]) & (bdf.tY2>=
                    tdf_tmp = tdf[(tdf.global_X_pos>=range_x[i]) & (tdf.global_X_pos<=range_x
                    p,r,q_th = PrecRecall(bdf_tmp,tdf_tmp,0)

                    res.append(pd.DataFrame({'Precision':p, 'Recall':r, 'Q_th': q_th,'dataset
                    k = k+1
            res = pd.concat(res)
            r = np.linspace(max([res[res.patch==x+1].Recall.min() for x in range(k-1)]),min([r

            for i in range(k-1):
                f = interp1d(res[res.patch==i+1].Recall, res[res.patch==i+1].Precision, kind=
                res_df.append(pd.DataFrame({'Precision':f(r), 'Recall':r, 'dataset':df, 'patch
        res_df = pd.concat(res_df)

        sns.lineplot(x="Recall", y="Precision",
                     hue="dataset",
                     data=res_df)
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe8cb0bce10>




In [ ]: