If there is any question, Please also review my code on Github: https://github.com/thomasyangrenqin/Neuron-signal-process-and-machine-learning/blob/master/548%20final%20project.ipynb

In [1]:
```
import numpy as np
# load data from ReachData.npz

data=np.load('/Users/yangrenqin/GitHub/HW5/ReachData.npz')
r=data['r']
targets=data['targets']
target_index=data['cfr']
data.close()
```

In [2]:
```
targets
```

Out[2]:
```
array([[-98, -17],
       [-86,  50],
       [-64, -76],
       [-34,  93],
       [ 34,  93],
       [ 64, -76],
       [ 86,  50],
       [ 98, -17]], dtype=int16)
```

In [3]:
```
# convert x,y coordiantes to respective degreees

import math
degrees=[]
for i in targets:
    degree=math.degrees(math.atan2(i[1],i[0]))
    if degree < 0:
        degree=360+degree
    degrees.append(degree)
```

In [4]:
```
degrees
```

Out[4]:
```
[189.84113175963438,
 149.82647997035568,
 229.89909245378774,
 110.08197748418075,
 69.91802251581925,
 310.1009075462122,
 30.173520029644333,
 350.1588682403656]
```

In [5]:
```
import pandas as pd
import random
```

```
cfr=pd.Series(target_index)
training_data=np.array([])
testing_data=np.array([])
# randomly select 400 trials(50 trials for each target) as traning data, and also pick out remaini
ng data as test data
for i in range(8):
    i+=1
    cfr_i=cfr[cfr.values==i]
    t1=random.sample(range(len(cfr_i.index)),50)
    t1.sort()
    t2=[cfr_i.index[l] for l in t1]
    t3=list(set(cfr_i.index)-set(t2))
    training_data=np.append(training_data,t2)
    testing_data=np.append(testing_data,t3)
```

In [6]:

```
training_data.sort()
training_data=np.int_(training_data)
# calculate spikes in plan, move and combined window individually, and its respective time with al
l the 190 neurons.
N=[]
N_time=[]
n_plan=[]
n_plantime=[]
n_move=[]
n_movetime=[]
for i in range(len(training_data)):
    p1=r[training_data[i]].timeTouchHeld
    p2=r[training_data[i]].timeGoCue
    p3=r[training_data[i]].timeTargetAcquire
    N2,n_plan2,n_move2=np.array([]),np.array([]),np.array([])
    for l in range(190):
        if type(r[training_data[i]].unit[l].spikeTimes) == float:   # when there is only one spike
 and its spiketime
            N0=(r[training_data[i]].unit[l].spikeTimes>p1) & (r[training_data[i]].unit[l].spikeTim
es<p3)
            N1=np.sum(N0)
            n_plan0=(r[training_data[i]].unit[l].spikeTimes>p1) & (r[training_data[i]].unit[l].spi
keTimes<p2)
            n_plan1=np.sum(n_plan0)
            n_move0=(r[training_data[i]].unit[l].spikeTimes>p2) & (r[training_data[i]].unit[l].spi
keTimes<p3)
            n_move1=np.sum(n_move0)
        elif list(r[training_data[i]].unit[l].spikeTimes) == []:   # when there is no spike and it
s spiketime
```

```
            N1=0
            n_plan1=0
            n_move1=0
        else:
            N0=(r[training_data[i]].unit[l].spikeTimes>p1) & (r[training_data[i]].unit[l].spikeTim
es<p3)
            N1=np.sum(N0)
            n_plan0=(r[training_data[i]].unit[l].spikeTimes>p1) & (r[training_data[i]].unit[l].spi
keTimes<p2)
            n_plan1=np.sum(n_plan0)
            n_move0=(r[training_data[i]].unit[l].spikeTimes>p2) & (r[training_data[i]].unit[l].spi
keTimes<p3)
            n_move1=np.sum(n_move0)
        N_time1=p3-p1
        n_movetime1=p3-p2
        n_plantime1=p2-p1

        N2=np.append(N2,N1)
        n_plan2=np.append(n_plan2,n_plan1)
        n_move2=np.append(n_move2,n_move1)

    N.append(N2)
    N_time.append(N_time1)
    n_plan.append(n_plan2)
    n_plantime.append(n_plantime1)
    n_move.append(n_move2)
    n_movetime.append(n_movetime1)
```

In [ ]:

```
target0=[cfr[i] for i in training_data]
table1=pd.DataFrame(target0,index=training_data,columns=['targets']) # index represent the i th tr
ials
table1['Combined']=N
table1['Combined_time']=N_time
table1['n_plan']=n_plan
table1['n_plantime']=n_plantime
table1['n_move']=n_move
table1['n_movetime']=n_movetime
table1['combined_rate']=table1['Combined']/table1['Combined_time']
table1['plan_rate']=table1['n_plan']/table1['n_plantime']
table1['move_rate']=table1['n_move']/table1['n_movetime']
```

In [8]:

```
# Group different rates(combined, plan and move window rates) by eight targets,
# then calculate the mean and covariance matrix for each targets through different rates
# For any neuron whose averaged mean rates equals zero, delete them from dataset, and record which
```

neurons are deleted

```
combined_mean=[]
combined_cov=[]
combined_deleted_targets=[]
combined_deleted_index=[]
plan_mean=[]
plan_cov=[]
plan_deleted_targets=[]
plan_deleted_index=[]
move_mean=[]
move_cov=[]
move_deleted_targets=[]
move_deleted_index=[]
for i in range(8):
    i=i+1
    combined=np.array(list(table1[table1.targets==i]['combined_rate']))
    combined_mean1=np.mean(combined,axis=0)
    plan=np.array(list(table1[table1.targets==i]['plan_rate']))
    plan_mean1=np.mean(plan,axis=0)
    move=np.array(list(table1[table1.targets==i]['move_rate']))
    move_mean1=np.mean(move,axis=0)

    if np.any(plan_mean1==0) or np.any(move_mean1==0):
        id1=np.array(list(set(np.append(np.where(plan_mean1==0)[0],np.where(move_mean1==0)[0]))))
        combined=np.delete(combined,id1,axis=1)
        combined_mean1=np.mean(combined,axis=0)
        combined_deleted_targets.append(i)
        combined_deleted_index.append(id1)

    combined_mean.append(combined_mean1)
    combined_cov.append(np.cov(combined.T))


    if np.any(plan_mean1==0):
        id2=np.where(plan_mean1==0)[0]
        plan=np.delete(plan,id2,axis=1)
        plan_mean1=np.mean(plan,axis=0)
        plan_deleted_targets.append(i)
        plan_deleted_index.append(id2)

    plan_mean.append(plan_mean1)
    plan_cov.append(np.cov(plan.T))


    if np.any(move_mean1==0):
```

```
    if np.any(move_mean1==0):
        id3=np.where(move_mean1==0)[0]
        move=np.delete(move,id3,axis=1)
        move_mean1=np.mean(move,axis=0)
        move_deleted_targets.append(i)
        move_deleted_index.append(id3)

    move_mean.append(move_mean1)
    move_cov.append(np.cov(move.T))
```

In [9]:

```
testing_data.sort()
testing_data=np.int_(testing_data)
test_N=[]
test_N_time=[]
test_n_plan=[]
test_n_plantime=[]
test_n_move=[]
test_n_movetime=[]
# calculate spikes in plan, move and combined window individually, and its respective time with al
l the 190 neurons.
for i in range(len(testing_data)):
    p1=r[testing_data[i]].timeTouchHeld
    p2=r[testing_data[i]].timeGoCue
    p3=r[testing_data[i]].timeTargetAcquire
    test_N2,test_n_plan2,test_n_move2=np.array([]),np.array([]),np.array([])
    for l in range(190):
        if type(r[testing_data[i]].unit[l].spikeTimes) == float:
            test_N0=(r[testing_data[i]].unit[l].spikeTimes>p1) & (r[testing_data[i]].unit[l].spike
Times<p3)
            test_N1=np.sum(test_N0)
            test_n_plan0=(r[testing_data[i]].unit[l].spikeTimes>p1) &
(r[testing_data[i]].unit[l].spikeTimes<p2)
            test_n_plan1=np.sum(test_n_plan0)
            test_n_move0=(r[testing_data[i]].unit[l].spikeTimes>p2) &
(r[testing_data[i]].unit[l].spikeTimes<p3)
            test_n_move1=np.sum(test_n_move0)
        elif list(r[testing_data[i]].unit[l].spikeTimes) == []:
            test_N1=0
            test_n_plan1=0
            test_n_move1=0
        else:
            test_N0=(r[testing_data[i]].unit[l].spikeTimes>p1) & (r[testing_data[i]].unit[l].spike
Times<p3)
            test_N1=np.sum(test_N0)
            test_n_plan0=(r[testing_data[i]].unit[l].spikeTimes>p1) &
(r[testing_data[i]].unit[l].spikeTimes<p2)
```

```
(r[testing_data[i]].unit[l].spikeTimes<p2)
            test_n_plan1=np.sum(test_n_plan0)
            test_n_move0=(r[testing_data[i]].unit[l].spikeTimes>p2) &
(r[testing_data[i]].unit[l].spikeTimes<p3)
            test_n_move1=np.sum(test_n_move0)
        test_N_time1=p3-p1
        test_n_movetime1=p3-p2
        test_n_plantime1=p2-p1

        test_N2=np.append(test_N2,test_N1)
        test_n_plan2=np.append(test_n_plan2,test_n_plan1)
        test_n_move2=np.append(test_n_move2,test_n_move1)

    test_N.append(test_N2)
    test_N_time.append(test_N_time1)
    test_n_plan.append(test_n_plan2)
    test_n_plantime.append(test_n_plantime1)
    test_n_move.append(test_n_move2)
    test_n_movetime.append(test_n_movetime1)
```

In [10]:
```
test_target0=[cfr[i] for i in testing_data]
test_table1=pd.DataFrame(test_target0,index=testing_data,columns=['targets']) # index represent th
e i th trials
test_table1['Combined']=test_N
test_table1['Combined_time']=test_N_time
test_table1['n_plan']=test_n_plan
test_table1['n_plantime']=test_n_plantime
test_table1['n_move']=test_n_move
test_table1['n_movetime']=test_n_movetime
test_table1['combined_rate']=test_table1['Combined']/test_table1['Combined_time']
test_table1['plan_rate']=test_table1['n_plan']/test_table1['n_plantime']
test_table1['move_rate']=test_table1['n_move']/test_table1['n_movetime']
```

Undifferentiated rate model(combined window)

In [11]:
```
# I fited the trial-by-trial firing rates and/or PC scores using a multivariate Gaussian distribut
ion(f(r|d)),
# which has a built in function in scipy. Then decoded reach direction using maximum likelihood:
# d=argmax P(d|r), ignoring items which remain the same for every direction.
# Fianlly, we got d=argmax f(r|d)
# Please note, I also deleted the same number and poistion of neurons, which deleted in the traini
ng dataset,
# for the testing dataset.
```

```
from scipy.stats import multivariate_normal
def combined_simulate(r1):
    f=[]
    for l in range(8):
        l=l+1
        if l in combined_deleted_targets:
            r1_deleted=np.delete(r1,combined_deleted_index[combined_deleted_targets.index(l)])

            f1=multivariate_normal.logpdf(r1_deleted, mean=combined_mean[l-1], cov=np.diag(np.diag(co
mbined_cov[l-1])))
        else:
            f1=multivariate_normal.logpdf(r1, mean=combined_mean[l-1], cov=np.diag(np.diag(combine
d_cov[l-1])))
        f.append(f1)
    simulate_target=f.index(max(f))+1
    return simulate_target
```

In [12]:

```
# Make inference for each trials in the testing dataset

combined_simulate_targets=[]
for i in range(len(test_table1)):
    r1=list(test_table1['combined_rate'])[i]
    simulate_target=combined_simulate(r1)
    combined_simulate_targets.append(simulate_target)
```

In [57]:

```
# Compare inference with the acctual targets, and calulate respective absolute angular error and a
ccuracy.

orginal_degrees=[degrees[i-1] for i in test_table1['targets']]
combined_simulate_degrees=[degrees[i-1] for i in combined_simulate_targets]

combined_e=abs(np.array(orginal_degrees)-np.array(combined_simulate_degrees))
correct_combined=[i==j for i,j in zip(test_table1['targets'],combined_simulate_targets)]
combined_percent=sum(correct_combined)/len(test_table1['targets'])
combined_d=np.mean(combined_e)
combined_d_sem=np.std(combined_e)/np.sqrt(len(combined_e))
print('Mean of angular error for the Undifferent rate model is %.4f'%combined_d)
print('Sem of angular error for the Undifferent rate model is %.4f'%combined_d_sem)
print('Simulation accuracy for the Undifferent rate model is %.4f%%'%(combined_percent*100))
```

```
Mean of angular error for the Undifferent rate model is 2.8030
Sem of angular error for the Undifferent rate model is 0.8073
Simulation accuracy for the Undifferent rate model is 95.8735%
```

Only used plan window and its rate

In [14]:
```
def plan_simulate(r1):
    f=[]
    for l in range(8):
        l=l+1
        if l in plan_deleted_targets:
            r1_deleted=np.delete(r1,plan_deleted_index[plan_deleted_targets.index(l)])
            f1=multivariate_normal.logpdf(r1_deleted, mean=plan_mean[l-1], cov=np.diag(np.diag(pla
n_cov[l-1])))
        else:
            f1=multivariate_normal.logpdf(r1, mean=plan_mean[l-1], cov=np.diag(np.diag(plan_cov[l-1])))

        f.append(f1)
    simulate_target=f.index(max(f))+1
    return simulate_target
```

In [15]:
```
plan_simulate_targets=[]
for i in range(len(test_table1)):
    r1=list(test_table1['plan_rate'])[i]
    simulate_target=plan_simulate(r1)
    plan_simulate_targets.append(simulate_target)
```

In [62]:
```
plan_simulate_degrees=[degrees[i-1] for i in plan_simulate_targets]

plan_e=abs(np.array(orginal_degrees)-np.array(plan_simulate_degrees))
correct_plan=[i==j for i,j in zip(test_table1['targets'],plan_simulate_targets)]
plan_percent=sum(correct_plan)/len(test_table1['targets'])
plan_d=np.mean(plan_e)
plan_d_sem=np.std(plan_e)/np.sqrt(len(plan_e))
print('Mean of angular error for the Plan rate model is %.4f'%plan_d)
print('Sem of angular error for the Plan rate model is %.4f'%plan_d_sem)
print('Simulation accuracy for the Plan rate model is %.4f%%'%(plan_percent*100))
```

```
Mean of angular error for the Plan rate model is 9.6189
Sem of angular error for the Plan rate model is 1.4500
Simulation accuracy for the Plan rate model is 86.1073%
```

Only used move window and its rate

```
In [17]: def move_simulate(r1):
             f=[]
             for l in range(8):
                 l=l+1
                 if l in move_deleted_targets:
                     r1_deleted=np.delete(r1,move_deleted_index[move_deleted_targets.index(l)])
                     f1=multivariate_normal.logpdf(r1_deleted, mean=move_mean[l-1], cov=np.diag(np.diag(move_cov[l-1])))
                 else:
                     f1=multivariate_normal.logpdf(r1, mean=move_mean[l-1], cov=np.diag(np.diag(move_cov[l-1])))

                 f.append(f1)
             simulate_target=f.index(max(f))+1
             return simulate_target
```

```
In [18]: move_simulate_targets=[]
         for i in range(len(test_table1)):
             r1=list(test_table1['move_rate'])[i]
             simulate_target=move_simulate(r1)
             move_simulate_targets.append(simulate_target)
```

```
In [63]: move_simulate_degrees=[degrees[i-1] for i in move_simulate_targets]

         move_e=abs(np.array(orginal_degrees)-np.array(move_simulate_degrees))
         correct_move=[i==j for i,j in zip(test_table1['targets'],move_simulate_targets)]
         move_percent=sum(correct_move)/len(test_table1['targets'])
         move_d=np.mean(move_e)
         move_d_sem=np.std(move_e)/np.sqrt(len(move_e))
         print('Mean of angular error for the Move rate model is %.4f'%move_d)
         print('Sem of angular error for the Move rate model is %.4f'%move_d_sem)
         print('Simulation accuracy for the Move rate model is %.4f%%'%(move_percent*100))
```

```
Mean of angular error for the Move rate model is 4.2354
Sem of angular error for the Move rate model is 1.0595
Simulation accuracy for the Move rate model is 94.9106%
```

Plan rate/Move rate model

```
In [20]: def P_M_rate_simulate(r1):
             f=[]
             for l in range(8):
                 l=l+1
                 if l in (plan_deleted_targets) or l in (move_deleted_targets):
```

```
        if l in (plan_deleted_targets) or l in (move_deleted_targets):
            r1_deleted=r1
            if l in plan_deleted_targets:
                r1_deleted1=np.delete(r1_deleted[:190],plan_deleted_index[plan_deleted_targets.ind
ex(l)])
            if l in move_deleted_targets:
                r1_deleted2=np.delete(r1_deleted[190:],move_deleted_index[move_deleted_targets.ind
ex(l)])
            r1_deleted=np.append(r1_deleted1,r1_deleted2)
            f1=multivariate_normal.logpdf(r1_deleted, \
                                          mean=np.append(plan_mean[l-1],move_mean[l-1]),\
                                          cov=np.diag(np.append(np.diag(plan_cov[l-1]),np.diag(move_c
ov[l-1]))))
        else:
            f1=multivariate_normal.logpdf(r1, \
                                          mean=np.append(plan_mean[l-1],move_mean[l-1]),\
                                          cov=np.diag(np.append(np.diag(plan_cov[l-1]),np.diag(move_c
ov[l-1]))))
        f.append(f1)
    simulate_target=f.index(max(f))+1
    return simulate_target
```

In [21]:

```
PMrate_simulate_targets=[]
for i in range(len(test_table1)):
    r1=np.append(list(test_table1['plan_rate'])[i],list(test_table1['move_rate'])[i])
    simulate_target=P_M_rate_simulate(r1)
    PMrate_simulate_targets.append(simulate_target)
```

In [64]:

```
PMrate_simulate_degrees=[degrees[i-1] for i in PMrate_simulate_targets]

PMrate_e=abs(np.array(orginal_degrees)-np.array(PMrate_simulate_degrees))
correct_PMrate=[i==j for i,j in zip(test_table1['targets'],PMrate_simulate_targets)]
PMrate_percent=sum(correct_PMrate)/len(test_table1['targets'])
PMrate_d=np.mean(PMrate_e)
PMrate_d_sem=np.std(PMrate_e)/np.sqrt(len(PMrate_e))
print('Mean of angular error for the Plan rate/Move rate model is %.4f'%PMrate_d)
print('Sem of angular error for the Plan rate/Move rate model is %.4f'%PMrate_d_sem)
print('Simulation accuracy for the Plan rate/Move rate model is %.4f%%'%(PMrate_percent*100))
```

```
Mean of angular error for the Plan rate/Move rate model is 2.3637
Sem of angular error for the Plan rate/Move rate model is 0.7942
Simulation accuracy for the Plan rate/Move rate model is 96.9739%
```

PC score

PC score

In [23]:
```
def pc_projection(X):
    mu = np.mean(X,axis=0) # calculate mean
    w,v = np.linalg.eig(np.cov(X.T)) # calculate eigenvalues of covariance matrix
    scores = np.dot((X - mu),v[:,0]) # project into lower dimensional space
    return scores
```

In [179]:
```
# For each neuron of a trial, used 5 ms bins to convert SpikeTimes array to impulse-like array wh
ich have same time series.
# Then used Gaussian kernel(50 ms length) to convolve this impulse-like spike train for each neur
on.
# Finally, performed PCA, and take the first PC score of each trial as the PC score for the tria
l.

from scipy import ndimage
plan_pc=[]
move_pc=[]
for i in range(len(training_data)):
    plan_pc1=[]
    move_pc1=[]
    p1=r[training_data[i]].timeTouchHeld
    p2=r[training_data[i]].timeGoCue
    p3=r[training_data[i]].timeTargetAcquire
    plan_series=np.linspace(p1,p2,5+1)
    move_series=np.linspace(p2,p3,5+1)

    for l in range(190):
        plan_bin=np.zeros(len(plan_series))
        move_bin=np.zeros(len(move_series))
        if type(r[training_data[i]].unit[l].spikeTimes) == float:

            if (r[training_data[i]].unit[l].spikeTimes>=p1) & (r[training_data[i]].unit[l].spikeT
imes<p2):
                id_plan=math.floor((r[training_data[i]].unit[l].spikeTimes-p1)/((p2-p1)/5))
                plan_bin[id_plan] += 1
            if (r[training_data[i]].unit[l].spikeTimes>=p2) & (r[training_data[i]].unit[l].spikeT
imes<p3):
                id_move=math.floor((r[training_data[i]].unit[l].spikeTimes-p2)/((p3-p2)/5))
                move_bin[id_move] += 1
        elif list(r[training_data[i]].unit[l].spikeTimes) == []:
            pass
        else:
            for m in r[training_data[i]].unit[l].spikeTimes:
                if (m>=p1) & (m<p2):
```

```
                id_plan=math.floor((m-p1)/((p2-p1)/5))
                plan_bin[id_plan] += 1
            if (m>=p2) & (m<p3):
                id_move=math.floor((m-p2)/((p3-p2)/5))
                move_bin[id_move] += 1
        plan_bin=plan_bin/((p2-p1)/5)
        move_bin=move_bin/((p3-p2)/5)
        plan_convolve=ndimage.filters.gaussian_filter(plan_bin,sigma=5,truncate=5)
        move_convolve=ndimage.filters.gaussian_filter(move_bin,sigma=5,truncate=5)
        plan_pc1.append(plan_convolve)
        move_pc1.append(move_convolve)
    plan_pc1=np.array(plan_pc1)
    move_pc1=np.array(move_pc1)
    plan_pcscore=abs(pc_projection(plan_pc1))
    move_pcscore=abs(pc_projection(move_pc1))
    plan_pc.append(plan_pcscore)
    move_pc.append(move_pcscore)
```

In [180]:

```
target0=[cfr[i] for i in training_data]
table_pc=pd.DataFrame(target0,index=training_data,columns=['targets']) # index represent the i th
 trials
table_pc['plan_pc']=plan_pc
table_pc['move_pc']=move_pc
table_pc
```

Out[180]:

	targets	plan_pc	move_pc
0	4	[0.00780183226331, 0.00235097068938, 0.0118659...	[0.0345652428083, 0.00316093899488, 0.02278946...
1	1	[0.0140786271749, 0.00392326467722, 0.01611027...	[0.0318171178605, 0.0121902862355, 0.027891921...
2	7	[0.0147733746052, 0.0107104636672, 0.012742354...	[0.0353218607303, 0.00199556504063, 0.03115619...
3	5	[0.0169975342163, 0.0012868596455, 0.012935537...	[0.034298602945, 0.0037757346514, 0.0304827717...
5	8	[0.0161205697113, 0.000126359738087, 0.0161205...	[0.0318616886008, 0.00112828242018, 0.02773718...
6	8	[0.0185557632731, 0.00233352728826, 0.01855576...	[0.0319825808298, 0.00334796233485, 0.03198258...
7	4	[0.0139592656084, 0.00584832812143, 0.00855244...	[0.0340899746019, 0.00516508820432, 0.02231256...
9	1	[0.00427605266715, 0.000213961522763, 0.012400...	[0.0267233115373, 0.00415338824951, 0.02329220...
10	1	[0.00709921660945, 0.00439772151994, 0.0179152...	[0.0326771728505, 0.00774319787779, 0.02055030...
12	7	[0.0169905178243, 0.0115829873049, 0.011583415...	[0.0341960769406, 0.00379401200813, 0.03419607...
21	3	[0.0180557509829, 0.00586662132892, 0.01602453...	[0.0325581569264, 0.0286704084898, 0.032558156...

22	8	[0.0165699575462, 0.000321945222504, 0.0165699...	[0.0361733589708, 0.000282881272102, 0.0361733...
23	6	[0.0167913069077, 0.00867894446176, 0.01679130...	[0.0349650200184, 0.0253433604207, 0.034965020...
24	8	[0.0164069388602, 0.00559307193811, 0.01640693...	[0.032833819817, 0.00531197919712, 0.028247006...
25	7	[0.0186760141312, 0.0044586560504, 0.014612936...	[0.0374347414615, 0.00792632336242, 0.02836255...
26	8	[0.0146989610574, 0.00454361456692, 0.01469896...	[0.0333718832968, 0.000649213080749, 0.0333718...
28	6	[0.0166913948571, 0.00493612510506, 0.01939605...	[0.0333661175456, 0.0167029387785, 0.029515594...
33	7	[0.0137603928095, 0.00565157871951, 0.01105657...	[0.0333569015802, 0.00957578561013, 0.02543014...
35	2	[0.00902103615927, 0.00361482930683, 0.0090219...	[0.0354004819398, 0.021189156753, 0.0232725445...
39	2	[0.0152016421872, 0.0111388212266, 0.015201642...	[0.0334022826311, 0.00865874740142, 0.03340228...
41	5	[0.0144642366412, 0.000246857679899, 0.0063380...	[0.0332849773049, 0.018164221848, 0.0295054804...
42	5	[0.0168342424681, 0.00872319418245, 0.01142637...	[0.0323317518744, 0.00151990101237, 0.02462786...
49	1	[0.0146775655129, 0.000459300708721, 0.0146775...	[0.0308971470795, 0.00312354365145, 0.02333702...
53	8	[0.0163237859675, 0.00616787111786, 0.01632378...	[0.0330988881826, 0.0200701550953, 0.024412640...
56	3	[0.0169194707508, 0.0115129768431, 0.014216264...	[0.0331459244309, 0.00119567142489, 0.02959650...
59	2	[0.0134056889371, 0.00529287552965, 0.01340568...	[0.0360791322327, 0.0100709226666, 0.036079132...
61	2	[0.0111494200271, 0.000994138063371, 0.0030278...	[0.0298002939149, 0.000722997627215, 0.0221695...
63	4	[0.0105018155122, 0.00238929610467, 0.00239089...	[0.033831538493, 0.0030210138156, 0.0261287481...
67	7	[0.0166020278027, 0.0118368808826, 0.014571239...	[0.033894604268, 0.00212852107385, 0.025889631...
74	3	[0.015823760283, 0.00771116754312, 0.013120418...	[0.0331796046225, 8.06267762815e-05, 0.0184714...
...	...	...	...
1058	7	[0.0181012071598, 0.00623465801304, 0.00728611...	[0.0311561569772, 0.00558490328857, 0.02299108...
1062	3	[0.0158216721048, 0.00160252323621, 0.00769784...	[0.0299967515055, 0.0138291231318, 0.005743770...
1065	4	[0.0113986438562, 0.00482323973116, 0.00869482...	[0.036547575932, 0.00983970020041, 0.032732179...
1067	5	[0.0132339927862, 0.00782779199243, 0.01053019...	[0.0327871937082, 0.00123355908907, 0.02522698...
1069	1	[0.0167726067199, 0.00661796315942, 0.01474183...	[0.0316135806202, 0.0154454227709, 0.027570938...
1071	8	[0.0155543983843, 0.000695063427186, 0.0053992...	[0.02893270661, 0.0208485836556, 0.02489053195...

1072	8	[0.0129494703388, 0.00483784683036, 0.01024643...	[0.0303229346994, 0.00532774428564, 0.03032293...
1074	3	[0.0139884147737, 0.00858154485627, 0.00046979...	[0.0314097282399, 0.0239870320944, 0.016564304...
1076	2	[0.01139578037, 0.0073328547647, 0.00936431756...	[0.0303691604722, 0.0243069396111, 0.030369160...
1080	6	[0.0165498921354, 0.00573218545328, 0.01654989...	[0.0281064759694, 0.00900703699658, 0.02398291...
1081	2	[0.00761236784061, 0.00761300261329, 0.0103167...	[0.030100764929, 0.0190668719428, 0.0301007649...
1086	1	[0.0119959035445, 0.00658856253357, 0.00658883...	[0.0281892333534, 0.0167557280398, 0.024443532...
1089	2	[0.0125180189657, 0.00439422723434, 0.00373088...	[0.0313334022684, 0.00358870362748, 0.02340629...
1090	6	[0.0145068919295, 0.00174412468339, 0.01044522...	[0.030036436012, 0.021529454058, 0.01284779235...
1093	7	[0.0174895899054, 0.00530237981036, 0.01748958...	[0.0294735541909, 0.00636502589282, 0.02947355...
1094	7	[0.0144147601307, 0.00450974131307, 0.01711905...	[0.0321686092578, 0.00861527985949, 0.02824375...
1096	8	[0.0113291361013, 0.00321347043685, 0.01403152...	[0.0294898668924, 0.0128271871661, 0.029489866...
1097	5	[0.016265082224, 4.13383227533e-05, 0.01626508...	[0.0307980293367, 0.0013738610022, 0.027119517...
1099	5	[0.0128495801461, 0.000662867998923, 0.0108180...	[0.0312180563494, 0.000186919558359, 0.0312180...
1100	1	[0.0130528344515, 0.000865475292586, 0.0089907...	[0.0316013134692, 0.0013876098795 5, 0.02747831...
1104	4	[0.0111992948655, 0.00232075715479, 0.01119929...	[0.0352740424253, 0.0088134598825, 0.027713940...
1105	4	[0.0106152468802, 0.00452281067829, 0.00452316...	[0.0323292041393, 0.00512345785202, 0.02858402...
1106	2	[0.0130494014338, 0.00223454799471, 0.00223670...	[0.030101609307, 0.0110244086313, 0.0224703157...
1113	3	[0.0171608678688, 0.00634533730821, 0.01986531...	[0.0345415104208, 0.0169835535058, 0.034541510...
1115	1	[0.00534384130958, 0.00737714433684, 0.0053484...	[0.0326269007491, 0.00701146262226, 0.03262690...
1116	4	[0.0186352807881, 0.0105078882647, 0.000352771...	[0.0329207258919, 0.000789527809643, 0.0291744...
1117	5	[0.0167945332597, 0.00866869099555, 0.01679453...	[0.0321004196594, 0.0156066473496, 0.032100419...
1122	8	[0.0159929455231, 0.000257121018402, 0.0139617...	[0.0314449414988, 0.0212336536547, 0.031444941...
1123	3	[0.016698339766, 0.00654201330075, 0.012637791...	[0.0322303658834, 0.00365268255379, 0.03223036...
1125	8	[0.0154153746898, 0.007289141127, 0.0154153746...	[0.0278909863649, 0.000126799347702, 0.0278909...

400 rows × 3 columns

```
In [1811]: plan_pc_mean=[]
```

```
plan_pc_cov=[]
plan_pc_deleted_targets=[]
plan_pc_deleted_index=[]
move_pc_mean=[]
move_pc_cov=[]
move_pc_deleted_targets=[]
move_pc_deleted_index=[]
for i in range(8):
    i=i+1
    plan_pc=np.array(list(table_pc[table_pc.targets==i]['plan_pc']))
    plan_pc_mean1=np.mean(plan_pc,axis=0)
    if np.any(plan_pc_mean1==0):
        id2=np.where(plan_pc_mean1==0)[0]
        plan_pc=np.delete(plan_pc,id2,axis=1)
        plan_pc_mean1=np.mean(plan_pc,axis=0)
        plan_pc_deleted_targets.append(i)
        plan_pc_deleted_index.append(id2)

    plan_pc_mean.append(plan_pc_mean1)
    plan_pc_cov.append(np.cov(plan_pc.T))

    move_pc=np.array(list(table_pc[table_pc.targets==i]['move_pc']))
    move_pc_mean1=np.mean(move_pc,axis=0)
    if np.any(move_pc_mean1==0):
        id3=np.where(move_pc_mean1==0)[0]
        move_pc=np.delete(move_pc,id3,axis=1)
        move_pc_mean1=np.mean(move_pc,axis=0)
        move_pc_deleted_targets.append(i)
        move_pc_deleted_index.append(id3)

    move_pc_mean.append(move_pc_mean1)
    move_pc_cov.append(np.cov(move_pc.T))
```

In [182]:

```
test_plan_pc=[]
test_move_pc=[]
for i in range(len(testing_data)):
    test_plan_pc1=[]
    test_move_pc1=[]
    p1=r[testing_data[i]].timeTouchHeld
    p2=r[testing_data[i]].timeGoCue
    p3=r[testing_data[i]].timeTargetAcquire
    test_plan_series=np.linspace(p1,p2,5+1)
    test_move_series=np.linspace(p2,p3,5+1)

    for l in range(190):
```

```
    for l in range(190):
        test_plan_bin=np.zeros(len(test_plan_series))
        test_move_bin=np.zeros(len(test_move_series))
        if type(r[testing_data[i]].unit[l].spikeTimes) == float:   # when there is only one spike and its spiketime

            if (r[testing_data[i]].unit[l].spikeTimes>=p1) & (r[testing_data[i]].unit[l].spikeTimes<p2):
                test_id_plan=math.floor((r[testing_data[i]].unit[l].spikeTimes-p1)/((p2-p1)/5))
                test_plan_bin[test_id_plan] += 1
            if (r[testing_data[i]].unit[l].spikeTimes>=p2) & (r[testing_data[i]].unit[l].spikeTimes<p3):
                test_id_move=math.floor((r[testing_data[i]].unit[l].spikeTimes-p2)/((p3-p2)/5))
                test_move_bin[test_id_move] += 1
        elif list(r[testing_data[i]].unit[l].spikeTimes) == []:   # when there is no spike and its spiketime
            pass
        else:
            for m in r[testing_data[i]].unit[l].spikeTimes:
                if (m>=p1) & (m<p2):
                    test_id_plan=math.floor((m-p1)/((p2-p1)/5))
                    test_plan_bin[test_id_plan] += 1
                if (m>=p2) & (m<p3):
                    test_id_move=math.floor((m-p2)/((p3-p2)/5))
                    test_move_bin[test_id_move] += 1
        test_plan_bin=test_plan_bin/((p2-p1)/5)
        test_move_bin=test_move_bin/((p3-p2)/5)
        test_plan_convolve=ndimage.filters.gaussian_filter(test_plan_bin,sigma=5,truncate=5)
        test_move_convolve=ndimage.filters.gaussian_filter(test_move_bin,sigma=5,truncate=5)
        test_plan_pc1.append(test_plan_convolve)
        test_move_pc1.append(test_move_convolve)
    test_plan_pc1=np.array(test_plan_pc1)
    test_move_pc1=np.array(test_move_pc1)
    test_plan_pc.append(abs(pc_projection(test_plan_pc1)))
    test_move_pc.append(abs(pc_projection(test_move_pc1)))
```

In [183]:

```
target0=[cfr[i] for i in testing_data]
test_table_pc=pd.DataFrame(target0,index=testing_data,columns=['targets']) # index represent the i th trials
test_table_pc['plan_pc']=test_plan_pc
test_table_pc['move_pc']=test_move_pc
test_table_pc
```

Out[183]:

	targets	plan_pc	move_pc
4	6	[0.017147726904, 0.00903687756324, 0.017147726	[0.032642804433, 0.0161487625344, 0.0326428044

4	6	[0.0171477269 04, 0.0090368773 6324, 0.017147726...	[0.032642804433, 0.0161487623344, 0.0326428044...
8	7	[0.013646726047, 0.00553537094576, 0.013646726...	[0.0302320067652, 0.0220672734308, 0.030232006...
11	5	[0.00752552583827, 0.00482655805428, 0.0156390...	[0.0379101010314, 0.0011396706453, 0.037910101...
13	2	[0.0164096088782, 0.00625357562421, 0.00828555...	[0.0316726658347, 0.0193575594326, 0.023820850...
14	3	[0.0178441417487, 0.0151401037745, 0.015140706...	[0.0303045373013, 0.0162277248911, 0.030304537...
15	8	[0.0164214623224, 0.00830893132834, 0.01642146...	[0.0301256606643, 0.01196123235, 0.02591675185...
16	7	[0.0176281981197, 0.00877653470271, 0.01153432...	[0.0324474030606, 0.0077046817752, 0.028323828...
17	4	[0.0112554091268, 0.00766774815594, 0.00855304...	[0.0326677374188, 0.0034606145571, 0.025442328...
18	2	[0.00199569867915, 0.00469472092316, 0.0088239...	[0.0321282744429, 0.00491172643707, 0.02435219...
19	5	[0.00911702113889, 0.00307133180231, 0.0050583...	[0.0362673690574, 0.00327699931867, 0.03214421...
20	4	[0.010101571839, 0.00400924415712, 0.008070931...	[0.0327391522101, 0.00665394311259, 0.02915818...
27	3	[0.0158952795592, 0.0104871497121, 0.015895279...	[0.0330278393904, 0.00485106291535, 0.02881860...
29	1	[0.0144148428072, 0.0117110609252, 0.014414842...	[0.031672741413, 0.00812124627434, 0.027747114...
30	1	[0.0132343881218, 0.00714022776379, 0.01323438...	[0.0369530264291, 0.0196356957399, 0.032910876...
31	4	[0.00843834172012, 0.00507927812524, 0.0111427...	[0.0350883917408, 0.0119797752558, 0.035088391...
32	5	[0.0171333182699, 0.0090235925268, 0.017133318...	[0.0383754234265, 0.01877960368, 0.03837542342...
34	3	[0.0187825885929, 0.0106574304372, 0.016751692...	[0.0351927285191, 0.00348579697095, 0.03519272...
36	4	[0.00525815035042, 0.00119680970374, 0.0093219...	[0.0357092768745, 0.0123809390816, 0.027933164...
37	5	[0.0124962896622, 0.00640268010235, 0.00437384...	[0.0339748414059, 0.00972106267432, 0.02993198...
38	6	[0.0181091048974, 0.00186013871777, 0.01607816...	[0.0334778588208, 0.00603107668688, 0.02908765...
40	8	[0.0139597342548, 0.00767100040064, 0.01395973...	[0.0316878066007, 0.00954836335208, 0.03168780...
43	7	[0.0181294650237, 0.00190834583025, 0.01812946...	[0.0335034292014, 0.0108241024726, 0.033503429...
44	2	[0.0108071759067, 0.002682206, 0.0067469062439...	[0.0305235297724, 0.0152615073067, 0.030523529...
45	6	[0.0152873880982, 0.0112251569126, 0.015287388...	[0.0341326570834, 0.029656097904, 0.0341326570...
46	3	[0.00849277387309, 0.0139021010564, 0.00579158...	[0.0293715914172, 0.00612767285915, 0.02227035...
47	2	[0.00879331347763, 0.00338848571016, 0.0006836...	[0.0318817427128, 0.0269647714733, 0.024525716...
48	7	[0.0160461704754, 0.000202156139334, 0.0160461...	[0.0342670881632, 0.00875214818467, 0.03426708...

50	6	[0.018228610989, 0.0101173766862, 0.0101173766...	[0.0313973346724, 0.0435876321737, 0.023065610...
51	4	[0.0095200901182, 0.00140784792239, 0.00670372...	[0.0357316885539, 0.00171105635035, 0.02061142...
52	4	[0.010743174078, 0.00753714978716, 0.002622940...	[0.0328269863212, 0.0125339300235, 0.025266829...
...	...	...	...
1075	6	[0.0126575152684, 0.00656238593345, 0.00656468...	[0.0301042741628, 0.0171682977557, 0.030104274...
1077	7	[0.0145433921061, 0.00372647136823, 0.01454339...	[0.0298230710474, 0.0147013954644, 0.029823071...
1078	5	[0.0132912803333, 0.00247909969768, 0.00788601...	[0.0342457154884, 0.00173061999442, 0.03063283...
1079	7	[0.0137692363637, 0.00767642663831, 0.01173801...	[0.0300068989619, 0.00549313615195, 0.02645704...
1082	1	[0.0145171300259, 0.00436159031062, 0.00842335...	[0.0305286644299, 0.0139361764065, 0.030528664...
1083	4	[0.00677521889906, 0.00677317259089, 0.0013666...	[0.0284146019793, 0.0131535056817, 0.024598928...
1084	3	[0.0176597638708, 0.00684679455411, 0.00954620...	[0.0321654353819, 0.0156689934318, 0.032165435...
1085	8	[0.0154683730544, 0.00886617170751, 0.00735801...	[0.0284929699615, 0.0097812185241, 0.032745150...
1087	3	[0.0140685757214, 0.007974191089, 0.0001488258...	[0.0298239454776, 0.00124718732078, 0.01757705...
1088	8	[0.0151375823729, 0.00314453430235, 0.01310602...	[0.0297682239807, 0.0127573712716, 0.025515841...
1091	4	[0.0101236471203, 3.28195486264e-05, 0.0020618...	[0.0316931780077, 1.97874079484e-05, 0.0246537...
1092	5	[0.0142929704185, 0.00616879924608, 7.60995172...	[0.030196733706, 0.0229065170082, 0.0265513617...
1095	3	[0.0151481973406, 0.0110849133066, 0.013116983...	[0.0322307696988, 0.0021462366819, 0.032230769...
1098	6	[0.0196949022892, 0.00617703142446, 0.01428839...	[0.0308972979189, 0.00664596303008, 0.03089729...
1101	3	[0.0122954840913, 0.0122954840913, 0.009591663...	[0.0275782883696, 0.00809368936913, 0.02757828...
1102	1	[0.00920606558956, 0.00650352177841, 0.0038011...	[0.0305235290552, 0.0190771278024, 0.030523529...
1103	6	[0.0151589052147, 0.0070335877347, 0.009066678...	[0.0299475076489, 0.0083272488579, 0.029947507...
1107	8	[0.0139828103835, 0.00226670317335, 0.00992099...	[0.0302368560728, 0.0118494606857, 0.030236856...
1108	2	[0.0129142857708, 0.00478944651949, 0.00478944...	[0.0299812115064, 0.0196194680521, 0.026166007...
1109	1	[0.0120811562969, 0.00396989290107, 0.00396962...	[0.0281714579642, 0.0171894020236, 0.010724913...
1110	2	[0.0091288373043, 0.00506714211514, 0.00303543...	[0.0313666400203, 0.000843397230667, 0.0275512...
1111	8	[0.0129343996864, 0.00752626346618, 0.01293469...	[0.0337344877304, 0.0111713213872, 0.025568404...
1112	6	[0.0175049490636, 0.00939179531256, 0.01209647...	[0.0312624082631, 0.0270574663435, 0.026775500...

[illegible]	[illegible]	[illegible]	[illegible]
1114	2	[0.011212981726, 0.00500732011751, 0.011213491...	[0.0293450533352, 0.0099048669944, 0.021568795...
1118	6	[0.0155110632285, 0.00276852645295, 0.01347933...	[0.035360330635, 0.0312009029024, 0.0353603306...
1119	4	[0.00889318362229, 0.00192213849785, 0.0115976...	[0.0327408763166, 0.00848839104537, 0.03274087...
1120	7	[0.0163454368527, 0.0019348374965, 0.010252430...	[0.0310242882144, 0.00186366091085, 0.02685854...
1121	5	[0.0135330478187, 0.00539280392522, 0.01353304...	[0.0330153460554, 0.00553763681281, 0.03301534...
1124	7	[0.0152832806117, 0.00364226762799, 0.01528328...	[0.0311942815925, 0.0129692959121, 0.031194281...
1126	3	[0.0108613579218, 0.00335941402605, 0.01492378...	[0.0319050943442, 0.0112842986443, 0.023657518...

727 rows × 3 columns

Plan PC and Move PC

In [184]:

```
def P_M_pcscore_simulate(r1):
    f=[]
    for l in range(8):
        l=l+1
        if l in (plan_pc_deleted_targets) or l in (move_pc_deleted_targets):
            r1_deleted=r1
            r1_deleted1=r1[:190]
            r1_deleted2=r1[190:]
            if l in plan_pc_deleted_targets:
                r1_deleted1=np.delete(r1_deleted[:190],plan_pc_deleted_index[plan_pc_deleted_targ
ets.index(l)])
            if l in move_pc_deleted_targets:
                r1_deleted2=np.delete(r1_deleted[190:],move_pc_deleted_index[move_pc_deleted_targ
ets.index(l)])
            r1_deleted=np.append(r1_deleted1,r1_deleted2)
            f1=multivariate_normal.logpdf(r1_deleted, \
                                          mean=np.append(plan_pc_mean[l-1],move_pc_mean[l-1]),\
                                          cov=np.diag(np.append(np.diag(plan_pc_cov[l-1]),np.diag(mo
ve_pc_cov[l-1]))))
        else:
            f1=multivariate_normal.logpdf(r1, \
                                          mean=np.append(plan_pc_mean[l-1],move_pc_mean[l-1]),\
                                          cov=np.diag(np.append(np.diag(plan_pc_cov[l-1]),np.diag(mo
ve_pc_cov[l-1]))))
        f.append(f1)
```

```
    simulate_target=f.index(max(f))+1
    return simulate_target
```

In [185]:
```
PMpcscore_simulate_targets=[]
for i in range(len(test_table_pc)):
    r1=np.append(list(test_table_pc['plan_pc'])[i],list(test_table_pc['move_pc'])[i])
    simulate_target=P_M_pcscore_simulate(r1)
    PMpcscore_simulate_targets.append(simulate_target)
```

In [186]:
```
PMpcscore_simulate_degrees=[degrees[i-1] for i in PMpcscore_simulate_targets]

PMpcscore_e=abs(np.array(orginal_degrees)-np.array(PMpcscore_simulate_degrees))
correct_PMpcscore=[i==j for i,j in zip(test_table_pc['targets'],PMpcscore_simulate_targets)]
PMpcscore_percent=sum(correct_PMpcscore)/len(test_table_pc['targets'])
PMpcscore_d=np.mean(PMpcscore_e)
PMpcscore_d_sem=np.std(PMpcscore_e)/np.sqrt(len(PMpcscore_e))
print('Mean of angular error for the Plan PC score/Move PC score model is %.4f'%PMpcscore_d)
print('Sem of angular error for the Plan PC score/Move PC score model is %.4f'%PMpcscore_d_sem)
print('Simulation accuracy for the Plan PC score/Move PC score model is %.4f%%'%(PMpcscore_percen
t*100))
```

```
Mean of angular error for the Plan PC score/Move PC score model is 2.8054
Sem of angular error for the Plan PC score/Move PC score model is 0.8112
Simulation accuracy for the Plan PC score/Move PC score model is 96.0110%
```

Plan rate and Move PC

In [187]:
```
def Prate_Mpc_simulate(r1):
    f=[]
    for l in range(8):
        l=l+1
        if l in (plan_deleted_targets) or l in (move_pc_deleted_targets):
            r1_deleted=r1
            r1_deleted1=r1[:190]
            r1_deleted2=r1[190:]
            if l in plan_deleted_targets:
                r1_deleted1=np.delete(r1_deleted[:190],plan_deleted_index[plan_deleted_targets.in
dex(l)])
            if l in move_pc_deleted_targets:
                r1_deleted2=np.delete(r1_deleted[190:],move_pc_deleted_index[move_pc_deleted_targ
ets.index(l)])
            r1_deleted=np.append(r1_deleted1,r1_deleted2)
            f1=multivariate_normal.logpdf(r1_deleted, \
```

```
                                       mean=np.append(plan_mean[l-1],move_pc_mean[l-1]),\
                                       cov=np.diag(np.append(np.diag(plan_cov[l-1]),np.diag(move_
pc_cov[l-1]))))
        else:
            f1=multivariate_normal.logpdf(r1, \
                                       mean=np.append(plan_mean[l-1],move_pc_mean[l-1]),\
                                       cov=np.diag(np.append(np.diag(plan_cov[l-1]),np.diag(move_
pc_cov[l-1]))))
        f.append(f1)
    simulate_target=f.index(max(f))+1
    return simulate_target
```

In [188]:

```
Prate_Mpc_simulate_targets=[]
for i in range(len(test_table_pc)):
    r1=np.append(list(test_table1['plan_rate'])[i],list(test_table_pc['move_pc'])[i])
    simulate_target=Prate_Mpc_simulate(r1)
    Prate_Mpc_simulate_targets.append(simulate_target)
```

In [189]:

```
Prate_Mpc_simulate_degrees=[degrees[i-1] for i in Prate_Mpc_simulate_targets]

Prate_Mpc_e=abs(np.array(orginal_degrees)-np.array(Prate_Mpc_simulate_degrees))
correct_Prate_Mpc=[i==j for i,j in zip(test_table_pc['targets'],Prate_Mpc_simulate_targets)]
Prate_Mpc_percent=sum(correct_Prate_Mpc)/len(test_table_pc['targets'])
Prate_Mpc_d=np.mean(Prate_Mpc_e)
Prate_Mpc_d_sem=np.std(Prate_Mpc_e)/np.sqrt(len(Prate_Mpc_e))
print('Mean of angular error for the Plan rate/Move PC score model is %.4f'%Prate_Mpc_d)
print('Sem of angular error for the Plan rate/Move PC score model is %.4f'%Prate_Mpc_d_sem)
print('Simulation accuracy for the Plan rate/Move PC score model is %.4f%%'%(Prate_Mpc_percent*10
0))
```

```
Mean of angular error for the Plan rate/Move PC score model is 3.1905
Sem of angular error for the Plan rate/Move PC score model is 1.0045
Simulation accuracy for the Plan rate/Move PC score model is 96.8363%
```

Plan PC and move rate

In [35]:

```
def Ppc_Mrate_simulate(r1):
    f=[]
    for l in range(8):
        l=l+1
        if l in (plan_pc_deleted_targets) or l in (move_deleted_targets):
            r1_deleted=r1
            r1_deleted1=r1[:190]
```

```
            r1_deleted1=r1[:190]
            r1_deleted2=r1[190:]
            if l in plan_pc_deleted_targets:
                r1_deleted1=np.delete(r1_deleted[:190],plan_pc_deleted_index[plan_pc_deleted_targe
ts.index(l)])
            if l in move_deleted_targets:
                r1_deleted2=np.delete(r1_deleted[190:],move_deleted_index[move_deleted_targets.ind
ex(l)])
            r1_deleted=np.append(r1_deleted1,r1_deleted2)
            f1=multivariate_normal.logpdf(r1_deleted, \
                                          mean=np.append(plan_pc_mean[l-1],move_mean[l-1]),\
                                          cov=np.diag(np.append(np.diag(plan_pc_cov[l-1]),np.diag(mov
e_cov[l-1]))))
        else:
            f1=multivariate_normal.logpdf(r1, \
                                          mean=np.append(plan_pc_mean[l-1],move_mean[l-1]),\
                                          cov=np.diag(np.append(np.diag(plan_pc_cov[l-1]),np.diag(mov
e_cov[l-1]))))
        f.append(f1)
    simulate_target=f.index(max(f))+1
    return simulate_target
```

In [36]:
```
Ppc_Mrate_simulate_targets=[]
for i in range(len(test_table_pc)):
    r1=np.append(list(test_table_pc['plan_pc'])[i],list(test_table1['move_rate'])[i])
    simulate_target=Ppc_Mrate_simulate(r1)
    Ppc_Mrate_simulate_targets.append(simulate_target)
```

In [67]:
```
Ppc_Mrate_simulate_degrees=[degrees[i-1] for i in Ppc_Mrate_simulate_targets]

Ppc_Mrate_e=abs(np.array(orginal_degrees)-np.array(Ppc_Mrate_simulate_degrees))
correct_Ppc_Mrate=[i==j for i,j in zip(test_table_pc['targets'],Ppc_Mrate_simulate_targets)]
Ppc_Mrate_percent=sum(correct_Ppc_Mrate)/len(test_table_pc['targets'])
Ppc_Mrate_d=np.mean(Ppc_Mrate_e)
Ppc_Mrate_d_sem=np.std(Ppc_Mrate_e)/np.sqrt(len(Ppc_Mrate_e))
print('Mean of angular error for the Plan PC score/Move rate model is %.4f'%Ppc_Mrate_d)
print('Sem of angular error for the Plan PC score/Move rate model is %.4f'%Ppc_Mrate_d_sem)
print('Simulation accuracy for the Plan PC score/Move rate model is %.4f%%'%
(Ppc_Mrate_percent*100))
```

```
Mean of angular error for the Plan PC score/Move rate model is 3.6856
Sem of angular error for the Plan PC score/Move rate model is 1.0157
Simulation accuracy for the Plan PC score/Move rate model is 95.5983%
```

Results

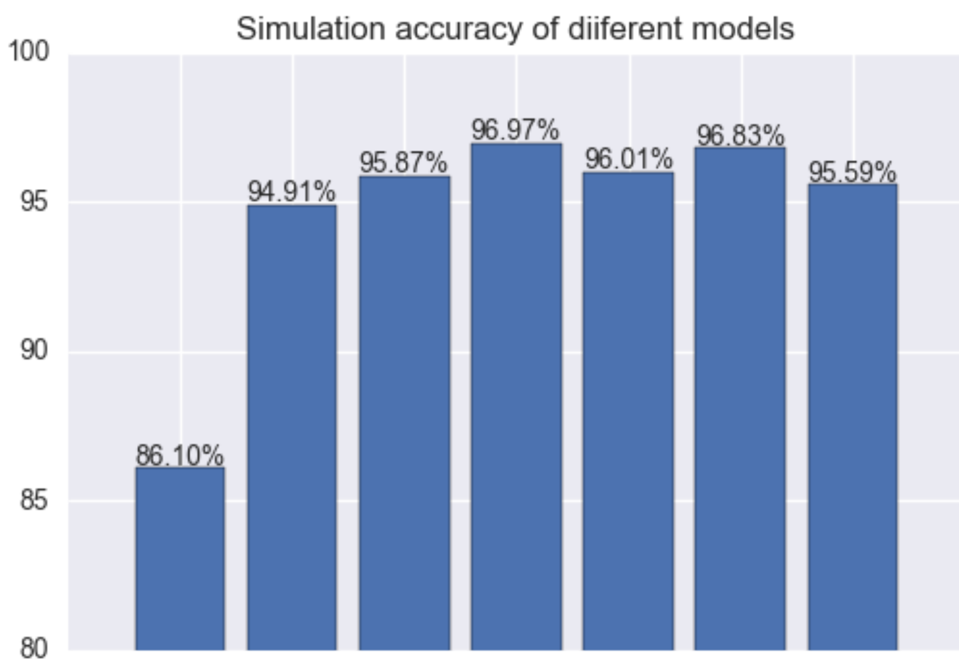
In [199]:

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

results_accuracy=[plan_percent,move_percent,combined_percent,PMrate_percent,\
                  PMpcscore_percent,Prate_Mpc_percent,Ppc_Mrate_percent]
results_degrees=[plan_d,move_d,combined_d,PMrate_d,\
                  PMpcscore_d,Prate_Mpc_d,Ppc_Mrate_d]
results_sem=[plan_d_sem,move_d_sem,combined_d_sem,PMrate_d_sem,\
            PMpcscore_d_sem,Prate_Mpc_d_sem,Ppc_Mrate_d_sem]
category=['Move\n Rate','Plan\n Rate','Undiff.\n Rate','Plan Rate/\n Move Rate',\
          'Plan PC/\n Move PC','Plan Rate/\n Move PC','Plan PC/\n Move Rate']
```

In [217]:

```
x=np.arange(len(results_accuracy))+1
plt.bar(left=x,height=np.array(results_accuracy)*100,align='center',tick_label=category)
plt.xticks(horizontalalignment='center',fontsize=8)
plt.ylim(80,100)
plt.title('Simulation accuracy of diiferent models')
for a,b in zip(x,np.array(results_accuracy)*100):
    c=str(b)[:5]+'%'
    plt.text(a,b+0.1,c,horizontalalignment='center')
```



Move Rate | Plan Rate | Undiff. Rate | Plan Rate/ Move Rate | Plan PC/ Move PC | Plan Rate/ Move PC | Plan PC/ Move Rate

In [218]:
```
x=np.arange(len(results_accuracy))+1
plt.bar(left=x,height=results_degrees,align='center',tick_label=category)
plt.xticks(horizontalalignment='center',fontsize=8)
plt.ylim(0,12)
plt.title('Absolute error(mean$\pm$sem) of different models')
for a,b in zip(x,results_degrees):
    c=str(b)[:5]+'$^{\circ}$'+'\n''$\pm$'+str(results_sem[a-1])[:5]
    plt.text(a,b+0.1,c,horizontalalignment='center')
```