**Support Vector Machine Prediction of   
Finger Tapping Frequency from fMRI Data**Jonathan Ng  
August 2009

**Abstract**

The ultimate goal of this project was to create a regression model for a simple motor task that can predict the behavioral state of the subject for each fMRI time point collected. In this process, we also examined classification performance and regions of brain activation for our task.

**Introduction/Background**

**Functional Magnetic Resonance Imaging**

Blood oxygenation level dependent functional magnetic resonance imaging (BOLD fMRI) is a technique widely used in neuroscience that utilizes vascular correlates of neural activity to obtain spatially and temporally resolved measurements of the brain as an individual performs a stimulus driv en task. As neurons do not have stores of glucose or oxygen, they require an outside energy source in order to fire. Thus, a hemodynamic response is necessary, during which blood releases oxygen to the neurons that are firing. The difference in the MR signal of diamagnetic oxygenated hemoglobin and paramagnetic deoxygenated hemoglobin is then used to calculate BOLD (blood oxygenation level dependent) signal intensities, which are caused by an increase in the concentration of oxygenated hemoglobin causes the blood magnetic susceptibility to more closely match that of the tissue.

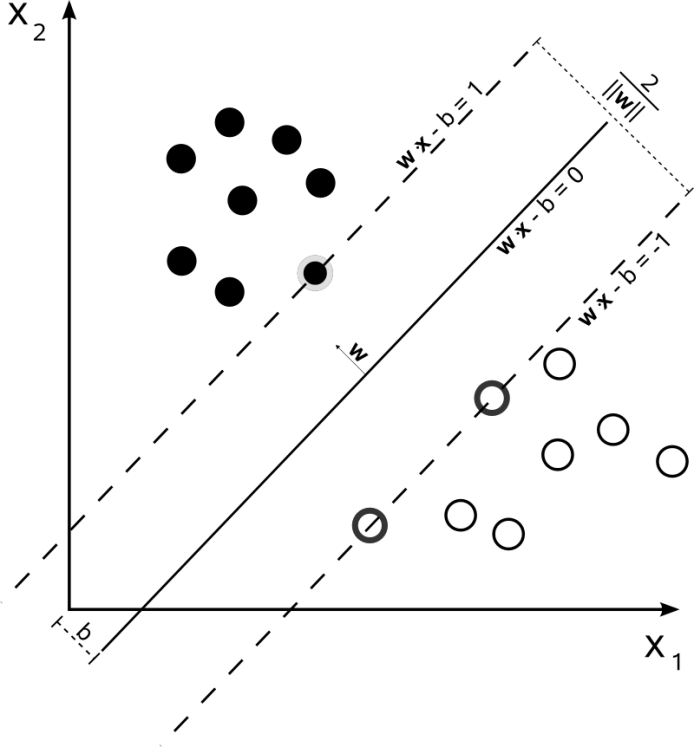
**Machine Learning**

Machine learning concerns the development and employment of algorithms to make intelligent decisions based on patterns from data. Supervised learning involves training data, which is data that has assigned labels that are used together to create an algorithm for classifying the data; this algorithm can then be used on sets of data without labels to make intelligent predictions. Unsupervised learning, however, does not use training data, and clusters sets of data based on different parameters.

**Support Vector Machines**

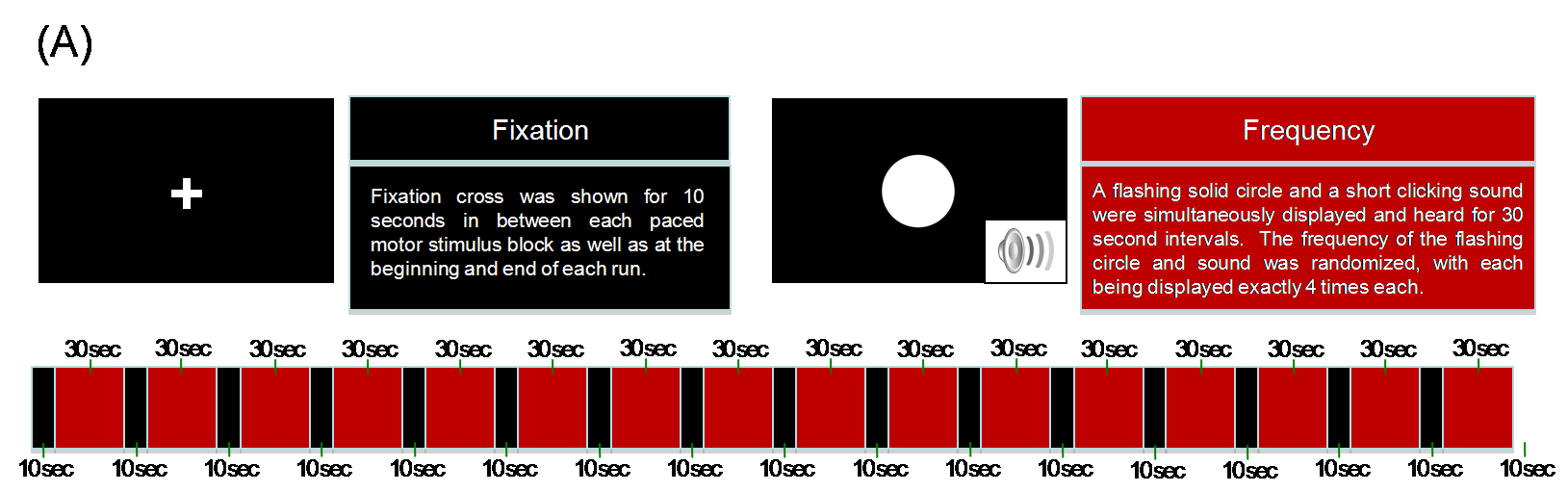
Support vector machines (SVMs), a machine learning method, allow for classification and regression analysis to enable prediction of stimulus conditions from fMRI data. By collecting training data—data with labels related to the task condition—it is possible to train SVMs to predict subsequent sets of unlabeled data, virtually making it possible to estimate what the subject’s sensory/behavioral state was for each fMRI time point. Support vector regression is mathematically similar to support vector classification, but has the potential to allow prediction of continuously varying task “levels,” while classification is used for prediction of categorical stimuli.

For support vector classification, the two dimensional problem can be solved and then generalized to higher dimensions. As seen in Figure , for two classes, we want to optimally separate them with a hyperplane.



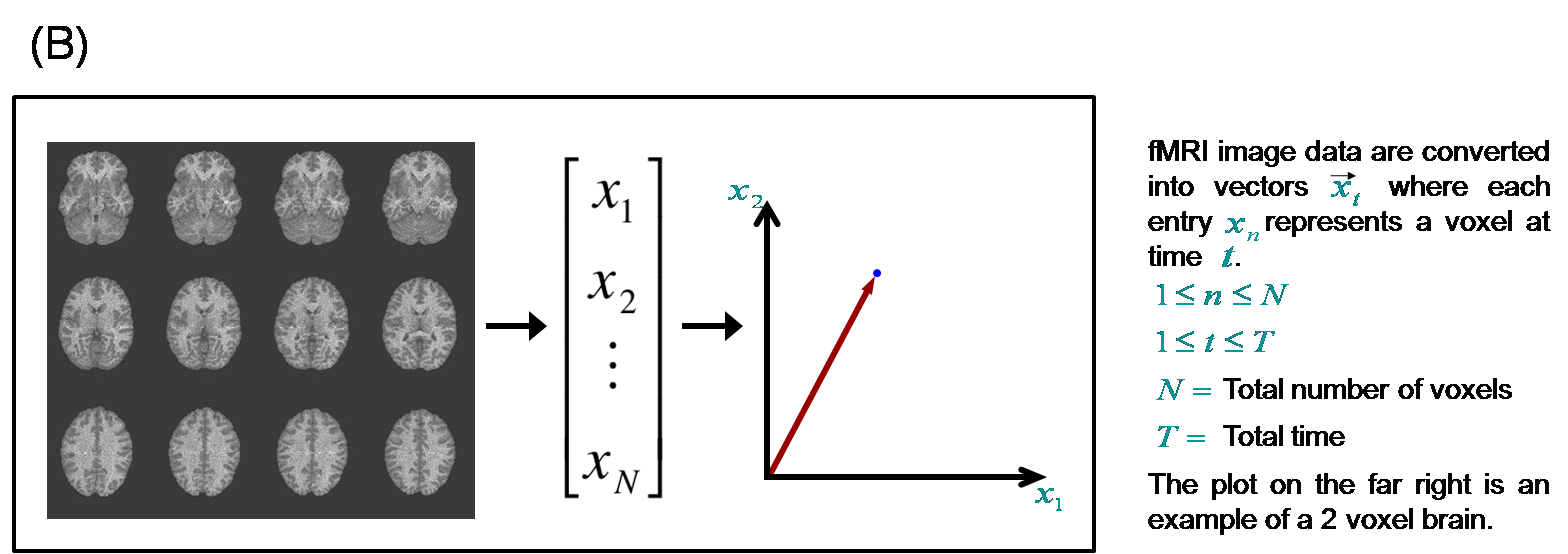
**Methods**

One healthy right-handed male volunteer performed the finger tapping task portrayed in figure (A). The subject was given two button boxes to hold, one in each hand. He was instructed to press the button on each button box with the index fingers on each respective box simultaneously at a certain frequency, guided by both a visual and auditory metronome (a flashing solid circle located at the center of the screen and short clicking sound played simultaneously—see [5] for a review of fMRI finger tapping tasks). The subject’s button presses were recorded by the stimulus presentation software and saved in a data file that was subsequently used to analyze the data. Each motor block lasted 30 seconds, and paced frequencies for each block were randomly presented at frequencies of {2, 3, 4, 5} Hz (the tapping frequency for the entire 30 seconds was fixed at one of these levels). Each frequency was displayed 4 times per run. Before and after each button tapping block was a 10 second control period during which a fixation cross would appear in the center of the screen. The subject performed 3 runs of this task during the scanning session.



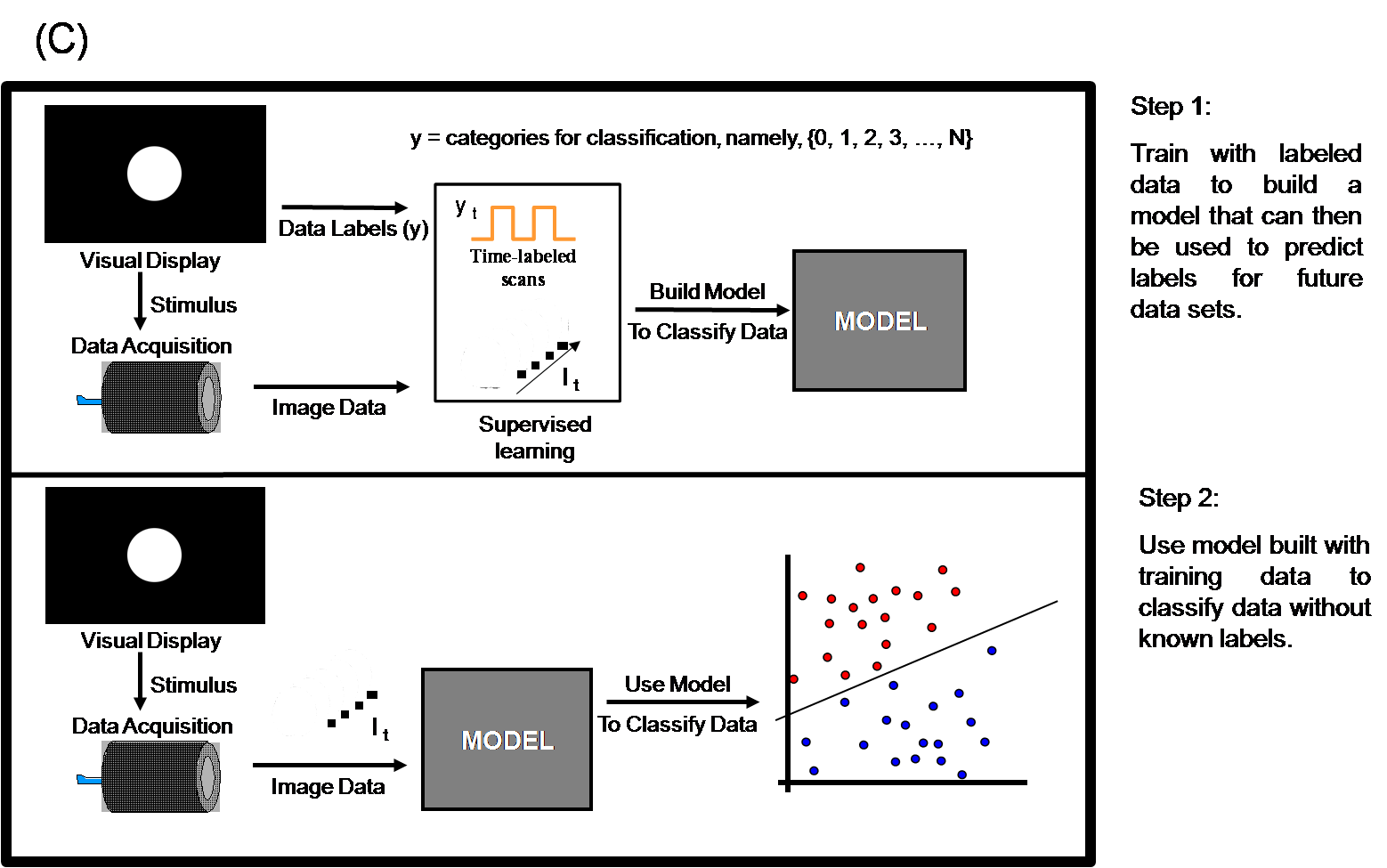
The stimulus for the task was created and displayed with Vision Egg. The design and development of the Vision Egg software constituted a major portion of the summer project. fMRI data were collected using a 3 Tesla Siemens Trio Scanner, with 30 axial EPI slices (TR/TE = 2000/31 msec).

Image data were then used for classification through the comparison of brain states containing voxels of neurons with distinct activation.



The written data files that recorded button presses from the scan were analyzed in MATLAB. The mean and variance of the rate at which the subject was actually tapping were found for each motor block. Using this data in MATLAB, the trends in left versus right tapping and the accuracy with which the subject tapped with each hand were also analyzed.

The fMRI data for the right hand were analyzed using AFNI, a set of programs used for mapping human brain activity. 3dsvm, the SVM algorithm plug-in for AFNI, was used for classification of whole brain volumes. Each possible combination of single training runs used to predict single test runs were analyzed for each frequency using 3dsvm. The prediction accuracy for each of these combinations as well as for the multiclass classification was found.



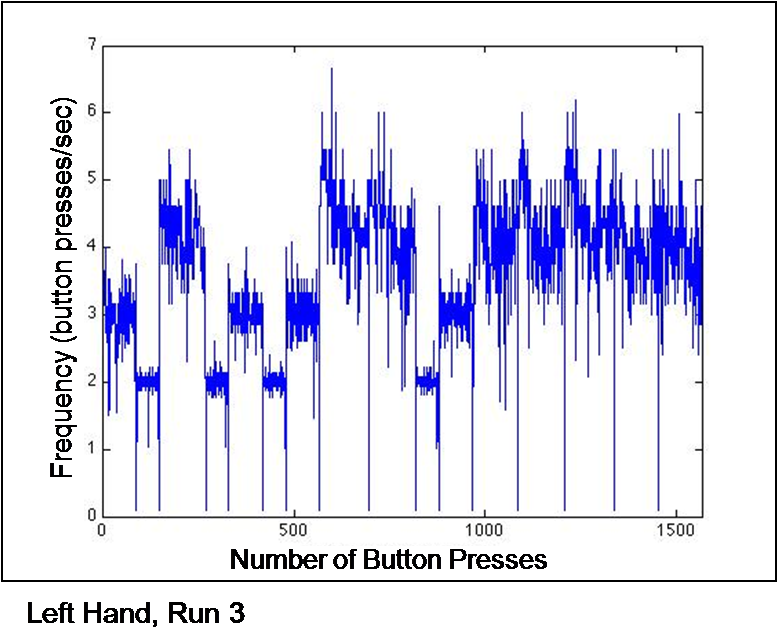
For the classification and regression models, the first two TRs (4000 msec) of each motor block were omitted to account for the time the subject would need to recognize and adjust to the frequency at which the visual and auditory stimuli were paced. Data sets and label files were concatenated to train on 2 runs and test on the third for the regression. All permutations of this combination were performed to obtain better predictions.

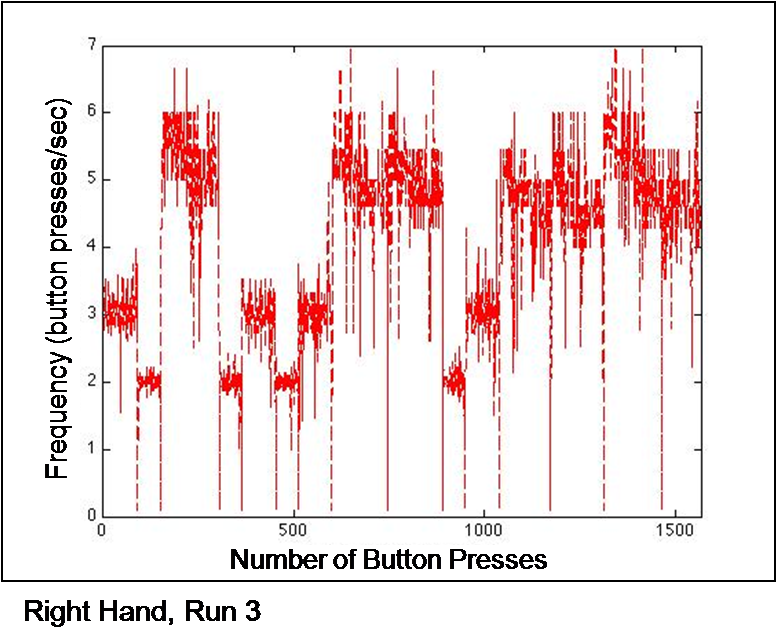
**Results**

**Behavioral Data**

Performance by the subject was similar in all three runs in terms of tapping accuracy. The subject’s tapping with his left hand was slower than that of his right hand, which was generally more accurate. For runs 1, 2, and 3, there were 71, 79, and 168 more button presses with the right hand than the left, respectively. The variance in the actual frequency that the subject was tapping at seemed to correspond with the paced frequency of the motor blocks; variance was mostly below .10 and .15 for the 2 Hz and 3 Hz blocks, respectively. The variance for the 4 Hz block was generally between .25 and .50 while that for the 5 Hz block was in the range of .25-.60.

The task with only a visual stimulus was performed one day before the mentioned runs due to technical difficulties. This task was shown to provide different results than that of the task with a visual and an auditory stimulus, which was consistent with the results found in [5]. The subject was much more accurate in the finger tapping task when the auditory stimulus was presented along with the visual stimulus.

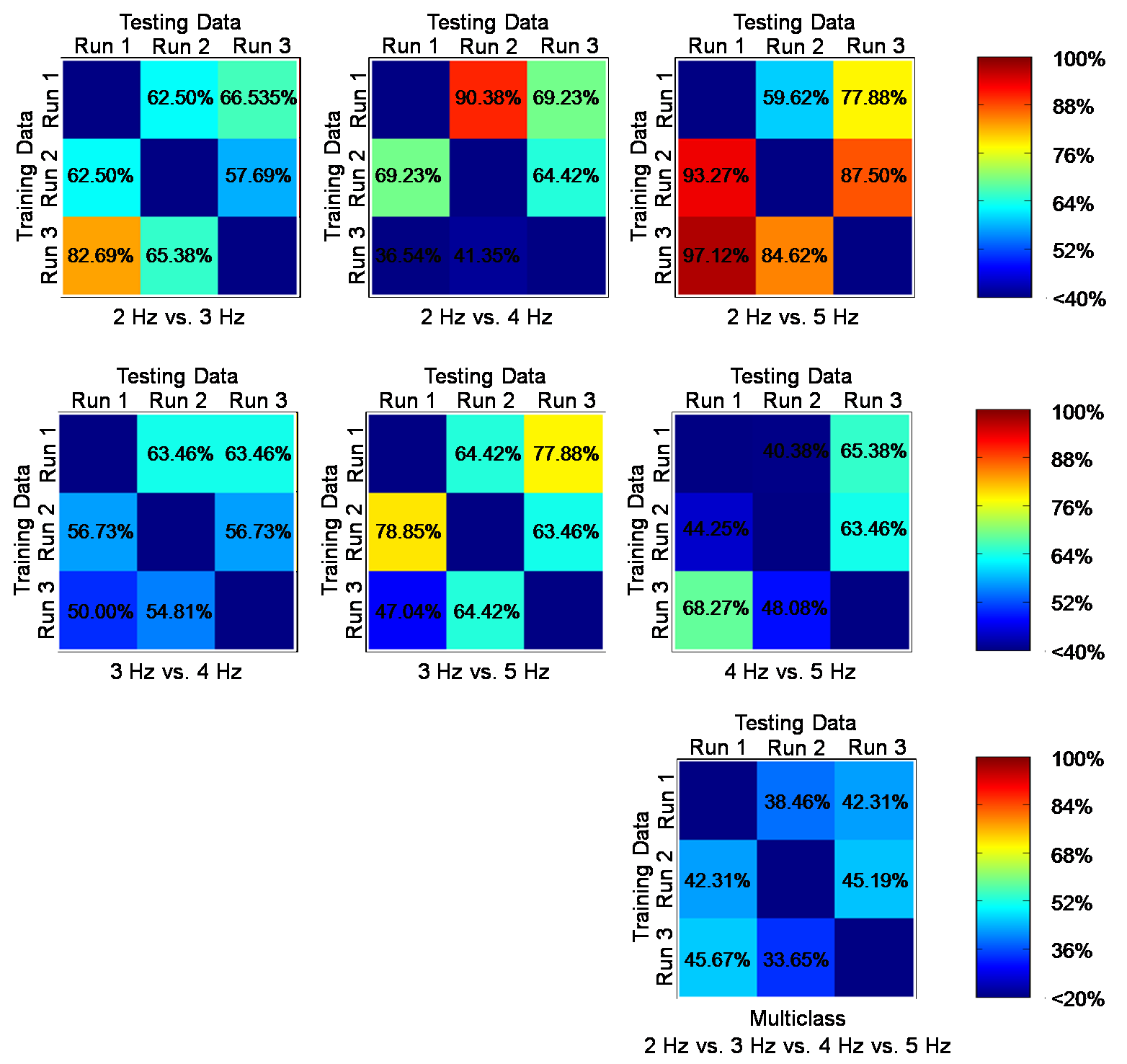




**Classification**

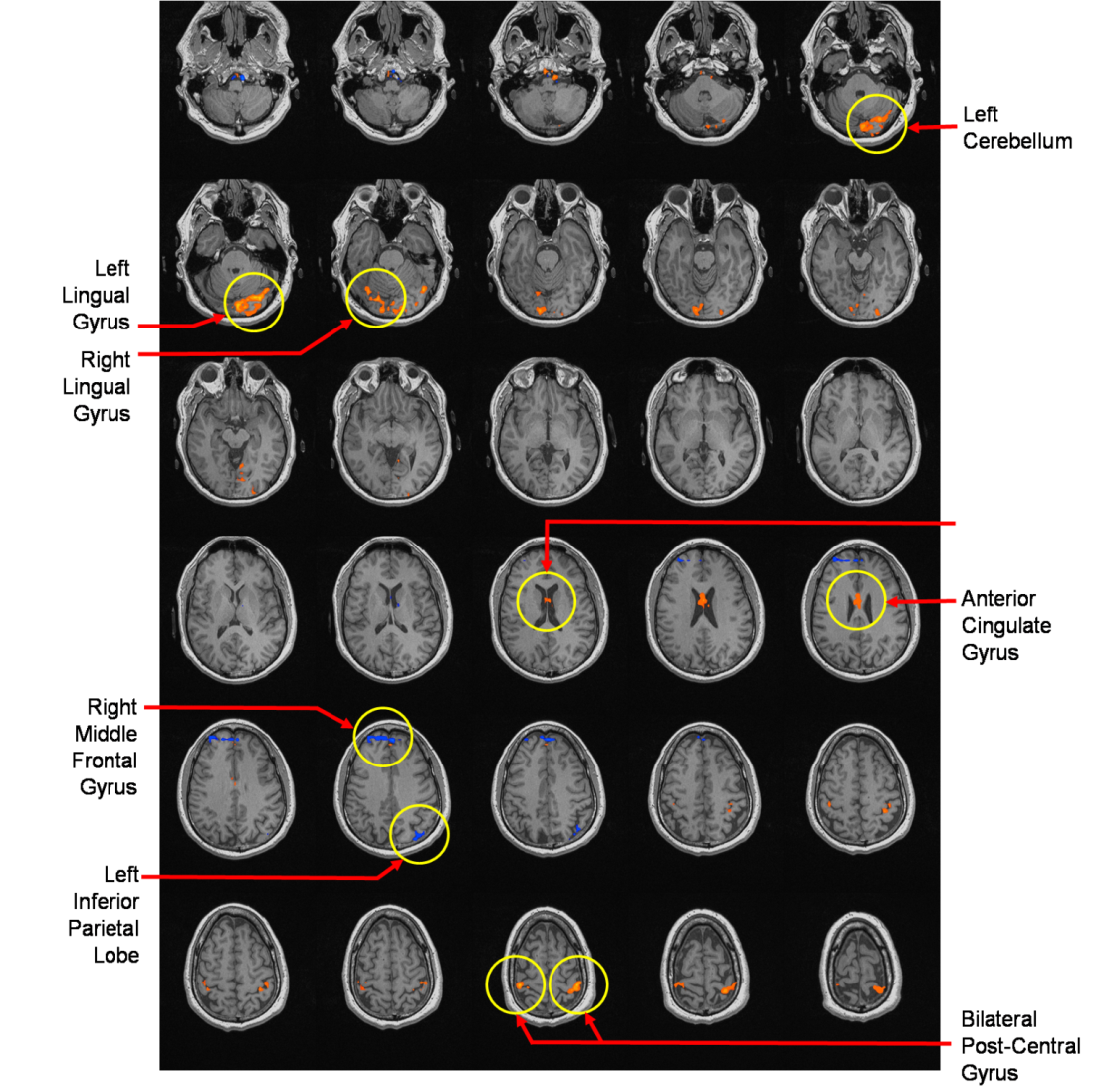
**Prediction Accuracy**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Using run 1 to predict run 2 | Using run 1 to predict run 3 | Using run 2 to predict run 1 | Using run 2 to predict run 3 | Using run 3 to predict run 1 | Using run 3 to predict run 2 |
| 2 Hz vs 3 Hz | 62.50% | 66.35% | 62.50% | 57.69% | 82.69% | 65.38% |
| 2 Hz vs 4 Hz | 90.38% | 69.23% | 69.23% | 64.42% | 36.54% | 41.35% |
| 2 Hz vs 5 Hz | 59.62% | 77.88% | 93.27% | 87.50% | 97.12% | 84.62% |
| 3 Hz vs 4 Hz | 63.46% | 63.46% | 56.73% | 56.73% | 50.00% | 54.81% |
| 3 Hz vs 5 Hz | 64.42% | 77.88% | 78.85% | 63.46% | 47.04% | 64.42% |
| 4 Hz vs 5 Hz | 40.38% | 65.38% | 44.25% | 63.46% | 68.27% | 48.08% |
| multiclass | 38.46% | 42.31% | 42.31% | 45.19% | 45.67% | 33.65% |
| * 2 Hz | 67.31% | 53.85% | 55.77% | 57.69% | 30.77% | 36.54% |
| * 3 Hz | 26.92% | 30.77% | 34.62% | 17.31% | 34.62% | 25.00% |
| * 4 Hz | 28.85% | 21.15% | 50.00% | 48.08% | 40.38% | 28.85% |
| * 5 Hz | 30.77% | 63.46% | 28.85% | 57.69% | 76.92% | 44.23% |



Using SVM classification to predict labels for other data sets, the SVM did fair to good, nearly always classifying the data correctly at a higher percentage than chance. The SVM did a particularly well at distinguishing 2 Hz vs. 5 Hz. For frequencies that were closer to each other, the SVM performed slightly worse. For the multiclass prediction, the SVM also generally performed better than chance.

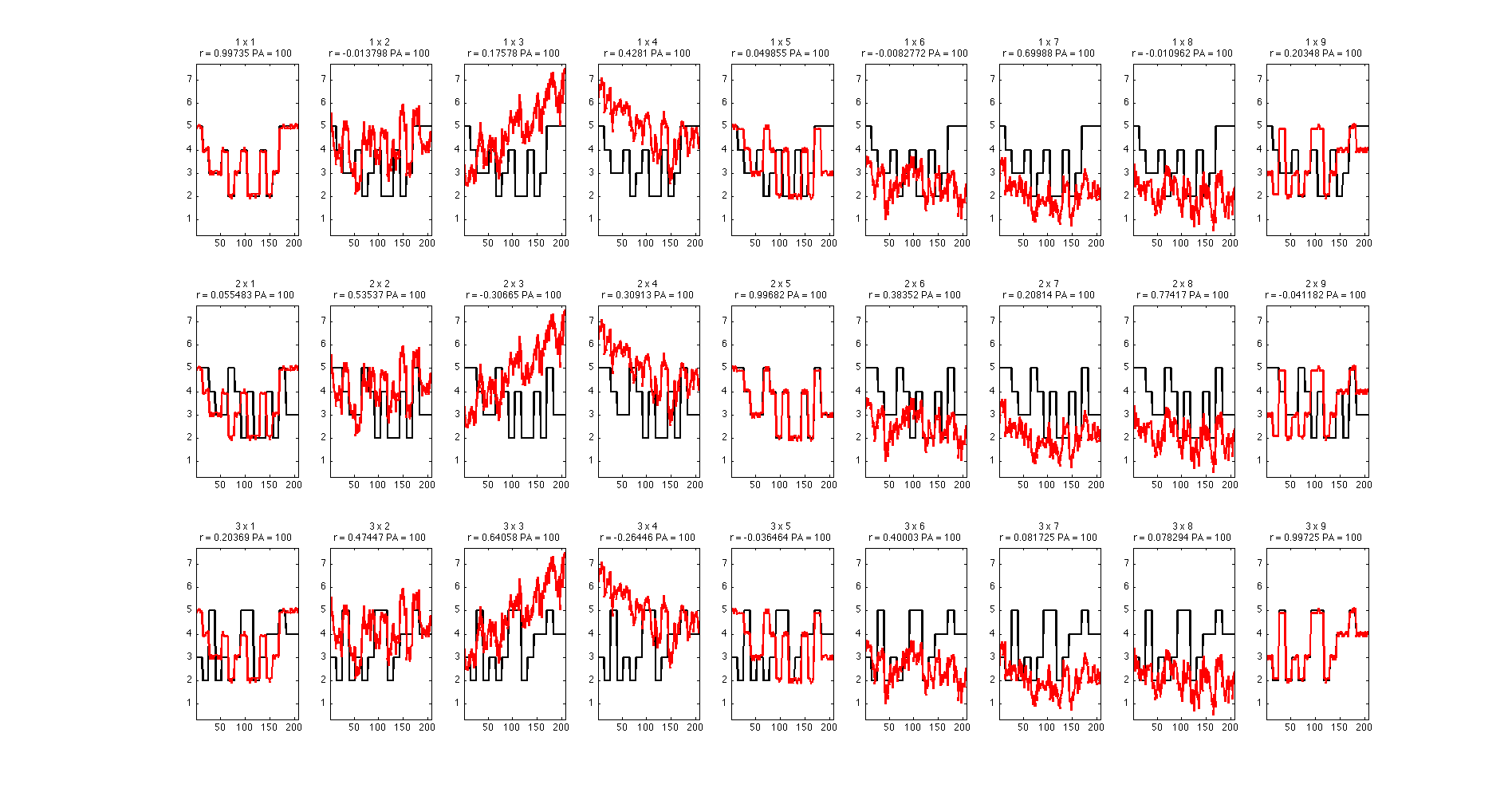
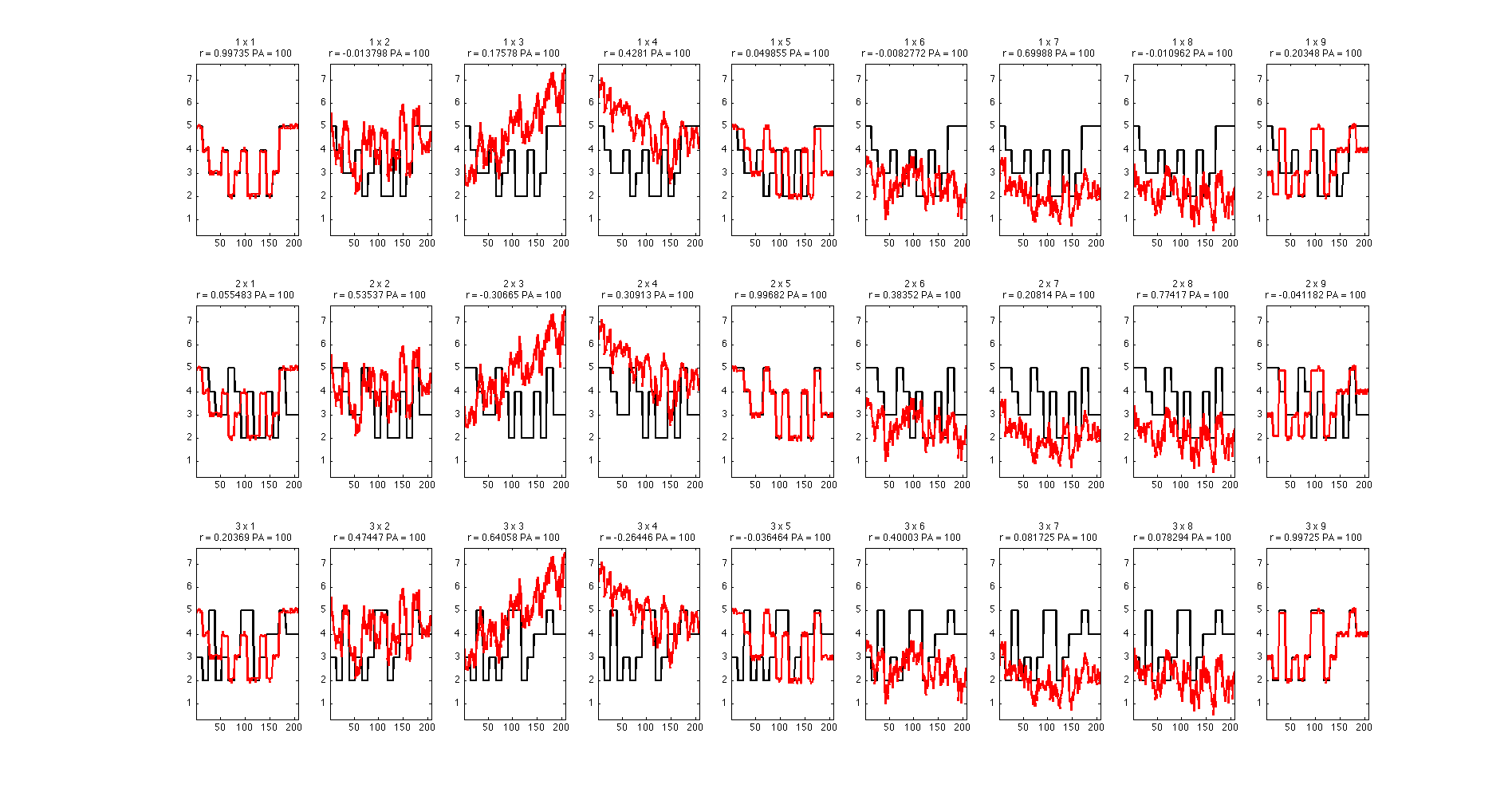
**Neural Activation**

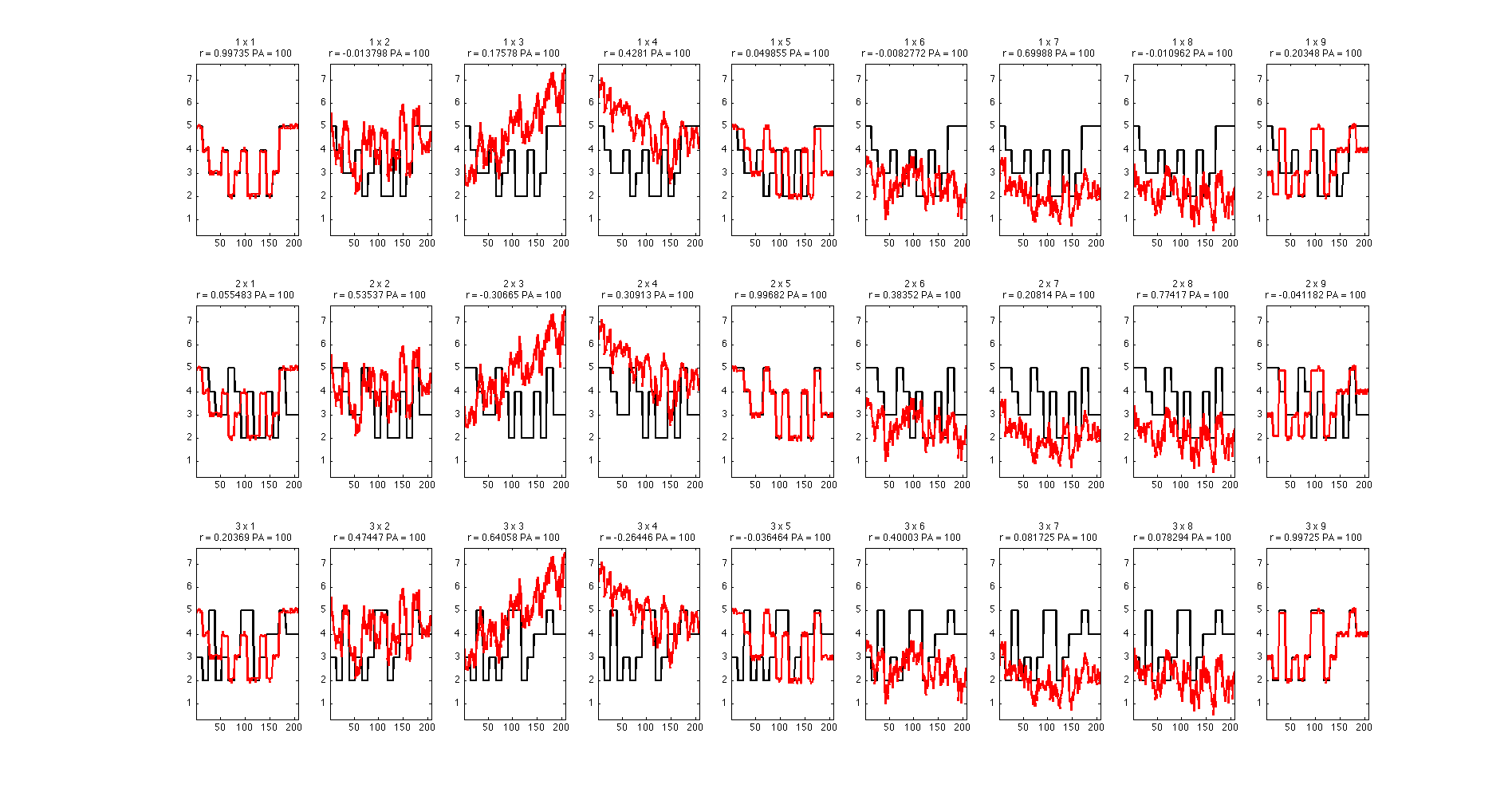
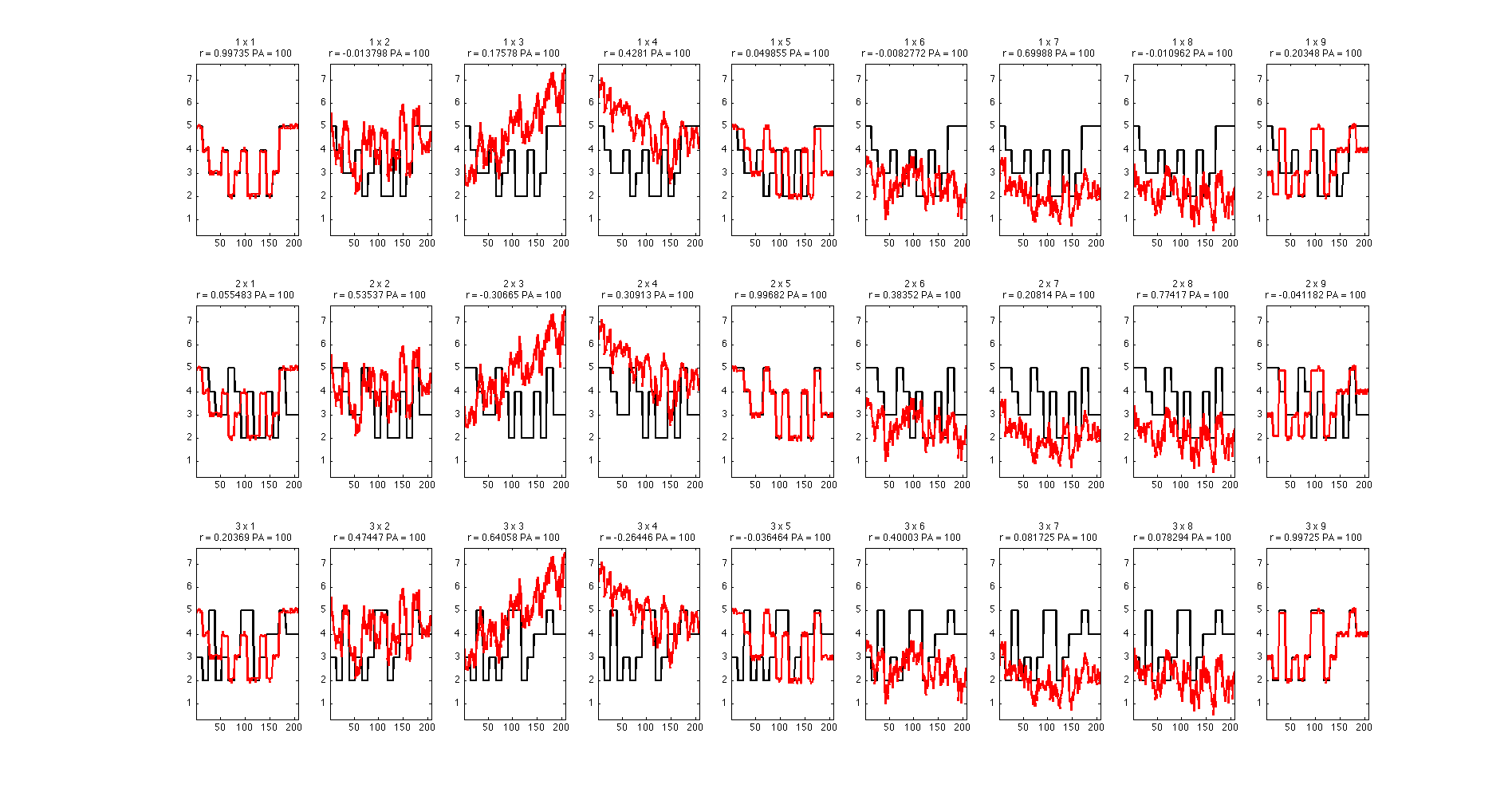


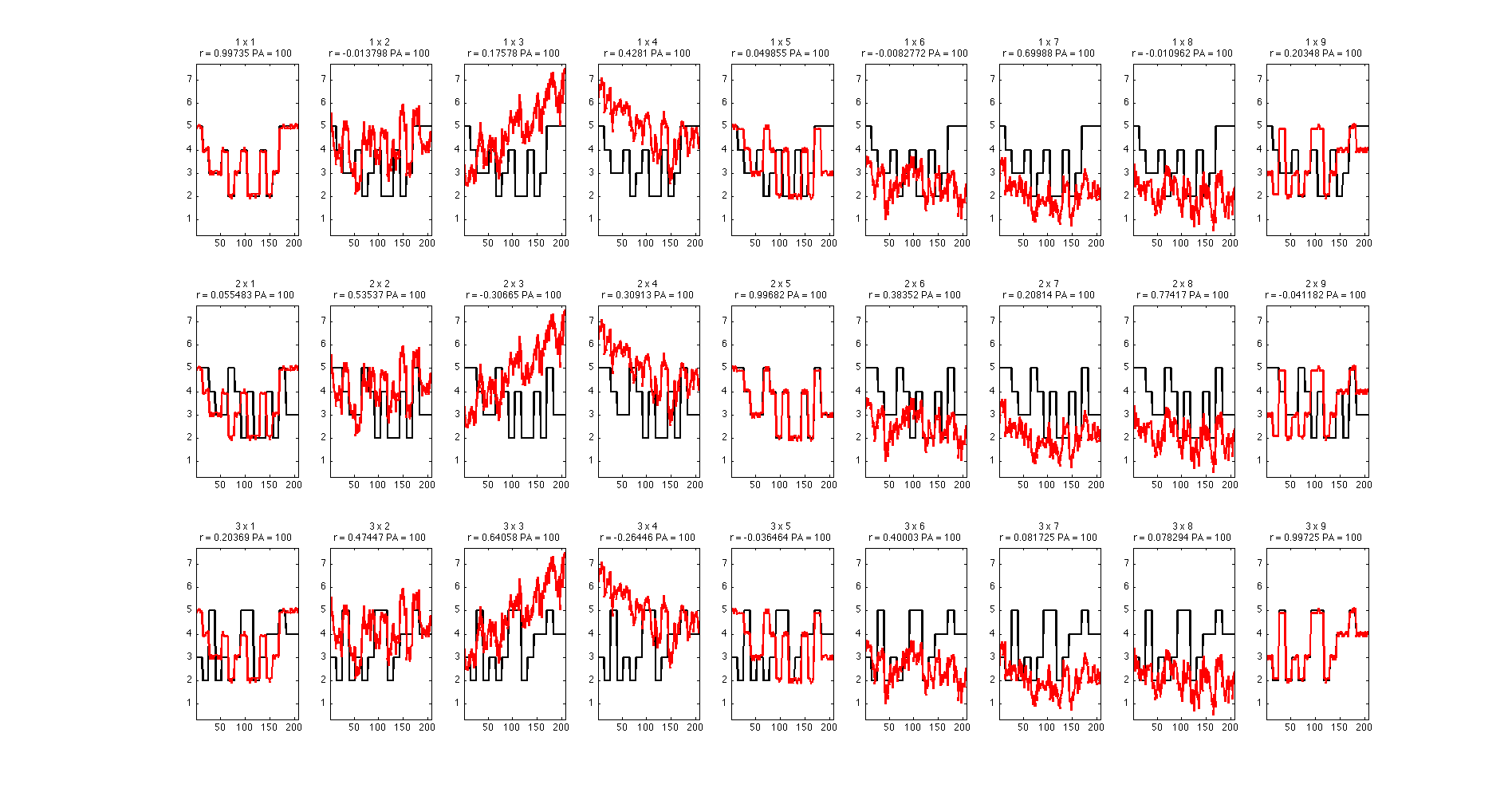
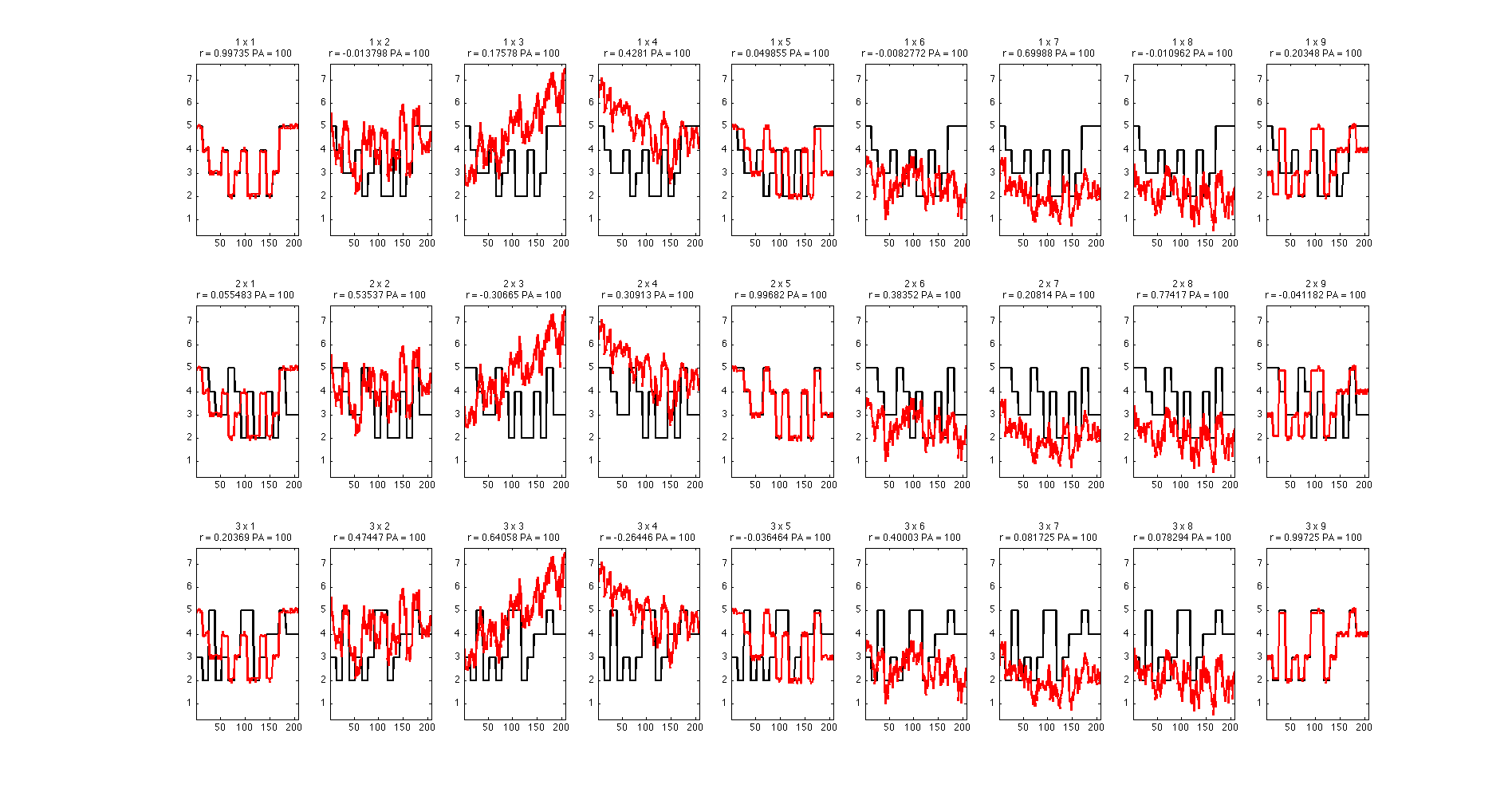
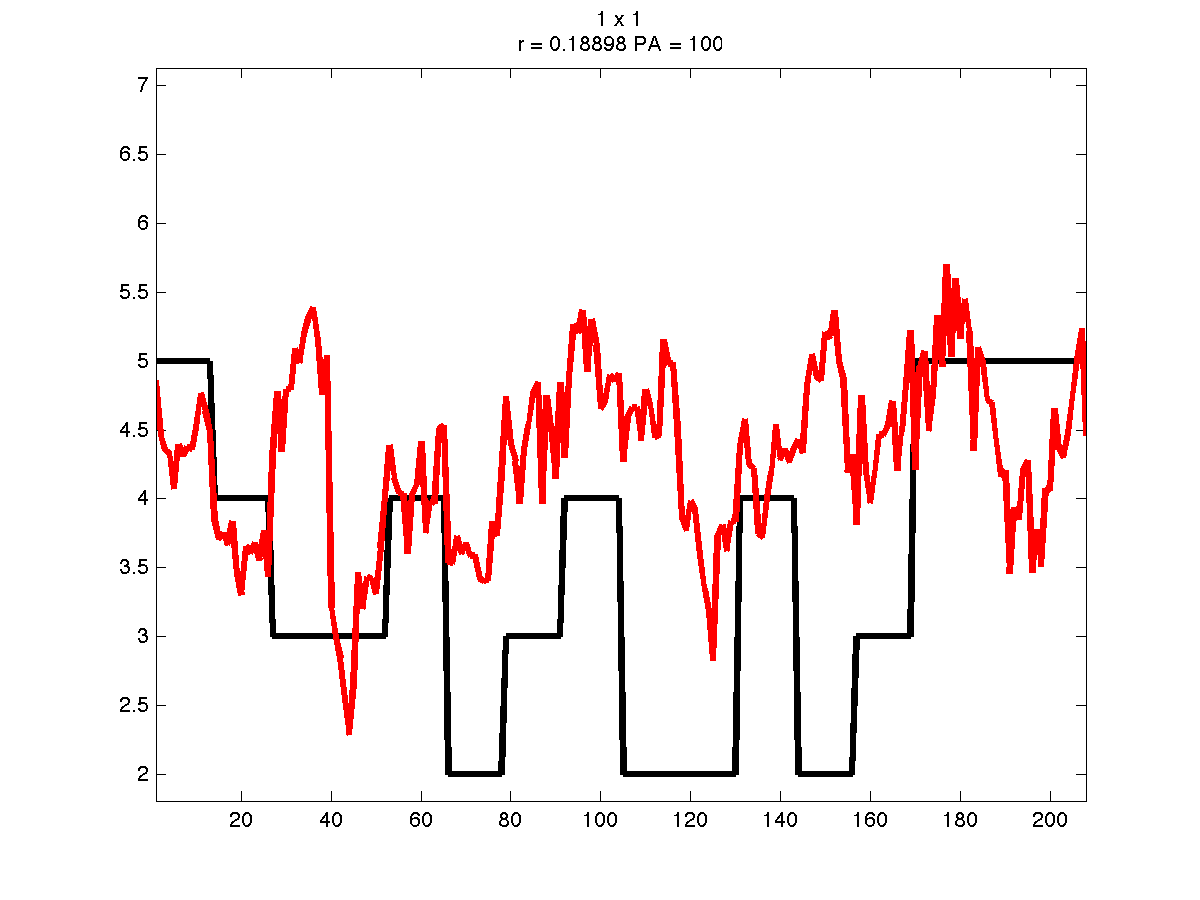
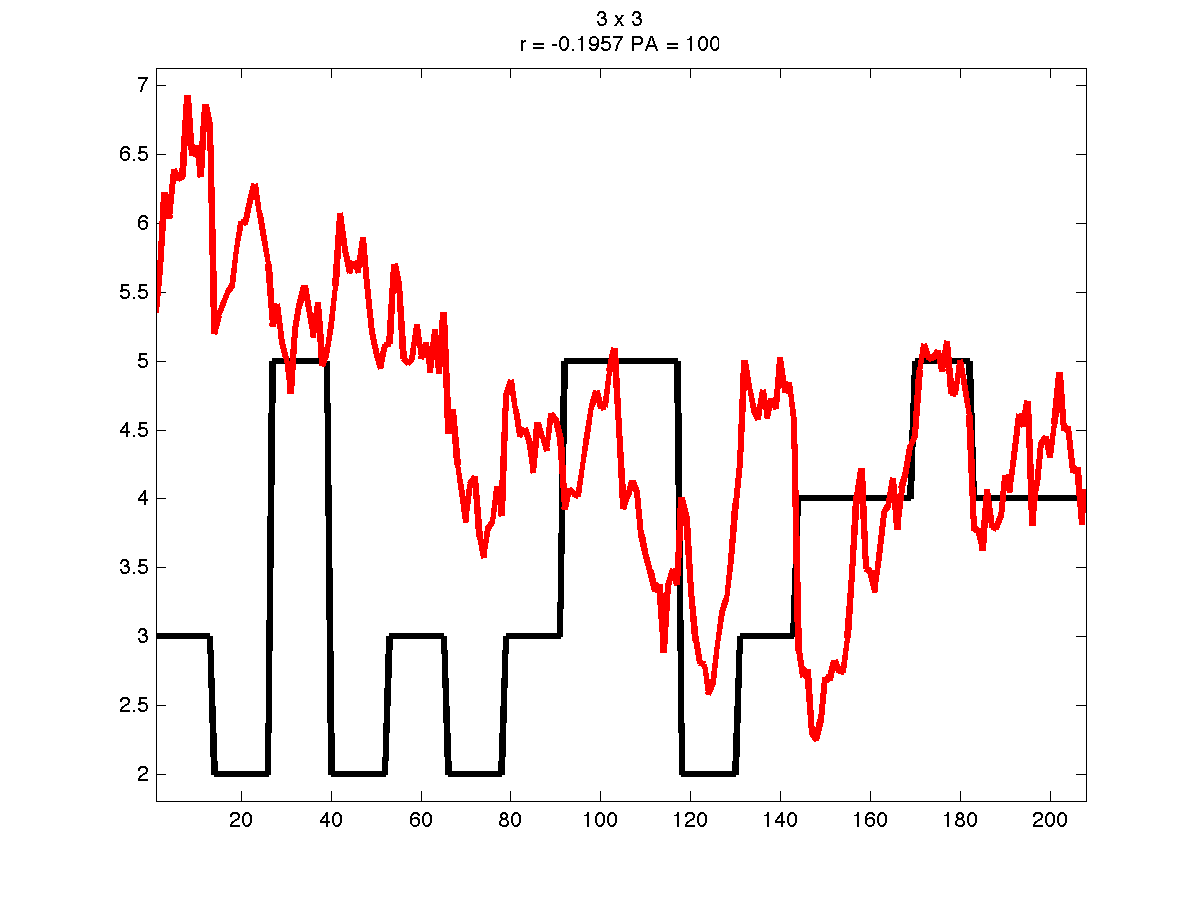
A spatial map derived from the SVM classification depicting the differences in neural activity between the 2 Hz and 5 Hz conditions for the third run is shown in (F). Areas observed include the post-central gyrus, right middle frontal gyrus, lingual gyrus, and left cerebellum.

**Regression**

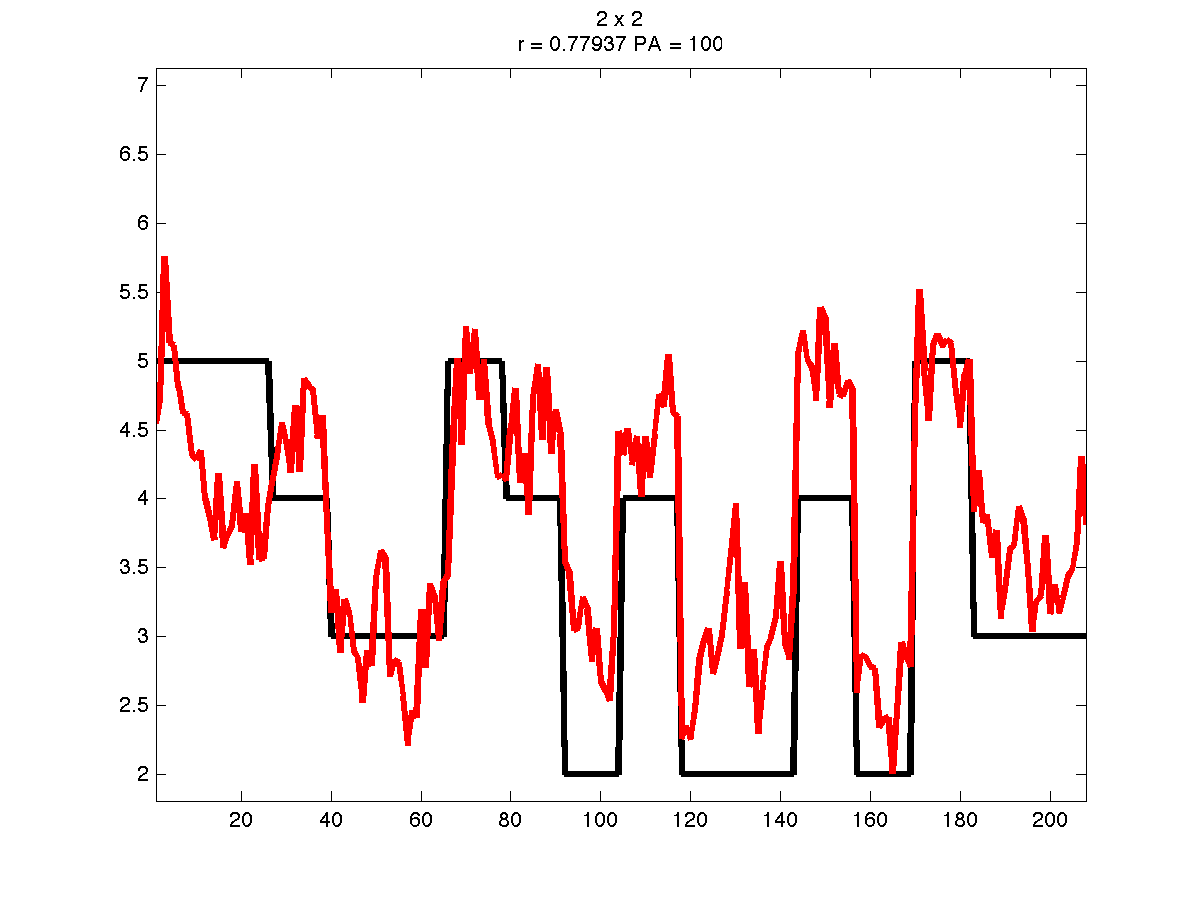
Using data from a single run to predict for one of the other two runs did not provide a very accurate regression model. However, concatenating data from two runs and testing on the third provided better results. Figure (G) portrays the target frequency and the predicted frequency of the second run after training on data from runs one and three.

Train on 1, Test on 2Train on 1, Test on 3

Train on 2, Test on 1Train on 2, Test on 3

Train on 3, Test on 1Train on 3, Test on 2

Training on data from runs 1 and 2. Testing on data from run 3.



**Conclusion/Discussion**

Support vector machines were used to analyze data from a finger tapping task and classification results above chance were achieved. Important regions for discriminating stimulus conditions were consistent with the finger-tapping literature [2,4,5]. The regression model did not perform as well as expected, which might have been caused by a variety of reasons. Only one data set was available to analyze; upon collection of more data, results could be more accurate. Increasing the number of runs or number of blocks might also help to achieve more accurate predictions in the future. Behaviorally, the subject performed better with his right (dominant) hand than his left.

Further studies include working to create a better regression model that is able to predict more accurately a person’s tapping rate. After achieving a good regression model, it would be interesting to predict the behavioral data, which are continuous values (rather than discrete frequency levels). Behaviorally, it would also be interesting to study subjects over a range of skill levels.

The work from this study would hopefully generalize to other parametrically varying tasks and other situations in which the sensory and behavioral conditions are most naturally represented with continuous values.

References

[1] Jancke, L., Specht, K., Mirzazade, S., Loose, R., Himmelbach, M., Lutz, K., Shah, N.J. (1998), A parametric analysis of the ‘rate effect’ in the sensorimotor cortex: a functional magnetic resonance imaging analysis in human subjects. Neuroscience Letters 252, 37-40

[2] LaConte, S.M., Peltier, S.J., Hu, X.P. (2007), Real-Time fMRI Using Brain-State Classification. Human Brain Mapping 28: 1033-1044

[3] LaConte, S., Strother, S., Cherkassky, V., Anderson, J., Hu, X. (2005), Support vector machines for temporal classification of block design fMRI data. NeuroImage 26, 317-329

[4] Rao, S.M., Bandettini, P.A., Binder, J.R., Bobholz, J.A., Hammeke, T.A., Stein, E.A., Hyde, J.S. (1996), Relationship Between Finger Movement Rate and Functional Magnetic Resonance Signal Change in Human Primary Motor Cortex. Journal of Cerebral Blood Flow and Metabolism 16: 1250-1254

[5] Witt, T.W., Laird, A.R., Meyerand, M.E. (2008), Functional neuroimaging correlates of finger-tapping task variations: An ALE meta-analysis. NeuroImage 42, 343-356

**Appendix A**

#!/usr/bin/env python

###############################

# Paradigm for finger tapping #

# #

# June 2009 #

# Jonathan Ng #

# #

###############################

############################

# Import various modules #

############################

import VisionEgg

VisionEgg.start\_default\_logging(); VisionEgg.watch\_exceptions()

from VisionEgg.Core import \*

from VisionEgg.FlowControl import Presentation, Controller, FunctionController

from VisionEgg.MoreStimuli import \*

from VisionEgg.Textures import \*

from math import \*

import pygame

import OpenGL.GL as gl

from VisionEgg.DaqKeyboard import \*

from VisionEgg.Text import \*

from VisionEgg.Textures import \*

#from VisionEgg.ResponseControl import \*

from string import \*

import Image, ImageDraw # Python Imaging Library (PIL)

import os, sys

import random

import os.path

import pygame.mixer, pygame.time

mixer = pygame.mixer

times = pygame.time

#############################

# Files, directories etc. #

#############################

DBG = 1 #Set 1 to print info/debugging statements

TRN = 1 #Set 1 for training and 0 for testing

WINDOWS = 0 #Set 1 on Windows and 0 on Linux/Mac

# Files, images and directories #

label\_file\_name = 'tap\_vegg.dat'

img\_dir\_name='images'

############################

# Initilization #

############################

base\_dir = os.getcwd()

log\_file\_name = time.strftime ('%m-%d-%Y\_%Hh-%Mm.txt');

prog\_name = split (os.path.basename (sys.argv[0]), '.')

log\_file\_name = str ('log\_trn\_') + log\_file\_name

if WINDOWS:

tmp\_dir = base\_dir + str ('\..')

img\_dir = '\\'.join([tmp\_dir, img\_dir\_name,'']);

else:

tmp\_dir = base\_dir + str ('/..')

img\_dir ='/'.join([tmp\_dir, img\_dir\_name, ''])

if DBG:

print '\n============================ Initialization =========================='

print ' Label file: %s' % (label\_file\_name)

print ' Log file: %s' % (log\_file\_name)

print ' Directory current: %s' % (base\_dir)

print ' Directory images: %s' % (img\_dir)

print '========================================================================\n'

if DBG: print 'Opening files ',

label\_file=open(label\_file\_name, 'r')

log\_file=open(log\_file\_name,'w')

if DBG: print '...done.'

# Some initial variable definitions

TR\_n = int(label\_file.readline())

TR\_len = 2 # in seconds

sec\_n = TR\_n\*TR\_len

current\_block = 0

block\_array = []

visual\_stim = []

TR\_array = []

# Reading label

if DBG: print 'Reading labels ',

for line in label\_file:

b, v, r, xxx = line.split(";", 3)

block\_array = block\_array + [b]

visual\_stim = visual\_stim + [v]

TR\_array = TR\_array + [r]

label\_file.close

if DBG: print '...done.'

img\_name = '%scircle.png' % (img\_dir)

txtr\_circle = Texture(img\_name)

#choose a desired audio format

mixer.init(42000)

stims="""

Click.wav

"""

# print block\_len\_array

#################################

# Initialize the various bits #

#################################

# Initialize OpenGL graphics screen.

screen = get\_default\_screen()

# Set the background color to white (RGBA).

screen.parameters.bgcolor = (0.0,0.0,0.0,0.0)

screen\_half\_x = screen.size[0]/2

screen\_half\_y = screen.size[1]/2

ss0 = screen.size[0]

ss1 = screen.size[1]

scrn\_x = [0.5]

scrn\_y = [0.5]

text\_instruct\_1 = Text(text="Finger Tapping Paradigm:",

color=(1.0,0.0,0.0),

position=(screen\_half\_x,screen\_half\_y+120),

font\_size=50,

anchor='center')

text\_instruct\_2 = Text(text="Auditory and Visual Stimulus",

color=(1.0,1.0,1.0),

position=(screen\_half\_x,screen\_half\_y+40),

font\_size=40,

anchor='center')

#stimTextVis = Text(text="A",

# color=(1.0,1.0,1.0),

# position=(screen\_half\_x+250,screen\_half\_y+50),

# font\_size=40,

# anchor='center')

#stimTextAud = Text(text="A",

# color=(1.0,1.0,1.0),

# position=(screen\_half\_x+250,screen\_half\_y),

# font\_size=40,

# anchor='center')

fixation = Text(text="A",

color=(1.0,1.0,1.0),

position=(screen\_half\_x,screen\_half\_y),

font\_size=100,

anchor='center')

circle = TextureStimulus(texture=txtr\_circle,

internal\_format = gl.GL\_RGBA,

max\_alpha = 1.0,

size = (400,300),

position = (screen\_half\_x,screen\_half\_y),

anchor='center')

# Create a Viewport instance

viewportIntro = Viewport(screen=screen)

viewport = Viewport(screen=screen, stimuli=[text\_instruct\_1, text\_instruct\_2, fixation, circle])

#viewport = Viewport(screen=screen, stimuli=[text\_instruct\_1, text\_instruct\_2, stimTextVis, stimTextAud, fixation, circle])

p = Presentation(

go\_duration=(sec\_n,'seconds'),

trigger\_go\_if\_armed=0, #wait for trigger

viewports=[viewport,viewportIntro])

# Record key presses to text file, end stimulus with 'esc'

def keydown(event):

if event.key == pygame.locals.K\_ESCAPE: # Quit presentation 'p' with esc press

p.parameters.go\_duration = (0, 'frames')

# calculate a few variables

next\_TR\_time = 0

prev\_TR\_time = 0

next\_vis\_time = 0.0

prev\_vis\_time = 0.0

first\_loop = 1

start\_time = 0

TRcount = -1

circleTxtr = txtr\_circle

alpha\_min = 0.0

alpha\_max = 1.0

currBlock = 0

prevBlock = block\_array[0]

currTextVis = ''

currTextAud = ''

currfixation = ''

currFreq = 0.0

currVisTime = 0.0

displaytime = 0.0

currVisStim = 0

framerate=60

count=0

#initialize log file

if TRN:

log\_file.write("# LOGFILE: %s, %s (TRAINING)\n" %(prog\_name[0], time.strftime ('%m-%d-%Y %H:%M')))

else:

log\_file.write("# LOGFILE: %s, %s (TESTING)\n" %(prog\_name[0], time.strftime ('%m-%d-%Y %H:%M')))

log\_file.write("# button press;time;currVisStim;frequency;\n")

def play(file):

sound=mixer.Sound(file)

channel=sound.play()

while channel.get\_busy():

times.wait(10)

keystroke\_left = 0

keystroke\_right = 0

def getState(t):

global TR\_len, next\_TR\_time, prev\_TR\_time, currBlock

global first\_loop, start\_time, TRcount

global prevBlock, currTextVis, currTextAud, currfixation, currFreq, currVisTime, currVisStim, displaytime

global prev\_vis\_time, next\_vis\_time, count, circleTxtr, alpha\_min, alpha\_max, keystroke\_left, keystroke\_right

mod = 0

currTime = 0

if (first\_loop == 1) & (p.parameters.trigger\_go\_if\_armed):

first\_loop = 0

start\_time = VisionEgg.time\_func()

if t > next\_TR\_time:

TRcount = TRcount + 1

prev\_TR\_time = next\_TR\_time

next\_TR\_time = next\_TR\_time + TR\_len\*(int(TR\_array[TRcount]))

currBlock = int(block\_array[TRcount])

if currBlock == 1:

currFreq = int(visual\_stim[TRcount])

currVisTime = 1/float(currFreq)

currfixation = ''

# currTextVis = 'Visual: %s Hz' % str(currFreq)

# currTextAud = 'Auditory: %s Hz' % str(currFreq)

displaytime = 1.0\*currVisTime

count = -1

else:

currVisStim = 0

currfixation = '+'

currTextVis = ''

currTextAud = ''

count = (count + 1)

if currBlock == 1:

mod = count % (framerate/int(currFreq))

if mod == 0:

currVisStim = 1

circleTxtr = txtr\_circle

count = 0

currTime = t

play('Click.wav')

if t > currTime + displaytime:

currVisStim = 0

log\_file.write("%d; %d; %5f; %d; %5f\n" %(keystroke\_left, keystroke\_right, t, currVisStim, currFreq))

keystroke\_left = 0

keystroke\_right = 0

#print '%5f; %d; %5f' %(t, currVisStim, currFreq)

return 1

#Record key presses to text file, end stimulus with 'esc'

def keydown(event):

global keystroke\_left, keystroke\_right

if event.key == pygame.locals.K\_1 or event.key == pygame.locals.K\_2:

keystroke\_left = 1

if event.key == pygame.locals.K\_3 or event.key == pygame.locals.K\_4:

keystroke\_right = 3

if event.key == pygame.locals.K\_ESCAPE:

p.parameters.go\_duration = (0, 'frames')

# Quit presentation 'p' with esc press

def getcircleTexture(t):

global circleTxtr

return circleTxtr

def getcircleAlpha(t):

global currVisStim, alpha\_min, alpha\_max

alpha = 0

if currVisStim == 0:

alpha = alpha\_min

elif currVisStim == 1:

alpha = alpha\_max

return alpha

def myStimTextVis(t):

global currTextVis

return currTextVis

def myStimTextAud(t):

global currTextAud

return currTextAud

def myfixation(t):

global currfixation

return currfixation

#######################

# Define controllers #

#######################

###### Create an instance of the Controller class

trigger\_in\_controller = KeyboardTriggerInController(pygame.locals.K\_5)

stimulus\_on\_controller = ConstantController(during\_go\_value=1,between\_go\_value=0)

stimulus\_off\_controller = ConstantController(during\_go\_value=0,between\_go\_value=1)

state\_controller = FunctionController(during\_go\_func=getState)

#stimTextVis\_controller = FunctionController(during\_go\_func=myStimTextVis)

#stimTextAud\_controller = FunctionController(during\_go\_func=myStimTextAud)

fixation\_controller = FunctionController(during\_go\_func=myfixation)

circleTexture\_controller = FunctionController(during\_go\_func=getcircleTexture)

circleAlpha\_controller = FunctionController(during\_go\_func=getcircleAlpha)

#p.add\_controller(flashing,'on', stimulus\_on\_controller )

#############################################################

# Connect the controllers with the variables they control #

#############################################################

p.add\_controller(p,'trigger\_go\_if\_armed',trigger\_in\_controller)

p.add\_controller(text\_instruct\_1,'on', stimulus\_off\_controller)

p.add\_controller(text\_instruct\_2,'on', stimulus\_off\_controller)

#p.add\_controller(stimTextVis,'on', stimulus\_on\_controller)

#p.add\_controller(stimTextVis,'text', stimTextVis\_controller)

#p.add\_controller(stimTextAud,'on', stimulus\_on\_controller)

#p.add\_controller(stimTextAud,'text', stimTextAud\_controller)

p.add\_controller(fixation,'on', stimulus\_on\_controller)

p.add\_controller(fixation,'text', fixation\_controller)

p.add\_controller(circle,'on', stimulus\_on\_controller)

p.add\_controller(circle,'max\_alpha', circleAlpha\_controller)

p.add\_controller(circle,'texture', circleTexture\_controller)

p.add\_controller(p, 'trigger\_go\_if\_armed', state\_controller)

p.parameters.handle\_event\_callbacks = [(pygame.locals.KEYDOWN, keydown)]

#######################

# Run the stimulus! #

#######################

p.go()

log\_file.close