

BRIAN MINGUS / GRADUATE THESIS

How practice becomes skill.

An Investigation of Motor Chunking

Cognitive psychology / CU Boulder



MASTER'S THESIS | 2012 / THESIS MANUSCRIPT

An Investigation of Motor Chunking

Brian Mingus

University of Colorado Boulder

Experiments examine how people group movement sequences into chunks and whether chunk size is constrained by working-memory capacity.

The supplied manuscript is dated 2012; the degree was awarded later.

Collection cover added September 2026. Original manuscript follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

An Investigation of Motor Chunking

by

B. J. Mingus

B.A., University of Colorado at Boulder, 2008

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Master of Arts
Department of Psychology
2012

BRIAN MINGUS / BM-A9E0CEBE342A

This thesis entitled:
An Investigation of Motor Chunking
written by B. J. Mingus
has been approved for the Department of Psychology

Randy O'Reilly

Prof. Matt Jones

Prof. Tor Wager

Date _____

BRIAN MINGUS / BM-A9E0CEBE342A

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Mingus, B. J. (M.A., Cognitive Psychology)

An Investigation of Motor Chunking

Thesis directed by Prof. Randy O'Reilly

Chunking is widely theorized to be a fundamental building block of cognition, underlying language, motor control and general problem solving abilities. Recent studies of chunking have focused on a sequence learning paradigm known as the Discrete Sequence Production task, which is thought to reliably elicit chunks. These chunks have been documented in sequences that are executed at remarkable speeds, greatly exceeding the documented abilities of working memory in simple reaction time task contexts. These chunks further greatly exceed working memory capacity, with previously documented chunk sizes consisting of up to ten elements. Here, we question whether what other researchers have identified as chunks really are chunks, demonstrate that subjects can create putative chunks on demand, and further demonstrate that subjects may be able to create chunks of up to size 20 with extensive practice, with the salient prediction that there is no limit to chunk size. These findings question theories of chunking that rely on working memory, question whether chunk detection algorithms are tagging random features as chunks, and point to directions for future research that might clarify these issues.

Contents

Chapter	
1	Introduction
	1
2	Pilot Study
	6
2.1	Methods
	6
2.2	Results
	7
3	Dual Task Experiment
	13
3.1	Methods
	13
3.2	Results
	15
4	Discussion
	21
5	Appendix 1: Experiment Python Code
	23
6	Appendix 2: Analysis Python Code
	30
	Bibliography
	41

Chapter 1

Introduction

Chunking is thought to be a fundamental building block of cognition, underlying language, motor control and general problem solving ability. More specifically, chunking has been defined as the ability to compile a sequence of responses into a single motor chunk whose size has recently been found to correlate with working memory capacity (WMC) [1], but has also been found to be as large as size seven [10, 3] and even ten [9]. This compilation process theoretically allows one to think at a higher level; rather than executing each lower level action, one can instead initiate the chunk which is then executed automatically, freeing up the mind to think of other things. Going further, chunking may allow one to predict the outcome of a chunk executed in the present context with the intention of accomplishing some goal, without ever actually executing the chunk. Increasing sophistication in a task domain would thus correspond to an increased ability to model the effects of increasingly abstract actions on the world. This can be thought of intuitively in the context of chess; whereas beginners must painstakingly execute one movement at a time, more advanced players can play at a higher level, intuitively assessing the value and envisioning the board configuration that is the result of a chunked action consisting of multiple moves into the future.

Recent theories and neuroimaging studies have suggested that chunking is composed of both a segmentation and a concatenation process [7]. The segmentation process is thought to reflect the contribution of visuospatial working memory (WM), which segments an incoming sequence into chunks that in principle should not exceed WMC. The concatenation process is thought to occur in sensorimotor coordinates and is responsible for ensuring that the sequential execution of

chunks occurs fluidly, with minimal disruption in motor performance due to chunk boundaries. Another neuroimaging study found converging evidence for this theory suggesting that motor sequence learning follows a trajectory from visuospatial prefrontal cortex-basal ganglia (PFC-BG) to sensorimotor PFC-BG to the ultimate locus of the motor skill in a direct path through parietomotor cortex [8].

This data supports at least one model with three phases: in the early phase WM is filled up to some point not to exceed WMC, creating stronger associations for items within a chunk than for items between chunks; in the middle phase WM transitions to executing “pointers to chunks” (see [5]), with the actual execution of the chunk occurring in sensorimotor PFC-BG; in the late phase of learning, although the chunks may still be available for use to PFC-BG, the sequence has also been reconsolidated into a direct parietomotor cortical path that supports the fastest possible execution next to reflexive action and is WM independent.

This model is, however, too simple to explain at least two additional lines of evidence. The first is that chunks larger than modern estimates of WMC have been documented, as mentioned previously. While a recent theory suggests that WMC ought to be defined not in terms of items but in terms of the maximally compressed representation of a set of items [2] up to about 10, most recent chunking studies have carefully selected sequences that are hard to compress. The second line of evidence comes from the simple reaction time (sRT) paradigm, in which it has been found that, although there are many modulating factors, a reasonable estimate for the fastest that a person can respond to a visual stimulus is approximately 200 ms [4]. Many studies of chunking have documented reaction times considerably faster than this value for durations that exceed the modern estimate of WMC of 3-4. For instance, even when averaging across subjects [11] document a sequence of five reaction times (RTs) that is less than 160 ms.

An extension of our model might be able to explain chunks larger than WMC. If the result of the sensorimotor PFC-BG segmentation process is not only to concatenate chunks in an online fashion, but to work to minimize the number of chunks by moving the first element of successive chunks onto the end of previous chunks, then chunks could putatively grow in size so as to exceed

WMC. This would require WM to execute a pointer to the second element of the chunk whose first element was just reassigned to the end of another chunk. A model analogous to this one would need to be invoked to explain the finding of [10] in which, over the course of five days of training, chunks gradually asymptoted to approximately size seven. Interestingly, chunk size remained stable for the next five days of training, indicating that the process could somehow be capacity limited according to Miller's more famous definition of WMC [6], and leaving the chunk of size ten found by [9] as an anomaly. Alternately, this finding by [10] may not generalize well seeing as most studies of chunking have used much shorter sequences (the authors used a sequence of length 32) and have tended to find that chunk size is constant with decreases in RTs.

What of RTs faster than have been documented in the sRT paradigm? Any explanation for the continuous stream of fast RTs seen during "chunking mode" [11] is likely to involve associations between items. Assuming our model in which WM executes pointers to chunks, this model predicts that there are associations between the pointers themselves leading to their faster execution. A natural extension of our model suggests that the pointers themselves become chunks, and that the entire sequence could become one pointer to pointers to chunks, or a chunk of chunks, or, in other words, what is effectively one big chunk. If this were the case, then after extensive amounts of training we might expect to see chunk boundaries disappear, as they are presumably the result of WMC limitations. However, this is not necessarily the case - for instance, the same limitation on visuospatial PFC-BG capacity could also be an analogous capacity limit on sensorimotor PFC-BG. Further, apparent chunk boundaries in RTs could conceivably persist to the final representation of the sequence in parietomotor cortex if we assume that the PFC-BG processes that were initially responsible for executing the sequence served to train up a timing profile in parietomotor cortex that is now nothing more than a confusing reflection.

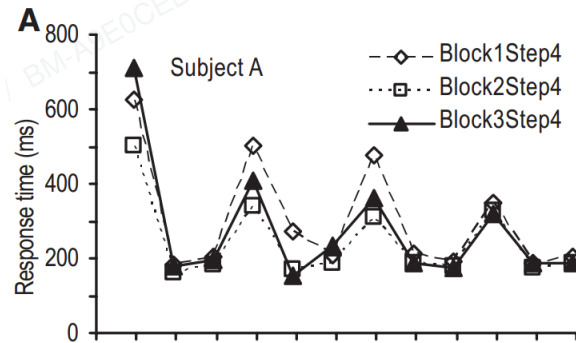
The space of models that can explain chunking is large. For instance, both hierarchical models (such as pointers to pointers to chunks) and "flat" models (where each item is optionally connected to every other item) can be parameterized so as to produce the same reaction time profile. Further, there could be hierarchical or flat models at any stage of the consolidation trajectory, including

in parietomotor cortex. This indeterminacy makes it challenging to study, for instance, whether or not the “ripples” we see in the RT data are really chunks, or whether they merely reflect the prior existence of chunks, or something different altogether. One possible manipulation is to reorganize sequences along subject specific chunk boundaries to see if the increase in RT due to the reorganization is less than the increase seen when sequences are organized along random boundaries. However, one could simply argue that the reorganization along chunk boundaries triggers the use of chunks that were used at an earlier stage of the sequence but that are no longer being used, with what’s being seen in the RTs being a misleading reflection. Further, one could argue that of course the increase would be greater in the random boundary condition, for it includes the pauses due to the previous chunk boundaries and due to the reorganization along random boundaries.

Designing experiments to differentiate between the multitude of possible models underlying chunking is challenging, as is the perhaps more basic problem of identifying chunks in RT data. While the standard definition of a chunk is a long RT followed by a burst of fast RTs (see Figure 1.1), this definition is unsatisfying. This is primarily because this definition of a chunk generalizes to the identification of putative “chunks” in any RT profile, even that produced by a random walk. Going further, while there is no known limit to how fast a person can become at a task with extended practice, there is presumably a limit to how fast a person can execute a sequence with WM involvement. Thus, once a sequence has been reconsolidated into parietomotor cortex and is executed faster than could possibly be explained by WM, this definition of a chunk will always identify chunks irrespective of whether chunks are actually used.

Despite this confusion identifying chunks in RT data is still of great interest. It seems likely that the “chunking mode” documented by [11] does reflect a WM algorithm at early and middle stages of learning that produces RTs faster than have previously been documented in WM studies. This faster execution is likely due to context, where prior context leads to the anticipation of future responses, facilitating their execution. However, as appears to be indicated by [8], it is also likely to be the case that in late stages of learning sequence execution becomes WM-independent. Thus, we are left with the question as to exactly what defines early, middle and late stage learning, and

Figure 1.1: An ideal chunk profile where chunk size correlates with WMC taken from [1]. The chunks are elements 1-3, 4-6, 7-9, 10-12, and the subject's measured WMC is 3.



whether any or all the stages actually use chunks, and whether late stage learning merely reflects chunks in early stages.

While we cannot hope to answer these questions in the present work, two experiments have been run in order to shed light on some aspects of them. In the first, we had two subjects practice a 20-element sequence for one hour per day for 10 weekdays over a two week period. This experiment is meant to probe just how fast subjects can become at entering the sequence, and to give us a glimpse of what a “chunk” looks like in such a scenario. In the second, a challenging dual task is mixed with the learning of a sequence. Under the assumption that chunking involves WM, it is hoped that the dual task will in some way interfere with chunking.

Chapter 2

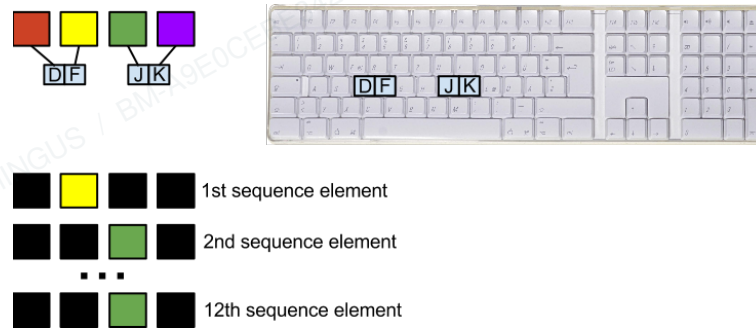
Pilot Study

2.1 Methods

In order to observe chunking phenomena in action a pilot study was run. The experimental procedures were approved by the Institutional Review Board of the University of Colorado at Boulder. Two subjects between the ages 18 - 29 were recruited via flyers posted in the Muenzinger building on the University of Colorado at Boulder campus and were paid \$20 per hour. They performed a Discrete Sequence Production task [1, 11] for one hour per day for ten consecutive weekdays, M-F. The task begins with a 3-2-1 countdown, where each number is displayed for one second. During this time subjects were shown plots with the first indicating the sum of RTs per trial for the current day and the second showing this figure across all days to give a sense of their improvement. They were instructed to learn to enter the sequence as fast as possible. As soon as the 1 has been shown for one second four large boxes are presented horizontally across the middle of the screen. As soon as the correct key is pressed the current box turns black and the next box turns its associated color, corresponding to a response-to-stimulus (RSI) interval of 0, which is standard for DSP tasks [11]. If an incorrect key is pressed there is no change in the display, thus, the correct key must be pressed in order to progress to the next element of the sequence. The subjects used a USB keyboard precise to one millisecond and sat unconstrained with their heads between 1' and 2' from the screen. They used their left and right middle and pointer fingers to press keys associated with either D, F, J and K, respectively.

The specific sequence subjects practiced is *yellow, green, red, green, yellow, purple, red, pur-*

Figure 2.1: Basic Discrete Sequence Production Design



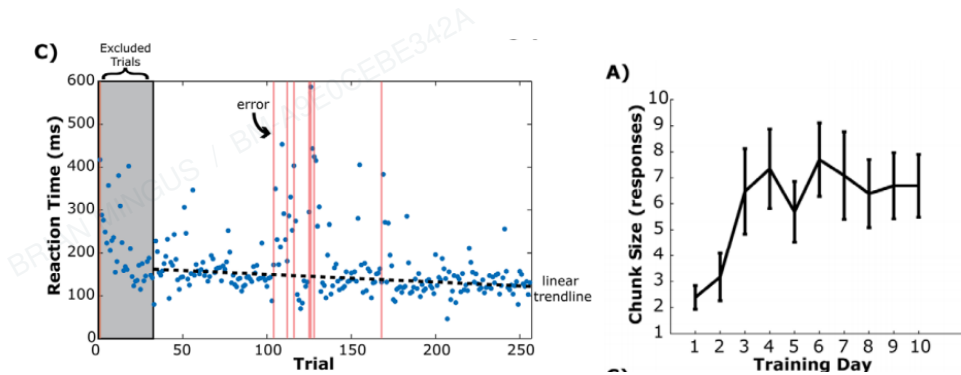
ple, green, red, yellow, purple, green, purple, red, yellow, red, purple, yellow, green. The sequence was chosen based on the criteria listed in [1], namely that 1) each element occurs with equal probability; 2) there is no fixed grouping pattern such as DFJK; 3) there are no runs of three (DDD) or trills (DFDF). The bi, tri and quadrigram structure of the sequence for elements that occur more than once is as follows: KF, FJ, KJ (x2), JD, DF, FK (x3), KFJ, JDF, DFK (x2), JDFK (x2).

2.2 Results

At the time that the pilot study was run the 20-element sequence was longer than had been published and 10 days of training was also more training than had been published. Shortly thereafter, however, [10] was published which contains an analysis of a 32-element DSP sequence with 10 days of practice. They document that over training subjects become faster and more accurate, with increases in speed and accuracy uncorrelated, that they become increasingly explicitly aware of the sequence over days of training and that the size of chunks increases over the first five days of training from two to seven (with a standard error of 6-9). A plot of chunk size from their paper can be seen in Figure 2.2, along with an RT plot from one subject late in training demonstrating that subjects become extremely fast, with the mean RT on one trial apparently below 50 ms.

We will now consider the results from this pilot study. First, in Figure 2.3.a we can see that both subjects asymptote to approximately the same trial time of 2 seconds, with both subjects

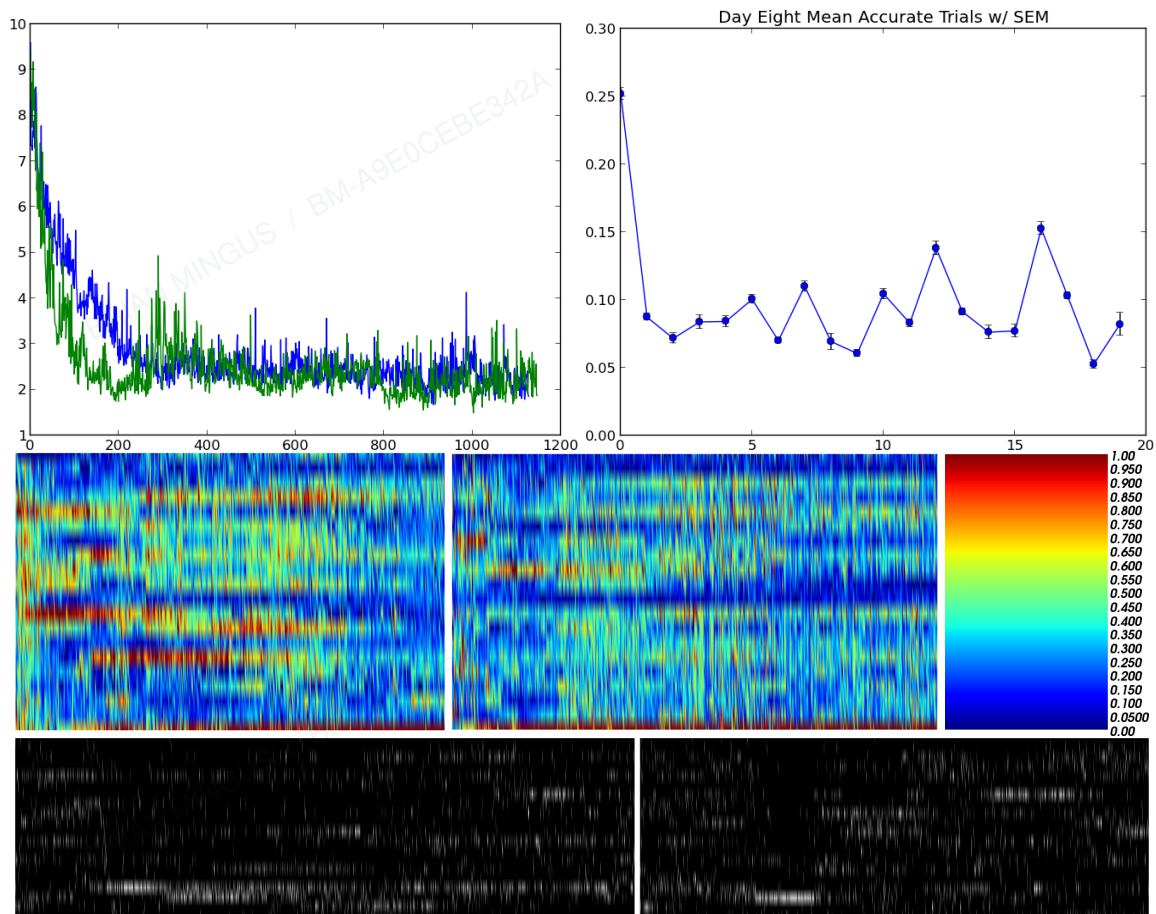
Figure 2.2: Figures from [10]. Figure 5c (left) and 7a (right).



reaching approximately 1.5 seconds on their best trials, corresponding to a mean RT of 75 milliseconds. It is interesting to compare this RT to several real world examples. For instance, the song Through the Fire and Flames by DragonForce has a well-known guitar section called Red Snake which is played at 27 notes per second, or one action every 37 milliseconds. Professional Starcraft player Park Sung-Joon can execute 818 actions per minute, or one action every 73 milliseconds. Finally, the typing speed world record of 216 words per minute set by Stella Pajuna corresponds to 1 character every 56 milliseconds. If we further consider that the first RT of these subjects is generally slower at around 225 milliseconds, and that the fastest observed sequence was actually 1.45 seconds, then the mean RT for all but the first RT is less than 65 milliseconds. Thus, these subjects came to execute the sequence at a rate that gets close to some of the fastest consecutive RTs that have ever been recorded, not counting the apparent flukes in their RTs that are faster than 1 millisecond (faster than even the upgraded millisecond sensitive keyboards can detect). While it is challenging to prove that WM is not involved during these feats, and that the fast RTs are not due to planning per se, this seems to be evidence in favor of the subjects learning to execute the sequence with the involvement of WM being limited to the its role in initiating the sequence, reflected in the slow first RT.

Next, let us turn out attention to the RT distributions in Figure 2.3.c. Each of these plots represents the RTs for a given subject, with each trial independently normalized between zero and

Figure 2.3: **a)** Trial time across all days for both pilot subjects in seconds. **b)** RTs on day eight for the first subject with SEM indicators. **c)** RTs for subjects across all ten days of training on successful trials. Each trial starts with the first sequence element on the bottom and ends with the 20th sequence element on the top. The trials move from left to right. Each trial was independently normalized from zero to one so that trends independent of RT could be observed. **d)** The position of the first error in error trials for each subject.



one so as to preserve patterns controlling for decreases in RTs. The standard definition of a chunk is that there is a relatively slow RT followed by a burst of fast RTs that can be as short as one element. A “visual regression” of this data reveals that there are many instances of redder features with substantial horizontal continuation that border bluer features with substantial horizontal continuation. The more consistent a redder feature bordering a bluer feature the higher the power with which a chunk detection algorithm might plausibly call that a chunk boundary. Note that many of these features persist in the data from early days of training, when RTs were slow, to late days of training, when RTs were fast. It is also notable that some chunks are transient, as if there were an optional “chunk library” that subjects can idiosyncratically recruit from, and that our power to detect “chunks” in this data allows us to detect many more chunk-like features than might be explained by a correlation with WMC.

Next we will shift to Figure 2.3.b. Here we can see the RTs averaged across accurate day eight trials for the first subject with SEM bars demonstrating very low variance for each point, and that all except the first RT are faster than about 160 ms. If we implement a visually intuitive chunk detection algorithm which follows the definition of a slow RT followed by a burst of fast RTs, then it might result in the following chunks: 0-19 (the entire sequence), 0-4, 5-6, 7-9, 10-11, 12-15, 16-19. Not counting the entire sequence as a chunk, this is six chunks with sizes of 5, 2, 3, 2, 4 and 4, and a mean size of 3.33. This is close to 3.5, which is inbetween the modern estimate of WMC of 3-4 elements depending on the subject [1], (an estimate of which [10] seems to be unaware). However, this chunk algorithm must be considered highly speculative - it predicts that the slowdown in RT due to refilling WM is between 25 - 75 ms, and that this period of time decreases even as subjects become faster at the sequence, a progression to which there is no known limit.

Figure 2.4.a shows a surface plot for one subject in which each row along the long axis represents the mean RT for a different day. Here, the contiguous ridges in RT represent chunk boundaries and the plot reveals that there is some movement of them from day to day, although overall the patterns are fairly consistent. 2.4.b shows the mean of the errors by position over all 10 days and 2.4.c shows the correlation of RTs and errors demonstrating that there is no simple

direct relationship between the two.

Figure 2.4: **a)** Top right matrix: Sequence elements one through twenty (long axis) averaged across days one through ten (short axis). The ridges are candidate chunk boundaries that are contiguous across days. **b)** Middle matrix: Position of errors by day. **c)** Bottom left matrix: Correlation of these two matrices, showing that errors and chunk boundaries are not directly correlated.

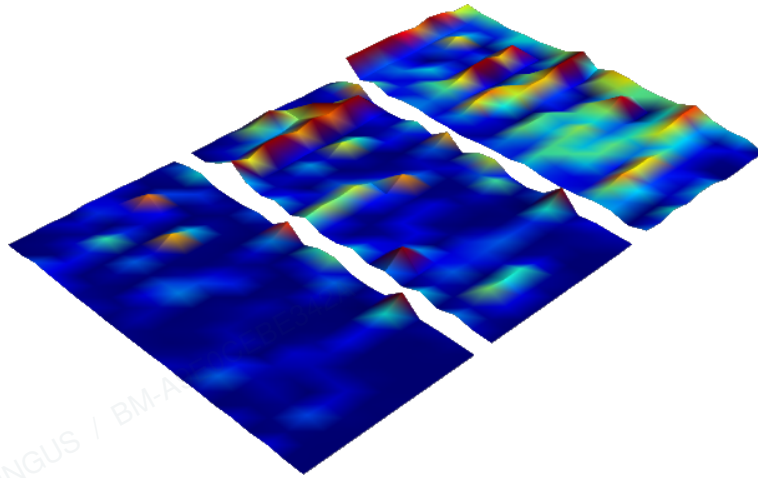


Figure 2.5 is due to a separate, third pilot subject who was already aware of chunking before entering the study. This subject was used to test various hypotheses, including the one demonstrated here: that subjects can willingly create chunks of a given size. While this subject found creating a chunk of size ten in one hour of training to be challenging or impossible (not shown), they were easily able to create five chunks of size four, using the same sequence as the pilot subjects above.

Figure 2.6 allows one to ascertain the amount of consistency in RTs by plotting RT averages over various windows. Several putative phenomena can be observed here, including chunk consistency, chunk movement and chunk splitting,

Figure 2.5: A separate pilot subject who knew about chunking was asked to create five chunks of four in the 20-element sequence of the pilot study.

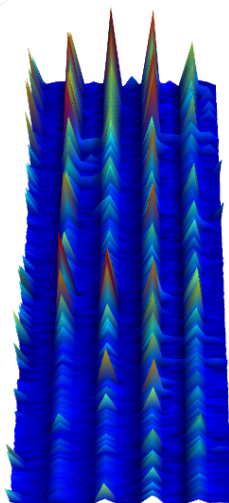
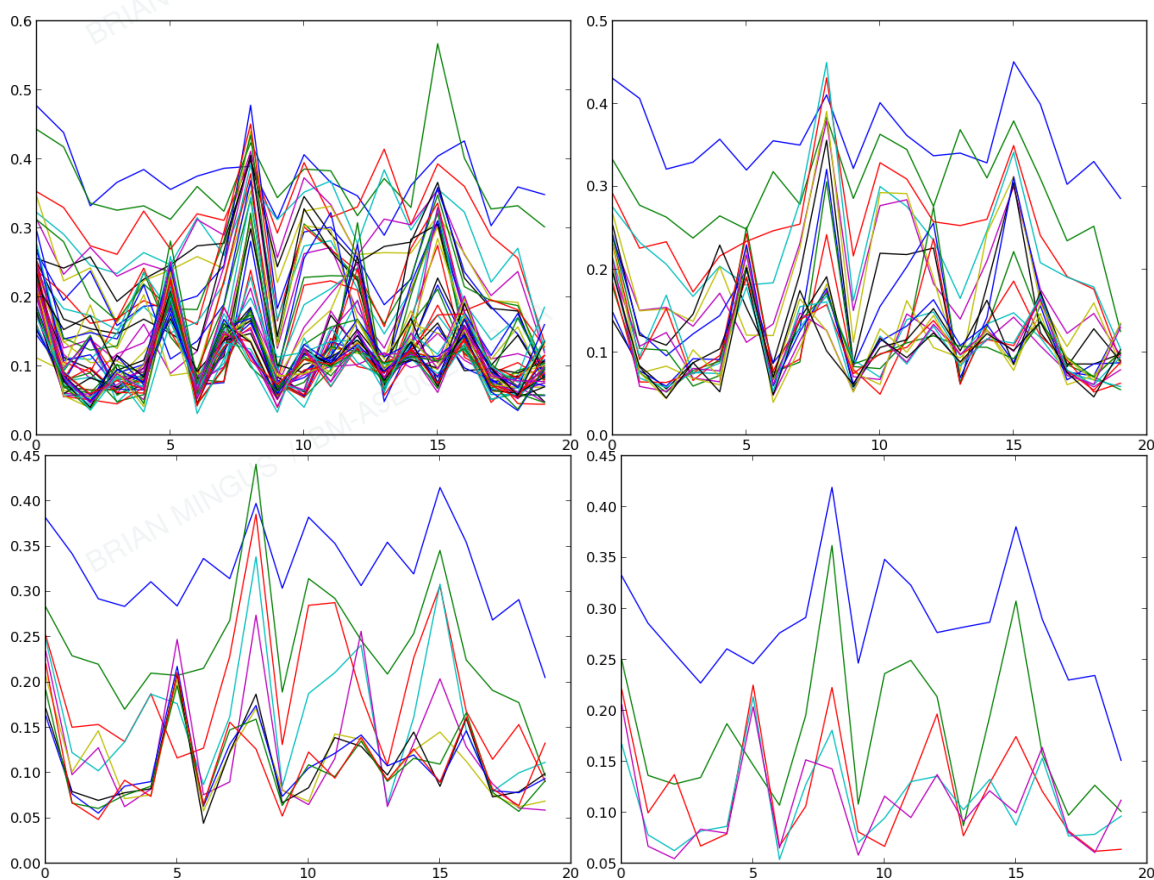


Figure 2.6: Averaging over every 10, 25, 50 and 100 trials respectively for one subject in the pilot. Lower lines generally correspond to later periods of training as the average RT lowered.



Chapter 3

Dual Task Experiment

3.1 Methods

In order to probe the contribution of WM to chunking a dual task paradigm was run. The experimental procedures were approved by the Institutional Review Board of the University of Colorado at Boulder. 42 subjects were recruited, however, six subjects were dropped due to failure to complete all three days. In this experiment there are two conditions. In both conditions subjects practice the same 12-element sequence as used in [1], which is *purple, yellow, green, purple, red, green, red, yellow, green, red, yellow, purple*. In the first condition, completed by 19 subjects, the sequence is practiced for one hour per day for two days. On the second day they practice the sequence under dual task conditions. The dual task manipulation is that in addition to executing the sequence subjects had to count how many times the box for an element had a thick black border, which happened a random number of times not to exceed 9 per trial, and indicate this count at the end of a trial. In the second condition, completed by 17 subjects, the sequence was practiced in a dual task paradigm for two days followed by one day without it. There were differences between the DSP in the pilot study and here, namely that there was only a one second break between sequences and subjects were not shown any plots. However, they were given feedback as discussed below. Additionally, subjects could press the spacebar in the break between sequences to take a break of arbitrary duration to prevent fatigue, as even at a relatively low number of sequences per day in this experiment, i.e., 400, subjects are still pressing nearly 5,000 keys, with some subjects pressing up to 15,000 keys per hour.

We learned in the pilot study that there is a challenge with the standard DSP paradigm due to variability in where individual subjects tend to lie on the speed-accuracy tradeoff. Some subjects will tend to begin by trying to enter the sequence as quickly as possible without regard for accuracy, resulting in a few trials in which they are extremely fast and accurate, but not leaving enough data for us to confidently determine that they have in fact learned the sequence. Other subjects will tend to enter the sequence accurately and slowly, never bothering to get much faster so as to make fewer mistakes. Still other subjects might tend to perform the task “correctly,” entering the sequence accurately at first and allowing themselves to naturally begin to perform the sequence faster as they are able to without sacrificing accuracy.

In order to encourage subjects to perform the task correctly we modified the DSP paradigm to make it somewhat more like a video game. The first manipulation is that subjects can gain levels; they start at level 0 and can (theoretically, but not practically) get to level 50. The current level is shown in the lower left corner of the screen during the 1 second break between sequences. It is shown in red with a “-1” if they lost a level, in black with a “-” if their level stayed the same and in blue with a “+1” if they gained a level. Level 0 corresponds to subjects needing to press all 12 keys with each individual RT being less than 500 milliseconds. In order to gain a level subjects must both enter the sequence accurately and have all RTs less than the current threshold. When a level is gained the threshold for entering keys decreases by 10 milliseconds. If the subject makes a mistake when entering the sequence they lose up to 1 level per sequence. In addition to this leveling manipulation, subjects were also given auditory feedback as to how fast they pressed the keys. If an individual key was pressed faster than the threshold determined by their current level then they heard a high pitched tone (1500 Hz), whereas if it was slower than the threshold they heard a low pitched tone (750 Hz). In the dual task manipulation a level could only be gained if subjects were faster than the threshold determined by their current level, entered the sequence accurately and correctly counted the number of boxes. A level was not lost due to answering the dual task incorrectly, however, subjects could not gain a level unless they did answer it correctly.

3.2 Results

The second experiment is designed to test whether adding a second task that occupies WM interferes with chunking. In theory, if subjects are obligated to use WM to enter the sequence then the dual task will prevent them from getting faster than 200 ms and it will additionally prevent them from chunking. However, there are a number of issues with this approach, including that subjects may learn to execute the two tasks as a single task [12] and that subjects may use a strategy such as “retrospective counting” that does not actually involve WM in an online counting sense, but instead relies on retrospective analysis of distributed memory traces after the boxes have been seen, or a sense of familiarity. Additionally, although some subjects in this experiment get as many trials of practice in a single day as our pilot subjects got over ten days, they do not have the benefit of twelve nights of sleep between the first and last sessions. Thus, we may not be giving subjects enough practice to allow them to get fast.

An ideal analysis of the levels and RT data might involve a multivariate mixed effects model where the dependent variables are the parameters of a sigmoid fit to the data. Additionally, power analyses would be nice. However, given time constraints much simpler analyses of the data must suffice.

From Figure 3.1 we can clearly see that there is a difference between the two groups due to the dual task manipulation. There are several important interpretational caveats when analyzing the data in this figure, however. The first is that just because a subject has attained a level does not imply that they are always executing the sequence at or near the RT threshold for that level. It simply implies that they managed to execute the sequence that quickly once, and that they haven’t made any mistakes when entering the sequence since then that would cause them to lose a level. Second, since each subject executed a different number of trials per day, our power to detect an effect trails off as the number of trials per day increases. Third, in the dual task conditions just because a subject has a low level due to not getting the dual task correct does not necessarily mean that they are slow at entering the sequence.

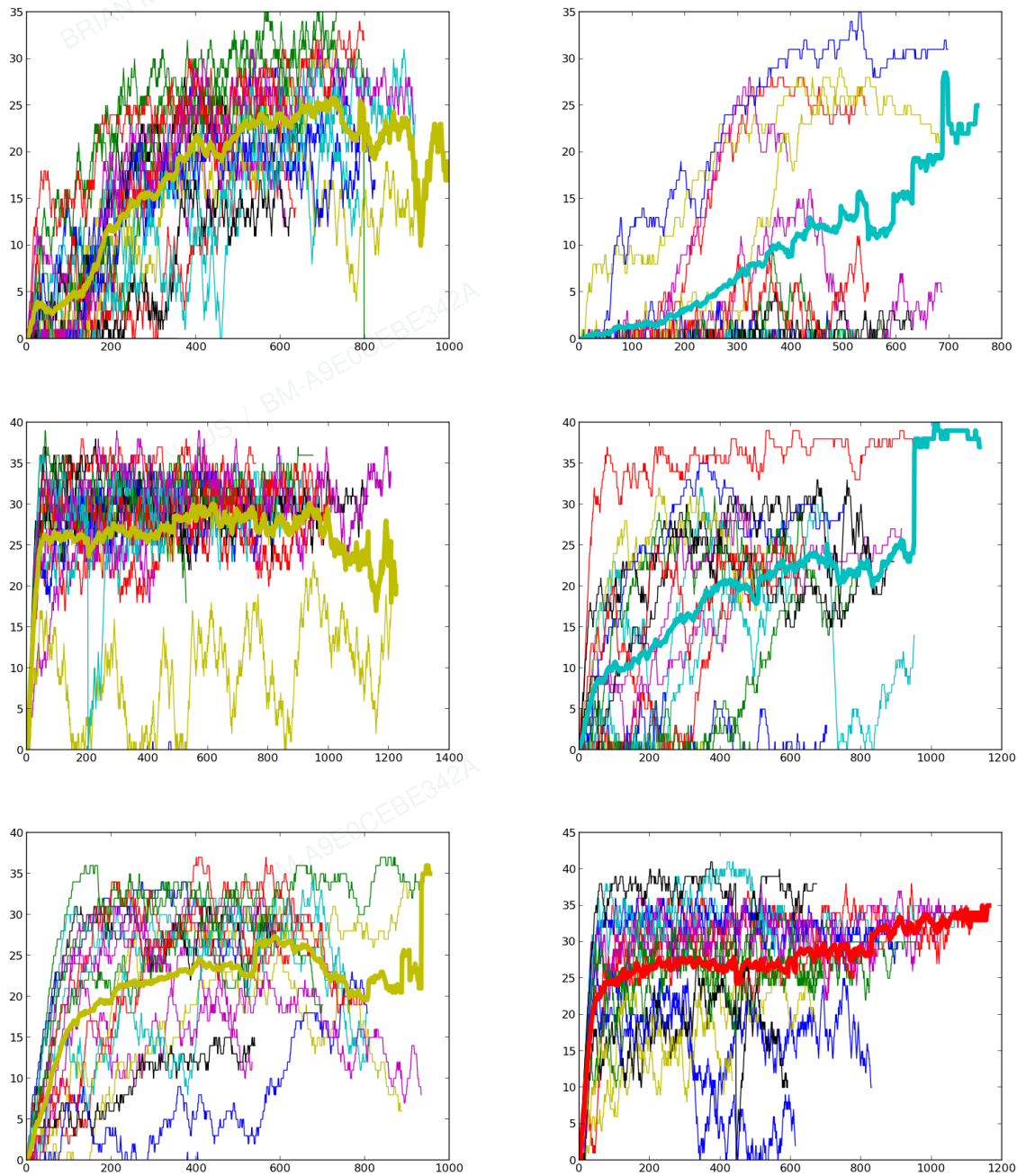
Nevertheless, it is convenient to work with the levels data as it is both clean and does in practice tend to strongly reflect the subject's actual ability to enter the sequence. In the following comparisons C1 and C2 will refer to conditions 1 and 2, respectively, and D1, D2 and D3 will refer to days 1, 2 and 3 respectively.

The difference between the conditions in the maximum level attained by subjects is $31.11 - 22.78 = 8.33$, a difference equivalent to the first condition being 83.3 milliseconds faster ($t = 11.82, p < 1e - 15$). This indicates that the dual task manipulation did indeed interfere with subject's ability to enter the sequence, and that in C1 the group mean maximum level passed the sRT threshold of 200 ms whereas in C2 it did not. However, a more apt comparison for testing whether the dual task manipulation interferes is perhaps between D1 of each condition and equivalently between D2 of each condition, after subjects had a night of rest. The difference between D1s is $28.33 - 12.05 = 16.27$, equivalent to 162.7 ms ($t = -5.3, p < 1e - 5$), and the difference between D2s is $34.5 - 23.82 = 10.67$ ($t = -10.67, p = .0015$), equivalent to 106.7 ms. Thus, in all three cases thus far analyzed, the dual task prevented subjects from being faster than 200 ms.

There are other interesting comparisons to make based on the levels, for instance, between D2 and D3 of C1 when subjects transitioned to the dual task, and between days D2 and D3 of C2 when subjects transitioned from the dual task. In the first case, the difference is $34.5 - 30.5 = 4$ ($t = -3.14, p = .006$). In the second case the difference is $23.82 - 33.06 = -9.2$ ($t = 6.75, p < 1e - 6$). Thus, when subjects transition to a dual task after practicing the sequence, they are 40 ms slower, which is not quite slower than 200 ms. When subjects transition from a dual task to just the sequence, they are 92 milliseconds faster, dropping to 170 ms.

While it is clearly the case that the dual task interferes with the speed with which subjects enter the sequence, what of chunking? In order to test for differences we will need a measure of chunking, and ideally we would be able to predict the effect of the dual task on chunking patterns. For instance, does the dual task lead to more or fewer chunks? One might predict that because subjects are distracted that the variability in their RTs expands, resulting in an associated increase

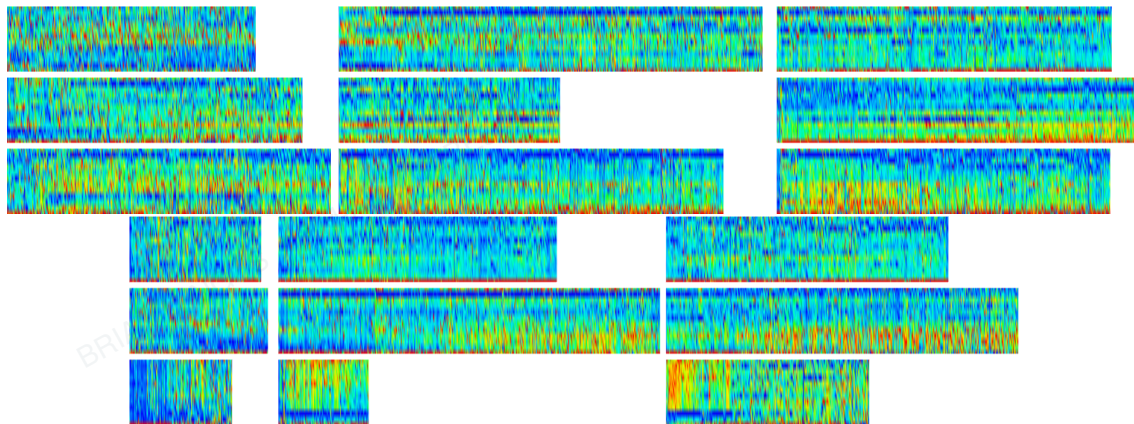
Figure 3.1: The left and right columns show the trajectory of levels for each subject in each condition, respectively, with group means in bold. Each day of the experiment is on a new row. In order to convert a level into milliseconds subtract it from 50 and multiply by 10.



in the number of chunks, or equivalently, an associated decrease in the average size of chunks. In order to test this hypothesis an automatic chunk detection algorithm was written. The Python source code for this algorithm can be seen on Page 3 of Appendix 2. The algorithm proceeds as follows:

Each RT is a member of one chunk. A chunk is defined by a long IKI followed by a burst of fast IKIs, each of which is not significantly different from each other. Each new IKI is compared to the maximum IKI within the current chunk, not counting the first element. As soon as an IKI is significantly larger than this maximum, the current chunk is ended and a new chunk is started. Notably, chunks of size one are possible in this algorithm and when an item is both significantly larger than both the item preceding it and following it. Example results of the chunking algorithm can be seen in Figure 3.3.

Figure 3.2: The top three rows correspond to three random subjects from C1, each of whose RTs are plotted across all three days. The bottom three rows contain three random subjects across days from C2. In each row the first RT of a trial is on the bottom and the last RT is on the top, and the first trial is on the left and the last trial is on the right. Features that have horizontal continuation are candidate chunk members. Notably, this data is visibly less chunky than the pilot data, and yet the existence of chunks is quite apparent.



Given this chunking algorithm it will be interesting to make the same comparisons as made for the maximum levels data above for the number of chunks per subject. The difference between conditions is $4.85 - 4.71 = .13$ which is not significant ($t = -.48, p = .62$). The difference between

Figure 3.3: Mean RTs for the same three random subjects as in Figure 3.2 (rows) from C1 (left column) and C2 (right column). The plot titles contain the classification of chunks from the algorithm described above for that subject for each day. Note: The right column titles errantly say C1, but are C2.



D1s is $5.68 - 4.3 = 1.4$ ($t = -3.13, p = .005$), indicating that the dual task leads to a significant reduction in the number of chunks, which is opposite to our prediction. The difference between D2s is $4.1 - 5.1 = -1$ ($t = 1.7, p = .1$) which is not significant, indicating that by the second day the groups did not differ in the mean number or size of chunks. The difference between D2 and D3 in C1 when the dual task is added is $4.1 - 4.8 = -.68$ ($t = 1.2, p = .24$) which is not significant. The difference between D2 and D3 in C2 when the dual task is removed is $5.2 - 4.75 = .36$ ($t = -.79, p = .43$) which is also not significant.

Thus, while the dual task clearly leads to differences in speed, it does not convincingly lead to differences in the tendency to chunk.

Chapter 4

Discussion

Herein we have demonstrated in the pilot study that in the presence of sequential context subjects can become much faster than has been documented in visual sRT studies, although that this is possible is by now common knowledge. The pilot also demonstrated that at least one subject was able to create chunks on demand, a finding that has not been previously documented and suggests that chunk creation is optionally under explicit control, but that it also has limits in that the subject struggled to create a chunk of size ten in one session. It has further been demonstrated that while there is remarkable consistency in chunking patterns, that they tend to change over days, and that they can even be idiosyncratic from trial to trial.

The dual task experiment demonstrates that adding a challenging dual task does affect RTs in the predicted directions. Further, the dual task prevents subjects from getting faster than the RTs documented in sRT studies, which could theoretically indicate that the dual task causes subjects to enter the sequence without the benefit of multiple items of context, corresponding to the “associative mode” of [11]. Interestingly, the dual task does not seem to affect chunking patterns, which could be due either to our chunk detection algorithm not picking up on chunks, chunks existing independent of whether a sequence is executed with WM or parietomotor cortex, or, perhaps more likely, consistent with [12] it is due to the dual task becoming entrained with the sequence learning task, or in other words, the lack of a difference is due to subjects accomplishing both tasks as a single task. This theory is consistent with the data in that a reduction in RT owing to the increased complexity of the unified task might be expected, but associated differences in

chunking might not be expected.

Future studies of chunking should focus on even more extensive periods of practice, allowing subjects to continue practicing until they cease to get faster. They should additionally focus on instructing subjects to create chunks of a specific size in order to test capacity limit theories which have confusingly linked chunking to WMC, despite chunk sizes produced by some chunk detection algorithms apparently greatly exceeding it. While dual task approaches to chunk investigation are appealing, their practical application suffers from not only the possibility of task integration, but from the speed with which subjects enter the sequence, which makes it challenging to argue that the dual task utilizes WM online rather than familiarity-based retrospective counting. Future studies should further limit the conclusions they draw from chunk detection algorithms, seeking first to prove that what is measured in the RT data really corresponds to chunks and defining what exactly a chunk is in terms of neural processes before making the strong claim that the RT ripples represent a unified action. Perhaps the most novel claim from the present work is that the pilot subjects seem to create a chunk of size 20, and that what otherwise appear to be chunks in their mean RT profiles may well be idiosyncratic ripples which do not meaningfully reflect on a neural process of interest. This theory could perhaps be tested by allowing the subjects even more days of practice, perhaps an entire month, and then reorganizing the sequence along the chunk boundaries defined by various author's chunk detection algorithms in order to demonstrate that random reorganizations slow subjects just as much.

Chapter 5

Appendix 1: Experiment Python Code

```

1  #!/bin/python
2  s = ' mingus '
3  day = 1
4  tagone = ' _one_chunk_of_twelve_two_tones_level_game_D ' + str(day)
5  dual_task = True
6  trialtime = 3600
7  play_sound = True
8  draw_plot = False
9
10 from psychopy import visual, core, event, sound, gui
11 from time import sleep, time
12 import numpy
13 from pprint import pprint
14 from sys import argv
15 import cPickle as pickle
16 from os.path import exists
17 from itertools import chain
18 from time import sleep
19 from random import randint, random
20
21 vres, hres = 1920, 1080
22
23 try:    subject = argv[1]
24 except: subject = s
25
26 def seq_to_chars(seq):
27     charseq = []
28     for el in seq:
29         if el == 0: charseq.append( ' d ' )
30         elif el == 1: charseq.append( ' f ' )
31         elif el == 2: charseq.append( ' j ' )
32         elif el == 3: charseq.append( ' k ' )
33     return charseq
34
35 seq = [3, 1, 2, 3, 0, 2, 0, 1, 2, 0, 1, 3] # BoSeidler09
36 #seq = [0,2,3,0,3,1,2,1,3,2,0,1] # Using two tones, one for over and under 200ms
37 #seq = [2,0,3,0,2,1,3,2,3,1,0,1] # Plus minus levels game seq
38 #seq = [3,2,3,1,3,0,1,0,2,1,2,0] # One chunk of twelve
39 #seq = [1, 2, 0, 2, 1, 3, 0, 3, 2, 0, 1, 3, 2, 3, 0, 1, 0, 3, 1, 2] # random 20-el seq
40 #seq = [0, 2, 3, 1, 2, 0, 1, 3, 0, 1, 3, 2, 0, 1, 3, 1, 2, 0, 3, 2] # My original sequence
41
42 seq_chars = seq_to_chars(seq)
43
44 win = visual.Window([vres,hres], rgb=[1,1,1], fullscr=True)
45
46 stim = visual.ElementArrayStim(win, nElements=4, fieldShape= ' sqr ', units= ' norm ', fieldPos=[0,0], sizes=.4,
47     colorSpace= ' rgb ',
48     xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
49     colors=[[-1,-1,-1], [-1,-1,-1], [-1,-1,-1], [-1,-1,-1]],
50     elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))
51
52 red = visual.ElementArrayStim(win, nElements=4, fieldShape= ' sqr ', units= ' norm ', fieldPos=[0,0], sizes=.4,
53     colorSpace= ' rgb ',
54     xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
55     colors=[[-1,-1,-1], [-1,-1,-1], [-1,-1,-1], [-1,-1,-1]],
56     elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))
57
58 yellow = visual.ElementArrayStim(win, nElements=4, fieldShape= ' sqr ', units= ' norm ', fieldPos=[0,0], sizes=.4,
59     colorSpace= ' rgb ',
60     xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
61     colors=[[-1,-1,-1], [1,1,-1], [-1,-1,-1], [-1,-1,-1]],
62     elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))

```

```

63
64 green = visual.ElementArrayStim(win, nElements=4, fieldShape= ' sqr ', units= ' norm ', fieldPos=[0,0], sizes=.4,
65                                     colorSpace= ' rgb ',
66                                     xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
67                                     colors=[[-1,-1,-1], [-1,-1,-1], [0,1,0], [-1,-1,-1]],
68                                     elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))
69
70 purple = visual.ElementArrayStim(win, nElements=4, fieldShape= ' sqr ', units= ' norm ', fieldPos=[0,0], sizes=.4,
71                                     colorSpace= ' rgb ',
72                                     xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
73                                     colors=[[-1,-1,-1], [-1,-1,-1], [-1,-1,-1], [0, -1, 0]],
74                                     elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))
75
76 black = visual.ElementArrayStim(win, nElements=4, fieldShape= ' sqr ', units= ' norm ', fieldPos=[0,0], sizes=.4,
77                                     colorSpace= ' rgb ',
78                                     xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
79                                     colors=[[-1,-1,-1], [-1,-1,-1], [-1,-1,-1], [-1,-1,-1]],
80                                     elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))
81
82 dual_dialog = gui.Dlg(title= " How many circles were there? ")
83 dual_dialog.addField( " Answer: ", choices=[ " 1 ", " 2 ", " 3 ", " 4 ", " 5 ", " 6 ", " 7 ", " 8 ", " 9 " ])
84
85
86 countdowntxt = visual.TextStim(win, text= " ",
87                                 units= ' norm ', height=.5, bold=True, color= ' black ')
88 pausetxt = visual.TextStim(win, text= " Press the spacebar to take a break ",
89                             units= ' norm ', height=.05, pos=(0,-.25), color= ' black ')
90 conttxt = visual.TextStim(win, text= " Press the spacebar to continue ",
91                             units= ' norm ', height=.05, pos=(0,-.5), color= ' black ')
92 dualtxt = visual.TextStim(win, text= " How many times did the boxes have borders? ",
93                             units= ' norm ', height=.05, pos=(0,-.5), color= ' black ')
94 banktxt = visual.TextStim(win, text= " ", units= ' norm ', height=.05, pos=(-.75,0), color= ' black ')
95
96 trialttimeplot = visual.ImageStim(win, image=None, pos=(0,.7))
97 alltimetrialtimeplot = visual.ImageStim(win, image=None, pos=(0,-.7))
98 realtimekiplot = visual.ShapeStim(win, vertices=(0,0), closeShape=False, units= " norm ")
99 realtimekiplot.setAutoDraw(True)
100 realtimekiplot.setPos((0, -1))
101
102 level_text = visual.TextStim(win, units= ' norm ', height = 0.2, pos=(-0.98, -0.98), text= ' +1 ',
103                             alignHoriz = ' left ', alignVert= ' bottom ')
104
105 realtimekiplot.setLineColor( " blue ")
106
107 def BlankScreen():
108     stim.setColors(color=[white, white, white, white])
109     stim.draw()
110     win.flip()
111
112 def DrawPlots(countdown=None, continuetxt=False):
113     event.clearEvents()
114
115     if not countdown == None:
116         countdowntxt.setText(countdown)
117         countdowntxt.draw()
118
119     banktxt.draw()
120     trialttimeplot.draw()
121     alltimetrialtimeplot.draw()
122
123     if continuetxt:
124         conttxt.draw()

```

```

125     else:
126         pausetxt.draw()
127
128     win.flip()
129     sleep(1)
130
131
132 def Draw(countdown):
133     DrawPlots(countdown)
134
135     k = event.getKeys(keyList=[ ' space ' ])
136     if ' space ' in k:
137         BlankScreen()
138         DrawPlots(continuetxt=True)
139         event.waitKeys(keyList=[ ' space ' ])
140         return True
141
142     return False
143
144 def Sleep():
145
146     sleep(1)
147
148     k = event.getKeys(keyList=[ ' space ' ])
149     if ' space ' in k:
150         event.waitKeys(keyList=[ ' space ' ])
151         return True
152
153     return False
154
155 def CountDown():
156     everslept = False
157     while True:
158         status = Sleep()
159         if status == True: everslept = True
160         break
161     if everslept: sleep(1)
162
163 def InitData(tag):
164     fname = subject + tag + ' .p '
165     data = {}
166
167     if not exists(fname):
168         pickle.dump(data, open(fname, " wb "))
169     else:
170         data = pickle.load(open(fname, " rb "))
171
172     if not data.has_key( " keylog " ):
173         # First session - set up dict structure
174         data[ " bank " ] = [[]]
175         data[ " keylog " ] = [[]]
176         data[ " timelog " ] = [[]]
177         data[ " ikilog " ] = [[]]
178         data[ " mtlog " ] = [[]]
179         data[ " accuratemt " ] = [[]]
180         data[ " log " ] = [[]]
181         data[ " level " ] = [[]]
182         data[ " dual_task " ] = [[]]
183     else:
184         data[ " bank " ].append([])
185         data[ " keylog " ].append([])

```

```

186         data[ " timelog " ].append([])
187         data[ " ikilog " ].append([])
188         data[ " mtlog " ].append([])
189         data[ " accuratemt " ].append([])
190         data[ " log " ].append([])
191         data[ " level " ].append([])
192         data[ " dual_task " ].append([])
193
194     return data, fname
195
196 data, fname = InitData(tagone)
197 clock = core.Clock()
198 start_time = clock.getTime()
199 stime = time()
200
201 norm_x = [-.83, -.66, -.5, -.33, -.16, 0, .16, .33, .5, .66, .83]
202
203 thresh = .5 # Threshold for shaping
204
205 while time() - stime < trialttime:
206
207     Countdown()
208
209     keytimes, times, iki, correct_ikis, verts, dual = [], [], [], [], [], []
210     dual_ctr = 0
211
212     iki_clock = core.Clock()
213
214     for el in range(len(seq)):
215
216         if len(correct_ikis) >= 2:
217             verts = [(norm_x[0], correct_ikis[0])] + [(norm_x[i+1], correct_ikis[i+1]-correct_ikis[i]) \
218                 for i in range(len(correct_ikis)) if not i == len(correct_ikis)-1]
219         elif len(iki) == 1:
220             verts = [(norm_x[0], correct_ikis[0])]
221
222         if draw_plot: realtimeikiplot.setVertices(verts)
223
224         if play_sound and len(verts) > 0:
225             rt = verts[-1][1]
226             if rt > thresh: sound.Sound(value=750, secs=.025).play()
227             else:         sound.Sound(value=1500, secs=.025).play()
228
229
230         if dual_task and random() < .3 and dual_ctr <= 9:
231             dual_ctr += 1
232             red.setSizes(.3)
233             yellow.setSizes(.3)
234             green.setSizes(.3)
235             purple.setSizes(.3)
236             black.draw()
237         else:
238             red.setSizes(.4)
239             yellow.setSizes(.4)
240             green.setSizes(.4)
241             purple.setSizes(.4)
242
243         if seq[el] == 0: red.draw()
244         elif seq[el] == 1: yellow.draw()
245         elif seq[el] == 2: green.draw()
246         elif seq[el] == 3: purple.draw()
247
248     win.flip()
249
250     while True:

```

```

251         clock.reset()
252
253         keys = event.waitKeys(keyList=[ ' d ', ' f ', ' j ', ' k ' ])
254
255         t = clock.getTime()
256         keytimes.append([keys[0], t])
257         times.append(t)
258         ikitime = iki_clock.getTime()
259         iki.append(ikitime)
260
261         if keys[0] == seq_chars[e1]:
262             correct_ikis.append(ikitime)
263             break
264
265         if e1 == len(seq) - 1:
266             black.draw()
267             win.flip()
268
269     if dual_task:
270         dualtxt.draw()
271         win.flip()
272         keys = event.waitKeys(keyList=[ ' 0 ', ' 1 ', ' 2 ', ' 3 ', ' 4 ', ' 5 ', ' 6 ', ' 7 ', ' 8 ', ' 9 ' ])
273
274     lvls = 0
275
276     # Got them all right
277     if (len(iki) == len(correct_ikis)):
278
279         # Were they all under threshold?
280         under_thresh = True
281         for rt in verts:
282             if not rt[1] < thresh:
283                 under_thresh = False
284
285         # Got them all right and they are under threshold
286         # Add a level
287         if under_thresh:
288
289             if dual_task:
290                 if (int(keys[0]) == dual_ctr):
291                     lvls += 1
292             else:
293                 lvls += 1
294
295         else:
296             # Did not get them all right
297             # Subtract a level
298             lvls -= 1
299
300     if lvls < 0:
301         for l in range(abs(lvls)):
302             if thresh < .5:
303                 thresh += .01
304
305         level_text.setText( " Level: " + str((.5 - thresh)*100).split( " . " )[0] + " ( " + str(lvls) + " ) " )
306         level_text.setColor( " red " )
307         level_text.draw()
308     elif lvls == 0:
309         level_text.setText( " Level: " + str((.5 - thresh)*100).split( " . " )[0] + " ( -- ) " )
310         level_text.setColor( " black " )
311         level_text.draw()
312     elif lvls > 0:
313         thresh -= .01
314         level_text.setText( " Level: " + str((.5 - thresh)*100).split( " . " )[0] + " ( +1 ) " )
315         level_text.setColor( " blue " )
316         level_text.draw()

```

```

317
318 win.flip()
319
320 data[ " keylog " ][-1].append(keytimes)
321 data[ " timelog " ][-1].append(times)
322 data[ " ikilog " ][-1].append(iki)
323 data[ " level " ][-1].append([str((.5 - thresh)*100).split(" . ")[0]])
324
325 if dual_task:
326     data[ " dual_task " ][-1].append([bool(int(keys[0]) == dual_ctr)])
327
328 mtime = iki_clock.getTime()
329 data[ " mtlog " ][-1].append(mtime)
330
331 if len(data[ " keylog " ][-1][-1]) == len(seq):
332     data[ " accuratemt " ][-1].append(mtime)
333
334 cash = 0
335 if len(data[ " keylog " ][-1][-1]) == len(seq):
336
337     alltimes = list(chain(*data[ " accuratemt " ]))
338
339     if alltimes[-1] == min(alltimes):
340         cash += 1
341
342     data[ " bank " ][-1].append(cash)
343
344 # Save subject data
345 pickle.dump(data, open(fname, " wb "))
346
347 win.close()

```

Chapter 6

Appendix 2: Analysis Python Code

```

1  #!/home/mingus/masters/epd-7.3-2-rh5-x86_64/bin/python
2  #####
3  # Configuration
4  #####
5  show_plots = False
6
7  #####
8  # Misc
9  #####
10 means = []
11 ns = []
12
13 #####
14 # Notes
15 #####
16
17 # keylog has [key, IKI] pairs including mistakes
18 # data["keylog"]: [['k', 0.7686469554901123], ['f', 0.4044618606567383], ...]
19
20 # level shows the level they were at on that trial
21 # data["level"]: [int(i[0]) for i in data["level"][0]]
22
23 # ikilog shows the elapsed time confusingly
24 # data["ikilog"][0][0]: [0.7848360538482666, 1.205543041229248, ...]
25
26 # timelog shows the IKIs
27 # data["timelog"][0][0]: [0.7686469554901123, 0.4044618606567383, ...]
28
29 # mtlog shows the cumulative movement time per trial
30 # data["mtlog"][0][0]: 5.543774127960205
31
32 # accuratemt shows the cumulative movement time per trial for accurate trials
33 # data["mtlog"][0][0]: 5.543774127960205
34
35
36 #####
37 # Imports
38 #####
39 from scipy import stats
40 from numpy.linalg import norm
41 from numpy import correlate
42 from pprint import pprint
43 from psychopy import visual, core, event
44 from time import sleep, time
45 import numpy
46 from numpy import array, polyfit
47 from pprint import pprint
48 import matplotlib.pyplot as plt
49 from matplotlib.pyplot import show
50 from matplotlib.pyplot import matshow, errorbar
51 from mpl_toolkits.mplot3d import Axes3D
52 import mpl_toolkits.mplot3d.axes3d as p3
53 from matplotlib.collections import PolyCollection
54 from matplotlib.colors import colorConverter
55 from matplotlib import cm
56 from sys import argv
57 import cPickle as pickle
58 from os.path import exists
59 from itertools import chain
60 import scipy.ndimage as ndimage
61 from subprocess import call
62 from mayavi import mlab
63 import operator
64 from scipy.cluster.hierarchy import *
65 from glob import glob

```

```

66
67 #####
68 # Code
69 #####
70 seq = [1,2,0,3,0,1,3,1,0,2,3,2] # BoSeidler09
71 seqlen = len(seq)
72
73 data_files = glob( " /home/mingus/Dropbox/MastersExperiment/Final/*//*.p " )
74
75 def norm(x):
76     return (array(x) - numpy.min(x)) / (float(numpy.max(x)) - float(numpy.min(x)))
77
78 #####
79 # Load up the data for each subject
80 #####
81
82 data = {}
83
84 for df in data_files:
85     c = df.split( " / " )[6] # condition
86     d = df.split( " / " )[7] # d
87     s = df.split( " / " )[-1].split( " _ " )[0] # s
88
89
90
91
92
93
94
95
96
97
98 if not data.has_key(c):
99     data[c] = {}
100
101 if not data[c].has_key(d):
102     data[c][d] = {}
103
104 if not data[c][d].has_key(s):
105     data[c][d][s] = {}
106
107
108 data[c][d][s] = pickle.load(open(df, " rb " ))
109
110 # Fix up s data where there are multiple entries due to restarting
111 if len(data[c][d][s][ " accuratemt " ]) > 1:
112     for k in data[c][d][s].keys():
113         data[c][d][s][k][0] = data[c][d][s][k][0] + data[c][d][s][k][1]
114         del data[c][d][s][k][1]
115
116 #####
117 # Create success_trials and error_trials
118 #####
119
120 for c in data.keys():
121     for d in data[c].keys():
122         for s in data[c][d].keys():
123             if not data[c][d][s].has_key( " success_trials " ): data[c][d][s][ " success_trials " ] = []
124             if not data[c][d][s].has_key( " success_trials_norm " ): data[c][d][s][ " success_trials_norm " ] = []
125             if not data[c][d][s].has_key( " error_trials " ): data[c][d][s][ " error_trials " ] = []
126
127             for t in data[c][d][s][ " timelog " ][0]:
128                 if len(t) == 12:
129                     data[c][d][s][ " success_trials " ].append(t)
130                     data[c][d][s][ " success_trials_norm " ].append(norm(t))

```

```

131         else:
132             data[c][d][s][ " error_trials " ].append(t)
133
134     #####
135     # Chunk detection algorithm:
136     # - Each item can only be part of one chunk.
137     # - A chunk is defined as a long IKI followed by a string of IKIs each of which are not significantly different.
138     # - As soon as the next item is significantly larger than the largest previous item in the chunk we start a new chunk.
139     # - Thus we need to keep track of the difference between each item and the starting item and the next item
140     #####
141
142     def reject_outliers(data, m=2):
143         return data[abs(data - numpy.mean(data)) < m * numpy.std(data)]
144
145     for c in data.keys():
146         for d in data[c].keys():
147             for s in data[c][d].keys():
148
149                 # The format of "chunks" is [i, end],
150                 data[c][d][s][ " chunks " ] = []
151                 i, j = 0, 0
152
153                 while i < 12:
154
155                     main_col = numpy.mean(reject_outliers(array(data[c][d][s][ " success_trials_norm " ])[0:,i]))
156                     max_col = 0
157
158                     if j > 11:
159                         data[c][d][s][ " chunks " ].append([i, 11])
160                         break
161
162                     j = i + 1
163                     while j < 12:
164
165                         compare_col = reject_outliers(array(data[c][d][s][ " success_trials_norm " ])[0:,j])
166
167                         if numpy.mean(compare_col) > max_col:
168                             max_col = numpy.mean(compare_col)
169
170                         if j+1 < 12:
171                             next_col = reject_outliers(array(data[c][d][s][ " success_trials_norm " ])[0:,j+1])
172
173                         t, p = stats.ttest_1samp(compare_col, main_col)
174
175
176                         # If this item is not significantly less than the first item, end the chunk.
177                         if not (t < 0 and p < .05/12):
178
179                             # If this item was actually just one past the last item, we have a chunk of size 1.
180                             if j == i + 1:
181                                 data[c][d][s][ " chunks " ].append([i, i])
182                                 i = i + 1
183                                 break
184
185                             i = j+1
186                             break
187
188                         # If the next item is significantly greater than greatest item in this chunk, end the chunk.
189                         if j+1 < 12:
190                             t_next, p_next = stats.ttest_1samp(next_col, max_col)
191
192                             if t_next > 0 and p_next < .05/12:
193                                 data[c][d][s][ " chunks " ].append([i, j])
194                                 i = j+1
195                                 break
196
197                     j += 1

```

```

198
199         data[c][d][s][ " chunks_n " ] = len(data[c][d][s][ " chunks " ])
200
201 #####
202 # Compare chunks between and within conditions
203 #####
204 C1_chunks_n = []
205 C2_chunks_n = []
206
207 C1_D1_chunks_n = []
208 C2_D1_chunks_n = []
209
210 C1_D2_chunks_n = []
211 C2_D2_chunks_n = []
212
213 C1_D3_chunks_n = []
214 C2_D3_chunks_n = []
215
216 for c in data.keys():
217     for d in data[c].keys():
218         for s in data[c][d].keys():
219             if c == " C1 " :
220                 C1_chunks_n.append(len(data[c][d][s][ " chunks " ]))
221
222                 if d == " D1 " :
223                     C1_D1_chunks_n.append(len(data[c][d][s][ " chunks " ]))
224
225                 if d == " D2 " :
226                     C1_D2_chunks_n.append(len(data[c][d][s][ " chunks " ]))
227
228                 if d == " D3 " :
229                     C1_D3_chunks_n.append(len(data[c][d][s][ " chunks " ]))
230
231             if c == " C2 " :
232                 C2_chunks_n.append(len(data[c][d][s][ " chunks " ]))
233
234                 if d == " D1 " :
235                     C2_D1_chunks_n.append(len(data[c][d][s][ " chunks " ]))
236
237                 if d == " D2 " :
238                     C2_D2_chunks_n.append(len(data[c][d][s][ " chunks " ]))
239
240                 if d == " D3 " :
241                     C2_D3_chunks_n.append(len(data[c][d][s][ " chunks " ]))
242
243 #####
244 # Fit curves to levels data
245 # TODO: Why not reaction time data? Need to remove outliers as well.
246 # http://glowingpython.blogspot.it/2011/07/polynomial-curve-fitting.html
247 #####
248 def Fit(y, order, out_range):
249     x = range(len(y))
250     z = numpy.polyfit(x, y, order)
251     p = numpy.poly1d(z) # so you can do i.e., p(xx)
252     xx = numpy.linspace(0, out_range, out_range)
253     fit = p(xx)
254
255     return y, z, fit
256
257
258 for c in data.keys():
259     for d in data[c].keys():
260         for s in data[c][d].keys():
261             y = [int(i[0]) for i in data[c][d][s][ " level " ][0]]
262

```

```

263         y, p, fit = Fit(y, 2, 1000)
264
265         data[c][d][s][ " levels_poly_coefs " ] = p
266         data[c][d][s][ " levels_poly_fit " ] = fit
267
268     #####
269     # Fit curves to accuratemtdata
270     # http://glowingpython.blogspot.it/2011/07/polynomial-curve-fitting.html
271     #####
272
273     for c in data.keys():
274         for d in data[c].keys():
275             for s in data[c][d].keys():
276                 y = data[c][d][s][ " accuratemt " ][0]
277
278                 y, p, fit = Fit(y, 2, len(y))
279
280                 data[c][d][s][ " accuratemt_poly_coefs " ] = p
281                 data[c][d][s][ " accuratemt_poly_fit " ] = fit
282
283     def PlotTrialTime(condition, day, title= " ", fname= " " ):
284         c = " C " + str(condition)
285         d = " D " + str(day)
286
287         plt.clf()
288         fig = plt.figure()
289         ax = fig.add_subplot(111)
290         if not title == " ":
291             ax.set_title(title)
292
293         for s in data[c][d].keys():
294             ikisum = [sum(data[c][d][s][ " success_trials " ][i])
295                       for i in range(len(data[c][d][s][ " success_trials " ]))]
296
297             for i in range(len(ikisum)):
298                 if ikisum[i] > 6:
299                     mn = numpy.mean(ikisum)
300                     if mn > 6: mn = 6
301                     ikisum[i] = mn
302
303             ax.plot(ikisum)
304
305         plt.savefig(fname)
306
307     #####
308     # Create a plot of their levels including mean level
309     #####
310
311     def PlotLevels(condition, day, title= " ", fname= " ", legend=True, print_means = False):
312         c = " C " + str(condition)
313         d = " D " + str(day)
314
315         plt.clf()
316         fig = plt.figure()
317         ax = fig.add_subplot(111)
318         if not title == " ":
319             ax.set_title(title)
320
321         j = 0
322         for s in data[c][d].keys():
323
324             ax.plot([int(i[0]) for i in data[c][d][s][ " level " ][0]], label= " S " + str(j))
325             j += 1
326
327         sums, means, sds, ns = [], [], [], []

```

```

328     n_trials_per_s = sorted([len(data[c][d][s][ " level " ][0]) for s in data[c][d].keys()])
329     max_trial = n_trials_per_s[-1]
330
331     for t in range(max_trial):
332
333         sums.append(0)
334         means.append(0)
335         ns.append(0)
336         sds.append(0)
337
338         for s in data[c][d]:
339
340             try:
341                 val = int(data[c][d][s][ " level " ][0][t][0])
342                 sums[t] += val
343                 ns[t] += 1
344             except:
345                 continue
346
347             try:
348                 means[t] = (float(sums[t]) / float(ns[t]))
349
350             except:
351                 continue
352
353     ax.plot(means, label= " Mean " , lw=5)
354
355     if legend:
356         ax.legend(loc= " best " )
357
358     if not fname == " " :
359         plt.savefig(fname)
360
361     if show_plots:
362         plt.show()
363
364     return means, ns, sds
365
366 def LouvaineOutput(condition, day, subject, intertrial = 1000, intratrial = 1, n_slices = -1, smooth = -1, run=False):
367     viz = {}
368
369     # The vector of time series expressed in seconds since trial start, up to n_slices
370     d = list(chain(*data[condition][day][subject][ " ikilog " ]))[:n_slices]
371
372     # Get rid of slices where the subject made errors
373     d_new = []
374     for row in d:
375         if len(row) == seqlen:
376             d_new.append(row)
377
378     d = d_new[:]
379
380     # Convert timeseries data into 2d numpy array
381     d_new = array([])
382     d_new.resize(len(d), seqlen)
383     for row in range(len(d)):
384         for el in range(seqlen):
385             d_new[row][el] = d[row][el]
386
387     d = d_new[:]
388
389     viz[ ' d_raw_timeseries_data ' ] = d[:]
390
391     # Create dataset of inter-key-intervals
392     # Each row will now have seqlen-1 elements
393     d_new = array([])

```

```

395     d_new.resize(len(d), seqlen-1)
396
397     for x in range(len(d)):
398         ts = d[x]
399         ts_shift = numpy.append(ts[1:], 0)
400         ikis = ts_shift - ts
401         ikis = ikis[:-1]
402
403         for el in range(seqlen-1):
404             d_new[x][el] = ikis[el]
405
406     d = d_new[:]
407
408     viz[ ' d_raw_ikis ' ] = d[:]
409
410     # # Normalize each row
411     d_norm = d[:]
412     for x in range(len(d_norm)):
413         v = d_norm[x]
414         d_norm[x] = (v - min(v))/max(v)
415
416     viz[ ' d_norm_ikis ' ] = d_norm[:]
417
418     # Compute the third standard deviation of each row
419     stdevs, means = array([]), array([])
420     stdevs.resize(seqlen-1,1)
421     means.resize(seqlen-1,1)
422
423     for x in range(seqlen-1):
424         stdevs[x][0] = numpy.std(d_norm[:,x])*3
425         means[x][0] = numpy.mean(d_norm[:,x])
426
427     #Compute rows with outlier values
428     outliers = []
429
430     for row in range(d_norm.shape[0]):
431         for col in range(seqlen-1):
432             val = d_norm[row][col]
433             sdmax = means[col] + stdevs[col]
434             sdmin = means[col] - stdevs[col]
435             if val > sdmax or val < sdmin:
436                 outliers.append(row)
437
438     outliers = numpy.unique(outliers)
439
440     # TODO: wtf broken? Delete rows with outlier values
441     d_norm = numpy.delete(d_norm, outliers, 0)
442
443     viz[ ' d_norm_iki_outliers_removed ' ] = d_norm[:]
444
445     # Compute the difference between successive IKIs for each row
446     d_sims = array([])
447     d_sims.resize(d_norm.shape[0], seqlen-2)
448
449     for x in range(d_norm.shape[0]):
450         for y in range(seqlen-2):
451             d_sims[x][y] = numpy.abs(d_norm[x][y] - d_norm[x][y+1])
452
453     # Compute the max of each trial of iki differences
454     maxs = array([])
455     maxs.resize(d_sims.shape[0])
456
457     for row in range(d_sims.shape[0]):
458         maxs[row] = numpy.max(d_sims[row])
459
460     # Normalize each row of iki differences based on the max
461     for row in range(d_sims.shape[0]):

```

```

462         for col in range(seqlen-2):
463             d_sims[row][col] = (maxs[row] - d_sims[row][col]) / maxs[row]
464
465     viz[ ' d_norm_iki_similarities ' ] = d_sims[:]
466
467     # Create the text output for Louvaine community detection
468     out = " "
469
470     out += " > \n "
471     for x in range(seqlen-1):
472         out += str(x) + " " + str(x) + " \n "
473     out += " > \n "
474
475     for x in range(d_sims.shape[0]):
476         out += str(x) + " " + str(x) + " \n "
477     out += " > \n "
478
479     for row in range(d_sims.shape[0]):
480         for col in range(seqlen-2):
481             out += str(col) + " " + str(col+1) + " " + str(d_sims[row][col]) + " " + str(row) + " \n "
482             out += str(col+1) + " " + str(col) + " " + str(d_sims[row][col]) + " " + str(row) + " \n "
483
484     out += " \n "
485
486     f = open(condition + " _ " + day + " _ " + subject + " _graph.txt ", " w ")
487     f.write(out)
488     f.close()
489
490     if run:
491
492         cmd = " ./slicer -i " + condition + " _ " + day + " _ " + subject + " _graph.txt -o " + \
493             condition + " _ " + day + " _ " + subject + " _graph.bin -n " + condition + " _ " + \
494             day + " _ " + subject + " .node -c " + condition + " _ " + day + " _ " + \
495             subject + " .conf -w " + str(intertrial)
496
497         call(cmd, shell=True)
498
499         cmd = " ./community " + condition + " _ " + day + " _ " + subject + " _graph.bin " + \
500             condition + " _ " + day + " _ " + subject + " .conf -o " + condition + " _ " + \
501             day + " _ " + subject + " .communities -pp " + strf(intratrial)
502
503         call(cmd, shell=True)
504
505         f = open(condition + " _ " + day + " _ " + subject + " .communities ")
506         communities = f.read()
507         f.close()
508         communities = communities.split( " \n " )[:-1]
509
510
511         coms = []
512         for row in range(len(communities)):
513             coms.append(int(communities[row].split( " \t " )[1]))
514
515         coms = array(coms)
516         coms.resize(10, coms.shape[0] / 10)
517
518     data[condition][day][subject][ " communities " ] = coms
519
520     return viz
521
522 def PlotVar(var, x_range, y_range, plot_mean=False):
523
524     for c in data.keys():

```

```

525
526     for d in data[c].keys():
527
528         plt.clf()
529         fig = plt.figure()
530         ax = fig.add_subplot(111)
531
532         mn = []
533
534         for s in data[c][d].keys():
535             dat = data[c][d][s][var]
536             mn.append(array(dat))
537             ax.plot(dat)
538
539         if plot_mean:
540             mn = numpy.mean(array(mn), axis=0)
541             ax.plot(mn, lw=3, color= " black ")
542             plt.axis([0, x_range, 0, y_range])
543             plt.savefig(c + "_" + d + "_" + var + ".png ")
544
545 def SurfacePlotsAllSubjects(c,d):
546     for s in data[c][d].keys():
547         mlab.surf(data[c][d][s][ " success_trials_norm "], colormap= " jet ", warp_scale= " auto ")
548
549 def Plot(x, fname, y=None, z=None, legend=False, labels= " ", title= " ", title_size= " "):
550     plt.clf()
551     fig = plt.figure()
552     ax = fig.add_subplot(111)
553     ax.plot(x)
554
555     if not y is None:
556         ax.plot(y)
557
558     if not z is None:
559         ax.plot(z)
560
561     if legend:
562         if labels == " ":
563             ax.legend()
564         else:
565             ax.legend(labels)
566
567     if not title == " ":
568         if title_size == " ":
569             ax.set_title(title)
570         else:
571             ax.set_title(title, size=title_size)
572
573     plt.savefig(fname)
574
575 means, ns, sds = PlotLevels(1, 1, " ", " C1_D1_levels.png ", legend=False)
576 Plot(norm(ns)*norm(means), " C1_D1_norm_ns_correlation.png ")
577 Plot(ns, " C1_D1_ns_means.png ", means)
578
579 means, ns, sds = PlotLevels(1, 2, " ", " C1_D2_levels.png ", legend=False)
580 Plot(norm(ns)*norm(means), " C1_D2_norm_ns_correlation.png ")
581 Plot(ns, " C1_D2_ns_means.png ", means)
582
583 PlotLevels(1, 3, " ", " C1_D3_levels.png ", legend=False)
584 PlotLevels(2, 1, " ", " C2_D1_levels.png ", legend=False)
585 PlotLevels(2, 2, " ", " C2_D2_levels.png ", legend=False)
586 PlotLevels(2, 3, " ", " C2_D3_levels.png ", legend=False)
587

```

```
588 PlotTrialTime(1, 1, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C1_D1_mean_trial_time.png ")
589 PlotTrialTime(1, 2, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C1_D2_mean_trial_time.png ")
590 PlotTrialTime(1, 3, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C1_D3_mean_trial_time.png ")
591 PlotTrialTime(2, 1, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C2_D1_mean_trial_time.png ")
592 PlotTrialTime(2, 2, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C2_D2_mean_trial_time.png ")
593 PlotTrialTime(2, 3, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C2_D3_mean_trial_time.png ")
```

Bibliography

- [1] J Bo and R D Seidler. Visuospatial working memory capacity predicts the organization of acquired explicit motor sequences. Journal of neurophysiology, 101, Jun 2009.
- [2] M. Chekaf and F. Mathy. Chunking memory span for objects varying in number, order and compressibility. In Review, 2012.
- [3] Steve W Kennerley, K Sakai, and M F S Rushworth. Organization of action sequences and the role of the pre-sma. Journal of neurophysiology, 91, Feb 2004.
- [4] Robert J. Kosinski. A literature review on reaction time: <http://goo.gl/lywyy>. 2008.
- [5] T Kriete, D Noelle, T Braver, J Cohen, and R O'Reilly. Indirection and generativity in the prefrontal cortex / basal ganglia system. In Prep.
- [6] G A Miller and G A Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. 1956. Psychological review, 101:81–97, 08 1994.
- [7] V S Chandrasekhar Pammi, K P Miyapuram, Kazuyuki Ahmed, Raju S Samejima, and Kenji Bapi. Changing the structure of complex visuo-motor sequences selectively activates the fronto-parietal network. NeuroImage, 59, Jan 2012.
- [8] Virginia B Penhune and Christopher J Steele. Parallel contributions of cerebellar, striatal and m1 mechanisms to motor sequence learning. Behavioural brain research, 226, Jan 2012.
- [9] Katsuyuki Sakai, Katsuya Kitaguchi, and Okihide Hikosaka. Chunking during human visuo-motor sequence learning. Experimental brain research, 152, Sep 2003.
- [10] Timothy Verstynen, Jeff Phillips, Emily Braun, Brett Workman, Christian Schunn, and Walter Schneider. Dynamic sensorimotor planning during long-term sequence learning: The role of variability, response chunking and planning errors. PloS one, 7, 10 2012.
- [11] Willem B. Verwey and Elger L. Abrahamse. Distinct modes of executing movement sequences: Reacting, associating, and chunking. Acta Psychologica, 140(3):274–282, Jul 2012.
- [12] Erica L Wohldmann, Alice F Healy, and Lyle E Bourne. Task integration in time production. Attention, perception & psychophysics, 72, May 2010.