

2008

Spatial ability cognitive model with ACT-R 6.0

Andre Lokasari
Iowa State University

Follow this and additional works at: <https://lib.dr.iastate.edu/rtd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Lokasari, Andre, "Spatial ability cognitive model with ACT-R 6.0" (2008). *Retrospective Theses and Dissertations*. 15429.
<https://lib.dr.iastate.edu/rtd/15429>

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

Spatial ability cognitive model with ACT-R 6.0

by

Andre Lokasari

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:

Leslie Miller, Major Professor

Sarah M. Nusser

Simanta Mitra

Iowa State University

Ames, Iowa

2008

Copyright © Andre Lokasari, 2008. All rights reserved.

UMI Number: 1454597

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 1454597
Copyright 2008 by ProQuest LLC
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

TABLE OF CONTENTS

LIST OF FIGURES	iv
LIST OF TABLES	v
ABSTRACT	vi
CHAPTER 1. INTRODUCTION	1
CHAPTER 2. THEORETICAL BACKGROUND	3
2.1 Spatial Reference System	6
2.2 Different aspects of spatial ability	6
CHAPTER 3. USER Study Experiment	8
3.1 Software developed for the user study	9
3.1.1 Software features	9
3.1.2 Software Randomization Input	13
3.1.3 Software Measures	14
3.1.4 Location learning	15
3.1.5 Technical details	17
3.2 Experiment Summary and Supporting Software	17
3.3 Result and discussion of user study experiment	19
3.3.1 Relevant results captured from the experiment	19
3.3.2 Limitation of user study experiment design	20
CHAPTER 4. Cognitive Model	21
4.1 Overview of ACT-R 6.0	21
4.1.1 Cognitive architectures comparison	21
4.1.2 ACT-R 6.0 cognitive architecture	22
CHAPTER 5. SUMMARY	39
CHAPTER 6. FUTURE DIRECTIONS	41
APPENDIX A. Graphical figures	42
A1. Exercise 3 labeling scheme (icons developed by Rusch (2008)).	42
A2. Map views generated by each button function.	43
A3. Trial Scenario	47
A4. Background Questionnaire	48
APPENDIX B. Code	51
B1. Control List Generator SAS Code	51
B2. Act-r 6.0 PRODUCTIONS CODE (COMPLETE)	66
APPENDIX C. PLOT APPENDIX	84
C1. Actual versus simulation total time performance plots	84
C2. Actual versus simulation summary statistics plots	105
C3. Actual versus simulation exercise total time plots	116

BIBLIOGRAPHY	119
ACKNOWLEDGEMENTS	121

LIST OF FIGURES

Figure 1.1: The role of ACT-R in modeling (Budiu 2008).....	5
Figure 3.1: Interface layout screenshots (target rectangle (a), circle layout (b), grid layout (c) and map layout (d)).....	10
Figure 3.2: Exercise one and two labeling schemes (pictures from Ehret (2000, 2002) and Rusch (2008)).....	11
Figure 3.3: Tooltips available to users (Circled area).....	13
Figure 3.4: Location learning test illustration: (a) beginning of test, (b) drag-and-dropped button being moved, and (c) the original exercise buttons' positions.	16
Figure 3.5: The proximity of the correct/hit area.....	16
Figure 3.6: Interface of the color training/test software showing the color being tested and the list of color names to choose from.	18
Figure 4.1: ACT-R 6.0 Architecture (Budiu 2008).	24
Figure 4.2 SAS program (compilation part)	26
Figure 4.3 Probability density graph (Gamma distribution-wikipedia 2008).....	27
Figure 4.4 Probability mass graph (Poisson distribution-wikipedia 2008).....	28
Figure 4.5 SAS program (Calculation part).....	29
Figure 4.6 SAS program (Generation part).....	30
Figure 4.7: ACT-R model productions	31
Figure 4.8 Initiate-attend production code (Productions code available in appendix B).....	32
Figure 4.9 Actual versus Simulation total time	35
Figure 4.10 Actual versus Simulation summary statistics	36
Figure 4.11 Actual versus Simulation exercise total time plots for all participants	37

LIST OF TABLES

Table 1: Preliminary tests (VZ-2, MV-2, and P-2 tests were taken from Ekstrom, et al (1976))	17
Table 2: Architecture feature comparison (taken from <i>Byrne</i> (2003)).	23

ABSTRACT

The effectiveness of software interface design is a lot of the times tied to the user of the software. In particular, it is tied to how users comprehend the spatial stimuli presented to them through the software interface (spatial information encoding). Thus, the goal of the research is to understand the aspects of different levels of spatial ability on a spatial software interface for creating a theoretical structure of spatial ability on a simple task as a stepping stone to understand a more complex demanding task. Why is this important? The importance of knowing the different spatial ability aspect relates to how one can design software interfaces to suit different levels of spatial ability.

To examine the impact of spatial ability on using user interfaces, we developed a set of user interfaces that required the user to interpret an instruction and choose the matching buttons. In total three interfaces were developed ranging from simply matching colors to an interface loosely based on the address canvassing task used by the Census Bureau to identify the correct location of housing units. A cognitive model was developed for the interfaces that incorporated spatial ability. The model was implemented using ACT/R and evaluated against user studies involving 63 participants.

CHAPTER 1. INTRODUCTION

In daily life, people interact with objects around them to complete a certain task. However, different people act differently toward these objects and sometimes use different strategies to accomplish their task. Many of these tasks involve the use of spatial information available on the corresponding spatial context. However, before one can use the information, it must be encoded and comprehended in a useful way so that it can be retrieved later. These whole processes regarding retrieval processing, and storing spatial information are widely accepted as a spatial ability (Gunzelmann 2006).

However, spatial transformation/processing of spatial information come with different dimensions and it can be broken down into several aspects. For example, one person might be more efficient in conducting a task specific to memorizing a location of an object on the map and another person is more efficient in orientation of that object with respect to himself. These different aspects are the determinant to how efficient a person can perform a task involving spatial ability. A person with a high level of spatial ability (e.g. has good wider range of spatial aspects/skills) might be able to perform spatial tasks better than a person with average spatial ability. Thus, understanding spatial ability is very important.

Our research has been designed to create a cognitive structure that verifies our understanding of spatial ability on a simple task. In order to achieve this goal, our research has been broken down into two parts: empirical and analytical.

The empirical approach makes use of user study to gather sample data corresponding to user performance on operating software interface. Based on these results we are interested in creating the same task. The task chosen builds on the task used by Ehret (2000, 2002) in his location learning studies.

Given the sample data, the analytical approach involves creating a cognitive model. This model will be a depiction of our understanding of how spatial ability impacts such a task and will be used to validate our presumptions on how spatial ability affects user performance on such a simple task (simple software interface task).

The validation will involve a statistical comparison between the data gathered from the empirical approach and the data simulated using the cognitive model. The thesis will be broken down into the following chapters:

1. Chapter 2, a theoretical background will be presented along with some existing comparable work with regard to modeling a cognitive structure on various fields, with an emphasize on spatial ability work in the end;
2. The next chapter (Chapter 3) will cover the user study experiment that covers the empirical approach of the research;
3. Chapter 4 discusses the cognitive model and its evaluation;
4. Chapter 5 summarizes the research; and
5. Chapter 6 gives some possible future directions of this research.

CHAPTER 2. THEORETICAL BACKGROUND

A great deal of research has investigated the importance of understanding human spatial information processing (Gunzelmann, 2006; Rusch, 2008; Kimura, 1999). People have varied levels of processing and the levels are related to how efficiently they process spatial information for navigation, map reading, and spatial transformation (Gunzelmann, 2006). All of these processes are complex and make use of some aspects of spatial ability.

Significant research has been aimed at solving the question “why do people use a specific strategy to solve a task?” For example, when two people are asked to read a map to find their way to a pre-determined location, they might make use of different processes and use different ways to get to the location. By understanding these processes, it is believed that certain steps can be taken to improve the efficiency of the required task that involves the use of spatial ability.

According to Gunzelmann and Lyon (2006), spatial ability is “a mechanism involved in encoding, processing, and using spatial knowledge for several purposes such as wayfinding, navigation, problem solving, language processing, orientation, map reading”, to name a few. By encoding, we mean that people observe some visual stimuli and try to register those stimuli in their brain. By processing, we mean that people do some mental processes within their brain to understand the stimuli that is perceived in the encoding stage. By using spatial knowledge, it is meant that people use the processed spatial information to guide their actions in order to perform their required task where required tasks can be of different nature such as map reading, mathematical operation, making a cup of coffee and so on. Thus, there are several aspects of spatial information processing that can be used to perform tasks: “representations of environmental information, visuospatial working memory (learning), reasoning with spatial mental models, mental imagery, and navigation” (Gunzelmann, 2006, 1-2). Identifying these aspects of spatial ability it is important to understand the involved procedures for doing a particular task. An understanding of spatial ability is necessary in order to build an appropriate cognitive model that takes spatial ability into account. Finally, the objective of a cognitive model is to incorporate information (what is known of the processes required for the task) “to facilitate developing more precise, and

psychologically valid quantitative accounts of human performance on complex spatially-demanding tasks” (Gunzelmann, 2006).

By providing precise and valid measurements, a model can provide an accurate description of important steps of a required task. For example, the study of Gugerty and Rodes (2007) created a cognitive model to demonstrate how theoretical constructs can be applied to a complex navigation task. In their case, they used the model to predict response times of using a map to determine the cardinal directions between two objects in the environment. Their model makes use of a cognitive modeling tool called ACT-R (Budi 2008). ACT-R has been used by a lot of psychologists to create cognitive models (a theory about how human cognition works). ACT-R is a cognitive mechanism built on top of LISP. Its constructs reflect assumptions about human cognition (Budi 2008). Researchers create models, using ACT-R programs, based on their assumptions about the particular task. These assumptions can be tested by comparing the results of the model with experimental results based on real participants (see Figure 1.1). The models usually involve the performance of the users in terms of quantitative measures of time and accuracy. Thus the benefit of ACT-R is that it allows researchers to collect quantitative measures that can be directly compared with quantitative measures gathered from the same experiments using real participants. ACT-R has been used to produce user models that can access different computer interfaces (Budi 2008). ACT-R is further overviewed in Chapter 4.

Why is modeling important? Building a cognitive model is a way to move towards an understanding the structure of tasks that make up an experiment using human participants. For example, in the field of HCI, a lot of modeling is done in terms of how people interact with computer programs with a given software interface stimuli such as computerized arithmetic problems (Lebiere 1998), Icons/buttons visual stimuli (Ehret 2000, 2002), diagram interface for map (Dye, 2007). A study of how people use map software, for example, is beneficial in order to identify the cognitive processes. By providing a valid model, human cognitive processes in operating the map software are understood and appropriate actions can be done with regards to the interface of the software (such as structuring the interface in a way that is conducive/optimal for map reading, etc.) to improve efficiency and performance of using the software.

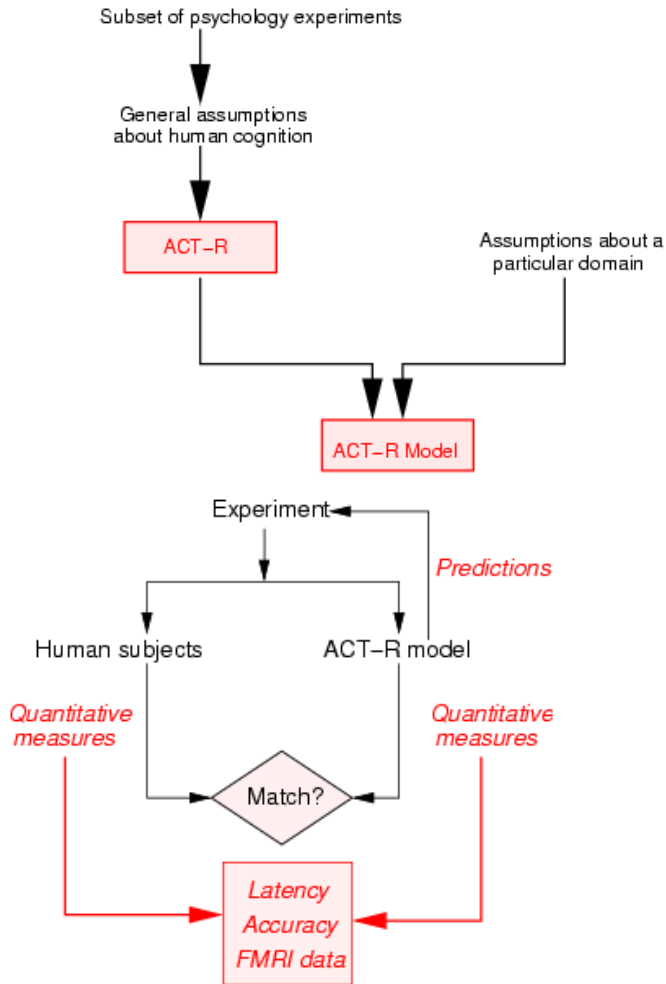


Figure 1.1: The role of ACT-R in modeling (Budiu 2008)

From the examples above, there is one underlying activity that needs to be considered, namely, how humans interact with their environment. Performance, then, can be attributed to how well humans are familiar with their environment (Ehret 2000). For example, when people interact with MS Word program, one person might be more familiar with where the edit menu is than another. Thus, the person that is more familiar can perform the editing much faster because they don't have to adapt to the interface. By adapting, people try to learn about how the interface is structured, and learn about the locations of

objects (edit menu, copy/paste button, etc.). This concept is important to this research as we will be investigating how users with different spatial ability interact with different software interfaces. Locations of objects on a particular interface can be represented in different ways depending on the underlying spatial reference system being used.

2.1 Spatial Reference System

Shelton and McNamara (2001) emphasize the importance of understanding spatial reference systems in order to represent object locations in the environment. A spatial reference system is a relational system between located objects, referenced objects, and spatial relations between them on orthogonal axes.

There are two major reference systems that are important to understand spatial processes: 1) *egocentric* reference system, and 2) *environmental* reference system or exocentric reference system according to Gunzelmann (2006). Egocentric reference system represents location of an object with respect to the observer. For example, the city of Des Moines, Iowa is located 20 miles south of the observer. Environmental reference system specifies location with respect to other objects. For example, the city of Des Moines, Iowa is located 20 miles south of Ames, Iowa. Another important view, which is an elaboration of the environmental view, has also been exposed. Previous studies done by Stevens, Coupe (1978), Maki (1981), and Wilton (1979), show that there is a clustering phenomenon of locations via observable features of the visual stimuli such as state boundaries, lines, etc. For example, Des Moines and Ames are cities located in Iowa State (via state boundary), where as Iowa and Illinois are two states located in USA (via country boundary). Furthermore, a theory developed by Stephen C. Hirtle and John Jonides (1985) strengthened this hierarchy reference system by showing that clustering of spatial locations can also be done subjectively (e.g. without any unambiguous visual features).

2.2 Different aspects of spatial ability

Kimura (1999) presented 6 aspects of spatial ability:

1. Spatial orientation: the ability to measure changes in the orientation of an object. This is testable with both 2D and 3D objects rotated in 2D or 3D spaces;
2. Visuospatial memory: the ability of a person to memorize the position of objects in an array. One possible test for this would be memory game.
3. Targeting: the ability to intercept moving targets or throw objects at a target. Based on Kitamura (1999), this task is not easily testable, however one measure can be done by throwing an object to a target;
4. Spatial visualization: the ability to tell the orientation of a static object with respect to the observer's position after moving around, or the ability to correctly imagine a result after folding or disassembling parts of an object;
5. Flexibility of closure/field independence: the ability to find a previously presented object after being obscured in a more complex figure. A test of this aspect can be done by asking a person to try to find an object that is obscured in a complex pattern;
6. Spatial perception: the ability to identify a visual pattern in a scene with distracting or complex pattern.

There are standardized tests for the six spatial ability aspects above as suggested by Ekstrom (1976): 1-cube comparison test (S-2), 2-building memory test (MV-2), 4-paper folding test (VZ-2), 5-hidden patterns test (CF-2), 6-number comparison test (P-2). We used some of these tests in our experiment (see Section 3.2) to base our spatial grouping of users by their test scores. The next chapter examines the software and the resulting experiment that is used as the basis of the cognitive model.

CHAPTER 3. USER STUDY EXPERIMENT

An experiment was conducted to gather data regarding users' performance in terms of operating a simple software interface. The cognitive model, which is discussed in Chapter 4, used the results of the user study to guide its behavior. The user performance data gathered from this experiment was also used as a standard to evaluate how well the model performs.

The experiment was conducted by Michelle Rusch (2008) using the software developed as the first component of my research. The experiment started with recruitment of 80 people with a range of background (age, gender, occupation, color vision, and spatial ability category). Thus, in order to get people from different background, preliminary tests (Table 1) were done as part of the recruitment process. The result of these preliminary tests would then decide the eligibility of the participants (e.g. sufficient color vision is needed) and the grouping of the result of the experiment. The results were categorized based on two different spatial categories (high and low). This categorization is particularly essential to guide the creation of the model so that it models the variety of the experiment performance.

The experiment itself was done by using a tablet PC as the media. With the tablet PC, instead of a mouse, a stylus was used. User study software was developed and deployed to this tablet PC. The discussion of the software is intended to both show the full extent of the software that I developed for the experiment and to illustrate the basis for the data used in the development of the cognitive model. Other software such as visual test/training and background questionnaire were also deployed to accommodate some of the tests mentioned in the previous paragraph.

As part of the experiment, a participant was required to operate the user study software, and their performance would be recorded. Then, the performance result would be compiled by using pre-developed compiler software and summarized. An analysis was conducted on the result of the preliminary test to see which spatial aspect matters most on the task required on the user study software (Rusch 2008). In the end, the result of this experiment would be used to restrict the structure and the behavior of the model.

3.1 Software developed for the user study

The discussion of the software in this section is designed to meet two goals. First, the experiment (especially the first two exercises) serves as the basis of the cognitive model presented in Chapter 4. Second, the description describes my contribution to the user study. As such, we will present some aspects of the software that do not directly contribute to the cognitive model.

There were two versions of the software: full and practice. The full version was the main software used in the experiment whereas the practice version was a stripped down version that was used to familiarize the user with the interface and what they were expected to do. By stripped-down, we mean that the program only contained enough features to get the participants to understand what they needed to do in the experiment. The primary purpose of the full version of the software was to gather data specific to user performance with respect to various levels of spatial ability on different software layouts. To achieve this goal, the user interface design incorporated several key features.

3.1.1 Software features

First, the interface was initiated with a configuration window that consisted of a text box for the user's identification (ID) number. Each user was assigned an ID number in advance. When the ID number was entered into the text box, the program would proceed to the main experiment window where the participant could see a colored rectangle with x's on the center (Figure 3.1a). Each ID number was associated with a unique order of the colors (exercise one and two) or instructions (exercise three) being presented (see Section 3.1.1.1). The users were instructed to select the correct button from the 12 buttons corresponding to the color/instruction of the target button.

3.1.1.1 Layout and labeling scheme

The software supported three different user interfaces. Each layout was the basis of an exercise. Exercise one and exercise two used either a circular button layout (essentially Ehret's interface layout (2000, 2002)) or a grid layout depending on which user ID was

entered in the configuration window. The idea was to randomize the order of the layout, between circular and grid, across users to see the impact of the layout presentation order on user performance. The purpose of the grid layout was to simulate a more typical button configuration (e.g. cell phone dial-pad, various software menu configurations such as MS Word, Excel, etc.).

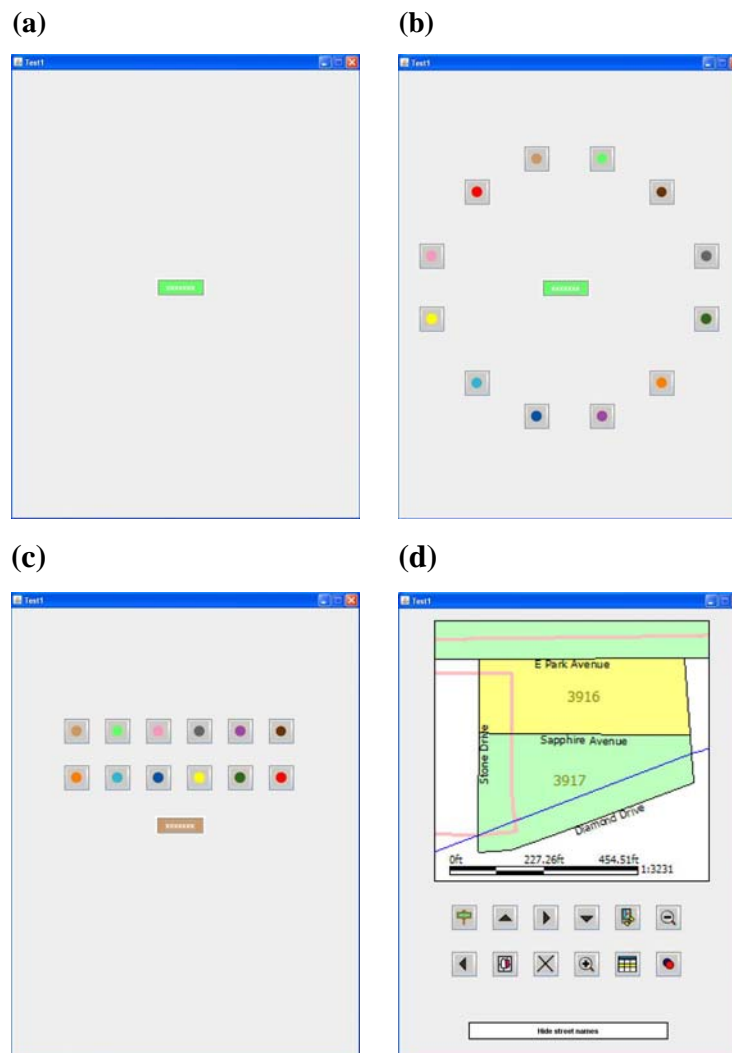


Figure 3.1: Interface layout screenshots (target rectangle (a), circle layout (b), grid layout (c) and map layout (d)).

After completing exercises one and two, users were presented a similar problem using a map interface. There were still 12 to-be-selected buttons and a target label. In this exercise, the

target label contained an instruction that corresponded to one of the 12 buttons. Figure 3.1 b, c, and d illustrate the screen layout for the three exercises, respectively.

We focus on discussing the button labeling schemes for exercises one and two because they form the basis of the modeling phase of the research discussed in Chapter 4. Exercise three was administered to collect additional data on user performance on a map-like interface and relevant to a separate future study.

Exercise one and two used button labeling scheme analogous to the one Ehret (2000, 2002) used. Ehret had four labeling schemes: (1) color-match, (2) meaningful text, (3) arbitrary icon, and (4) no-label.

- (1) The color-match scheme labeled the 12 buttons with 12 different colors;
- (2) Meaningful text scheme labeled the 12 buttons with 12 different texts describing the colors;
- (3) Arbitrary icon scheme labeled the 12 buttons with random icons, which were unrelated to the functions of the target button;
- (4) No-label scheme labeled the 12 buttons with no label.

All of the labeling schemes can be observed in Figure 3.2 below. All the graphics except for











Scheme	Label												
	1	2	3	4	5	6	7	8	9	10	11	12	
Color-match													
Meaningful-text	Green	Blue	Brown	Gray	Lt Blue	Lt Green	Green	Pink	Purple	Red	Tan	Yellow	
Arbitrary (Circle)													
Arbitrary (Grid)													
No-label													

Figure 3.2: Exercise one and two labeling schemes (pictures from Ehret (2000, 2002) and Rusch (2008)).

the arbitrary grid were the same as those developed by Ehret (2000, 2002). The arbitrary icons for the buttons in the grid exercise were selected by Michelle Rusch (2008). Each user ID was associated with a unique order of the 12 colors.

As for the labeling scheme for the map exercise (exercise 3), Rusch (2008) developed the icon graphics. Instead of colors, we used:

- (1) Meaningful icon scheme that labeled the 12 buttons with icons that were representative of the 12 different functions/instructions;
- (2) Meaningful text scheme that labeled the 12 buttons with instruction texts;
- (3) Arbitrary scheme that labeled the 12 buttons with unrelated random icons;
- (4) No-label scheme that labeled the 12 buttons with no label.

Since the map exercise is not used in the development of our model, the graphics are only shown in Appendix A.

The target button on exercise three displays a verbal instruction instead of a color. There were 12 possible instructions that could be displayed: Pan right, pan left, pan up, pan down, zoom in, zoom out, view assignment area, hide street names, show user's location, show address list, view map spots, and close map. Depending on which of these twelve instructions being displayed on the white label, the users selected the appropriate button. Exercise 3 had the additional feature that a map was displayed. Figure 3.1d illustrates the layout of screen for exercise 3. More details on Exercise three are presented in Appendix A.

3.1.1.2 Tooltips and trial scenarios

A tooltip feature was implemented for all the buttons used in the three exercises. The tooltips are triggered when the user hovers above the button for approximately 0.2-0.5 seconds. Java built-in tooltips were used to provide this functionality. The tooltips display information about the function of the buttons. They were especially useful to the users in the case of the arbitrary and no-label schemes. Figure 3.3 illustrates the tooltip functionality of the three exercises when the no label treatment is used.



Figure 3.3: Tooltips available to users (Circled area).

The labels (when present) and the tooltips are used to guide the users in terms of which button to select. Here is a scenario steps of what can happen in a trial (there is also a set of illustration figures of this sequences of scenario in Appendix A):

1. The target button/label is presented to the user;
2. The user clicks on the target button/label and 12 buttons will be displayed in a particular layout (circle, grid, or map) using one of the label treatments;
3. A user selects a button;
4. If the selection is incorrect, there will be a message box with a message saying that the selection is incorrect and asking the user to try again; otherwise, the correct button has been selected and the Xs or instructions on the target button/label will disappear for half second;
5. The next trial will begin with the new target color icon 1 second after the user makes a correct selection from the previous trial.

The users study experiment consisted of 3 (exercises) x 16 (blocks) x 12 trials.

3.1.2 Software Randomization Input

The software was designed so that the layout and trials were randomized. Therefore, all users had a different combination of layout, labeling scheme, and exercise order. The randomization was prepared external to the software using a SAS (Statistical Analysis Software) program (2008) developed by Prof. Sarah Nusser. The procedure involved the following randomization terms:

- The order of target color being presented on each block and on each location test;
- The ordering of the 12 colors on the 12 buttons;

- The order of the circle and grid exercises for users.

The SAS code generated a lookup spreadsheet (csv file), which was used as input into the software. The software then reads this spreadsheet in to determine which users belonged to which layout, labeling scheme, and exercise order depending on the inputted ID on the configuration window (The SAS code created by Prof. Nusser for the randomization is presented in Appendix B).

In order to achieve a good distribution in terms of user assignments and thus an unbiased result, the SAS program used the four factors below to determine user assignments to a particular labeling scheme and exercise order:

1. Button Label (Treatment: color-match, meaningful-text, arbitrary, and no-label);
2. Order in which exercises are given (whether grid or circle is first exercise);
3. Gender;
4. Age.

When a participant was recruited, depending on how many participants had been assigned so far to one of the four treatments, s/he would be assigned one of the button labeling schemes. Furthermore, s/he would also be assigned an exercise order, and these assignments are distributed equally based on their gender and age, thus we can get enough data for each age or gender category to do our modeling.

3.1.3 Software Measures

During the course of operating the software, all user activity was recorded on a log file for the purpose of analysis. For the individual trials, the following actions were recorded:

1. Current block and trial numbers;
2. Target color being generated and the time of generation;
3. Tooltip number being generated and time of generation;
4. The color of the button being clicked in response to the target color, time of the click, and whether the click is a correct click (e.g. the color of the button clicked match with the color of the target button);

The software also generated three summary log files: Summary log one (for exercise one), summary log two (for exercise two), and summary log three (for exercise three).

The summary log recorded the following data:

1. The total attempts for each block;
2. The mean time for each block;
3. The total attempts for all of the blocks;
4. Average number of attempts required to identify the correct button;
5. Location test related data (see Section 3.1.4):
 - a. Original button coordinate of tested color;
 - b. User's placement coordinate;
 - c. Target hit: correct/incorrect;
 - d. Distance between the user's placement coordinate and the original button coordinate.

The log file was originally formatted so that it is both human and machine readable. This is important because it is very critical to analyze the data by looking back at what happened in the course of experiment as indicated by the log file. The log has to be machine readable also in a sense that it is easy to interpret the log and gather the necessary data listed above. For this purpose, a compiler program was developed and associated with gathering, rearranging, and summarizing the data from the log. In the end, this compiler generates the master spreadsheet containing the necessary information for the model.

3.1.4 Location learning

After the three exercises, we used a location learning test similar to that used by Ehret (2000, 2002). In this test, participants were presented with 12 trials. On each trial, an interface of target button and a drag-and-drop button were presented to the participant. Depending on the target button color, the participant was instructed to place the drag-and-drop button on the location where button with the same color was originally located in the exercise. When the drag-and-drop button was being moved, its color changed to dark green to indicate that the button was being moved. Figure 3.4 shows an illustration of the location test.

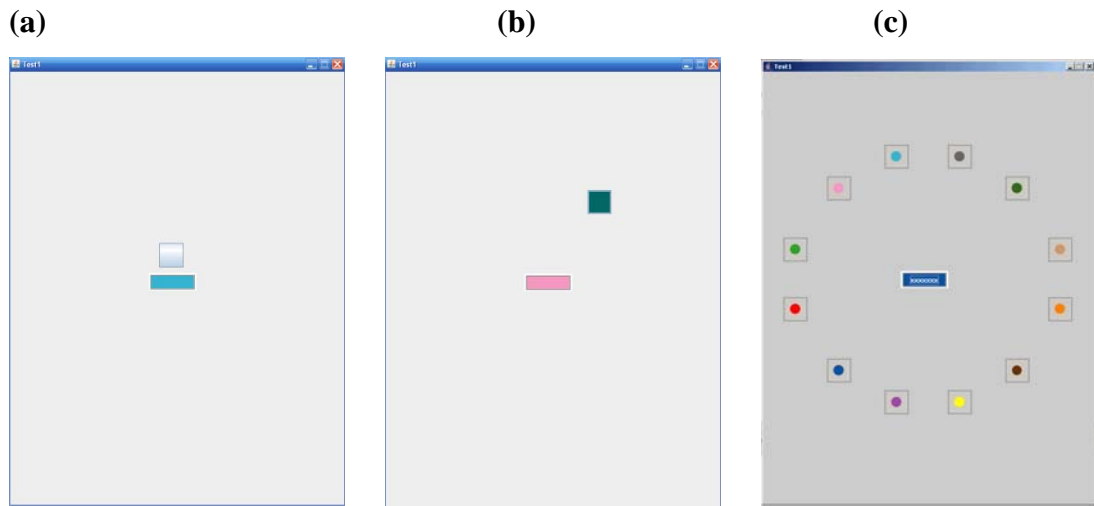


Figure 3.4: Location learning test illustration: (a) beginning of test, (b) drag-and-dropped button being moved, and (c) the original exercise buttons' positions.

Figure 3.4c above shows the original location of the 12 buttons in the current exercise.

The target hit data was based on a proximity analysis where proximity was half the distance to the next button (Figure 3.5).



Figure 3.5: The proximity of the correct/hit area.

The idea was when a user placed the drag-and-drop button on the appropriate circled area, that action would be recorded as a hit or correct placement. The same idea was used for the location learning test of the grid exercise and map exercise, but the circled area would be smaller on the grid and map because the distance between the buttons were closer.

3.1.5 Technical details

The software used was written in Eclipse IDE (Integrated Development Environment) with Java language. A java plug-in, window builder, was used for the ease of a drag-and-drop approach for building the interface.

3.2 Experiment Summary and Supporting Software

Michelle Rusch (2008) used the software discussed in Section 3.1 to conduct the user study. Here, only a brief summary of what was done in the user study will be discussed.

Table 1: Preliminary tests (VZ-2, MV-2, and P-2 tests were taken from Ekstrom, et al (1976))

Test	Method	Purpose
Background questionnaire	Computer-based	Collecting information for user background (age, gender, computer literacy, occupancy).
Color training/test	Computer-based	Confirming participant's eligibility.
Visualization test	Paper-based (VZ-2)	Measuring participant's spatial ability.
More cognitive tests: Memory and perceptual tests	Paper-based (MV-2 and P-2)	Measuring additional aspect of participant's spatial ability.

This summary is necessary to present the idea of how we obtained the data on which the model is based (Chapter 4). Michelle Rusch divided the recruited participants into different spatial categories based on the spatial ability test scores. The tests (Table 1) were given to the users before the user study experiment was conducted. The paper-based tests were adopted from Ekstrom, et al. (1976). The result of these tests was used to guide the grouping of the participants in terms of their spatial ability. This will be discussed further in the following subsection. The color training/test and the background questionnaire were administered by using different software. A snapshot of the color training/test software is shown in Figure 3.6.



Figure 3.6: Interface of the color training/test software showing the color being tested and the list of color names to choose from.

The color training/test was used to confirm that the user had sufficient color vision to operate the user study software and to train them on the color associations. The other test was a background questionnaire that collected personal data about the participant. The list of the questions displayed on the questionnaire software interface is included in Appendix A.

Rusch (2008) conducted the experiment with 80 participants by having them operate the user study software deployed to the tablet PC. However, some of the participants had to be eliminated due to failing to complete the exercise resulting in data from 63 participants forming the basis of the cognitive model described in Chapter 4.

3.3 Result and discussion of user study experiment

In this section, we will be focusing our discussion on the results of the experiment that relate to the development of a cognitive model incorporating spatial ability. These results defined the behavior of the model. The next subsection presents the results of the user study experiment that are relevant to the development of our cognitive model.

3.3.1 Relevant results captured from the experiment

One way to determine how well a user did on the experiment was to see how long it took them to complete one trial in the experiment. Thus, the time a user needed to select the correct button was recorded and this aspect would be used on the modeling phase (Chapter 4).

However, now the question is “what contributed to time?” Ehret in his studies (2000, 2002) indicated that the tasks of the experiment involved search time and evaluation time (see Section 4.1.2.2). These two aspects were necessary and formed the production structure of the cognitive model discussed in Chapter 4. Thus, it was important to record how many times a user had to evaluate a button that was captured from the number of movements to a particular button made by the user. Another thing that contributed to time was the number of clicks a user needed to complete one trial. For example, the more incorrect clicks a user had, the more time needed to complete one trial. Therefore, the number of attempts/clicks was also recorded.

Another important result of the experiment relevant to the model was the visualization score. The grouping of spatial ability was based on different aspects (visualization, perception, spatial location memory, spatial orientation, and so on). We conducted three tests with regard to visualization VZ-2, perception (P-2), and memory (MV-2). Rusch (2008) compiled the data and conducted an analysis on the results. But, only the effect of visualization was significant in the experiment. Therefore, only the visualization score was used for grouping. The result of the paper folding test gave a range of score from 1 to 19. Thus, the users were broken down into 2 groups: low spatial group (score 1-9) and

high spatial group (score 10-19). This grouping is an important baseline for our spatial ability modeling.

All these results were recorded and compiled by the compiler program mentioned in the previous Section. A final spreadsheet was generated and this would be used accordingly in the cognitive modeling process discussed in Chapter 4.

3.3.2 Limitation of user study experiment design

There was an important limitation on the experiment with regard to the use of tablet PC, which used stylus instead of a mouse. There were two different ways of using stylus that were observed. First, a user would hold it close to the screen so the cursor movement was logged. Second, the user would pull the stylus away from the screen and only get close to the screen to tap the buttons as necessary. In the latter case, the recorded cursor movement was jumpy so the buttons in between the pull-up of the stylus and the tap were skipped and, thus not logged/recorded as tooltips activations. In the first case, the tooltips were logged as the cursor was hovered over the buttons whether it was intentional or unintentional.

The second problem was that tooltips were recorded whenever the user had the stylus close enough to the tablet screen and the user entered the button area. This issue is addressed further in the next chapter and a solution to the model design is also explained.

CHAPTER 4. COGNITIVE MODEL

In this chapter, the cognitive model of the user study described in Chapter 3 is developed using ACT-R 6.0 (Budi 2008) and SAS (Statistical software 2008). This model is to validate our understanding of the different components that show the impact of different levels of spatial ability. The result from first two exercises of the study described in Chapter 3 will be used to provide the basis of the behavior of the model. Due to the limitations of ACT-R 6.0 with respect to spatial considerations, the model consisted of two parts: the cognitive mechanism part (ACT-R 6.0) and the user behavior control part (SAS program). The ACT-R part is the framework through which, the control list generated from SAS program (control part) runs. The mechanism includes the productions and the ACT-R 6.0 parameters. The model interacts with the interface from the user study. The success of the model would be tied to how well the model simulates the time of the first two exercises. Thus, the objective of the model would be tested and evaluated based on the simulation performance. The following is how the chapter will be broken down.

We will start with an overview of the ACT-R 6.0 components that are necessary to generate the model and some discussion of why we picked ACT-R 6.0.

4.1 Overview of ACT-R 6.0

ACT-R is a cognitive modeling mechanism (Budi 2008). The ACT-R program was built using the LISP language. The ACT-R API was written to support coding the model with some built-in functions. The important aspects of ACT-R 6.0 are explained in the reference manual by Dan Bothell (2007).

4.1.1 Cognitive architectures comparison

Besides the rich API support that ease the process of building a model with the ACT-R cognitive architecture, one question is why ACT-R? As mentioned in the theoretical background chapter that there are several major cognitive architectures being developed that include LICAI/CoLiDes, Soar, and EPIC (Brain, Cognition, and Action Laboratory 2002).

What we have to know is the major features that different architectures provide and how well they suit our needs. A detailed comparison is given by Michael D. Byrne (2003).

LICAI (Kitamura 2001), which was recently upgraded to CoLiDes (Comprehension-based Linked model of Deliberate Search) (Kitamura 2002) is a modeling architecture that is focused on text comprehension for tasks that requires representing and understanding texts for doing the desired tasks. However, for our purpose, LICAI is not adequate since it has no learning mechanism and no perceptual and motor systems that provide understanding software interface and conducting motor actions such as mouse movements, eye attention shift, and many more, required by our experiment. EPIC on the other hand is a cognitive modeling architecture that provides a detailed perceptual-motor system. However, the drawback of EPIC is that it has no learning mechanism which is needed in the user study experiment. Even though EPIC has strength in parallel processing of multiple tasks, the lack of learning mechanism seems to be a downfall of the architecture with regard to modeling spatial ability, especially the visual memory aspect. Soar (Newell 1990) provides both a perceptual-motor system and learning mechanism that could facilitate the development of our model. Like ACT-R, Soar uses a production system that acts as a procedural memory that encompasses the atomic steps in human cognitive action. But, Soar cannot handle high-performance perception like ACT-R does. An example of high-performance situations would be a driving task simulation developed by Salvucci (2001b) to illustrate user driving performance when interrupted by other task such as dialing on a cell phone at the same time. Since ACT-R provides high-performance tasks, learning mechanism, and perceptual-motor systems, it is the architecture model that is selected for this experiment. Table 2 (adopted from Byrne (2003)) summarizes the different architecture modeling features.

4.1.2 ACT-R 6.0 cognitive architecture

ACT-R is a program to instantiate human cognition by developing architecture with interconnecting building blocks. The major building blocks of ACT-R are the productions (procedural memory module), declarative memory module, other modules (motor and visual), buffers, and some parameters that control the way the whole system works. This is

Table 2: Architecture feature comparison (taken from Byrne (2003)).

	LICAI/ CoLiDeS	Soar	EPIC	ACT-R/PM
Original focus	text comprehension	learning and problem-solving	multiple-task performance	memory and problem-solving
Basic cycle	construction-integration	decision cycle	production cycle (parallel)	production cycle (serial)
Symbolic or activation-based?	both	symbolic	symbolic	both
Architectural goal management	special cycle types, supports vague goals	universal subgoaling	none	goal stack
Detailed perceptual-motor systems	no	no	yes	yes
Learning mechanisms	no	yes, pervasive	no	yes, but not pervasive
Large, integrated models	no	yes	no	no
Extensive natural language	yes	yes	no	no
Support for learning system	none	FAQ, some tutorial materials	none	extensive tutorial materials, summer school
User community*	none	some, primarily AI	none	growing, primarily psychology

only the framework, which a programmer can use to fill in details about the design of the productions, the setup of the parameters, and the flow of the productions to represent a set of psychological processes involved in a given task. Thus, in short, ACT-R is a series of productions, which are pairs of conditions and actions. ACT-R is based on a production cycle system where a run of an ACT-R model is basically a run of cycles where on each cycle, based on the state of the system, a matching condition will fire an action and transition the system to a different state. The cycles will end when there are no more productions with a matching condition (end state).

The state of the system is comprised of the buffers, such as, the goal buffers, retrieval buffers, imaginal buffers, vision buffers, and manual buffers. The buffers act as two things,

interfaces between the processing module and the production systems, and properties of modules. For example, the retrieval buffer is associated with the declarative memory module. The declarative memory module is a module that represents a long-term memory where facts and information are stored as chunks. A chunk represents information/fact by its type, and by its properties (name, value, and many more). For example, information about number “5” may be stored in the declarative memory as a chunk of type number (*isa* number) and a numerical value 5. The declarative memory module handles the processing of memory including extraction and storage of chunks. Buffer as a property module can be observed in the motor module with a manual buffer that represents a command property. For example, the manual buffer tells the motor module the information about what command (key-press, mouse-move, mouse-click, and so on) to execute. Figure 4.1 illustrates the cognitive architecture of ACT-R.

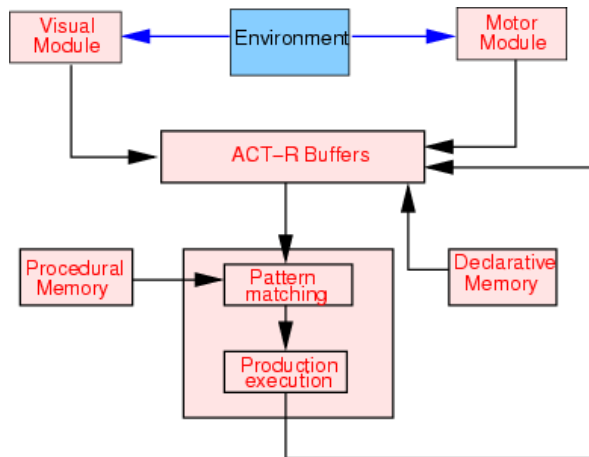


Figure 4.1: ACT-R 6.0 Architecture (Budiu 2008).

However, as promising as it sounds, ACT-R does not come without limitations. Some of the problems that we realized after a year of working with ACT-R 6.0 are that the architecture is lacking the mechanism to support performance variability based on variability of spatial ability levels (the exact thing that we are trying to model). In reality, people with higher level of spatial ability perform better in a task that requires processing of spatial information, but, there is no mechanism for spatial information processing in ACT-R 6.0 at this point. This issue has been discussed in Gunzelmann’s Mechanism of spatial competence (2006). He proposes a new spatial module for processing spatial information. In the absence

of Gunzelmann's spatial module, we introduce an external component built on a SAS program.

The issue is since we can't model delay time based on spatial transformation because it is not supported by ACT-R 6.0, we created a generator that generates a control list that specifies times associated with each action in a sequence of user behaviors. This control list drives our ACT-R model performance by introducing a delay time in between productions that substitute the lost spatial transformation delay time that we need. In the next section, the detail of the control list generation with the SAS program will be discussed.

4.1.2.1 SAS program (model control part)

The SAS program is needed to generate user behaviors as inputs to the ACT-R model. The program is working with the user behaviors recorded during a user study experiment through the compiler program of the test software, as discussed in Chapter 3. The program consists of three major parts: the compilation part for generating variables to describe user behaviors for each trial, the calculation part for estimating parameters of distributions that describe the range of user movement and clicking behaviors for each trial and spatial ability level, and the generation part for randomly generating values for behaviors for each trial performed by each individual. Relevant data for this program were:

1. Experiment related information (block #, trial #, user ID #, label treatment, and layout);
2. Trial related information (User total attempt time in a trial, number of attempts in a trial, and trial target color);
3. User related information (age, gender, and visualization score (VZ-2) used to classify each user into a spatial ability category);

First, the compilation part of the SAS program involved generating the variables that describe user behaviors for each trial. We began by importing the spreadsheet from the user study experiment and grouping the data into two groups: low spatial ability group ($VZ < 10$) and high spatial ability group ($VZ \geq 10$). The data was processed further to provide information for simulating individual behaviors for the user study task component. The task in a trial involved a sequence of actions applied to particular buttons. This action includes

click action (clicking a button) or movement action (moving to a button). From now on, these actions will be called events. Furthermore, there was a time associated with every event and also a selection of which button an event was applied. An event was described by these three entities (time, event type, and button selection). Based on the total trial time and the number of moves made by the user in a trial, we calculated an average time-per-move to represent movement time per event (time per move). For example, if the total attempt time was 22.5 seconds and there were 5 events, then the time-per-move is 4.5 seconds. We also calculated the number of movements (x_{moves}) and the total number of clicks (x_{clicks}) prior to the final correct click for the trial. Figure 4.2 is a partial snapshot of the SAS program that illustrates the compilation process.

```
PROC IMPORT OUT= WORK.events
           DATAFILE= "C:\Documents and Settings\and11241\Desktop\SAS\cogmodel_lists.csv"
           DBMS=CSV REPLACE;
           GETNAMES=YES;
           DATAROW=2;
RUN;

data events ; set events ;

           time_per_move = attempt_time / movement_sum ;
           x_clicks = click_sum - 1 ;
           x_moves = movement_sum - click_sum ;

           if vz < 10 then spatial_cat = 1 ;
           if vz >= 10 then spatial_cat = 2 ;

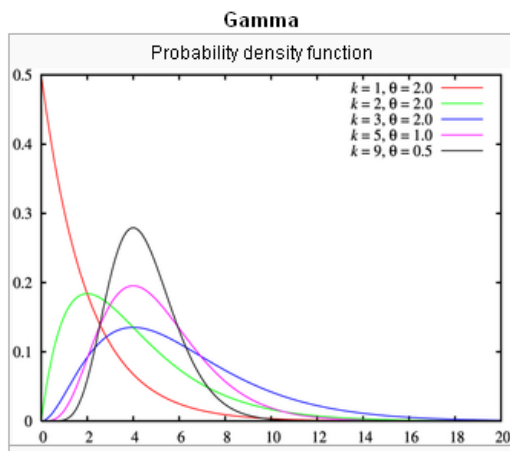
run ;
```

Figure 4.2 SAS program (compilation part)

The second part of the SAS program is the calculation part, which summarized the user data to provide parameter estimates for distributions used to simulate the number of individual moves and clicks and their corresponding times. Parameter estimates were calculated within spatial ability groups for each trial within each block corresponding to a given exercise. The number of individuals contributing data to each of these groups ranged from 2 to 14.

The gamma distribution was selected for generating the time associated with each event. Event times are continuous non-negative values, which was why Gamma distribution was selected. The Gamma distribution is parameterized by two parameters: shape parameter

and scale parameter (Gamma distribution-wikipedia 2008). In order to calculate these two parameters, the variance and the mean time per event (mtime_per_move and vtime_per_move on the code) had to be calculated from the compiled data for each trial. This was done with the SAS program by using the proc means routine (see Figure 4.5). The estimators of the scale parameter (beta on the code) and shape parameter (alpha on the code) are given by variance/mean and mean/beta respectively. Depending on the value of the alpha and beta, a probability density curve is returned by a probability density function as illustrated on Figure 4.3.



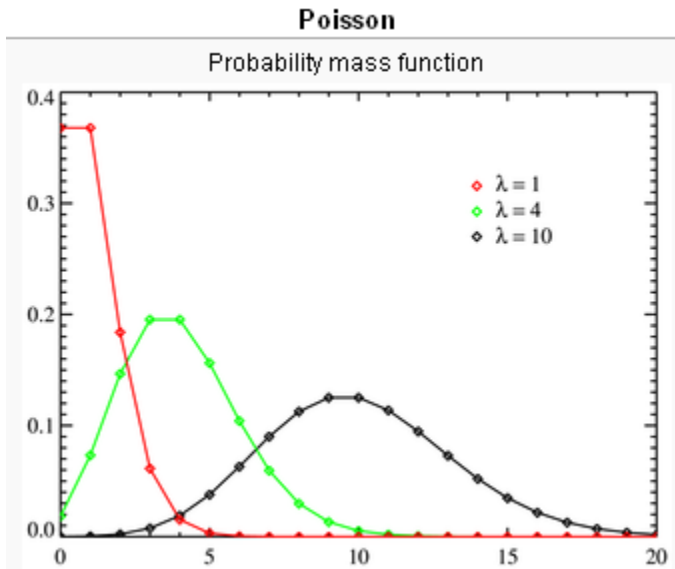
$$f(x; k, \theta) = x^{k-1} \frac{e^{-x/\theta}}{\theta^k \Gamma(k)} \text{ for } x > 0 \text{ and } k, \theta > 0.$$

Figure 4.3 Probability density graph (Gamma distribution-wikipedia 2008)

The area under the curve sums to one and this is called probability density function (e.g. the fatter the area under the curve, the more likely the value within that range is returned and this value is given as x (the horizontal axis on Figure 4.3)).

The type of event is the second entity that needed to be modeled. An event consisted of two types: click event and movement event. For click event, there was a minimum of one click for each trial because a user had to click the correct button to complete the trial. For the movement event, there was a minimum of zero movement because a user could immediately click the correct button without having to move around between buttons. Since the value of the number of events applied prior to the correct click is given by non-negative integer, it was modeled by Poisson distribution (Poisson distribution-wikipedia 2008). Because we

used a separate Poisson distribution for click events and movement events, both mean of click (mx_click on the code) and movement (mx_move on the code) needed to be calculated. The Poisson distribution mean (lambda) is the parameter and the distribution has probability mass function as illustrated on Figure 4.4.



$$f(k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!},$$

Figure 4.4 Probability mass graph (Poisson distribution-wikipedia 2008)

Unlike Gamma, Poisson distribution is a discrete probability of k occurrences (horizontal axis on Figure 4.4). For example, for lambda equals to zero and one (red line on Figure 4.4), the probabilities are both around 0.37. But, compared to the other k (three, four, five, and so on), zero and one are most probable since the curve is going down approaching zero for higher k.

The last entity that needed to be modeled was button selection. Selection corresponded to the button on which an event was operated in a Trial. Different kinds of buttons may be selected prior to clicking on the correct button for the trial. For example, within one trial, there may be 10 events (2 on red buttons, 4 on green buttons, and 4 on blue buttons), and the relative frequencies for each of these events would be 2/10 (for red), 4/10 (for green), 4/10 (for blue) and 0 for the rest of the buttons. Since there were up to 12

buttons to be selected, we used multinomial (instead of binomial) distribution to simulate button selection. This distribution is parameterized by the probability for each of the 12 events. The calculation SAS code is illustrated on Figure 4.5.

```

proc sort data=events ; by spatial_cat label_trt layout block trial ;
proc means data=events noprint ;
  by spatial_cat label_trt layout block trial;
  var time_per_move
      x_clicks
      x_moves
      position_1 position_2 position_3 position_4 position_5 position_6 position_7 position_8
      position_9 position_10 position_11 position_12;
output out=plotmeans
  mean = mtime_per_move/*attempt_time_mean*/ mx_clicks/*click_sum_mean*/ mx_moves /*movement_sum_mean*/
      mposition_1 mposition_2 mposition_3 mposition_4 mposition_5 mposition_6
      mposition_7 mposition_8 mposition_9 mposition_10 mposition_11 mposition_12
  var = vtime_per_move /*attempt_time_var*/
  n = nsubjects ;
run ;

proc sort data=events ; by spatial_cat label_trt layout block trial ;
proc sort data=plotmeans ; by spatial_cat label_trt layout block trial ;
run ;

data trial_merged ;
merge events plotmeans ; by spatial_cat label_trt layout block trial ;
run ;

data sim ;
set trial_merged ;
x=-1;          * random seed ;
*** preliminaries ;

* gamma parameters ;
beta = vtime_per_move / mtime_per_move ;
alpha = mtime_per_move / beta ;

* scaling position means to conform to multinomial ;
mposition_sum = mposition_1 + mposition_2 + mposition_3 + mposition_4 + mposition_5 + mposition_6 +
               mposition_7 + mposition_8 + mposition_9 + mposition_10 + mposition_11 + mposition_12 ;
array mposition {12} mposition_1 - mposition_12 ;
array p_position {12} p_position_1 - p_position_12 ;
do k = 1 to 12 ;
  p_position{k} = mposition{k} / mposition_sum ;
end ;

***continuing to the generation part...

```

Figure 4.5 SAS program (Calculation part).

The last part of the SAS program is the generation code. On this part, the estimated parameters were used to generate the time per event (using the gamma distribution), number of events (using the Poisson distribution), and the selection (using the Multinomial distribution) using SAS functions `rangam`, `ranpoi`, and `rantbl` respectively (Figure 4.6). The seed used for the randomization was -1, which meant that it changed in every generation using the clock/time of generation as the seed. The generation code is given on Figure 4.6.

```

*** create first event, which is a click at time 0 in the center ;
    event = 1 ;
    time = 0 ;
    /*pos_rand*/position = 0 ;
    output ;

*** draw the number of extra clicks (x_event1) beyond the first click
draw the number of extra movements (x_event2) beyond the number required for (x_event1 + 1) clicks
assume that distribution of number of events of either type is Poisson with mean equal to the mean from plotmeans ;
if mx_clicks > 0 then x_event1 = ranpoi(x,mx_clicks) ; * number of extra clicks ;
    else x_event1 = 0 ;
if mx_moves > 0 then x_event2 = ranpoi(x,mx_moves) ; * number of extra movements ;
    else x_event2 = 0 ;

*** generate extraneous movements (type 2 events) by setting time and button (position) for each of event2 movements
assume time follows gamma distribution with beta = var/mean and alpha = mean/beta (mean, var from time summary statistics)
assume button choice (position = 1, 2, 3, ..., 12) follows multinomial distribution with probs from summary stats ;
if x_event2 > 0 then do i = 1 to x_event2 by 1 ;
    event = 2 ;
    time = beta*rangam(x,alpha) ;
    position = rantbl(x, p_position_1,p_position_2,p_position_3,p_position_4,
        p_position_5,p_position_6,p_position_7,p_position_8,p_position_9,p_position_10,
        p_position_11,p_position_12) ;
    output ;
end ;

*** generate clicks and associated movement
excludes initial start click, extraneous movements, and final correct click
assume time follows normal distribution with mean and sd from summary statistics, set minimum time to .1 second
assume button choice (position = 1, 2, 3, ..., 12) follows multinomial distribution with probs from summary stats ;
if x_event2 > 0 then do i = 1 to x_event1 by 1 ;
    event = 1 ;
    time = beta*rangam(x,alpha) ;
    position = rantbl(x, p_position_1,p_position_2,p_position_3,p_position_4,
        p_position_5,p_position_6,p_position_7,p_position_8,p_position_9,p_position_10,
        p_position_11,p_position_12) ;
    output ;
end ;

*** generate last click ;
    event = 1 ;
    time = beta*rangam(x,alpha) ;
    position = target_color ;
    output ;
run;

```

Figure 4.6 SAS program (Generation part)

The SAS program ended and it generated a list, which would be used to control the behavior of the cognitive model developed by ACT-R. This part will be discussed in the next section.

4.1.2.2 Cognitive model

The Cognitive model is an ACT-R 6.0 program written to provide a framework through which the control list can be run. Another important part of this model is setting up the ACT-R parameters correctly to give a realistic result. This model consists of three major parts: LISP setup code, ACT-R production code, and ACT-R parameters.

The setup code is LISP code that sets up the same interface as the one used in exercise one and two from the user study, only that this is a stripped down version. The technicality of the interface setup is out of the scope of this section and is thus made available in Appendix B.

The productions of the model were based on Ehret's notion (2008) on the underlying mechanism of the task that includes search time and evaluation time. Search time would

involve a user trying to find the location of the button and this, like explained on the theoretical background as several research suggested, included how people represented spatial information in their memory (exocentric, egocentric, or hierarchical). The evaluation time involves people trying to use the information about the function of the button to decide on a correct button. The button information could come from utilizing the tooltip, shifting attention to the button to gather the color information (color or text treatment), or retrieving information from their memory from previous experience on the button being evaluated.

Below are the productions of our model.

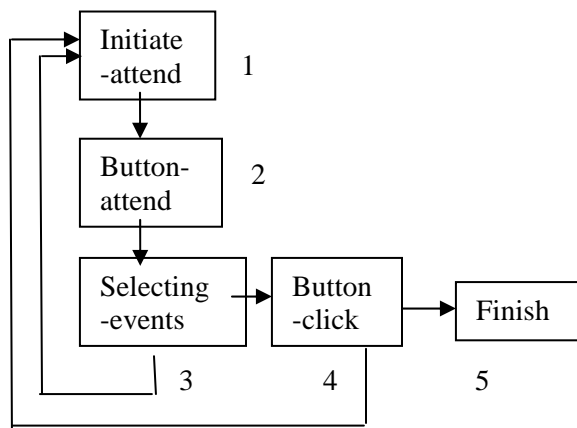


Figure 4.7: ACT-R model productions

The ACT-R code for the productions is available in Appendix B. The cognitive process described by the productions can be summarized as:

1. A user prepares to attend a button and is trying to decide which one;
2. A user is attending the button, which includes moving their attention to the button and moving their cursor to the button;
3. A user is trying to decide whether s/he should click this button or move his/her attention to a different button;
4. A user is clicking the button. When a user clicks on a button, the production either cycles back to the initiate-attend or to the finish depending whether there is more trials to run on the experiment;
5. A user finishes the experiment.

```

(p initiate-attend
  =goal>
  isa the-goal
  state start

  =imaginal>
  isa array
  - elements nil
  elements =list

  ?manual>
  state free

  ?visual>
  state free

  !safe-bind! =x (bind-x =list nil) ;harvesting the x position of the button
  !safe-bind! =y (bind-y =list nil) ;harvesting the y position of the button

  ==>

  !eval! (append-trial-time =list (mp-time))
  !eval! (append-block-time =list (mp-time))
  !eval! (append-exercise-time =list (mp-time))
  =goal>
  state attending

  =imaginal>

  +visual-location>
  isa      visual-location
  kind oval
  screen-x (+ 22 =x)
  screen-y (+ 22 =y)
)

```

Figure 4.8 Initiate-attend production code (Productions code available in appendix B)

Figure 4.8 illustrates one of the productions code. The left hand side is the “if” part of the production. If the condition matches (e.g. goal buffer isa the-goal, state start, imaginal buffer isa array, and so on), the right hand of the production will be executed. The detail of production structure is covered in ACT-R 6.0 reference manual (Bothell 2007).

The way the production works is controlled by ACT-R parameters. On this matter, we are more concerned about the timing of the production since this will affect how long a user spend his/her time trying to click a button, moving his/her attention to a button and so on. There are two kinds of parameters in ACT-R that concerns us on this matter: production parameters and general parameters. The production parameters control how the production is working including delay time before a production is fired. This delay time is controlled by the: at parameter. The global parameter controls other ACT-R components such as the

modules and the buffers (e.g. `:needs-mouse`, `:incremental-mouse-moves`, `:show-focus`, `:trace-detail`, `:do-not-harvest`, and many more). More parameters that can be set are explained in the ACT-R 6.0 reference manual (Bothell 2007). However, with regard to how the model works, the following are the parameters that must be set:

- Needs-mouse: this parameter decides whether the model needs the control of the mouse cursor to run the simulation. This parameter is set to true because obviously we need the control of the mouse to run the simulation of the user study experiment;
- Incremental-mouse-moves: this parameter tells the model to show the mouse movements in an incremental manner so we can see the mouse movement smoother. This is currently set to true, to see continuous cursor movement;
- Show-focus: this parameter tells the model to show the focus of the attention on the interface;
- Trace-detail: this parameter tells the model to show the trace of the model run with the given detail (high, medium, and low) as specified by the user;
- Do-not-harvest: this parameter tells a module related to the buffer being specified by the user to not harvest or set the buffer automatically when the model runs;
- At: the at parameter set the delay time of the firing of a production after it gets selected due to the matching condition (left hand side of the production);

The parameters mentioned above are the relevant ones that need to be set correctly so that the model runs in the way that we expect it to run.

This is when the control list comes into play. The expectation of how the model should run was controlled by the list generated from the SAS program. Recalling that the list contains 6-tuple (1 2 3 4 5 6) of information, the following was how the information was used within the model:

1. This is a control number on the exercise type. The model will use the exercise type (1= circle and 2 = grid) to load the correct layout of the interface;
2. This is a control number on the block. The block number will tell the model when the transition between blocks happen, which is when the mean time of the finished block being pushed to the simulated time list;

3. This is a trial number on the experiment. This does nothing to the model, but it is there to help with analyzing the trace of the simulation;
4. This is the event type (1 = click and 2 = move) that tells the model whether it is time to click a button or attend/move to a different button;
5. This is the position of the button that the model should click or move to;
6. This is the delay time that tells the model what value it should set the :at parameter to.

Other than these parameters, the model was controlled by the goal buffer that represented what task that the user was expected to do at a certain stage of the experiment (start, attending, clicking, selecting, and done). These were stored in the declarative memory as chunks and they controlled the transition of the production. For example, when the model was on the initiate-attend state and the goal buffer, on the right hand side of the production, was modified into attending, the model would select button-attend production and fired it. The end of the simulation was reached when the finish production was reached.

As the simulation ran, trial level, block level, and exercise level performance time were recorded. The performance of the simulation was then evaluated and compared with the original performance from the experiment to see how well the model simulated the variation of performance of different spatial groups of different layouts and treatments.

4.1.2.3 Model evaluation and discussion

The objective of the simulation was to give a good time performance model for individuals with different spatial ability (high and low). Thus, we want to see whether the model simulated time was close to the time of the original experiment. First, we tried to evaluate the model by looking at the total time comparison between the actual and the simulation per block. Figure 4.9 shows some plots for actual and simulation total time of different spatial groups, layout, and treatments. By looking at the plots, the result of the simulation was comparable to the actual performance overall (All total time comparison plots are available in Appendix C). However, there were several cases where the black line (simulation) failed to assume the shape of the blue line.

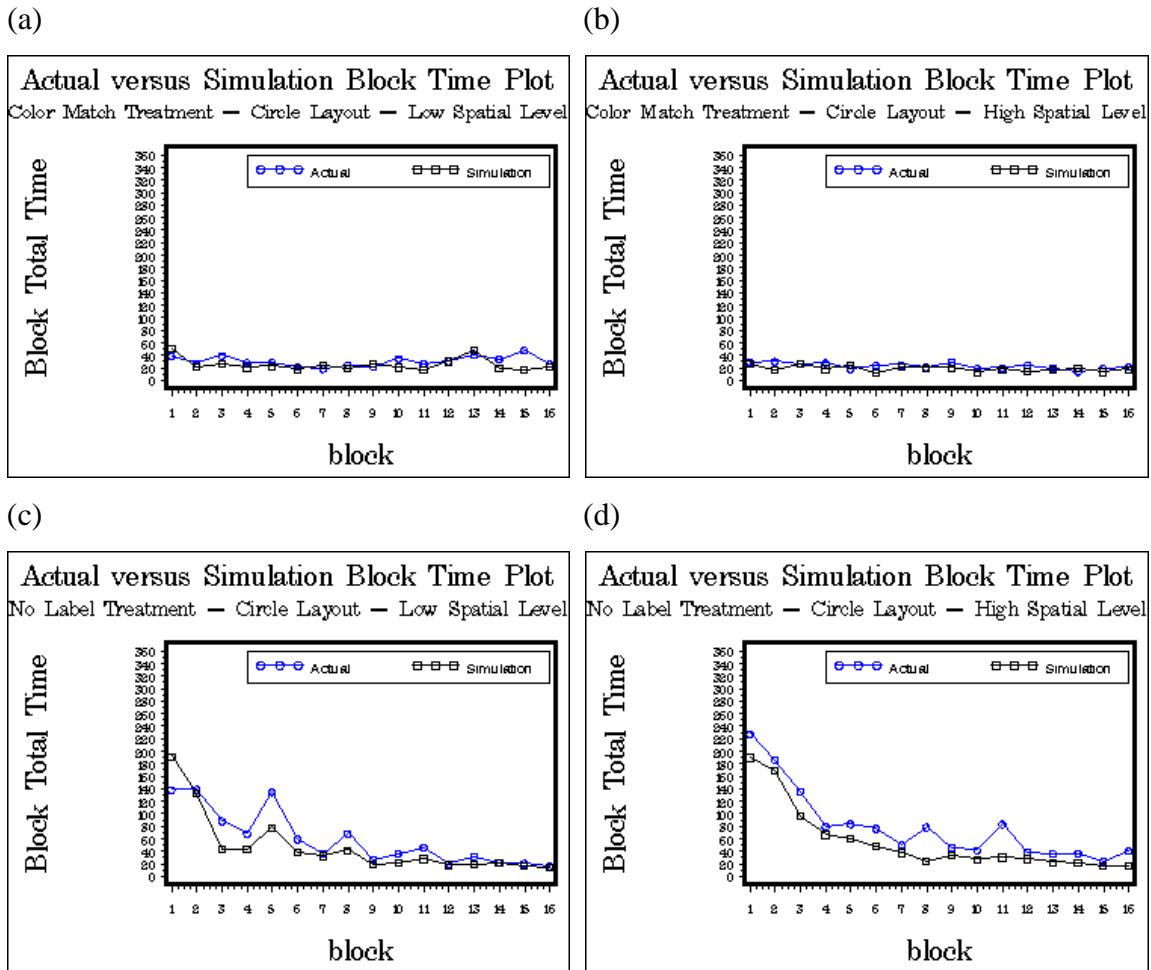


Figure 4.9 Actual versus Simulation total time

One case would be Figure 4.9d block time comparison of block 8 and 11. This suggested that the model only did a good job simulating the time to a certain degree that there were some variations that the model failed to capture.

In order to investigate further, we calculated the average time per block, the standard deviation for the time per block, and the min and max values for the time per block for each person for the simulated and actual data. Then, we plotted the summary statistics by treatment, layout, and spatial level category. Figure 4.10 shows the summary plots for one category (low spatial level, circle layout, and color-match treatment). The complete summary plots are also available in Appendix C.

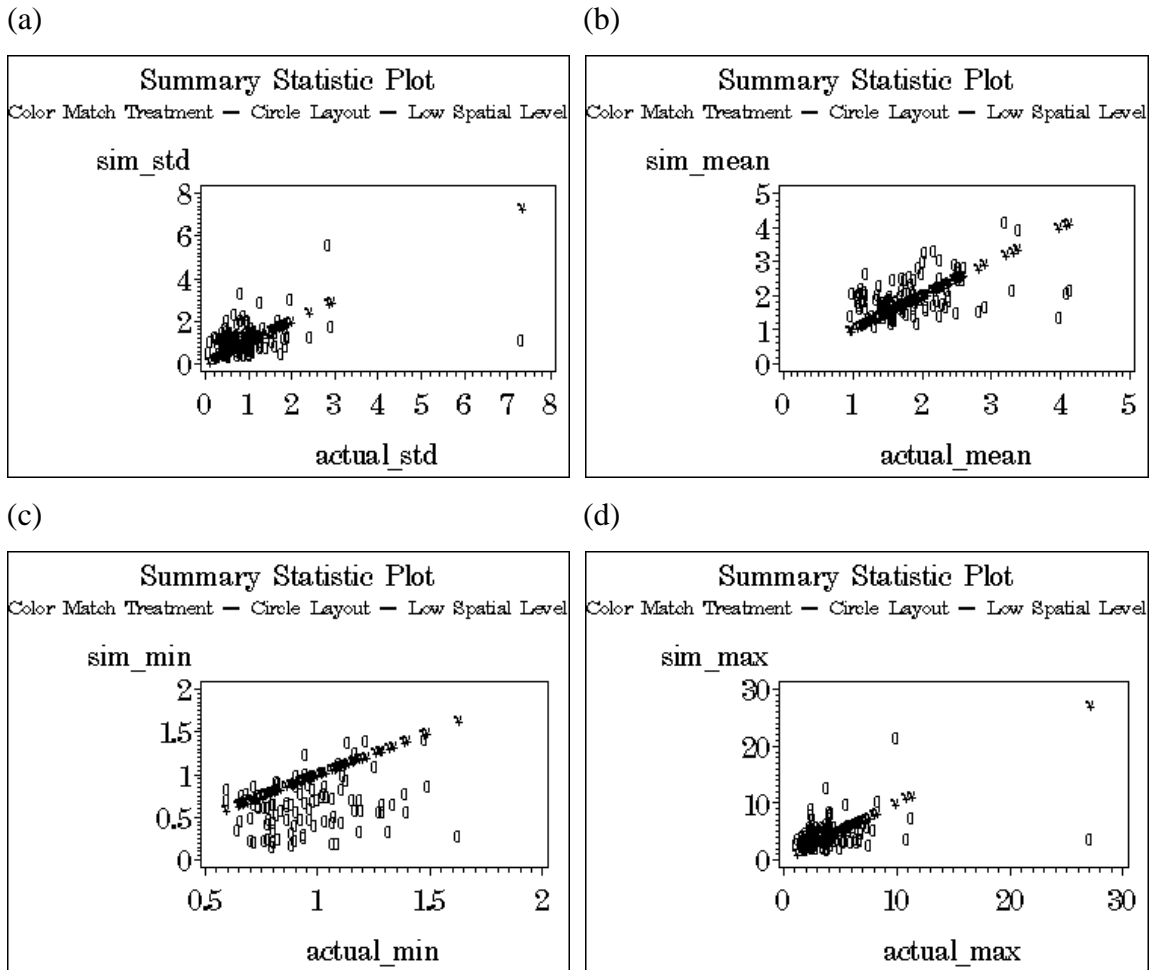


Figure 4.10 Actual versus Simulation summary statistics

The straight line is the line we would get if the simulation replicated the actual performance for all blocks. The circles indicate values comparing the summary statistic for the simulation (vertical axis) versus the actual (horizontal axis). Circles below the line means that the simulation summary statistics (min, max, standard deviation (std), or mean) was smaller than the summary statistics from the actual data, and above the line means that the simulation statistics were larger than the statistics for actual data. Circles are scattered along the line. Most of them are scattered very closely along the line, which is what we expect if the behavior of the model simulations was close to the actual experiment. However, the minimum time values for the model were generally too small relative to the actual

experiment. An explanation rests with the Gamma distribution used to generate the model time. It could be that the estimated Gamma parameters generated a distribution whose shape had a heavier left tail for smaller time values relative to actual data. This would mean that it was very probable for the model to generate low performance times. There are also circles that are way below the line such as those around the far right bottom corner for each of the summary statistics. This represents the behaviors from the actual data that somehow the model failed to capture. For example, small standard deviations from simulated data were paired with large standard deviations from actual data, perhaps due to outliers in the actual data. It is also possible that distributions for the number of clicks and moves per task did not capture those behaviors well.

The outliers in the actual data were investigated and it appeared to be connected to the factors that were not considered within the grouping process of the people. First, physical condition was a factor as it suggested how well a person could complete the experiment. For example, some older participants with high spatial ability performed poorly because their physical condition created difficulties in using the stylus. Another factor could be familiarity with the technology. The use of stylus was foreign to some participants and sometimes they had difficulty making the stylus function as desired.

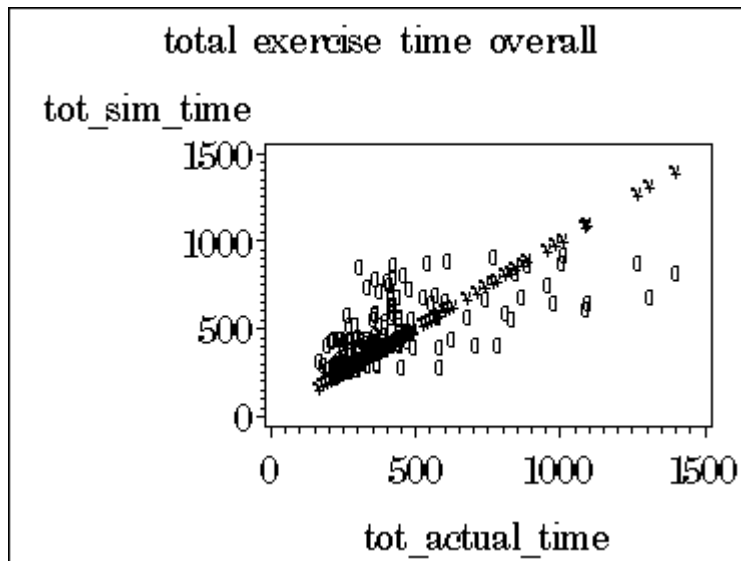


Figure 4.11 Actual versus Simulation exercise total time plots for all participants

Figure 4.11 shows the total time per exercise from actual and simulation performance for all the participants. This plot suggested further that though the model is not a perfect model, with a tendency to generate simulation times that were larger than actual times when the actual total time was small and a tendency to generate simulated times that were smaller than actual times when the actual total time was slow.

CHAPTER 5. SUMMARY

The effectiveness of a software interface design is tied to the skills of user of the software. In particular, it is tied to how users comprehend the spatial stimuli presented to them through the software interface (spatial information encoding). Thus, understanding the impact of different levels of spatial ability on software interfaces is necessary in order for good software design. The importance of this knowledge can then be applied to designing a more effective software interfaces to suit different levels of spatial ability.

In order to understand spatial ability, we looked at a few spatial ability aspects that had been identified as important to our research: visualization, perceptual speed, and visuospatial memory. Each aspect was testable using a protocol elaborated by Ekstrom (1976). Another important piece of the research was to provide a statistical comparison between the real users' performance and the model simulation performance.

In order to accomplish these objectives, the research required us to examine both empirical and analytical approaches. The empirical approach involved gathering performance data based on the task described in Chapter 3. The user study software was developed in cooperation with Michelle Rusch (2008). The data collected from this experiment was then divided into two spatial ability categories based on the spatial test scores. Furthermore, there were some limitations identified during the analysis of the data. These limitations and the result of the experiment were then used to guide the development of the cognitive model.

The purpose of developing the cognitive model was to verify our presumption of the impact of spatial ability. However, there are limitations of the cognitive architecture, which involved the lack of spatial information processing module in ACT-R version 6.0. This prevented the model from generating ACT-R time with regard to spatial information processing. An additional step was then introduced by providing an external component (SAS program) to control the behavior of the ACT-R model through a generated control list. The SAS program together with the ACT-R architecture then formed our cognitive model. There were 63 generated control lists based on the data from the 63 users with different spatial abilities and label treatments. The comparison between the actual performance and the simulation performance for two spatial ability levels were investigated. The model was

able to simulate the actual performance to a certain degree indicated by block level total time performance plots between the actual experiment and the simulation. However, to analyze the discrepancy between the two plots further, summary statistics for both performances were also plotted. In the end, even though the model was able to capture the performance trend of the real experiment, it failed to capture some variations of the real experiment performance. The minimum value of the model performance was mostly underestimated. Possible explanation for this would be the discrepancy between the density function curve estimated and the real distribution curve of the real experiment (e.g. fatter density area function along the lower interval for the estimated density function curve). Another variation was due to outliers and the model failed to capture this as well.

The outliers were believed to be caused by a combination of the user's physical condition, technology familiarity, and some software limitations. The software limitations included the inability to differentiate intentional or unintentional use of the tooltips based on the data that was logged. Another limitation from the experiment was the use of stylus that led to different ways of moving the cursor on the screen (jumpy or incremental). This led to skipped in-between processes and thus, some information about the user activity from one spot on the screen to another was lost due to the jumpy movement of the cursor.

CHAPTER 6. FUTURE DIRECTIONS

There are several issues that need to be addressed in this research. One issue regarding the classification method that is based on one spatial category that is visualization only might convey the different spatial ability on the experiment but only specific to this aspect. *Rusch* (2008) in her investigation indicated that visualization had an impact to the nature of the task in experiment one; however, other aspects of spatial ability such as spatial memory, perceptual speed, which were originally thought to have some impacts, ended up not being used because further investigation to how they contribute to spatial ability on user study experiment was still needed. Other than spatial ability aspects, miscellaneous aspects such as the user's physical condition and technology literacy might need to be considered as well.

Another route for future work would be to come up with a better experiment design that reduces as much noise on the observed data as possible (e.g. the usages of tooltips are only possible in one way and cursor movement is explicit). The cognitive model established using act-r 6.0 architecture could also be improved considerably by eliminating as many external components as possible. There was a paper by *Gunzelmann* (2006) that discussed a proposal to incorporate spatial information processing module to ACT-R. This is a future work that can still be addressed and done.

An interesting issue would be to model a more complex software interface. The current research would serve as the basis of looking at more complex tasks where spatial ability is important to user performance.

APPENDIX A. GRAPHICAL FIGURES

A1. Exercise 3 labeling scheme (icons developed by Rusch (2008)).

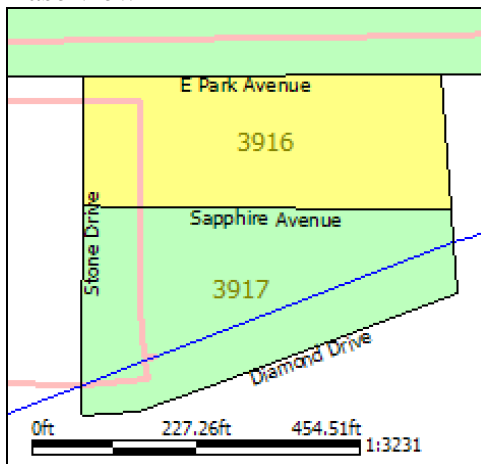
Below is the different labeling schemes for exercise 3 (map interface).

Scheme	Label											
	1	2	3	4	5	6	7	8	9	10	11	12
Meaningful-icon												
Meaningful-text	Pan right	Pan left	Pan up	Pan down	Zoom in	Zoom out	Assign. area	Street names	You are here	Address list	House loc.	Close map
Arbitrary												
No-label												

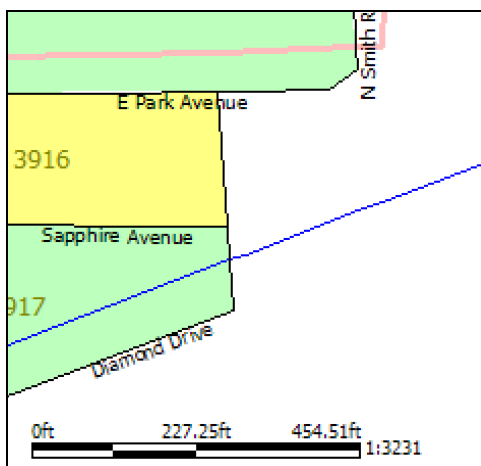
A2. Map views generated by each button function.

Below is the respective map views transitions from base view resulted from clicking the corresponding instruction buttons.

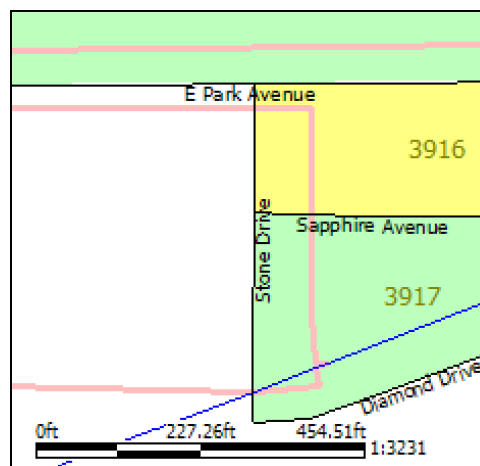
Base view



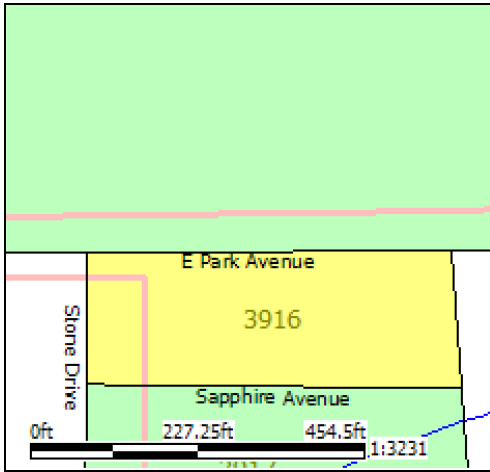
1. Pan right



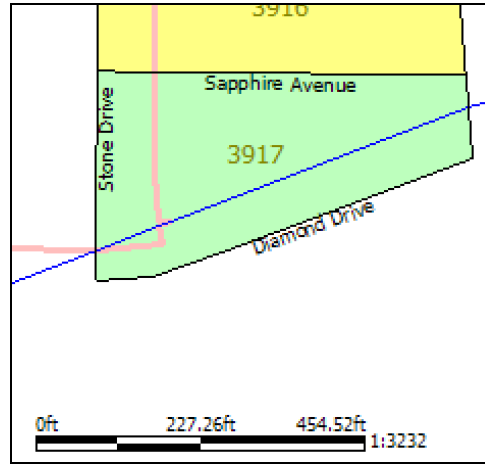
2. Pan left



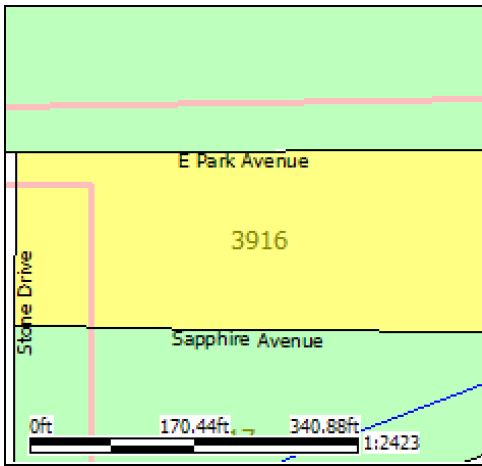
3. Pan up



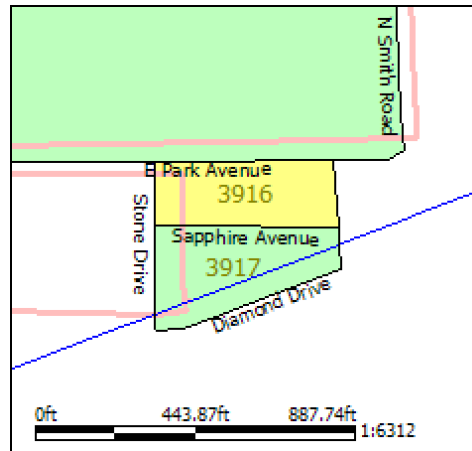
4. Pan down



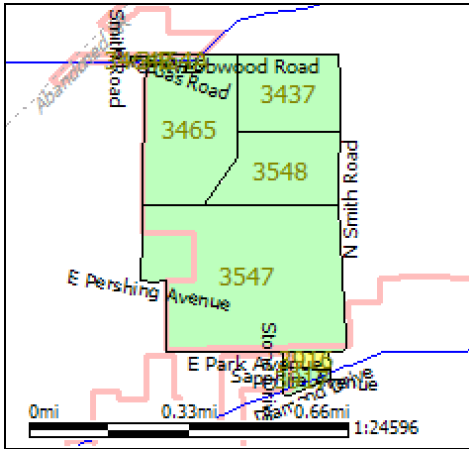
5. Zoom in



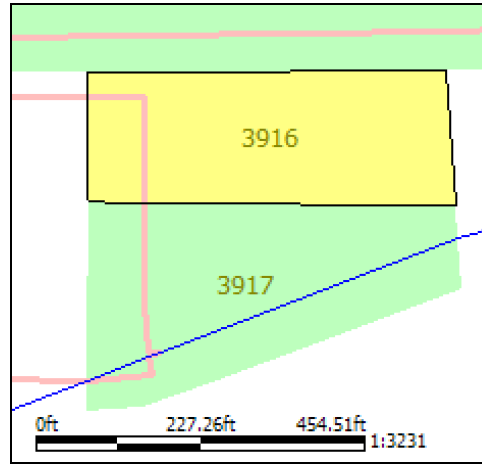
6. Zoom out



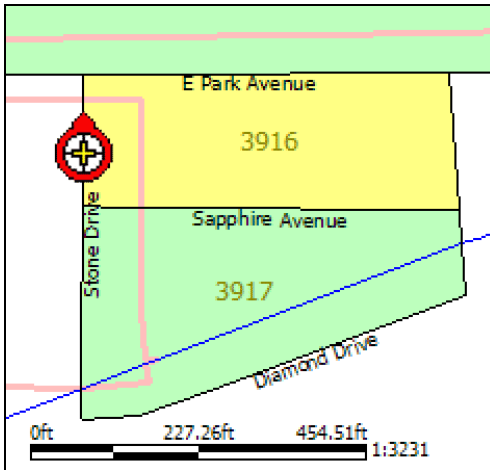
7. Show assignment area



8. Remove street names



9. Show “you are here” symbol

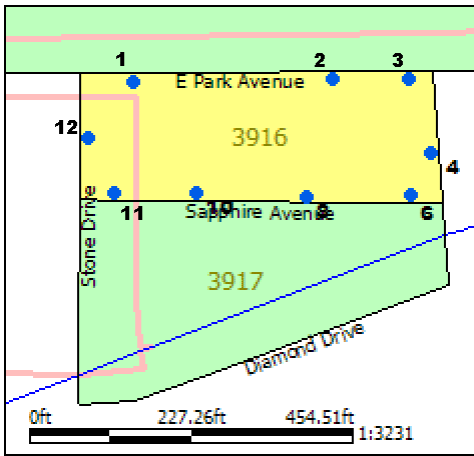


10. Show address list

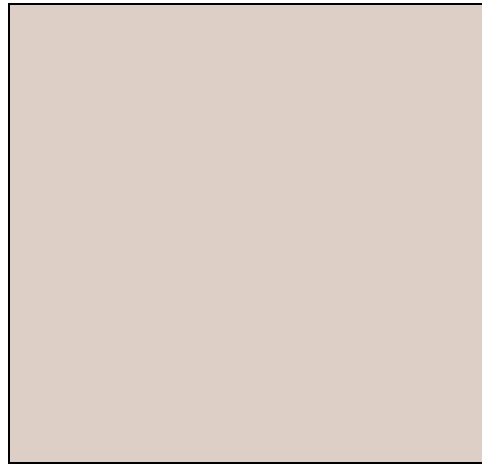
Address List - BLK: 3916 AA: 391041

House	Street Name	Unit
4302	E PARK AVE	

11. Show map spots

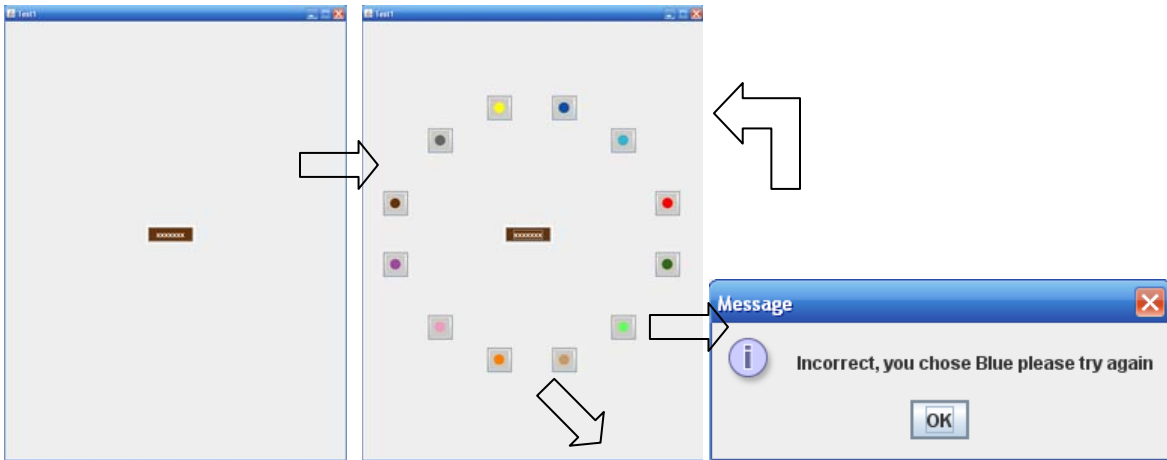


12. Close map



A3. Trial Scenario

Below are some window transitions that illustrate a trial scenario.



Next trial...

A4. Background Questionnaire

Below is interfaces of the background questionnaire software.

Background Information


Answer each of the questions by tapping the circle next to your selected answer.

1. How much experience do you have with the following computer and software tools?

	No experience		Moderately experienced		Very experienced
a. Personal or laptop computer	<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5
b. Tablet PC	<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5
c. Handheld computer	<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5
d. Map software (on your computer or internet)	<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5
e. GPS receiver (global positioning system)	<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5
f. Word processing applications	<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5
g. Drawing/Graphics applications	<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5
h. Web browsing	<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5
i. Email	<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5
j. Games	<input type="radio"/> 1	<input type="radio"/> 2	<input type="radio"/> 3	<input type="radio"/> 4	<input type="radio"/> 5

Next >>

Page 1 of 3

 Background Information 

2. Student Status

Undergraduate Student

Graduate Student

Not a Student

3. Employment status

Employed full time

Employed part time

Retired

Not employed or retired

Other

4. Age

18 - 24 years


25 - 35 years

36 - 50 years

51 - 65 years

66 years or older

Page 2 Of 3

 **Background Information**

5. Gender

Male

Female

6. How did you hear about this study?

Newspaper

Email announcement

On campus flier at Iowa State

Temporary employment services office

Word of mouth

Other

Finished

APPENDIX B. CODE

B1. Control List Generator SAS Code

Below is the code for the SAS program.

*** Import data - this is trial-level data from the experiment for circle and grid layouts only (no map) Includes attempt time, number of attempts, number of moves between buttons, number of clicks 63 subjects, 384 conditions [2 layout x 16 block x 12 trial] per subject = 24192 records ;

```
PROC IMPORT OUT= WORK.events
```

```
DATAFILE= "C:\Documents and Settings\andli241\Desktop\SAS\cogmodel_lists.csv"
```

```
DBMS=CSV REPLACE;
```

```
GETNAMES=YES;
```

```
DATAROW=2;
```

```
RUN;
```

* assume that trial starts with 1 click at the center on the start button (this is not recorded in click_sum) assume that each click after the first click is associated with a minimum of 1 movement thus, the minimum number of movements is the number of clicks in a trial these movements are associated with the target color all other movements can be associated with a button position (that is not clicked)

THIS IMPLIES A RESTRICTION ON RANDOM EVENTS FOR SIMULATION

for every trial, we must have at least one click and the number of movements must be at least as large as the number of clicks also, the attempt_time encompasses all of the movements and click actions, so we probably need a time per movement variable

thus we generate summary statistics for

x_clicks = number of clicks between the start click and the correct target click

x_moves = number of extra movements beyond the required 1 movement for the click

time_per_move = attempt_time / movement_sum ;

data events ; set events ;

time_per_move = attempt_time / movement_sum ;

x_clicks = click_sum - 1 ;

x_moves = movement_sum - click_sum ;

* reset spatial category to 2 levels to increase cell size for means to ensure sd > 0 ;

if vz < 10 then spatial_cat = 1 ;

if vz >= 10 then spatial_cat = 2 ;

* for testing: if subject_id = 25 or subject_id = 34 or subject_id = 69 ;

run ;

/*

proc print data=events ;

var subject_id spatial_cat label_trt layout block trial attempt_time click_sum

movement_sum time_per_move x_clicks x_moves ;

```

title 'testing new calculations for events data set' ;

run ;

*/

*** Generating trial-level summary statistics for each spatial_cat label_trt layout
block trial condition;

*****

should probably change output code to match new variables being used in means
statement time_per_move, x_clicks, x_moves ;

***** ;

* generate the estimated parameters for the Gamma, Poisson, and multinomial
distributions used below

for 3072 conditions (12 trials x 16 blocks x 2 layouts x 4 label_trt x 2 spatial_cat) ;

proc sort data=events ; by spatial_cat label_trt layout block trial ;

proc means data=events noprint ;

by spatial_cat label_trt layout block trial;

var time_per_move x_clicks x_moves position_1 position_2 position_3 position_4
position_5 position_6 position_7 position_8 position_9 position_10 position_11 position_12;

output out=plotmeans

```

```

mean = mtime_per_move mx_clicks mx_moves mposition_1 mposition_2
mposition_3 mposition_4 mposition_5 mposition_6 mposition_7 mposition_8 mposition_9
mposition_10 mposition_11 mposition_12

var = vtime_per_move /*attempt_time_var*/

n = nsubjects ;

run ;

* run this code to make sure each cell has 2 or more subjects and variance > 0 to
estimate above parameters ;

proc freq data=plotmeans ;

table nsubjects ;

title 'summary of sample size for each of 3072 conditions (12 trials x 16 blocks x 2
layouts x 4 label_trt x 2 spatial_cat)' ;

proc univariate data=plotmeans plot ;

var vtime_per_move ;

title 'summary of var to see if any 0 values' ;

run ;

*** Combine observed trial-level data on expt conditions with trial-level summary
statistics

(12 trials x 16 blocks x 2 layouts x 4 label_trt x 2 spatial_cat = 3072 means matching
with

24,192 records from observations) ;

```

```

proc sort data=events ; by spatial_cat label_trt layout block trial ;

proc sort data=plotmeans ; by spatial_cat label_trt layout block trial ;

run ;

Data trial_merged ;

merge events plotmeans ; by spatial_cat label_trt layout block trial ;

run ;

***Generating events (1=click and 2=movement) ;

data sim ;

set trial_merged ;

* for testing: if subject_id = 69 ;

*****

*****

*** FIX THE RANDOM NUMBER SEED APPROACH

*** CURRENTLY INCORRECT -- should probably use call functions too *** ;

x=-1; * random seed ;

*** preliminaries ;

* gamma parameters ;

beta = vtime_per_move / mtime_per_move ;

alpha = mtime_per_move / beta ;

*****

```

```
*****
```

```
*** CHECK THIS NEXT PART
```

```
*** the mposition variables are not probabilities as originally thought
```

```
*** this code makes them probabilities within the condition, but it may not be the
correct concept
```

```
*** ;
```

```
* scaling position means to conform to multinomial ;
```

```
mposition_sum = mposition_1 + mposition_2 + mposition_3 + mposition_4 +
mposition_5 + mposition_6 + mposition_7 + mposition_8 + mposition_9 + mposition_10 +
mposition_11 + mposition_12 ;
```

```
array mposition {12} mposition_1 - mposition_12 ;
```

```
array p_position {12} p_position_1 - p_position_12 ;
```

```
do k = 1 to 12 ;
```

```
    p_position{k} = mposition{k} / mposition_sum ;
```

```
end ;
```

```
*** create first event, which is a click at time 0 in the center ;
```

```
event = 1 ;
```

```
time = 0 ;
```

```
/*pos_rand*/position = 0 ;
```

```
output ;
```



```

*** draw the number of extra clicks (x_event1) beyond the first click

draw the number of extra movements (x_event2) beyond the number required for
(x_event1 + 1) clicks

assume that distribution of number of events of either type is Poisson with mean
equal to the mean from plotmeans ;

if mx_clicks > 0 then x_event1 = ranpoi(x,mx_clicks) ; * number of extra clicks ;
else x_event1 = 0 ;

if mx_moves > 0 then x_event2 = ranpoi(x,mx_moves) ; * number of extra
movements ;

else x_event2 = 0 ;

*** generate extraneous movements (type 2 events) by setting time and button
(position) for each of event2 movements

assume time follows gamma distribution with beta = var/mean and alpha =
mean/beta (mean, var from time summary statistics)

assume button choice (position = 1, 2, 3, ..., 12) follows multinomial
distribution with probs from summary stats ;

if x_event2 > 0 then do i = 1 to x_event2 by 1 ;

event = 2 ;

time = beta*rangam(x,alpha) ;

```

```

    position = rantbl(x,
p_position_1,p_position_2,p_position_3,p_position_4,p_position_5,p_position_6,p_position
_7,p_position_8,p_position_9,p_position_10,
    p_position_11,p_position_12) ;

    output ;

    end ;

*** generate clicks and associated movement

excludes initial start click, extraneous movements, and final correct click

assume time follows normal distribution with mean and sd from summary statistics,
set minimum time to .1 second

assume button choice (position = 1, 2, 3, ..., 12) follows multinomial distribution with
probs from summary stats ;

if x_event2 > 0 then do i = 1 to x_event1 by 1 ;

    event = 1 ;

    time = beta*rangam(x,alpha) ;

    position = rantbl(x, p_position_1,p_position_2,p_position_3,p_position_4,
    p_position_5,p_position_6,p_position_7,p_position_8,p_position_9,p_position_10,
p_position_11,p_position_12) ;

    output;

    end;

*** generate last click ;

```

```
event = 1 ;

time = beta*rangam(x,alpha) ;

position = target_color;

output;

run;

***Outputing the 6-tuple list file for each subject_id;

%macro output(subn,dsn) ;

data tuple; *'C:\Documents and Settings\andli241\Desktop\SAS\subject_69.txt';

set Sim;

if subject_id = &subn ;

file &dsn ;

put "(" layout " " block " " trial " " event " " position " " time " ");

run;

%mend ;

*cm sp 1;

%output(127,'C:\Documents and

Settings\andli241\Desktop\SAS\generated_list\cm_1\subject_127.txt');

%output(99,'C:\Documents and

Settings\andli241\Desktop\SAS\generated_list\cm_1\subject_99.txt');

%output(100,'C:\Documents and

Settings\andli241\Desktop\SAS\generated_list\cm_1\subject_100.txt');
```

```
%output(118,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_1\subject_118.txt');

%output(120,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_1\subject_120.txt');

%output(132,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_1\subject_132.txt');

*cm sp 2;

%output(3,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_3.txt');

%output(8,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_8.txt');

%output(18,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_18.txt');

%output(19,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_19.txt');

%output(36,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_36.txt');

%output(39,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_39.txt');

%output(50,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_50.txt');
```

```
%output(52,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_52.txt');

%output(67,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_67.txt');

%output(72,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_72.txt');

%output(82,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_82.txt');

%output(88,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_88.txt');

%output(121,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_121.txt');

%output(147,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\cm_2\subject_147.txt');

*text sp 1;

%output(98,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_1\subject_98.txt');

%output(37,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_1\subject_37.txt');

%output(103,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_1\subject_103.txt');
```

```
%output(119,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_1\subject_119.txt');

*text sp 2;

%output(4,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_4.txt');

%output(5,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_5.txt');

%output(20,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_20.txt');

%output(21,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_21.txt');

%output(38,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_38.txt');

%output(49,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_49.txt');

%output(51,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_51.txt');

%output(65,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_65.txt');

%output(71,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_71.txt');
```

```
%output(81,'C:\Documents and  
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_81.txt');
```

```
%output(83,'C:\Documents and  
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_83.txt');
```

```
%output(113,'C:\Documents and  
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_113.txt');
```

```
%output(126,'C:\Documents and  
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_126.txt');
```

```
%output(129,'C:\Documents and  
Settings\andli241\Desktop\SAS\generated_list\text_2\subject_129.txt');
```

```
*arbitrary sp 1;
```

```
%output(40,'C:\Documents and  
Settings\andli241\Desktop\SAS\generated_list\arbitrary_1\subject_40.txt');
```

```
%output(70,'C:\Documents and  
Settings\andli241\Desktop\SAS\generated_list\arbitrary_1\subject_70.txt');
```

```
%output(102,'C:\Documents and  
Settings\andli241\Desktop\SAS\generated_list\arbitrary_1\subject_102.txt');
```

```
*arbitrary sp 2;
```

```
%output(7,'C:\Documents and  
Settings\andli241\Desktop\SAS\generated_list\arbitrary_2\subject_7.txt');
```

```
%output(16,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\arbitrary_2\subject_16.txt');

%output(23,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\arbitrary_2\subject_23.txt');

%output(24,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\arbitrary_2\subject_24.txt');

%output(42,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\arbitrary_2\subject_42.txt');

%output(56,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\arbitrary_2\subject_56.txt');

%output(62,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\arbitrary_2\subject_62.txt');

%output(85,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\arbitrary_2\subject_85.txt');

%output(104,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\arbitrary_2\subject_104.txt');

*no label sp 1;

%output(35,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_1\subject_35.txt');

%output(68,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_1\subject_68.txt');
```



```
*no label sp 2;

%output(1,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_2\subject_1.txt');

%output(6,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_2\subject_6.txt');

%output(17,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_2\subject_17.txt');

%output(25,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_2\subject_25.txt');

%output(34,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_2\subject_34.txt');

%output(53,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_2\subject_53.txt');

%output(54,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_2\subject_54.txt');

%output(69,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_2\subject_69.txt');

%output(84,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_2\subject_84.txt');

%output(97,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_2\subject_97.txt');

%output(145,'C:\Documents and
Settings\andli241\Desktop\SAS\generated_list\nl_2\subject_145.txt');
```

B2. Act-r 6.0 PRODUCTIONS CODE (COMPLETE)

Below is ACT-R 6.0 simulation interface setup and productions code.

```
;;; LISP Input/Output EXPERIMENT code
```

```
;;;cmms machine
```

```
(defparameter *in* (open "C:/Documents and Settings/mlrusch/Desktop/actr6/input.txt" :if-
does-not-exist nil))
```

```
(defparameter *out* (open "C:/Documents and
Settings/mlrusch/Desktop/actr6/correlation.txt" :direction :output :if-exists :supersede))
```

```
;;;my machine
```

```
;(defparameter *s* (open "C:/Users/Andre/Desktop/actr6/input.txt" :if-does-not-exist nil))
```

```
;(defparameter *out* (open "C:/Users/Andre/Desktop/actr6/list.txt" :direction :output :if-
exists :supersede))
```

```
(defparameter *list* ())
```

```
(defparameter *block-time* ()) ;This is the list for the block time extracted from the
simulation
```

```
(defparameter *trial-time* ()) ;This is the list for the trial time extracted from the simulation
```

```
(defparameter *exercise-time* ()) ;This is the list for the exercise time extracted from the
simulation
```

```
(defparameter *control-time* '(10.59934833
```

```
7.7061346
```

```
6.259121
```

```
5.923454767
```

```
4.741019767
```

```
4.6985858
```

```
3.628430033
```

```
3.168462
```

```
3.1470028
```

```
3.187801767
```

```
2.617415733
```

```
2.6781359
```

```
2.319055533
```

```
2.355185867
```

```
1.975088567
```

```
2.0244694
```

```
11.12061
```

```
8.411149
```

```
5.5960025
```

```
3.9172621
```

```
3.524500067
```

```
3.529281733
```

2.466528633
 2.685951233
 1.977367
 2.1284567
 2.354814733
 1.826991567
 2.513465833
 1.976898433
 2.092001833
 1.856396067

));This is the list for the validation control time read from a file

;tsp2

;control data for sp1 cm now;;;;

(defparameter *block-current* 1)
 (defparameter *trial-current* 1)
 (defparameter *exercise-current* 1)

(defparameter *temp* ())
 (defparameter *result* ())
 (defparameter *thinkingtime* ())

(defparameter *circlePos* '((249 360) (330 130) (433 187) (510 297) (510 404) (433 514)
 (330 571)

(217 571) (114 514) (36 404) (36 297) (114 187) (217 130)))

(defparameter *gridPos* '((249 360) (90 190) (160 190) (230 190) (300 190) (370 190) (440
 190)

(90 270) (160 270) (230 270) (300 270) (370 270) (440 270)))

(defparameter *circleTooltips* '((249 340) (330 110) (433 167) (510 277) (510 384) (433
 494) (330 551)

(217 551) (114 494) (36 384) (36 277) (114 167) (217 110)))

(defparameter *gridTooltips* '((249 340) (90 170) (160 170) (230 170) (300 170) (370 170)
 (440 170)

(90 250) (160 250) (230 250) (300 250) (370 250) (440 250)))

(if *in*

(progn

(loop for tuple = (read *in* nil)

while tuple do (setq *list* (cons tuple *list*)))

)

(close *in*)

(setq *list* (reverse *list*))

;(write *list* :stream *out*)

;(close *out*)

```

)
(print "file does not exist or invalid path")
)

(defvar *response* nil)
(defvar layt 1) ;layout variable
(defvar *tooltip* nil) ;tooltip variable
(defvar *blockactionNumber* 0)
(defvar *trialactionNumber* 0)
(defvar *exerciseactionNumber* 0)
(setf *actionNumber* 0)
(defvar *mptrialTimeBefore* 0)
(setf *mptrialTimeBefore* 0)
(defvar *mpblockTime* 0)
(setf *mpblockTime* 0)
(defvar *mpexerciseTime* 0)
(setf *mpexerciseTime* 0)

(defmethod rpm-window-click-event-handler ((win rpm-window) pos)
  (setf *response* nil)
  (clear-exp-window)
  (when *actr-enabled-p*
    (proc-display)))

;;;Event-handlers function;;;;;
;;;;;
(defun button0-click (button)
  (progn
    (setf *response* 'clicked0)
    (reconfigure-interface layt)
  )
)
(defun button1-click (button)
  (progn
    (setf *response* 'clicked1)
    (if (= (fifth *result*) 0)
      (reconfigure-interface 0)
    )
  )
)
(defun button2-click (button)
  (progn
    (setf *response* 'clicked2)
    (if (= (fifth *result*) 0)
      (reconfigure-interface 0)
    )
  )
)

```

```
)  
)  
)  
(defun button3-click (button)  
  (progn  
    (setf *response* 'clicked3)  
    (if (= (fifth *result*) 0)  
        (reconfigure-interface 0)  
    )  
  )  
)  
)  
(defun button4-click (button)  
  (progn  
    (setf *response* 'clicked4)  
    (if (= (fifth *result*) 0)  
        (reconfigure-interface 0)  
    )  
  )  
)  
)  
(defun button5-click (button)  
  (progn  
    (setf *response* 'clicked5)  
    (if (= (fifth *result*) 0)  
        (reconfigure-interface 0)  
    )  
  )  
)  
)  
(defun button6-click (button)  
  (progn  
    (setf *response* 'clicked6)  
    (if (= (fifth *result*) 0)  
        (reconfigure-interface 0)  
    )  
  )  
)  
)  
(defun button7-click (button)  
  (progn  
    (setf *response* 'clicked7)  
    (if (= (fifth *result*) 0)  
        (reconfigure-interface 0)  
    )  
  )  
)  
)  
(defun button8-click (button)  
  (progn
```

```

    (setf *response* 'clicked8)
    (if (= (fifth *result*) 0)
        (reconfigure-interface 0)
    )
)
)
)
(defun button9-click (button)
  (progn
    (setf *response* 'clicked9)
    (if (= (fifth *result*) 0)
        (reconfigure-interface 0)
    )
  )
)
)
(defun button10-click (button)
  (progn
    (setf *response* 'clicked10)
    (if (= (fifth *result*) 0)
        (reconfigure-interface 0)
    )
  )
)
)
)
(defun button11-click (button)
  (progn
    (setf *response* 'clicked11)
    (if (= (fifth *result*) 0)
        (reconfigure-interface 0)
    )
  )
)
)
)
(defun button12-click (button)
  (progn
    (setf *response* 'clicked12)
    (if (= (fifth *result*) 0)
        (reconfigure-interface 0)
    )
  )
)
)
)
;:;interface and experiment setup function;:;
;:;
(defun d ()

  (reset)

```

```

(let* ((window (open-exp-window "Experiment"
                               :visible t
                               :width 600
                               :height 800
                               :x 300
                               :y 100)))

      (add-button-to-exp-window :action #'button0-click :height 43 :width 43 :x (first (nth 0
*circlePos*)) :y (second (nth 0 *circlePos*)))

      (setf *response* nil)

      (if *actr-enabled-p*
          (progn
            (install-device window)
            (start-hand-at-mouse)
            (setf *temp* (consume-list))
            (define-chunks-fct (list (list 'a1 'isa 'array 'elements *temp*)))
            (set-buffer-chunk 'imaginal 'a1)
            (proc-display)
            (run 10800 :real-time nil)
            )

          (while (null *response*)
                (allow-event-manager window)))

      *response*))
;,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
;:;supporting functions;:;,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
;:;,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
;layout 0: center button layout
;layout 1: circle layout
;layout 2: grid layout
(defun reconfigure-interface (layout)
  (cond ((= layout 0)
         (progn
           (clear-exp-window)
           (add-button-to-exp-window :action #'button0-click :height 43 :width 43 :x (first (nth 0
*circlePos*)) :y (second (nth 0 *circlePos*)))
           (proc-display)
           )
        )
        )
)

```

```

)
((= layout 1)

(progn
  (clear-exp-window)
  (add-button-to-exp-window :action #'button0-click :height 43 :width 43 :x (first (nth 0
*circlePos*)) :y (second (nth 0 *circlePos*)))
  (add-button-to-exp-window :action #'button1-click :height 43 :width 43 :x (first (nth 1
*circlePos*)) :y (second (nth 1 *circlePos*)))
  (add-button-to-exp-window :height 43 :action #'button2-click :width 43 :x (first (nth 2
*circlePos*)) :y (second (nth 2 *circlePos*)))
  (add-button-to-exp-window :height 43 :action #'button3-click :width 43 :x (first (nth 3
*circlePos*)) :y (second (nth 3 *circlePos*)))
  (add-button-to-exp-window :height 43 :action #'button4-click :width 43 :x (first (nth 4
*circlePos*)) :y (second (nth 4 *circlePos*)))
  (add-button-to-exp-window :height 43 :action #'button5-click :width 43 :x (first (nth 5
*circlePos*)) :y (second (nth 5 *circlePos*)))
  (add-button-to-exp-window :height 43 :action #'button6-click :width 43 :x (first (nth 6
*circlePos*)) :y (second (nth 6 *circlePos*)))
  (add-button-to-exp-window :height 43 :action #'button7-click :width 43 :x (first (nth 7
*circlePos*)) :y (second (nth 7 *circlePos*)))
  (add-button-to-exp-window :height 43 :action #'button8-click :width 43 :x (first (nth 8
*circlePos*)) :y (second (nth 8 *circlePos*)))
  (add-button-to-exp-window :height 43 :action #'button9-click :width 43 :x (first (nth 9
*circlePos*)) :y (second (nth 9 *circlePos*)))
  (add-button-to-exp-window :height 43 :action #'button10-click :width 43 :x (first (nth
10 *circlePos*)) :y (second (nth 10 *circlePos*)))
  (add-button-to-exp-window :height 43 :action #'button11-click :width 43 :x (first (nth
11 *circlePos*)) :y (second (nth 11 *circlePos*)))
  (add-button-to-exp-window :height 43 :action #'button12-click :width 43 :x (first (nth
12 *circlePos*)) :y (second (nth 12 *circlePos*)))
  (proc-display)
)
)
((= layout 2)
(progn
  (clear-exp-window)
  (add-button-to-exp-window :action #'button0-click :height 43 :width 43 :x (first (nth 0
*gridPos*)) :y (second (nth 0 *gridPos*)))
  (add-button-to-exp-window :action #'button1-click :height 43 :width 43 :x (first (nth 1
*gridPos*)) :y (second (nth 1 *gridPos*)))
  (add-button-to-exp-window :height 43 :action #'button2-click :width 43 :x (first (nth 2
*gridPos*)) :y (second (nth 2 *gridPos*)))
  (add-button-to-exp-window :height 43 :action #'button3-click :width 43 :x (first (nth 3
*gridPos*)) :y (second (nth 3 *gridPos*)))

```



```

    (add-button-to-exp-window :height 43 :action #'button4-click :width 43 :x (first (nth 4
*gridPos*)) :y (second (nth 4 *gridPos*)))
    (add-button-to-exp-window :height 43 :action #'button5-click :width 43 :x (first (nth 5
*gridPos*)) :y (second (nth 5 *gridPos*)))
    (add-button-to-exp-window :height 43 :action #'button6-click :width 43 :x (first (nth 6
*gridPos*)) :y (second (nth 6 *gridPos*)))
    (add-button-to-exp-window :height 43 :action #'button7-click :width 43 :x (first (nth 7
*gridPos*)) :y (second (nth 7 *gridPos*)))
    (add-button-to-exp-window :height 43 :action #'button8-click :width 43 :x (first (nth 8
*gridPos*)) :y (second (nth 8 *gridPos*)))
    (add-button-to-exp-window :height 43 :action #'button9-click :width 43 :x (first (nth 9
*gridPos*)) :y (second (nth 9 *gridPos*)))
    (add-button-to-exp-window :height 43 :action #'button10-click :width 43 :x (first (nth
10 *gridPos*)) :y (second (nth 10 *gridPos*)))
    (add-button-to-exp-window :height 43 :action #'button11-click :width 43 :x (first (nth
11 *gridPos*)) :y (second (nth 11 *gridPos*)))
    (add-button-to-exp-window :height 43 :action #'button12-click :width 43 :x (first (nth
12 *gridPos*)) :y (second (nth 12 *gridPos*)))
    (proc-display)
  )
)
)
)

(defun show-tooltip (tuple)
  (cond ((= layt 2)
    (setf *tooltip* (add-text-to-exp-window :text "x" :x (first (nth (fourth tuple)
*gridTooltips*)) :y (second (nth (fourth tuple) *gridTooltips*)))
    ))
    )
    ((= layt 1)
    (setf *tooltip* (add-text-to-exp-window :text "x" :x (first (nth (fourth tuple)
*circleTooltips*)) :y (second (nth (fourth tuple) *circleTooltips*)))
    ))
    )
  )
  (proc-display)
)

(defun remove-tooltip ()
  (remove-items-from-exp-window *tooltip*)
  (proc-display)
)

(defun consume-list ()

```

```

(if *list*
  (progn
    (setf *result* (car *list*))
    (setf layt (first *result*))
    (setf *list* (cdr *list*))
    *result*)
  nil
)
)

(defun return-response ()
  (if (null *result*) nil
      *result*)
)

(defun get-event (event)
  (setf *thinkingtime* (sixth *result*))
  (if (= (fourth event) 2)
    (progn
      (setf *blockactionNumber* (+ *blockactionNumber* 1))
      (setf *trialactionNumber* (+ *trialactionNumber* 1))
      (setf *exerciseactionNumber* (+ *exerciseactionNumber* 1))
      (spp-fct (list 'initiate-attend :at (- *thinkingtime* 0.15)))
      (spp-fct (list 'button-click :at 0))
      'start
    )
    (progn
      (if (not(= (fifth *result*) 0))
        (progn
          (setf *blockactionNumber* (+ *blockactionNumber* 1))
          (setf *trialactionNumber* (+ *trialactionNumber* 1))
          (setf *exerciseactionNumber* (+ *exerciseactionNumber* 1))
        )
        (progn
          (setf *blockactionNumber* *blockactionNumber*)
          (setf *trialactionNumber* *trialactionNumber*)
          (setf *exerciseactionNumber* *exerciseactionNumber*)
        )
      )
      (spp-fct (list 'initiate-attend :at 0))
      (if (= (sixth *result*) 0) (spp-fct (list 'button-click :at *thinkingtime*))
          (spp-fct (list 'button-click :at (- *thinkingtime* 0.15)))
      )
      'clicking
    )
  )
)

```

```

)
)

(defun bind-x (lis tooltip)
  (if tooltip
    (if (= layt 1)
      (first (nth (nth 4 lis) *circleTooltips*)) ;harvesting the x position
      (first (nth (nth 4 lis) *gridTooltips*))
    )
    (if (= layt 1)
      (first (nth (nth 4 lis) *circlePos*)) ;harvesting the x position
      (first (nth (nth 4 lis) *gridPos*))
    )
  )
)
)
)
(defun bind-y (lis tooltip)
  (if tooltip
    (if (= layt 1)
      (second (nth (nth 4 lis) *circleTooltips*)) ;harvesting the y position
      (second (nth (nth 4 lis) *gridTooltips*))
    )
    (if (= layt 1)
      (second (nth (nth 4 lis) *circlePos*)) ;harvesting the y position
      (second (nth (nth 4 lis) *gridPos*))
    )
  )
)
)
)

;;function for building the block time
(defun append-block-time (lis time)
  (if (not (= (second lis) *block-current*))
    (progn
      ;(setq *block-time* (cons (- (- (- time *mpblockTimeBefore*) (* 12 1.470)) (* 0.085 (-
*blockactionNumber* 1))) *block-time*))
      (setq *block-time* (cons *mpblocktime* *block-time*))
      ;(setf *mpblockTimeBefore* time)
      (setf *mpblockTime* 0)
      (setf *block-current* (second lis))
      ;(setf *blockactionNumber* 0)
    )
  )
  0
)
)
)

```

```
;;function for building the trial time
(defun append-trial-time (lis time)
  (if (not (= (third lis) *trial-current*))
      (progn
        (setq *trial-time* (cons (- (- (- time *mprialTimeBefore*) 1.470) (* 0.085 (-
*trialactionNumber* 1)) ) *trial-time*))
        (setf *mpblocktime* (+ *mpblocktime* (- (- (- time *mprialTimeBefore*) 1.470) (*
0.085 (- *trialactionNumber* 1) )))
        (setf *mpexercisetime* (+ *mpexercisetime* (- (- (- time *mprialTimeBefore*) 1.470)
(* 0.085 (- *trialactionNumber* 1) ))))

        (setf *mprialTimeBefore* time)
        (setf *trial-current* (third lis))
        (setf *trialactionNumber* 0)
      )
    0
  )
)
```

```
;;function for building the trial time
(defun append-exercise-time (lis time)
  (if (not (= (first lis) *exercise-current*))
      (progn
        ;(setq *exercise-time* (cons (- (- (- time *mpexerciseTimeBefore*) (* 192 1.470)) (*
0.085 (- *exerciseactionNumber* 1)) ) *exercise-time*))
        (setq *exercise-time* (cons *mpexercisetime* *exercise-time*))
        ;(setf *mpexerciseTimeBefore* time)
        (setf *mpexerciseTime* 0)
        (setf *exercise-current* (first lis))

        ;(setf *exerciseactionNumber* 0)
      )
    0
  )
)
```

```
(defun getActionNumber ()
  *actionNumber*
)
```

```
;(defun report-Actual-Data()
; (write-line "Actual Data:" :stream *out*)
; (let ((n 0))
; (loop
```

```
; (when (> n 31) (return))
; (write-line (nth n *control-time*) :stream *out*)
; (incf n)))
; )
```

```
;(defun report-Simulation-Data()
; (setq data (reverse *block-time*))
; (write-line "Simulation Data:" :stream *out*)
; (let ((n 0))
; (loop
; (when (> n 31) (return))
; (write-line (nth n *block-time*) :stream *out*)
; (incf n)))
; )
```

```
.....
```

```
(clear-all)
```

```
::;Model codes;.....
```

```
(define-model model
  (sgp :v t :needs-mouse t :incremental-mouse-moves t :show-focus t :trace-detail high :do-
not-harvest imaginal)
  :(sgp)
```

```
(chunk-type the-goal state output)
(chunk-type array elements)
```

```
(add-dm
  (g isa the-goal state start)
  (start isa chunk)
  (attending isa chunk)
  (clicking isa chunk)
  (selecting isa chunk)
  (done isa chunk)
  (tooltip isa chunk)
  (see-tooltip isa chunk)
  (attend-tooltip isa chunk)
  ;more dm can be added here
  )
(goal-focus g)
```

```
(p initiate-attend
```

```
  =goal>
  isa the-goal
  state start
```

```
  =imaginal>
  isa array
  - elements nil
  elements =list
```

```
  ;!eval! (append-block-time =list (mp-time))
  ;!safe-bind! =AN (getActionNumber)
```

```
  ?manual>
  state free
```

```
  ?visual>
  state free
```

```
  !safe-bind! =x (bind-x =list nil) ;harvesting the x position of the button
  !safe-bind! =y (bind-y =list nil) ;harvesting the y position of the button
```

```
  ==>
```

```
  !eval! (append-trial-time =list (mp-time))
  !eval! (append-block-time =list (mp-time))
  !eval! (append-exercise-time =list (mp-time))
  =goal>
  state attending
```

```
  =imaginal>
```

```
  +visual-location>
  isa visual-location
  kind oval
  screen-x (+ 22 =x)
  screen-y (+ 22 =y)
```

```
  ;!output! (=list)
  ;!output! (=AN)
```

```
)
```

```
(p button-attend
```

```
=goal>
isa the-goal
state attending
```

```
=imaginal>
isa array
```

```
=visual-location>
isa visual-location
```

```
?manual>
state free
```

```
?visual>
state free
==>
```

```
=imaginal>
```

```
=goal>
state selecting
```

```
+visual>
isa move-attention
screen-pos =visual-location
```

```
+manual>
isa move-cursor
loc =visual-location
```

```
;!eval! (remove-tooltip)
)
```

```
;(p display-tooltip
; =goal>
; isa the-goal
; state tooltip
;
; ?visual>
; state free
;
; ?manual>
; state free
;
; =imaginal>
```

```

; isa array
; elements =x
;
; !eval! (show-tooltip =x)
;
; ==>
;
; =goal>
; state selecting
;
; =imaginal>
;)

```

```

(p selecting-event
  =goal>
  isa the-goal
  state selecting

```

```

  =imaginal>
  isa array
  elements =x

```

```

  !safe-bind! =event (get-event =x)

```

```

  ==>

```

```

  =goal>
  state =event

```

```

  !safe-bind! =y (consume-list)
  !output! (=y)

```

```

  =imaginal>
  elements =y

```

```

; !eval! (spp-fct (list 'initiate-attend :at *thinkingtime*))
; !eval! (spp-fct (list 'button-attend :at *thinkingtime*))
; !eval! (spp-fct (list 'button-click :at *thinkingtime*))
)

```

```

;(p prepare-to-see-tooltip
;  =goal>
;  isa the-goal
;  state see-tooltip
;

```



```

; =imaginal>
; isa array
; elements =list
;
; !safe-bind! =x (bind-x =list t) ;harvesting the x position of the tooltip
; !safe-bind! =y (bind-y =list t) ;harvesting the y position of the tooltip
;
; ==>
;
; =goal>
; state attend-tooltip
;
; +visual-location>
; isa visual-location
; kind text
; screen-x (+ 4 =x)
; screen-y (+ 9 =y)
;
; =imaginal>
;)

```

```

;(p attend-tooltip
; =goal>
; isa the-goal
; state attend-tooltip
;
; =visual-location>
; isa visual-location
; screen-x =x
; screen-y =y
;
; ?visual>
; state free
;
; =imaginal>
; isa array
;
; ==>
;
; =goal>
; state start
;
; +visual>
; isa move-attention
; screen-pos =visual-location

```

```

;
; !safe-bind! =y (consume-list)
; !output! (=y)
;
; =imaginal>
; elements =y
;)

```

```
(p button-click
```

```

=goal>
isa the-goal
state clicking

```

```

?manual>
state free

```

```

=imaginal>
isa array

```

```

==>
=goal>
state start

```

```
;!safe-bind! =y (consume-list)
```

```

=imaginal>
;elements =y
;!output! (=y)

```

```

+manual>
isa click-mouse

```

```

;!eval! (spp-fct (list 'initiate-attend :at (sixth =y)))
)

```

```
(p finish
```

```

=goal>
isa the-goal
state start

```

```

=imaginal>
isa array
elements nil

```

```
;!bind! =out (return-response)
```

```
?manual>
state free
```

```
==>
```

```
!eval!(append-block-time '(0 0 0 0 0) (mp-time)) ;To get the last block time
!eval!(append-trial-time '(0 0 0 0 0) (mp-time)) ;To get the last trial time
!eval!(append-exercise-time '(0 0 0 0 0) (mp-time)) ;To get the last exercise time
```

```
=goal>
```

```
state done
```

```
;!eval! (report-Actual-Data)
```

```
;!eval! (report-Simulation-Data)
```

```
;!eval!(write-line "Simulation time:" :stream *out*)
```

```
!eval!(write *control-time* :stream *out*)
```

```
!eval!(write (reverse *trial-time*) :stream *out*)
```

```
!eval!(write (reverse *block-time*) :stream *out*)
```

```
!eval!(write (reverse *exercise-time*) :stream *out*)
```

```
;!eval!(write-line "Actual time:" :stream *out*)
```

```
;!eval!(write *control-time* :stream *out*)
```

```
;!eval! (correlation *control-time* (reverse *block-time*) :output *out*);"C:/Documents
and Settings/mlrusch/Desktop/actr6/correlation.txt")
```

```
!eval! (close *out*)
```

```
;!output! (=out)
```

```
)
```

```
(spp-fct (list 'button-attend :at 0.0))
```

```
(spp-fct (list 'button-click :at 0.0))
```

```
(spp-fct (list 'selecting-event :at 0.0))
```

```
(spp-fct (list 'finish :at 0.0))
```

```
(setf *actr-enabled-p* t)
```

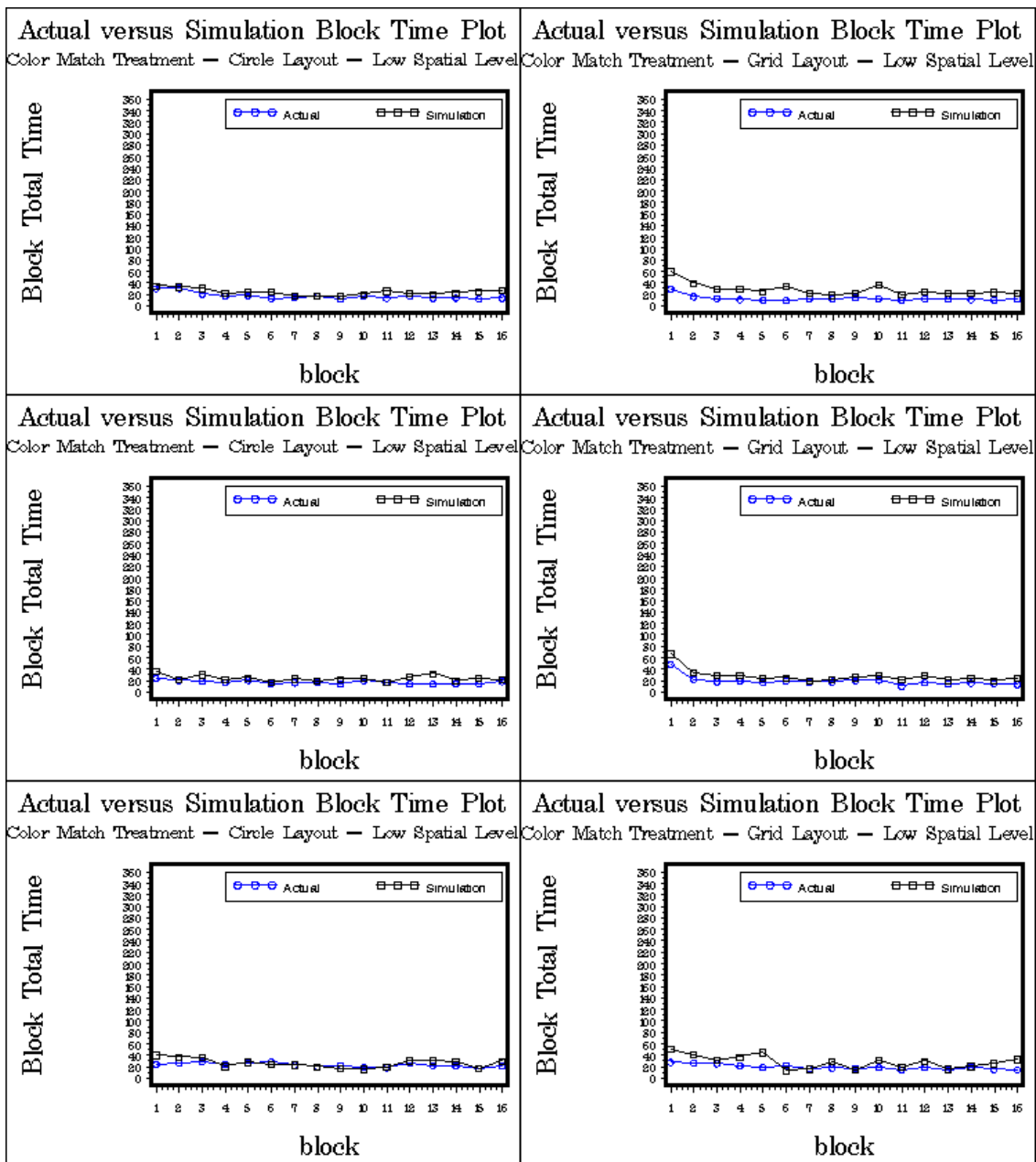
```
)
```

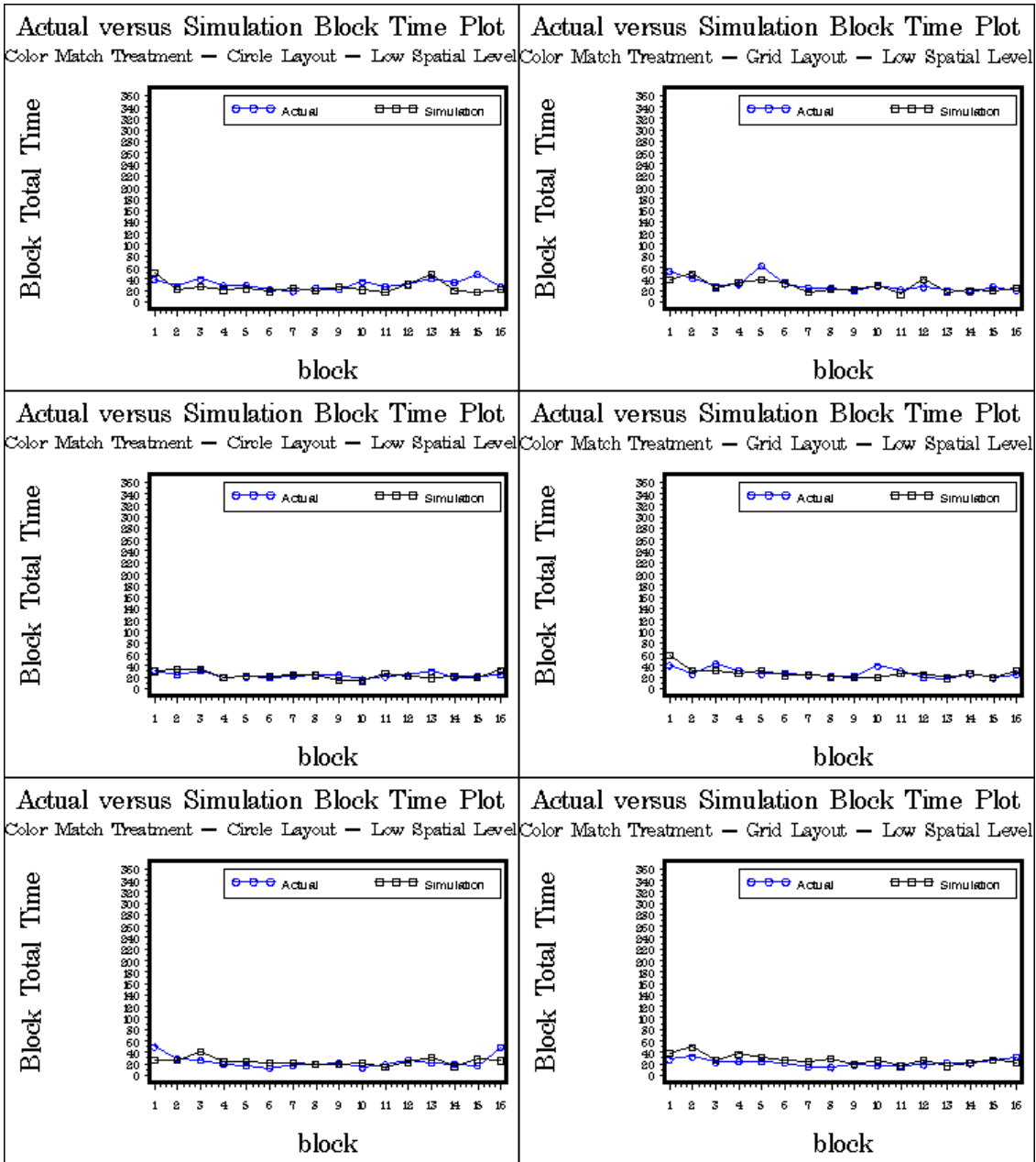
```
.....
```

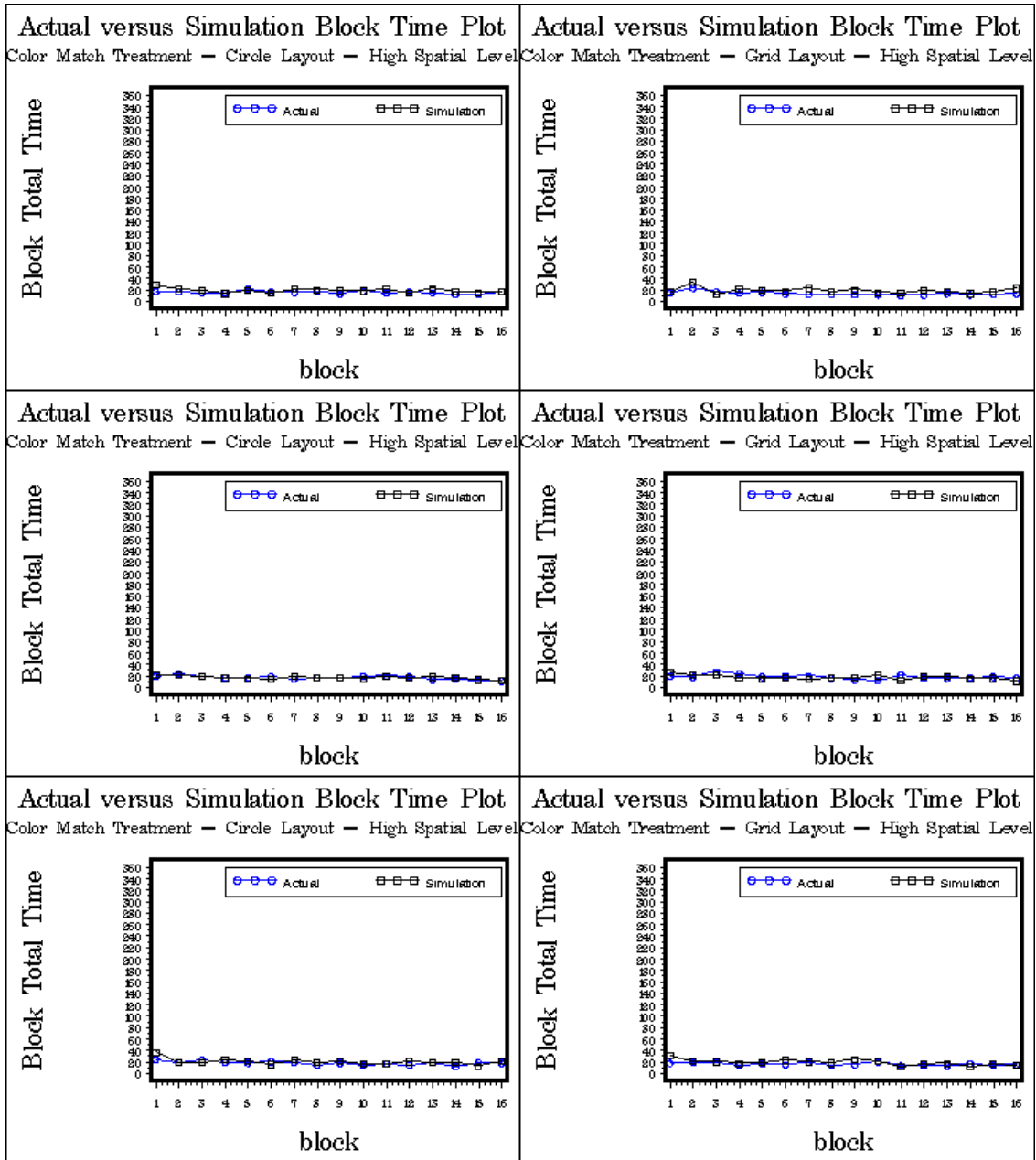
APPENDIX C. PLOT APPENDIX

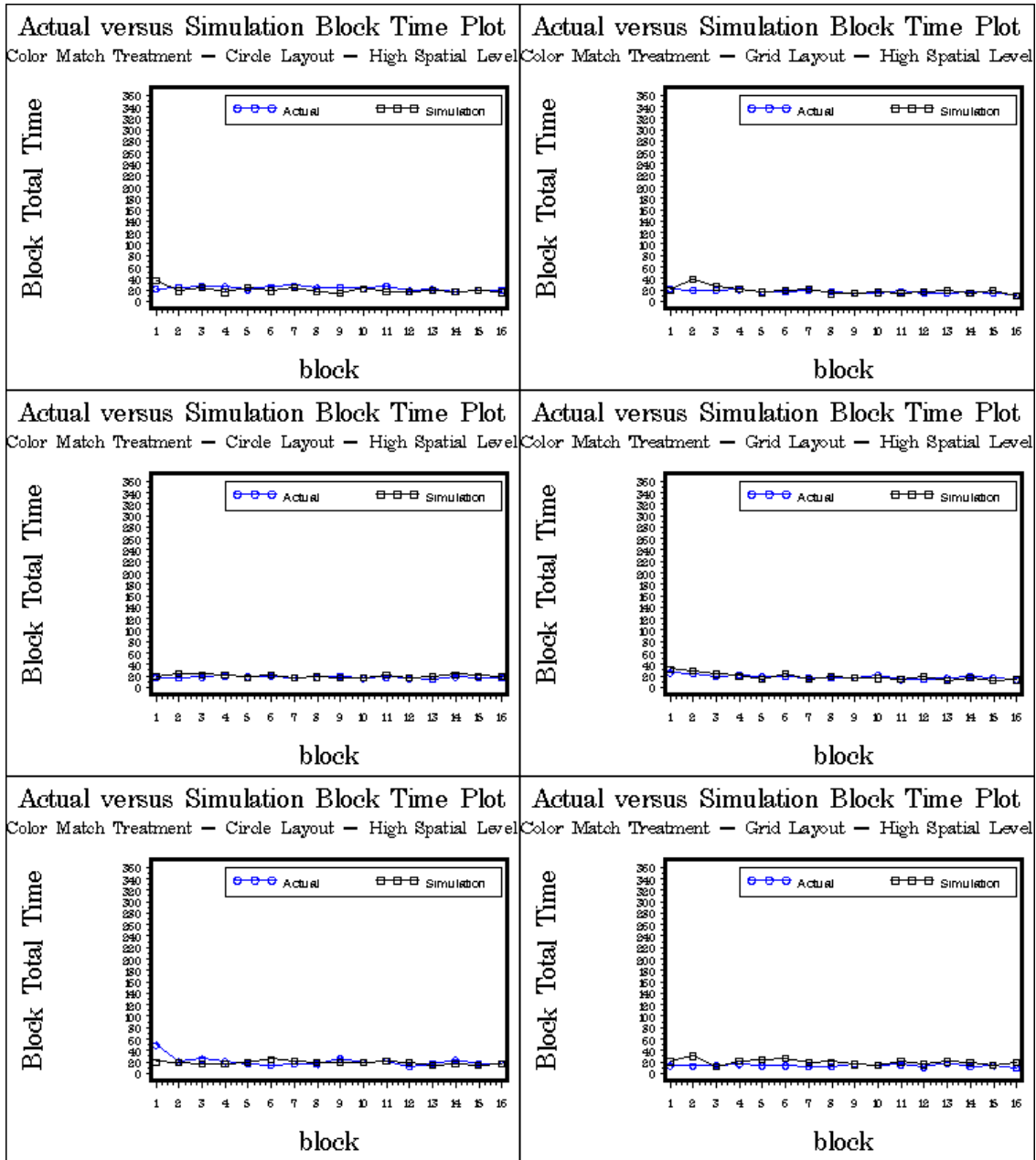
C1. Actual versus simulation total time performance plots

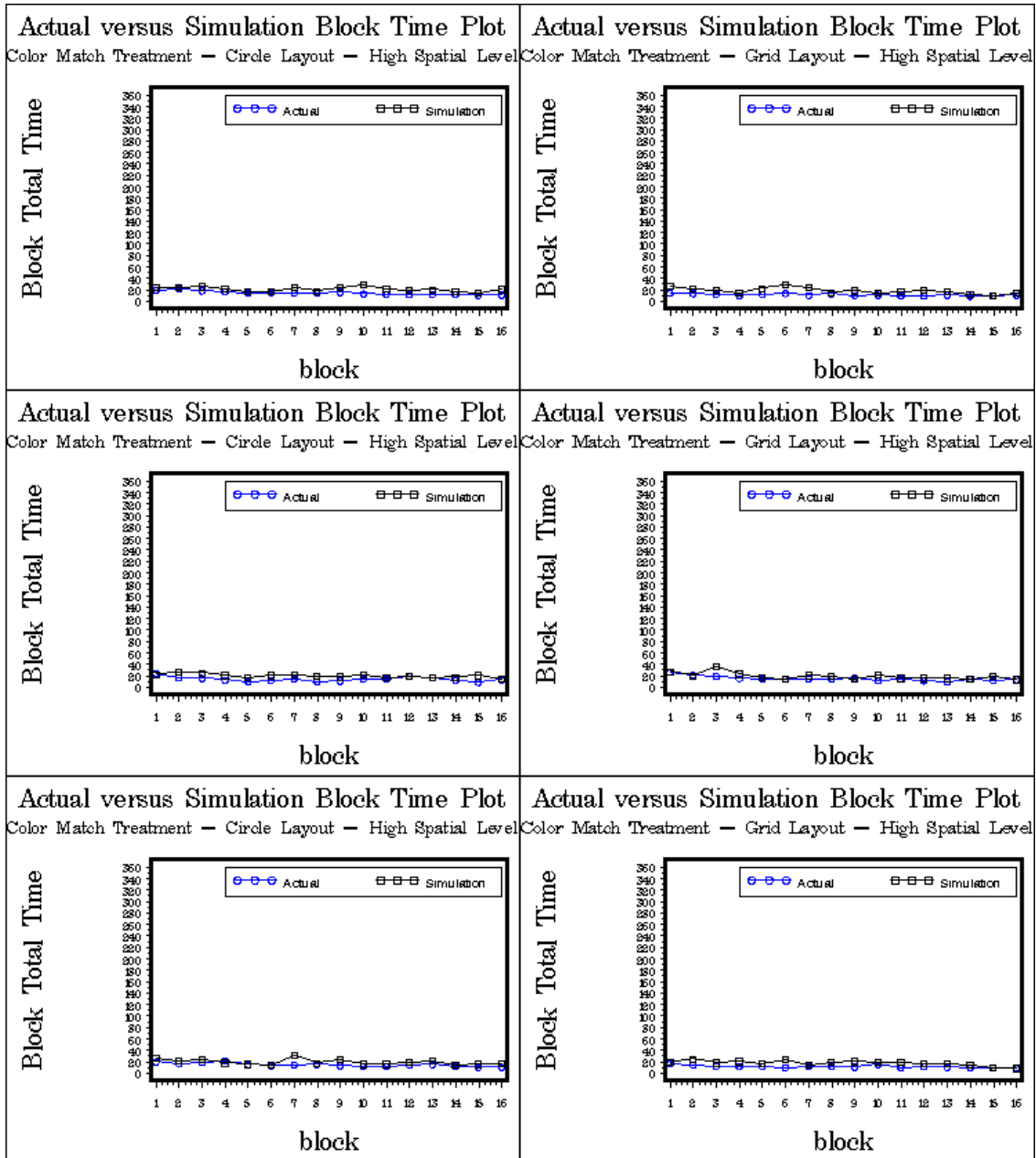
Below is the plotting of block level performance time between actual and simulation for each individual within spatial groups of different layout and label treatment.

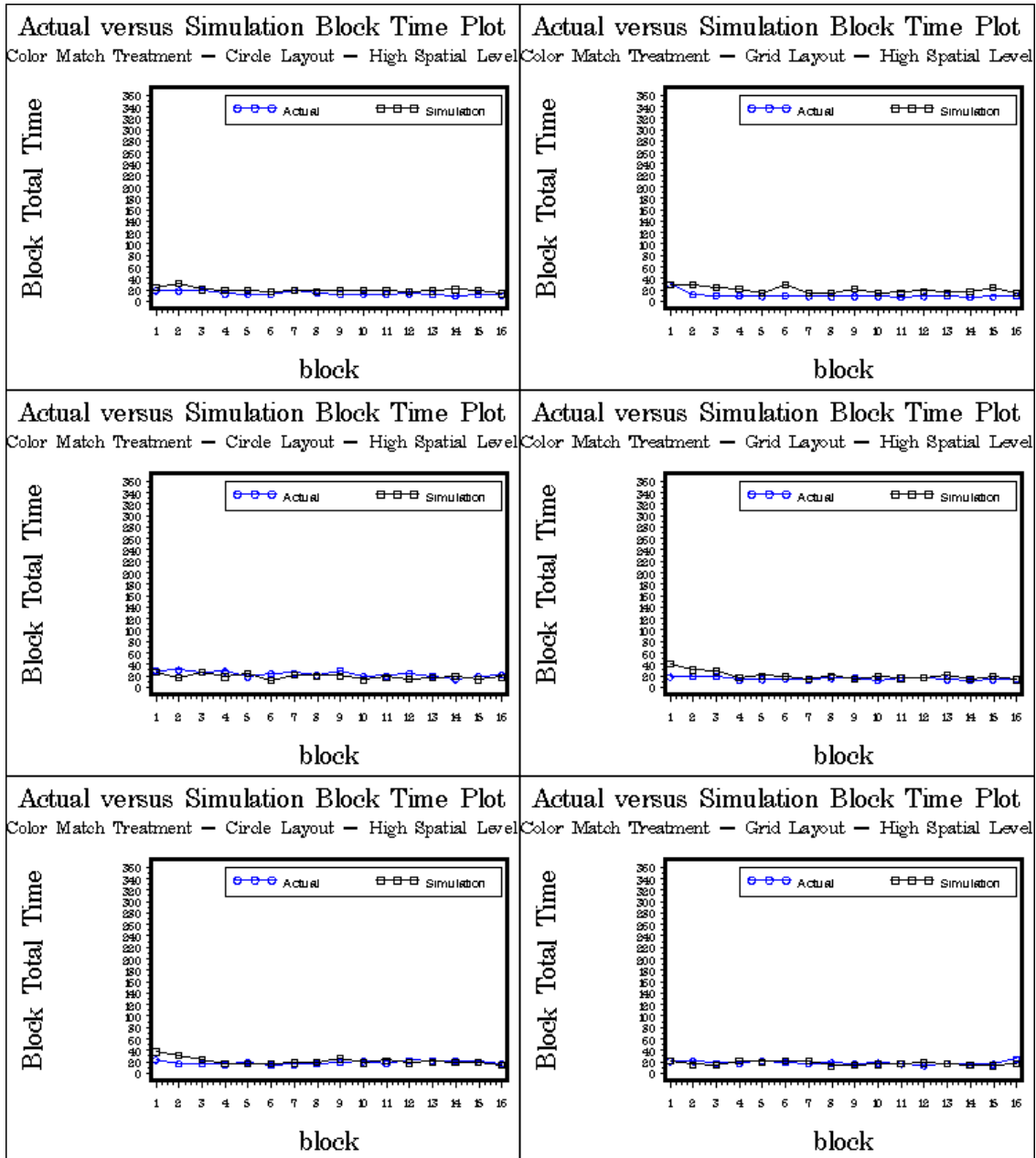


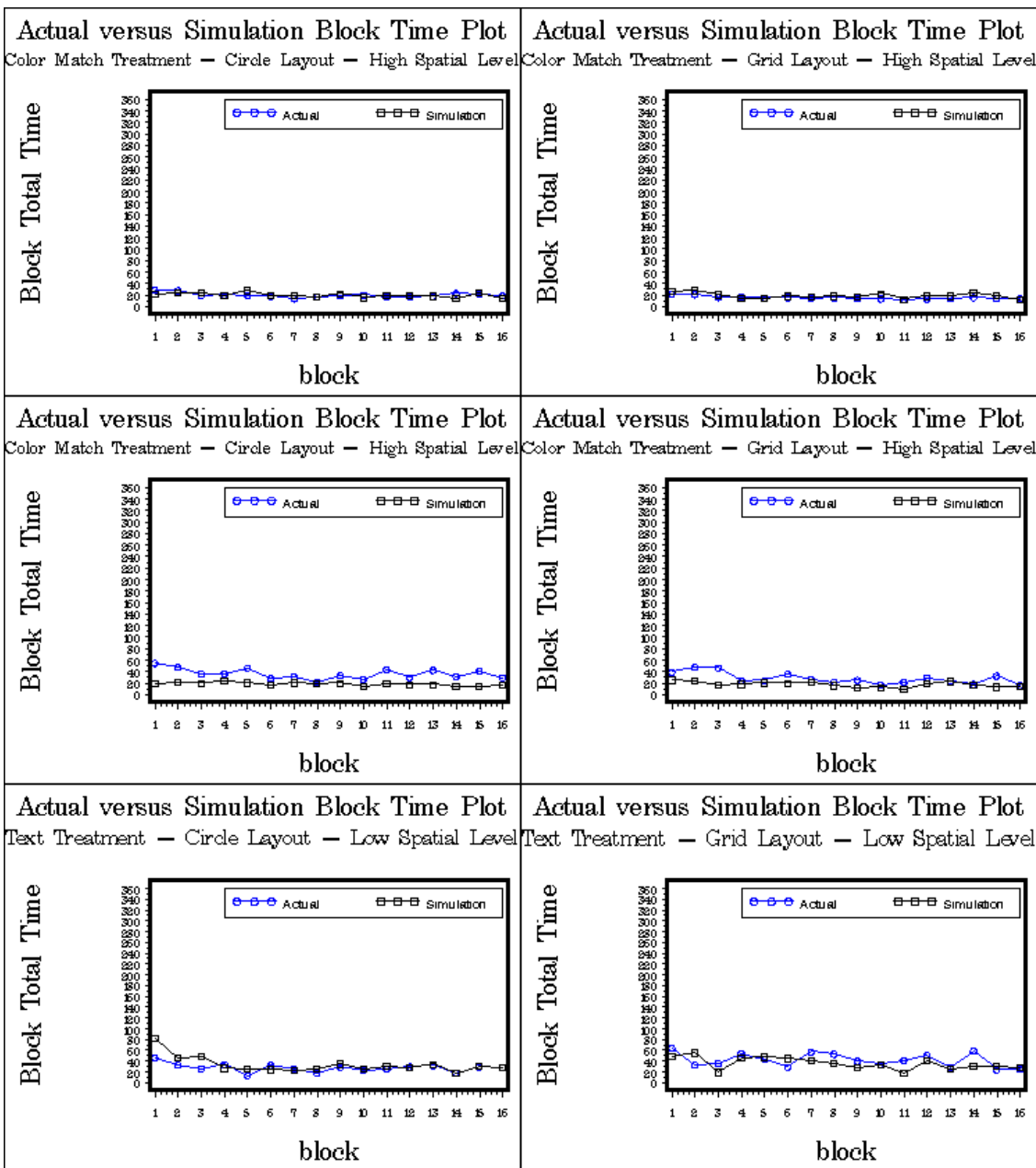


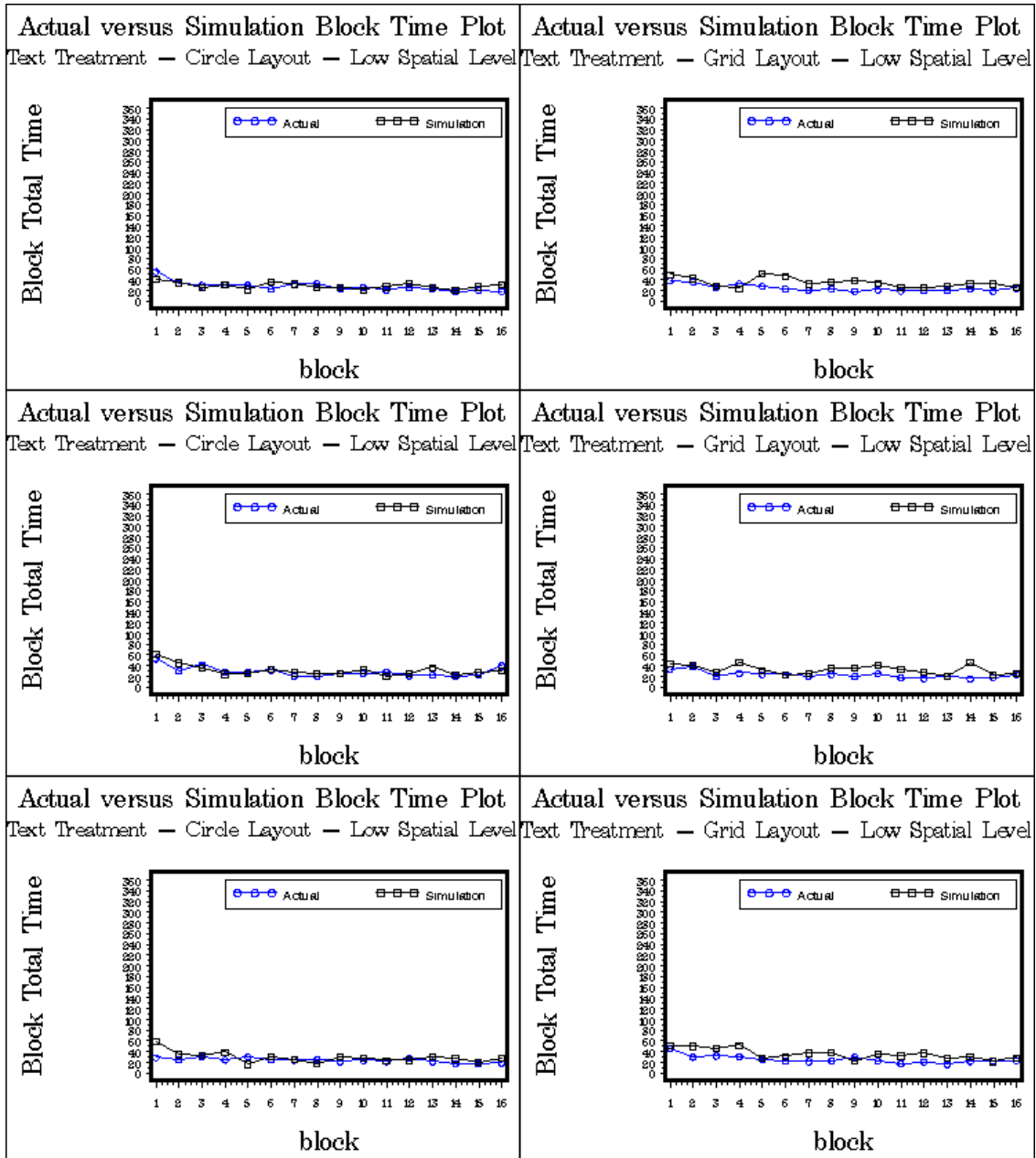


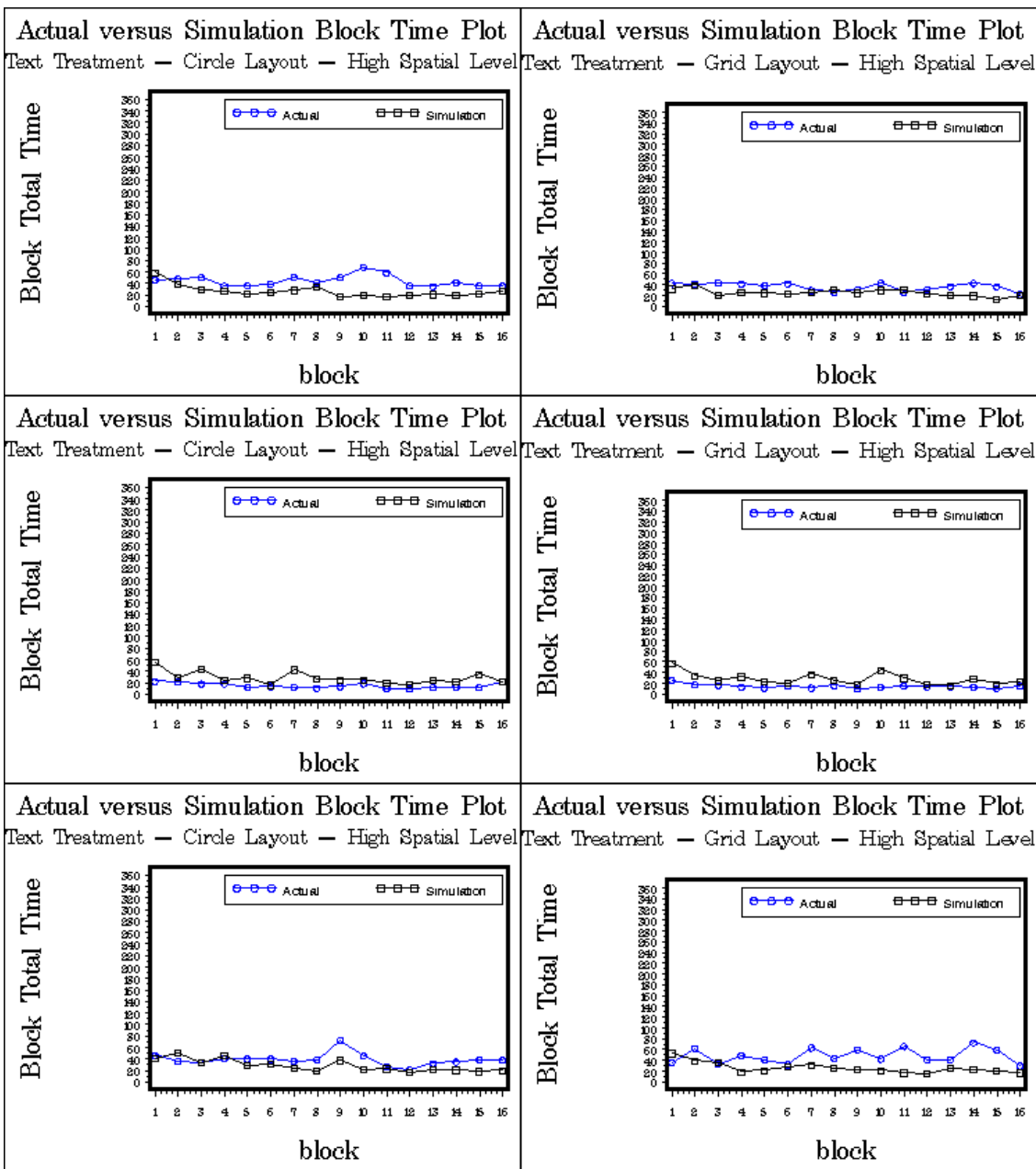


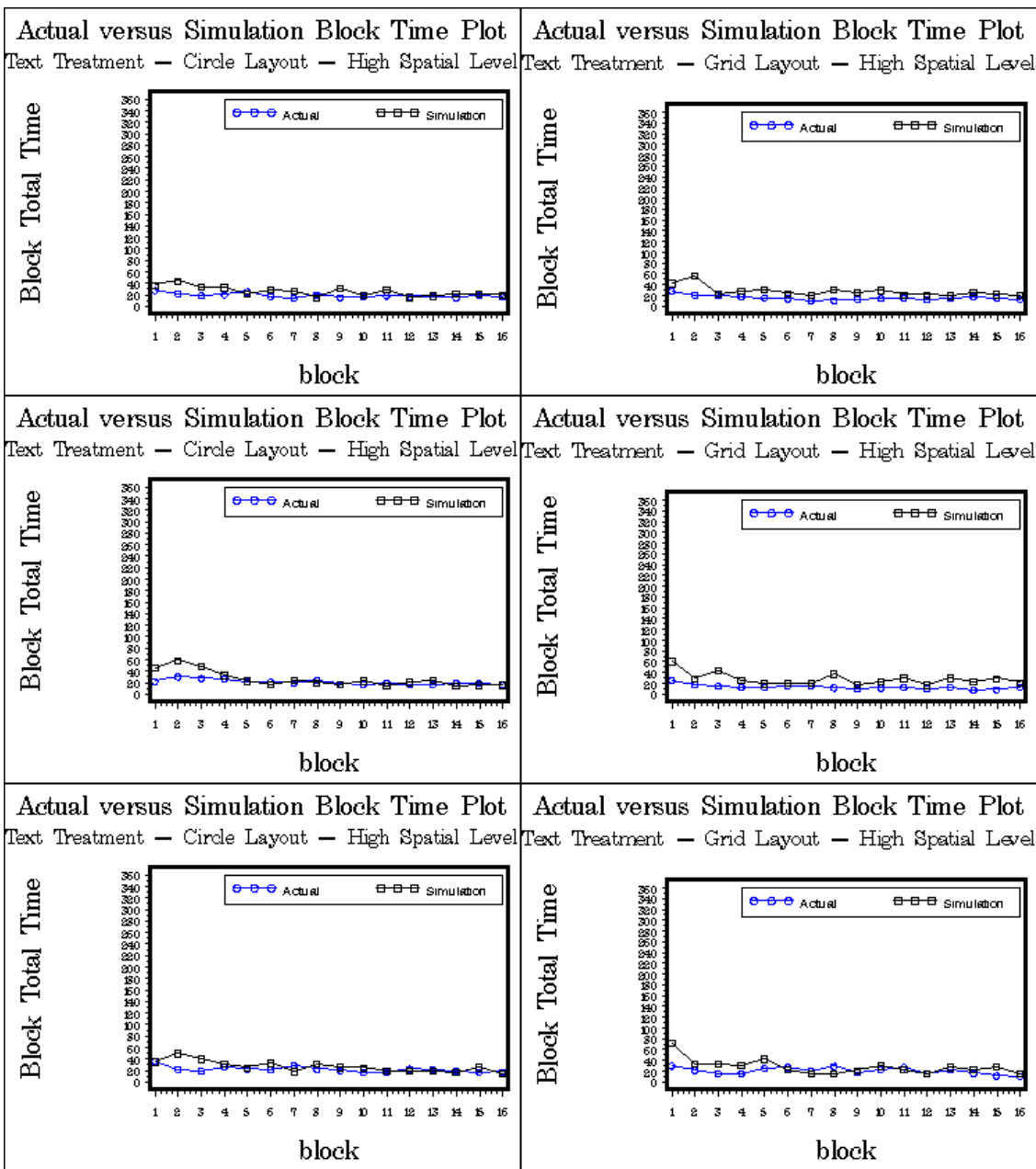


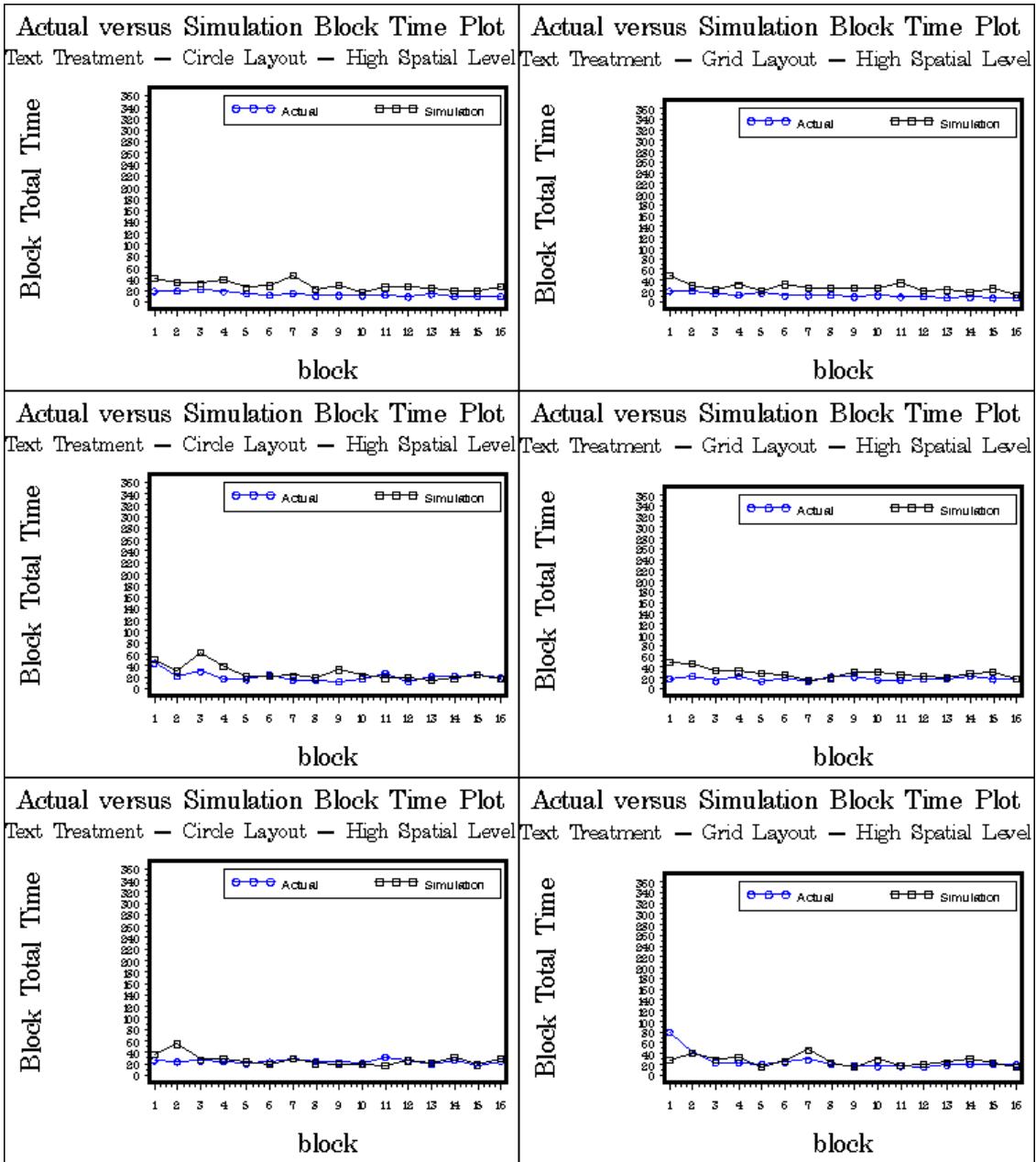


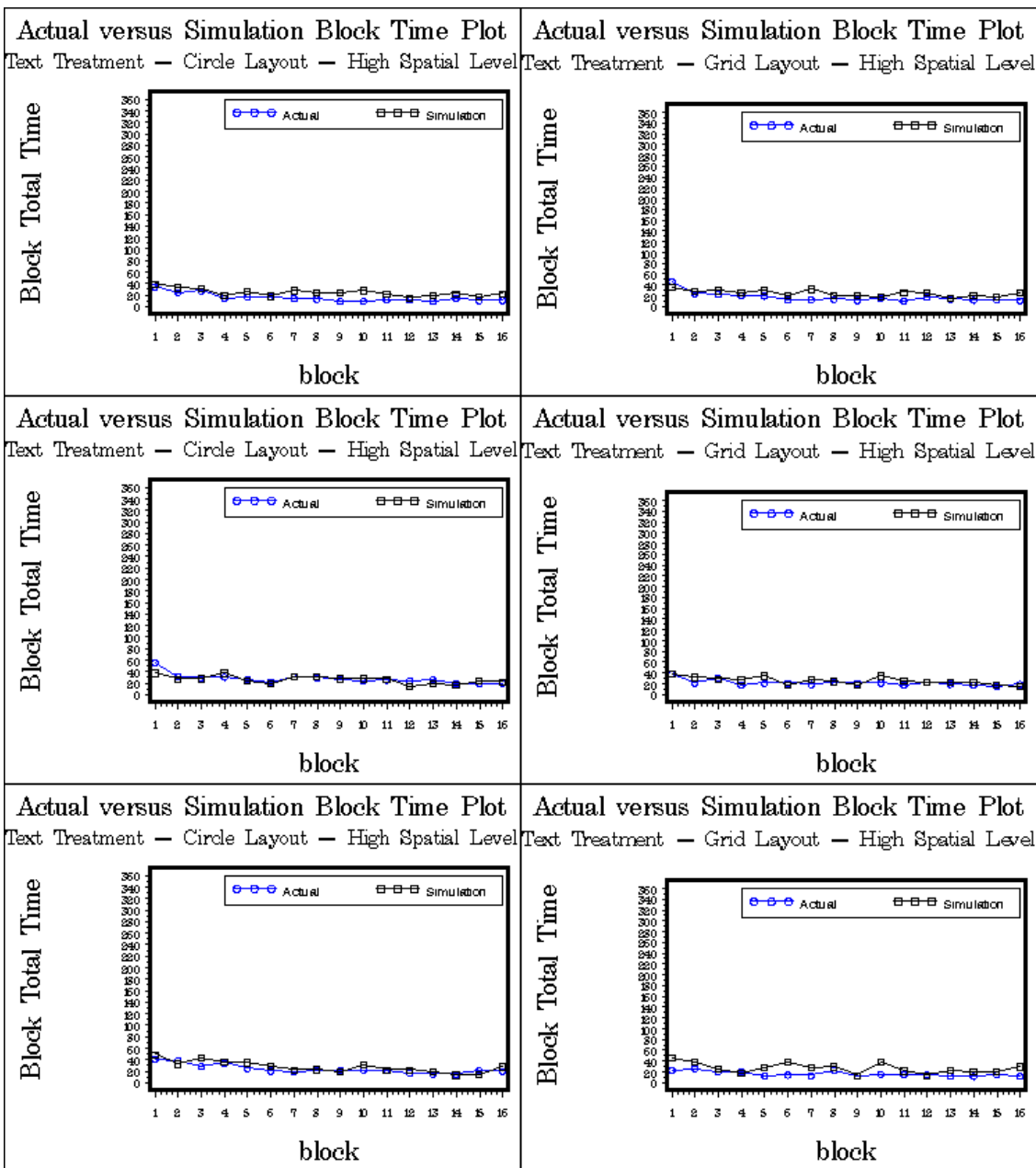


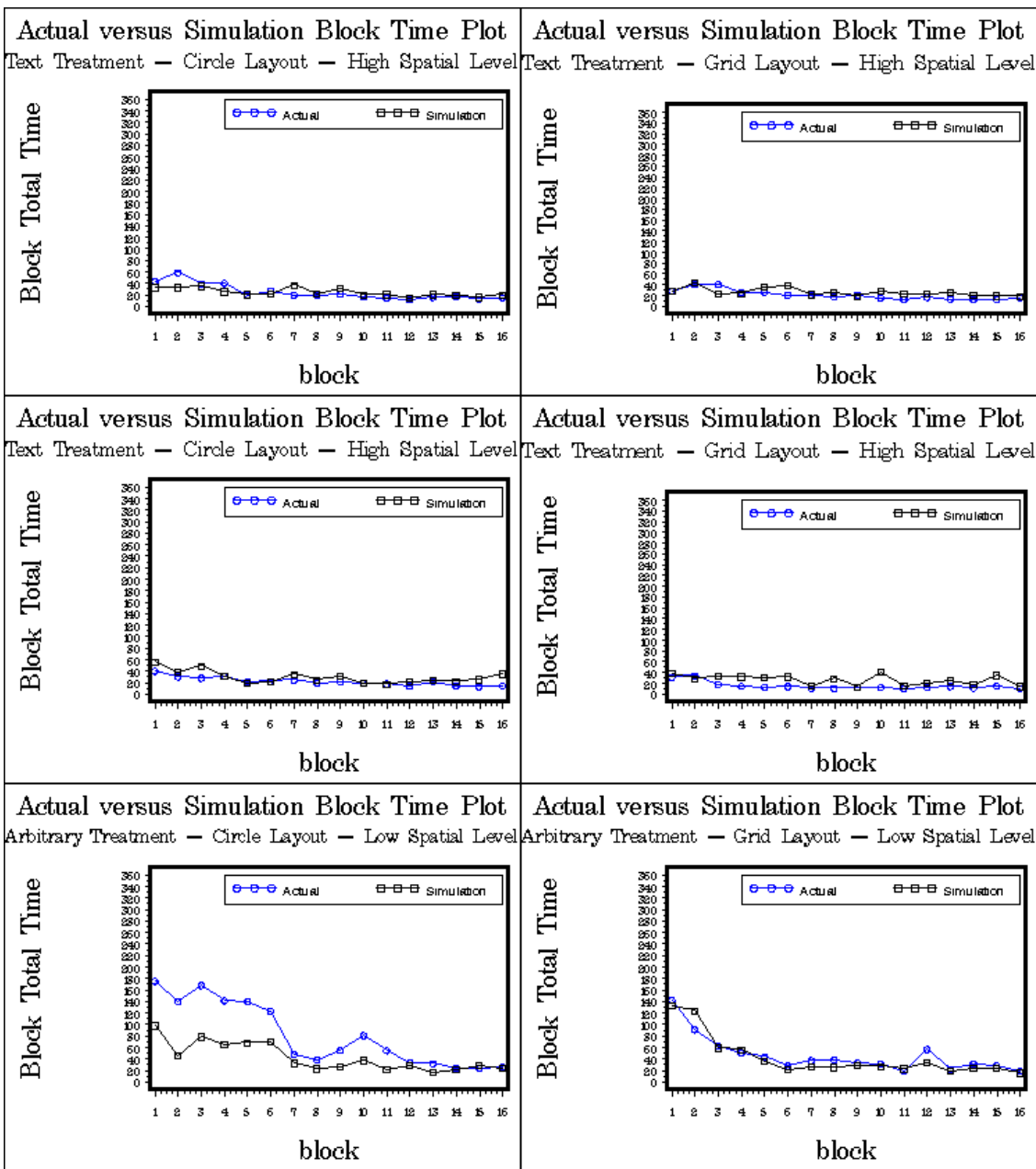


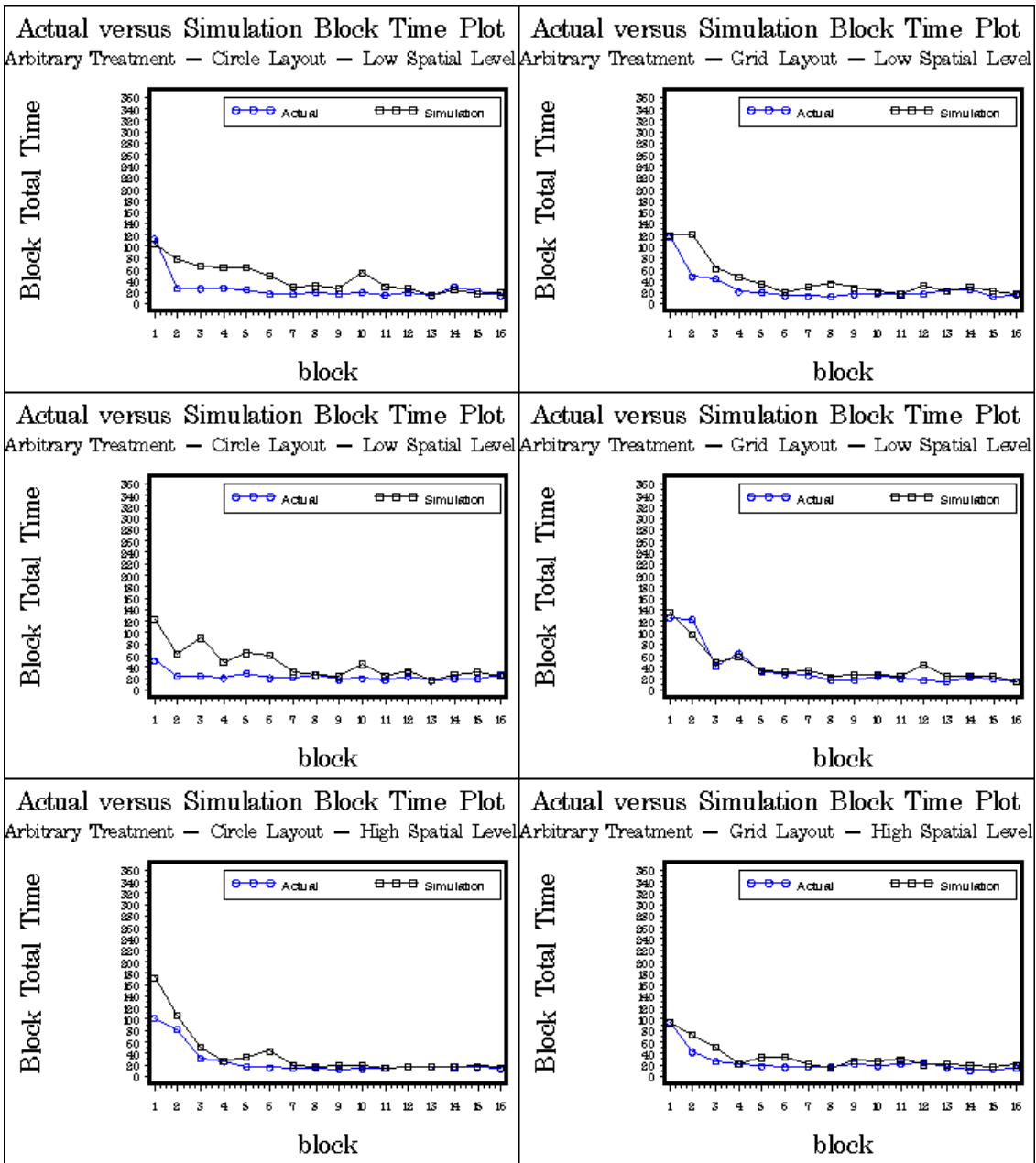


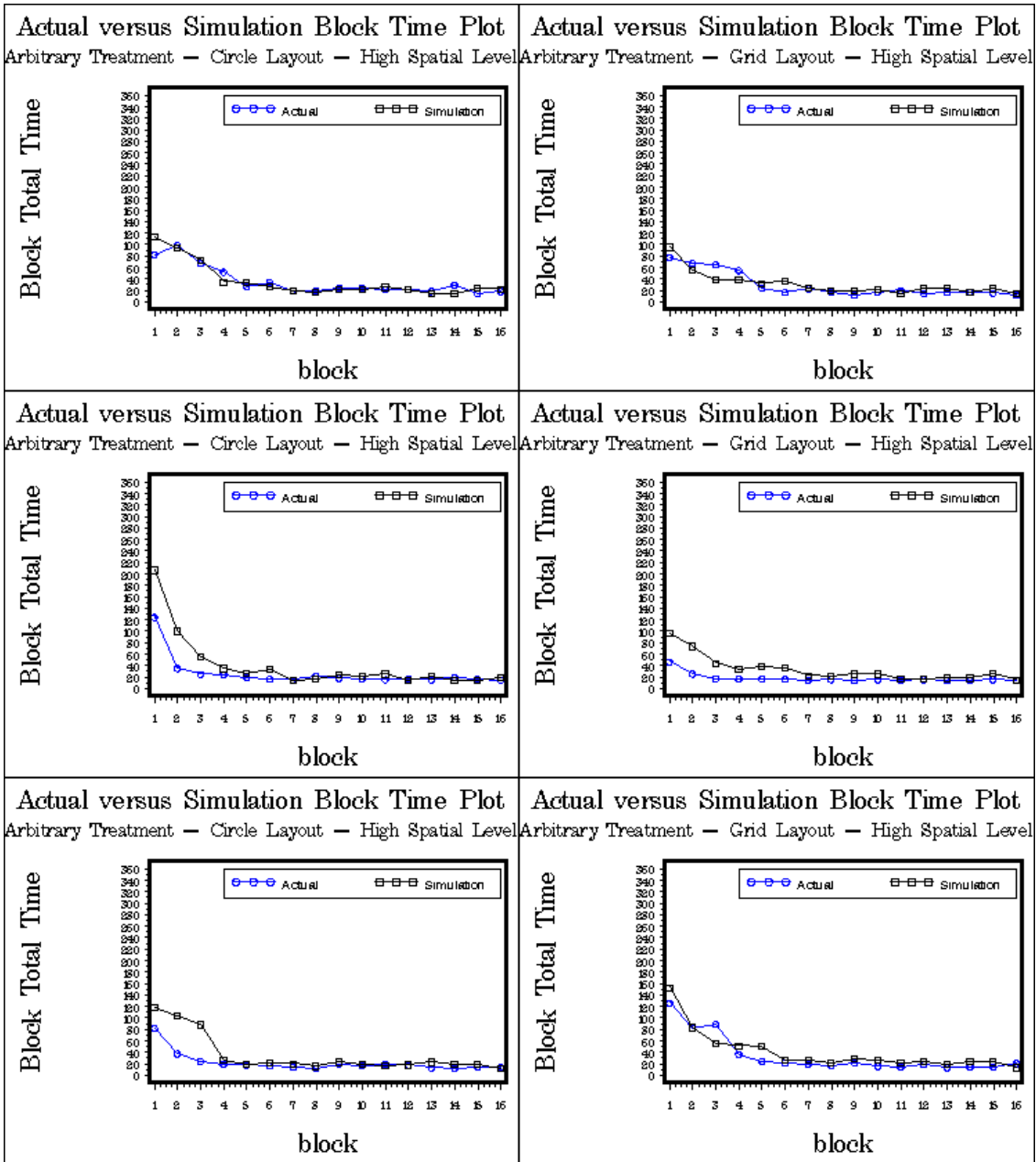


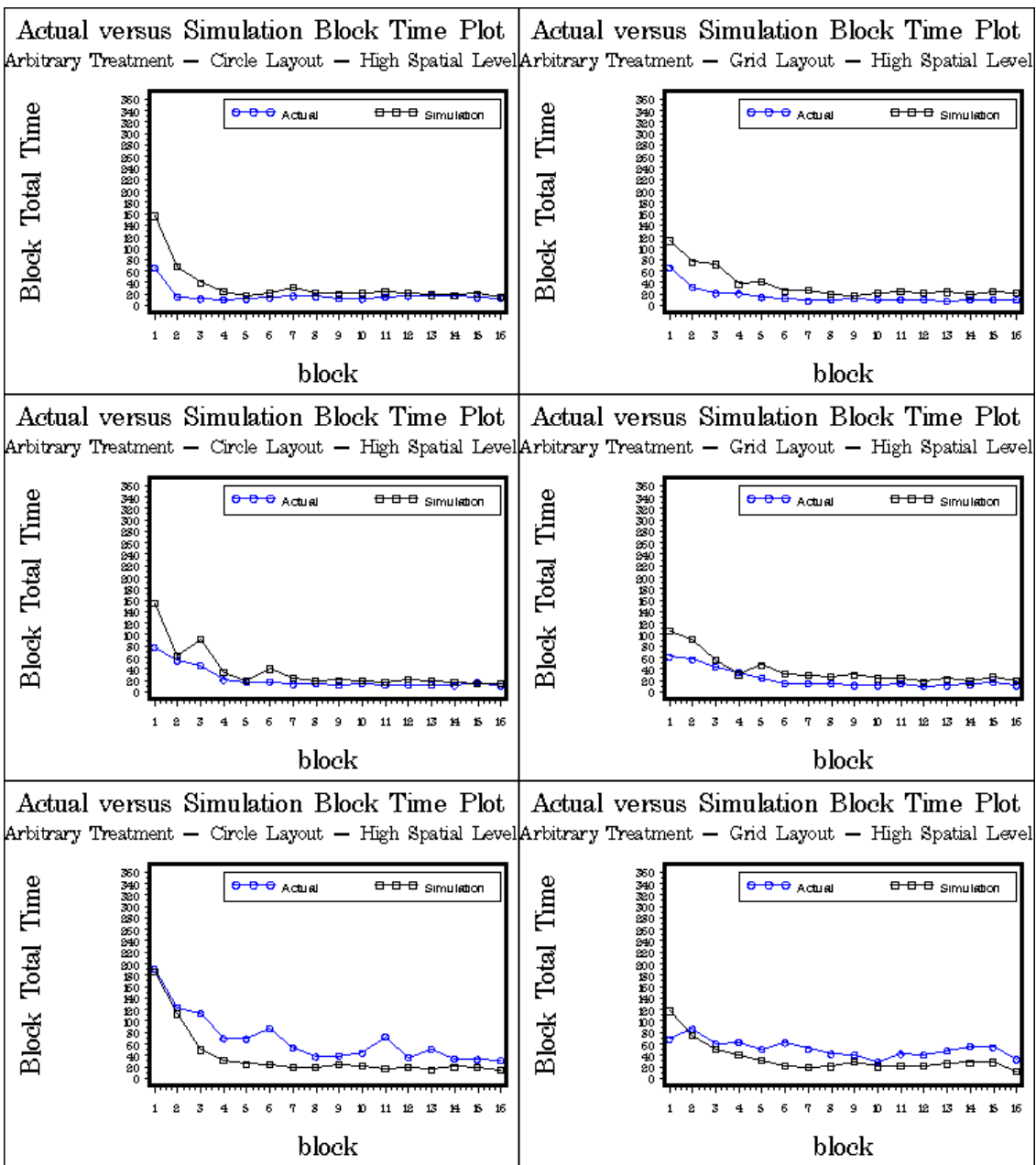


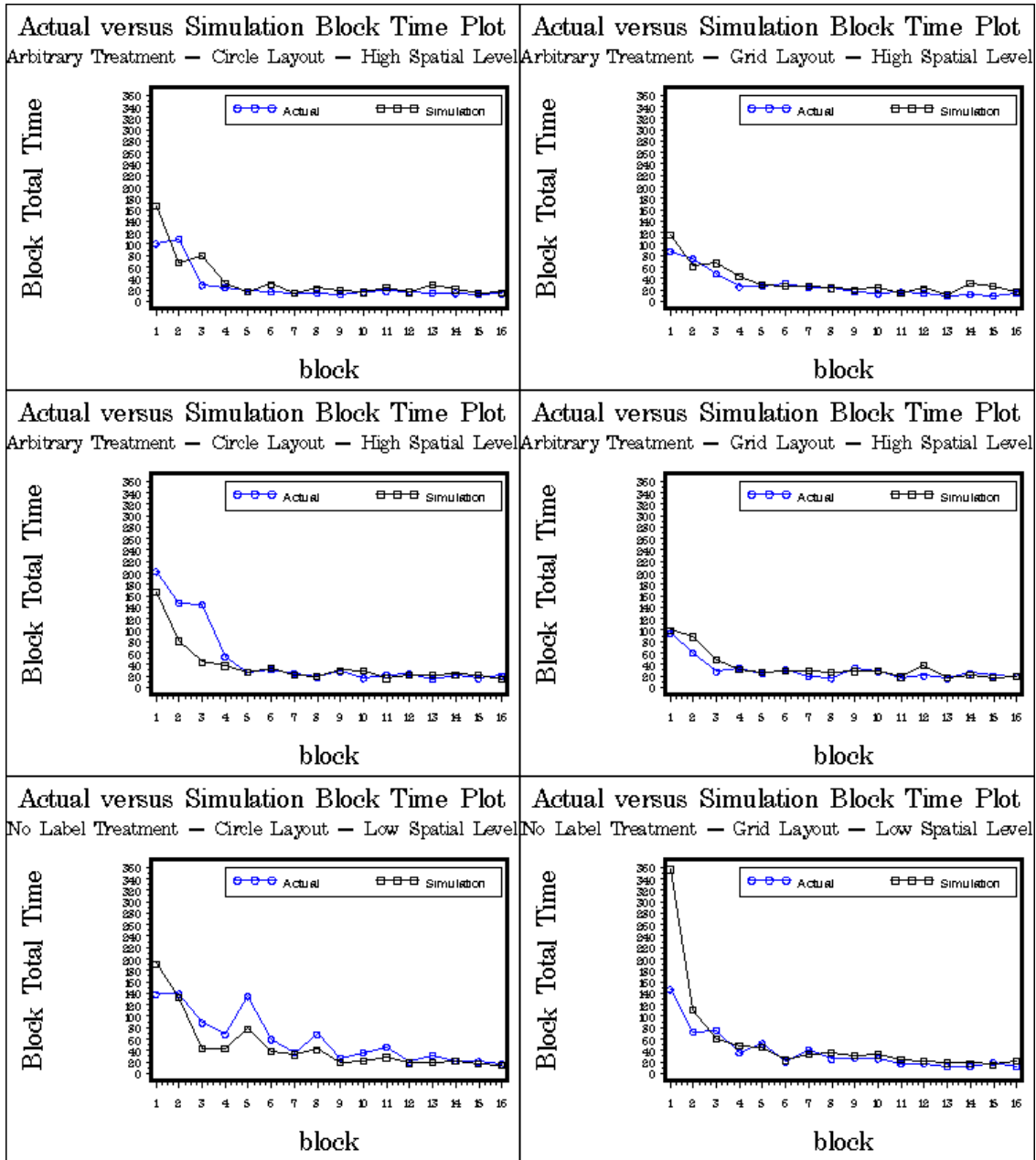


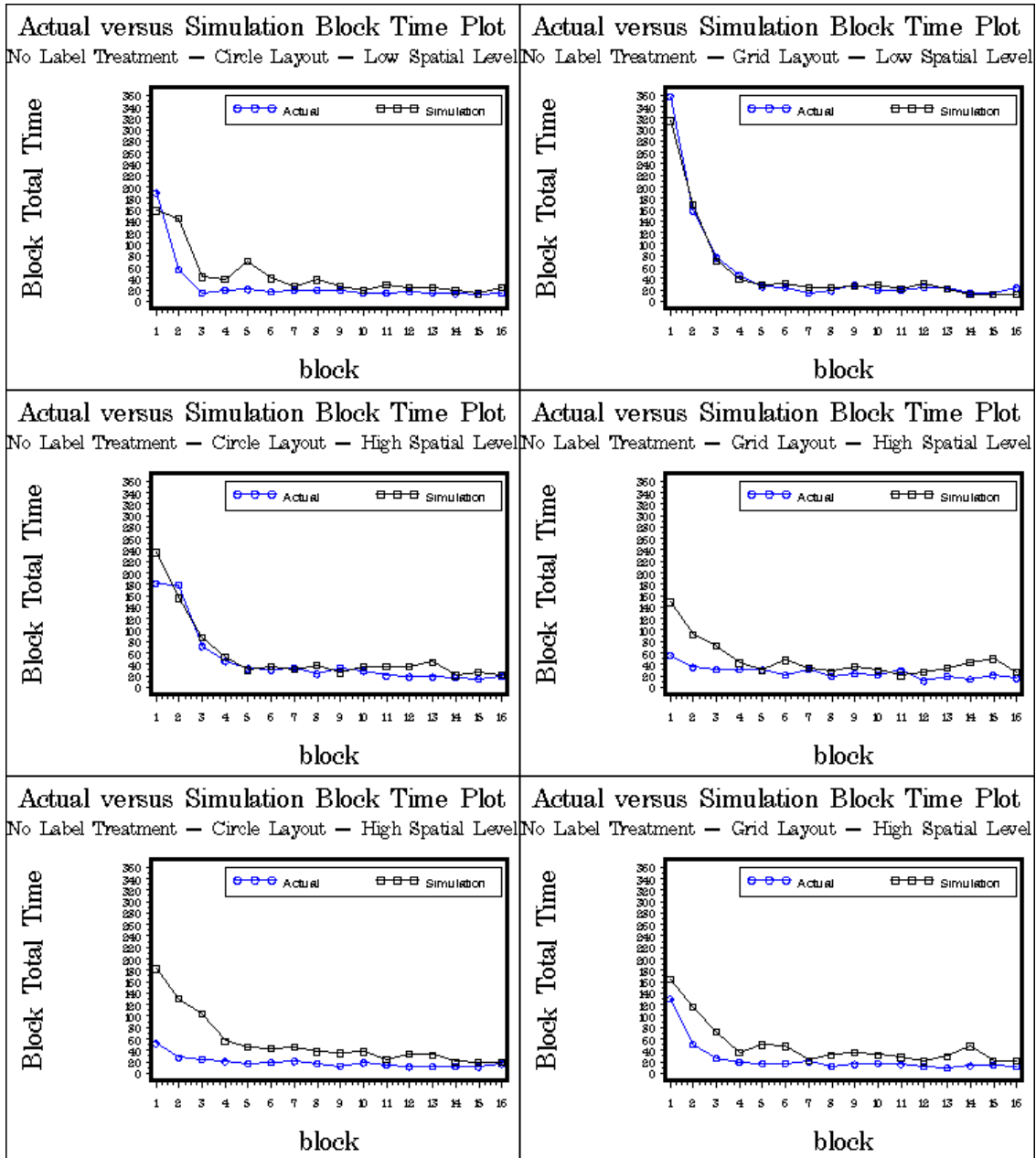


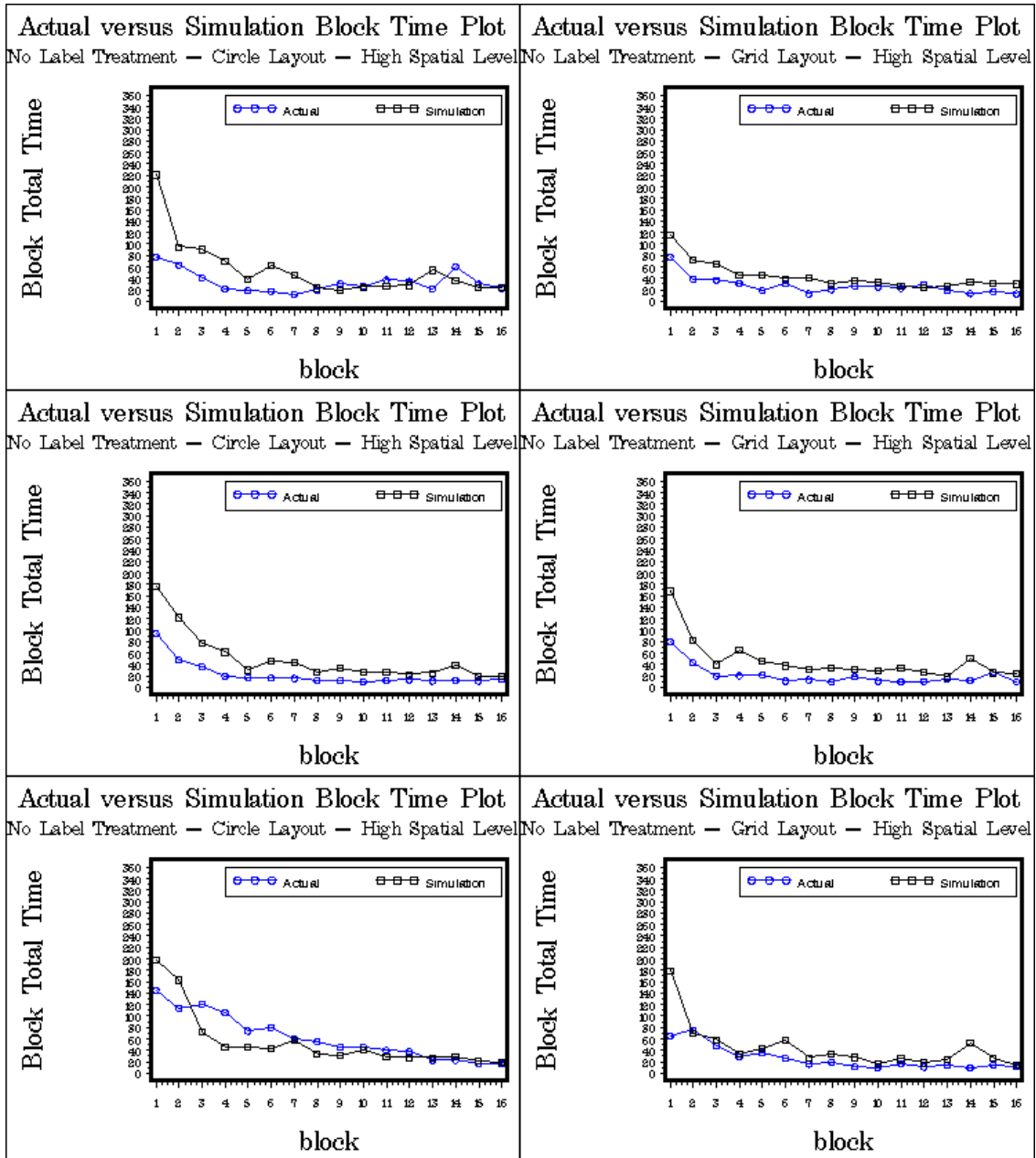


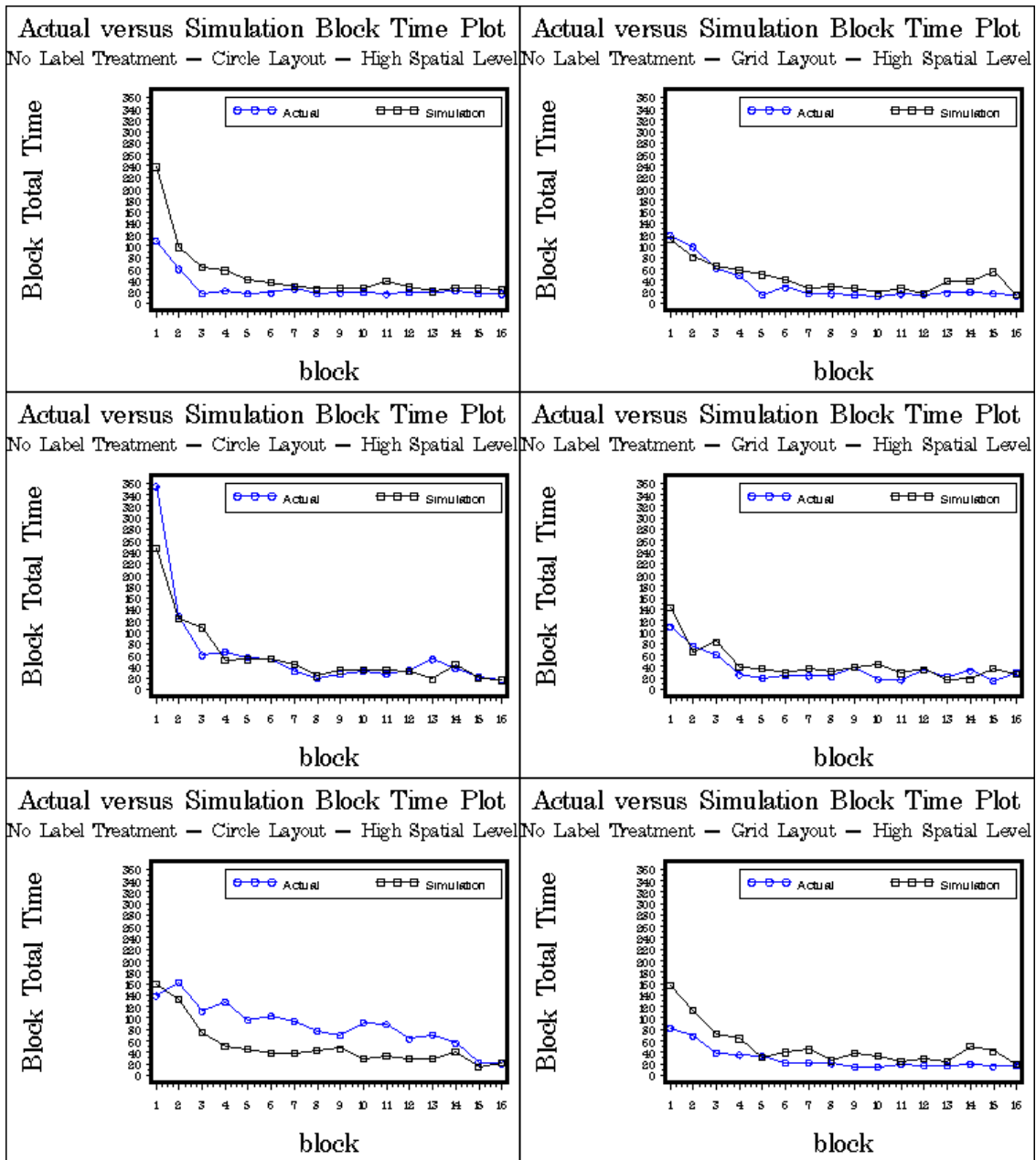


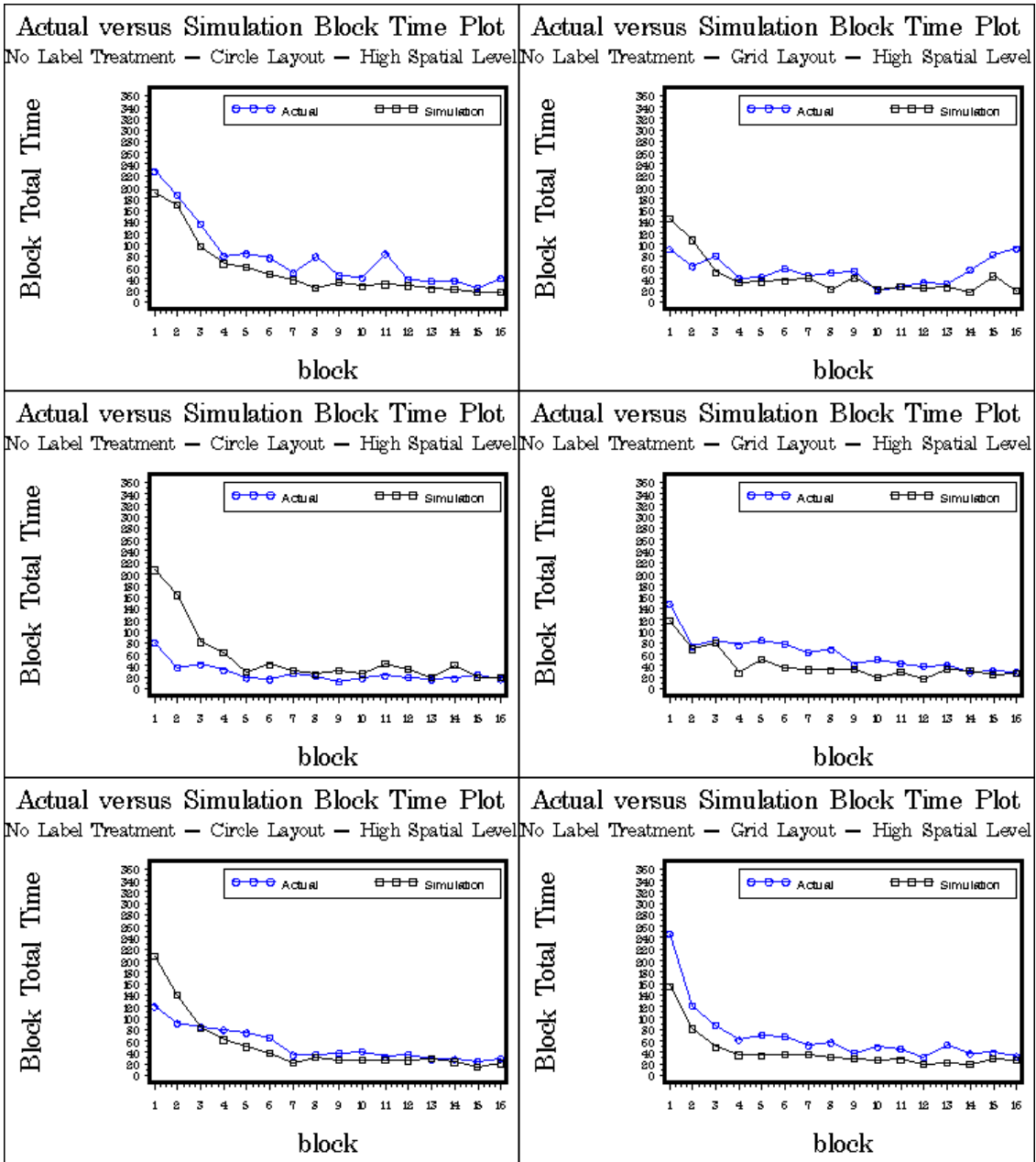






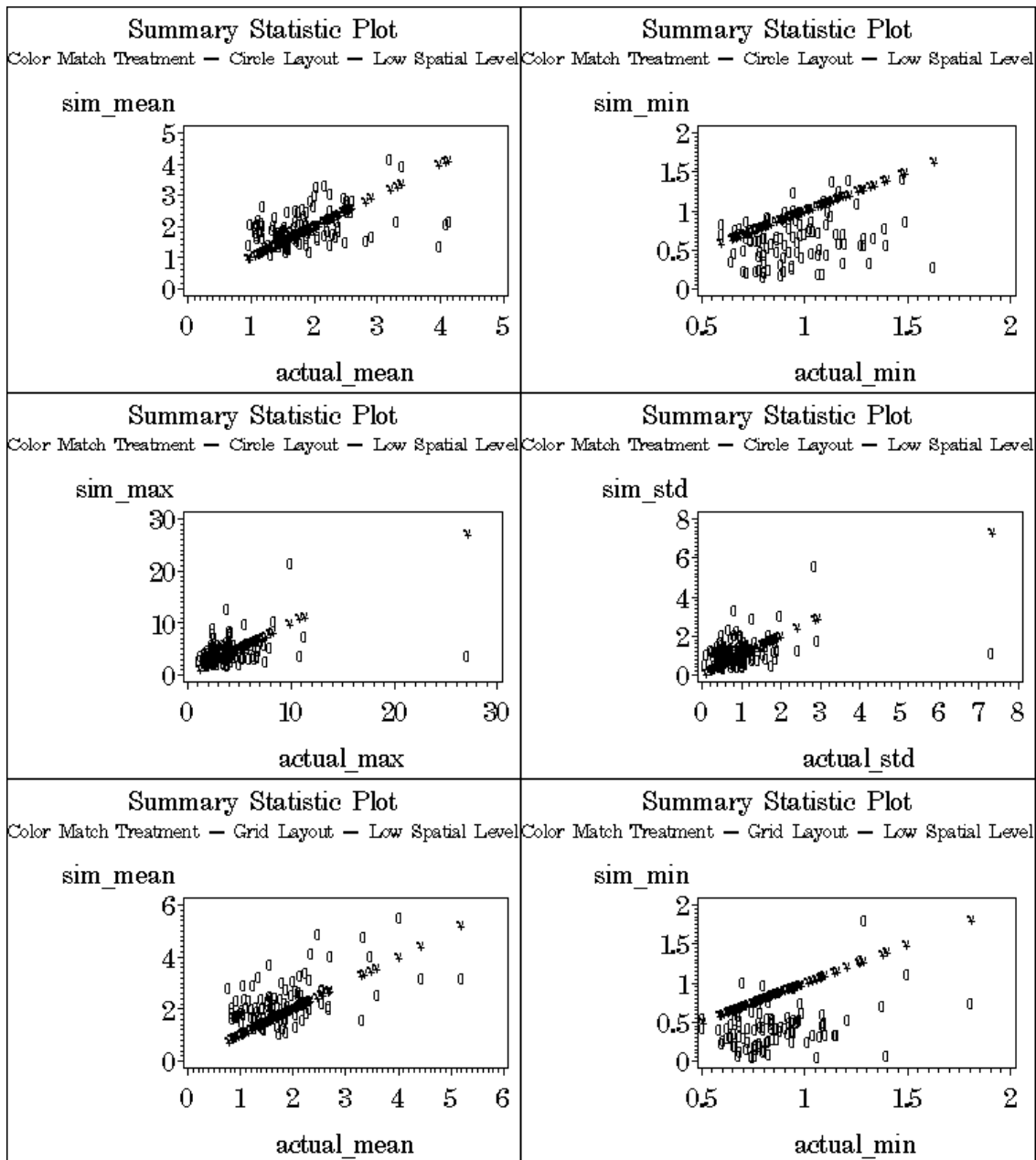


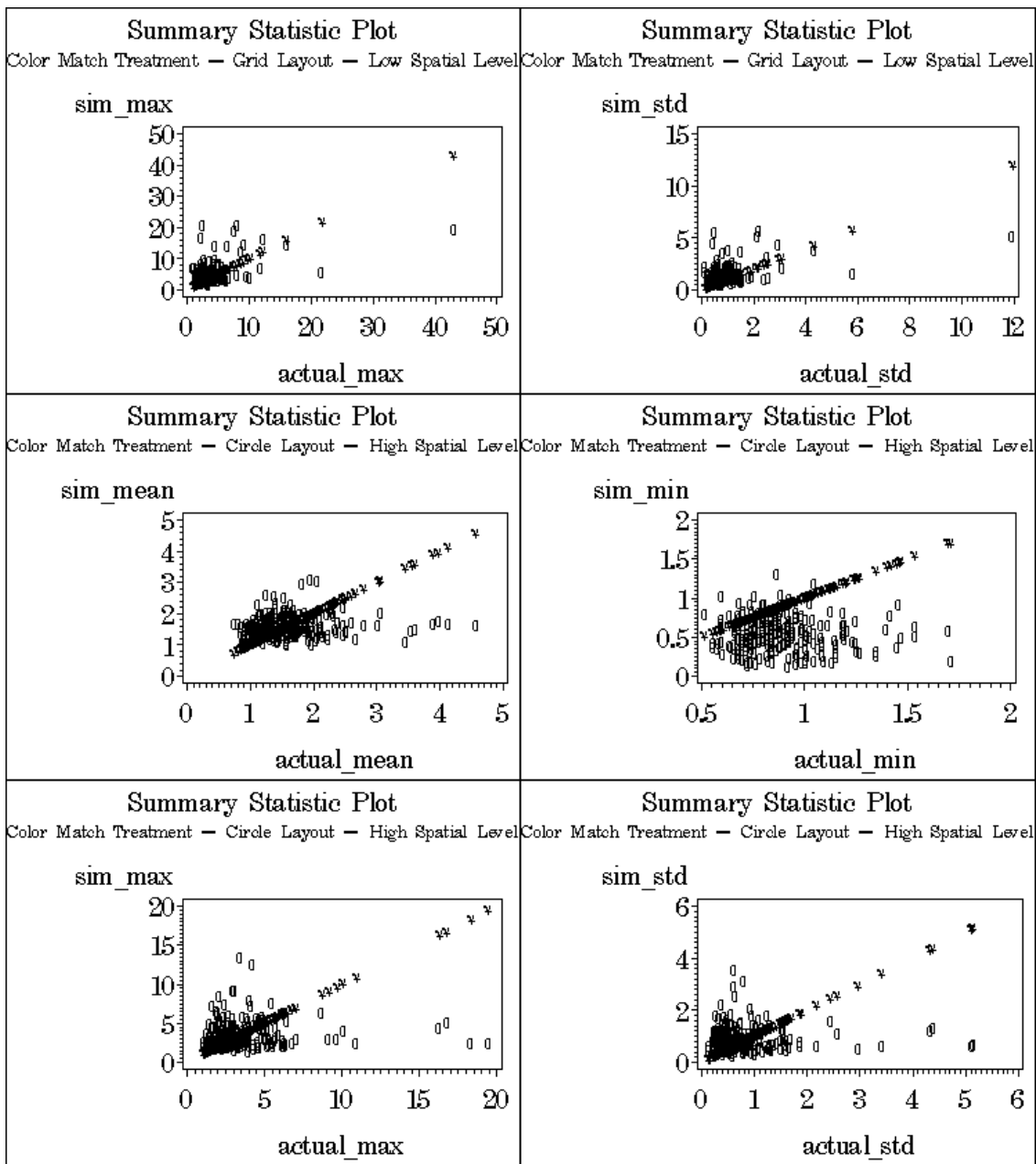


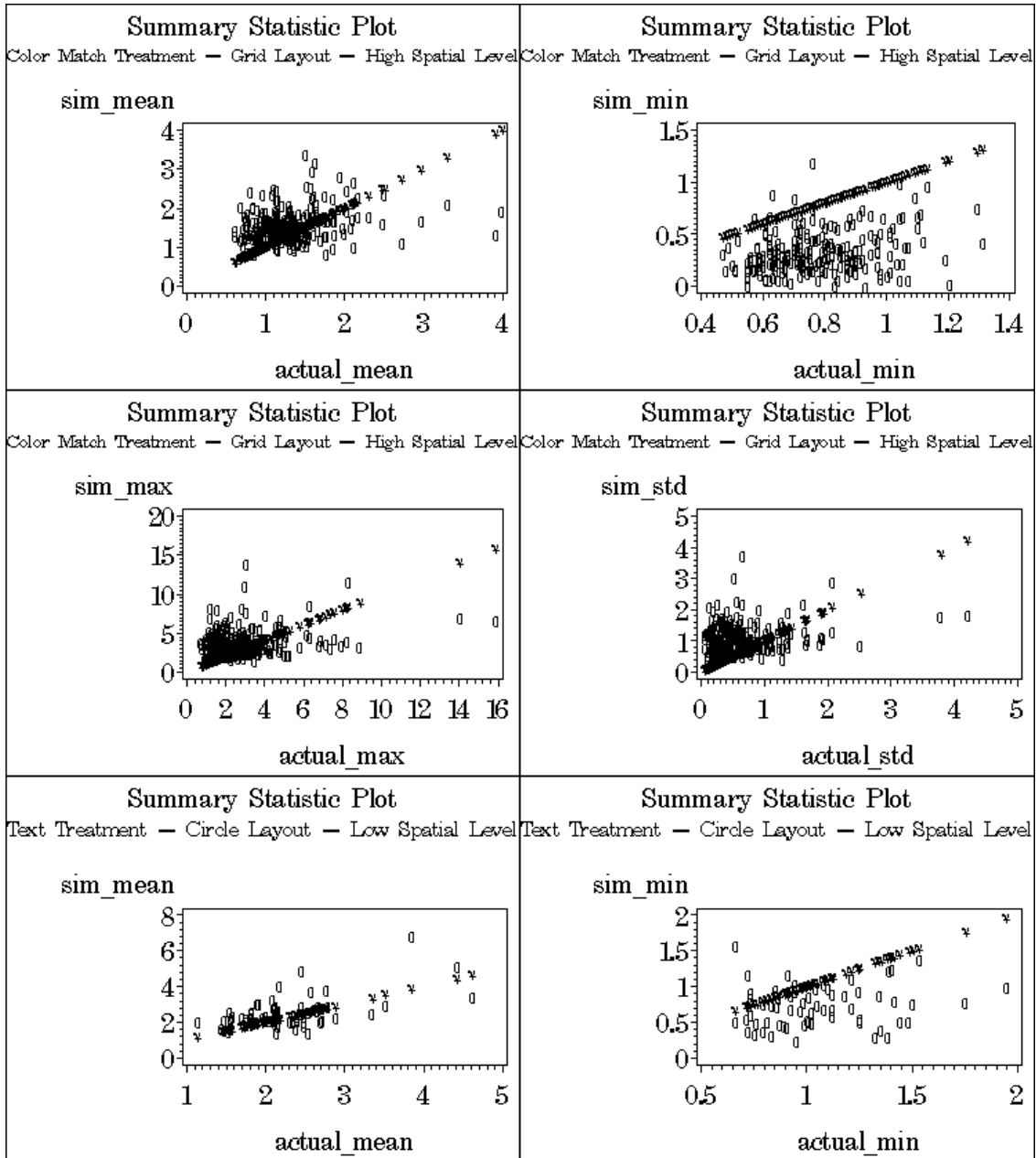


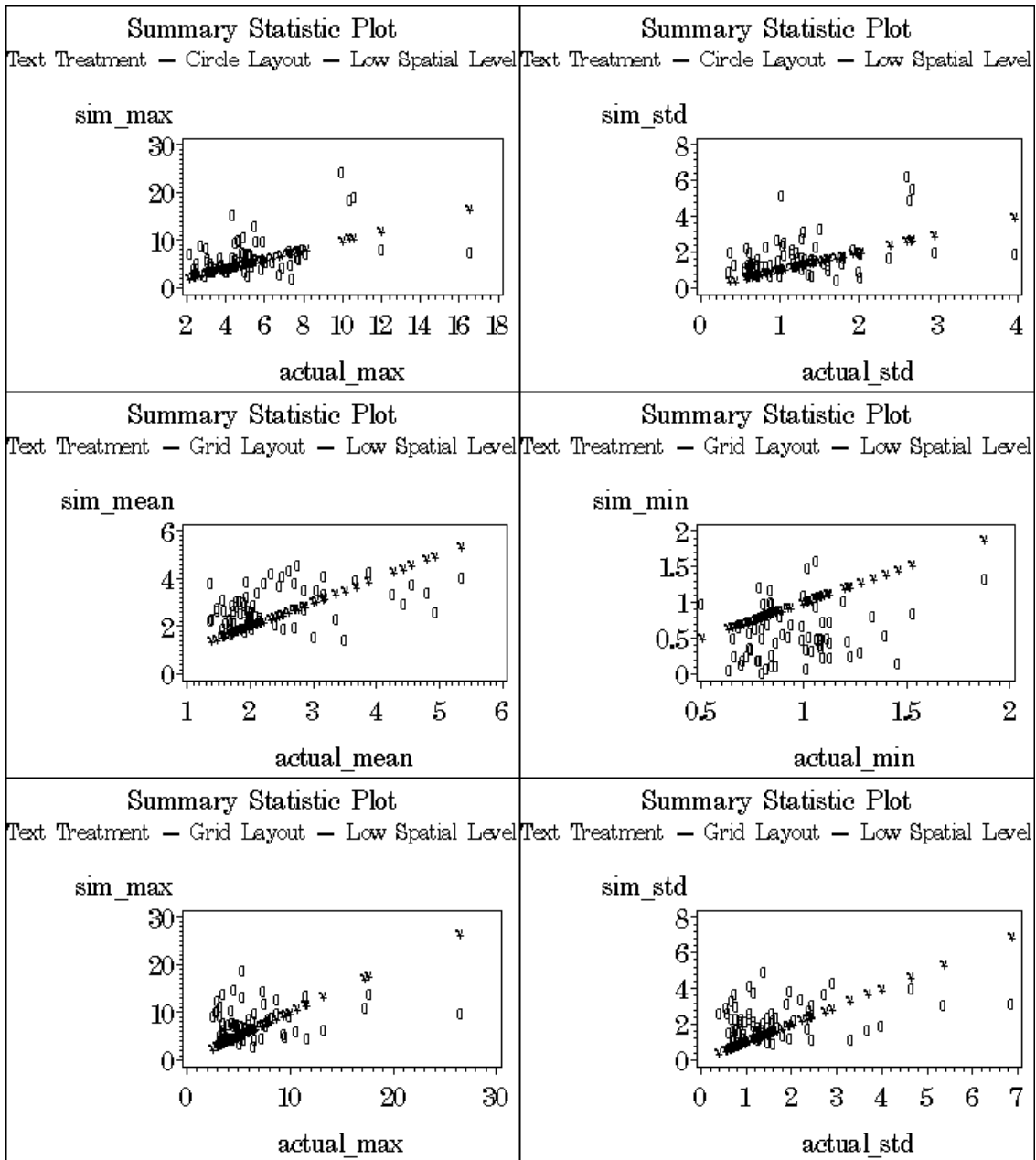
C2. Actual versus simulation summary statistics plots

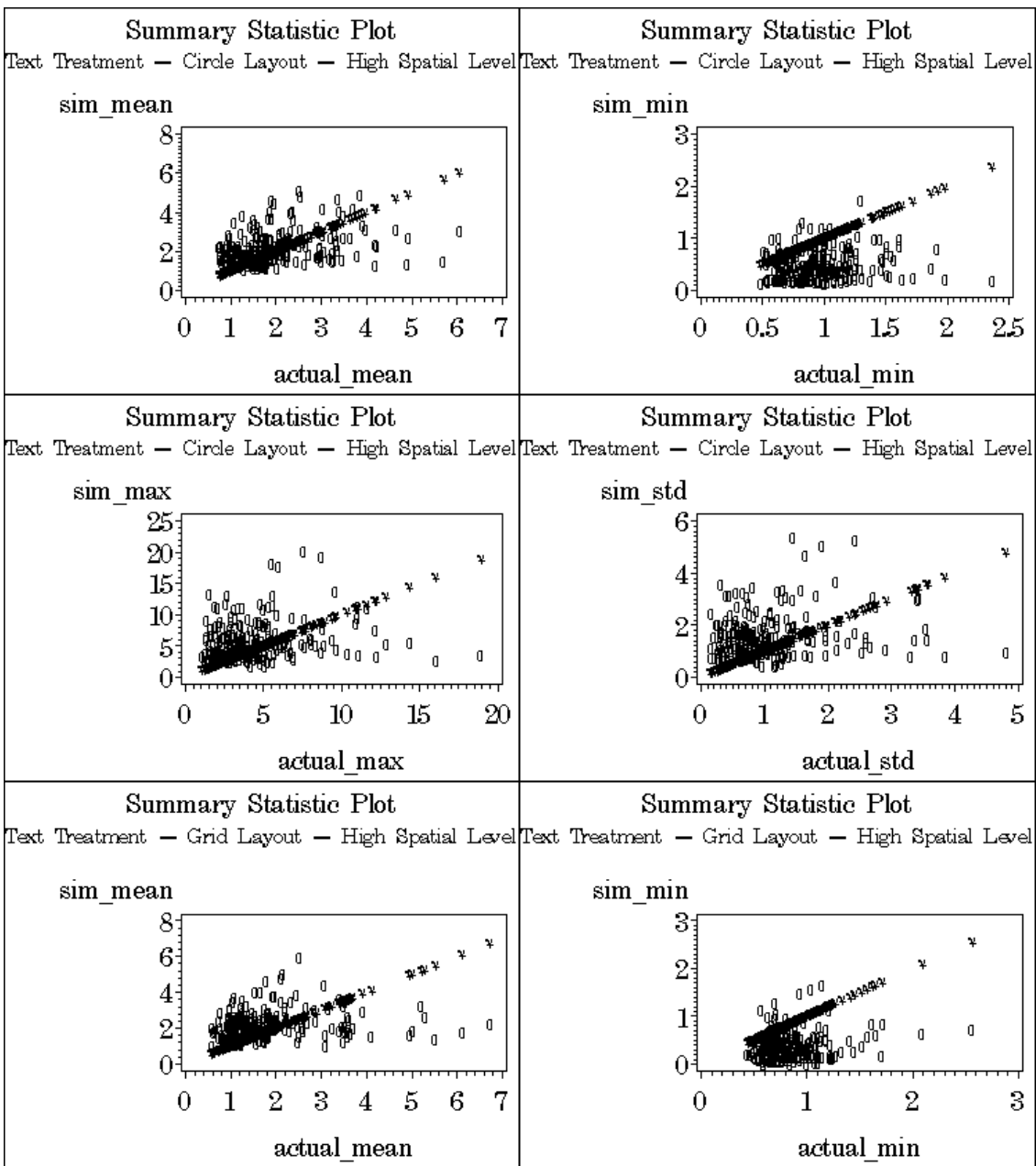
Below is the plotting of statistical summary between actual and simulation for different spatial groups, layout, and label treatment.

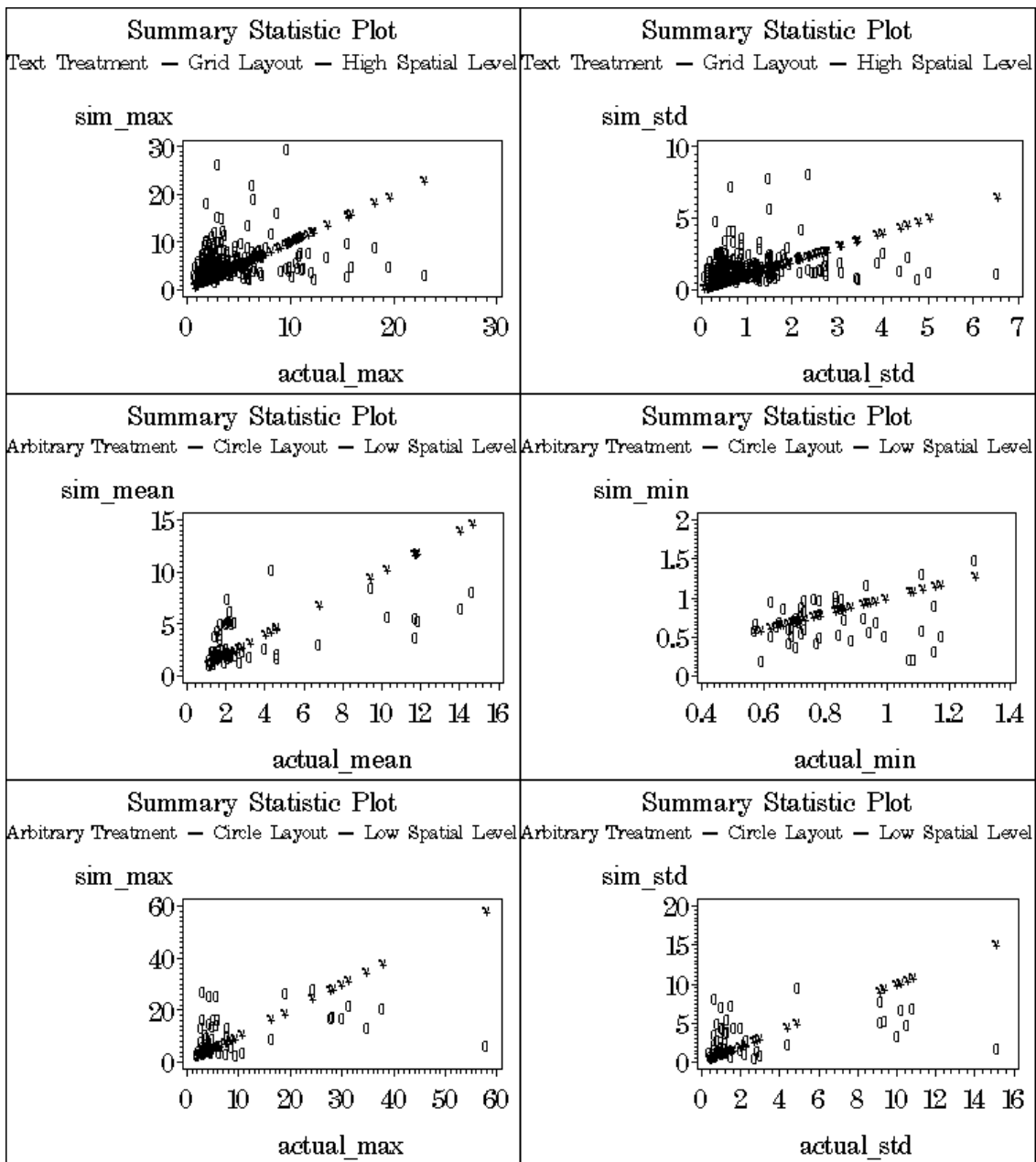


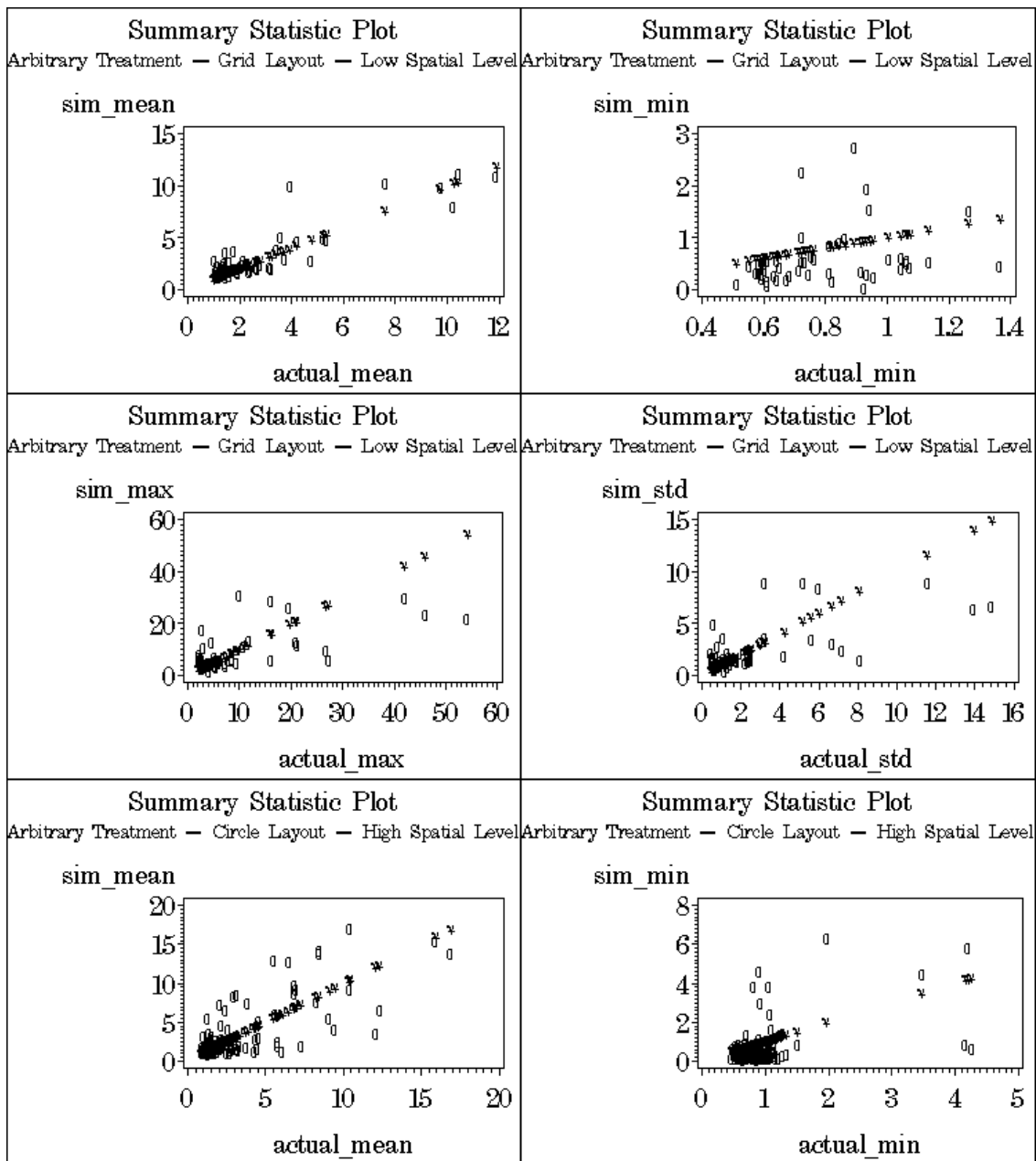


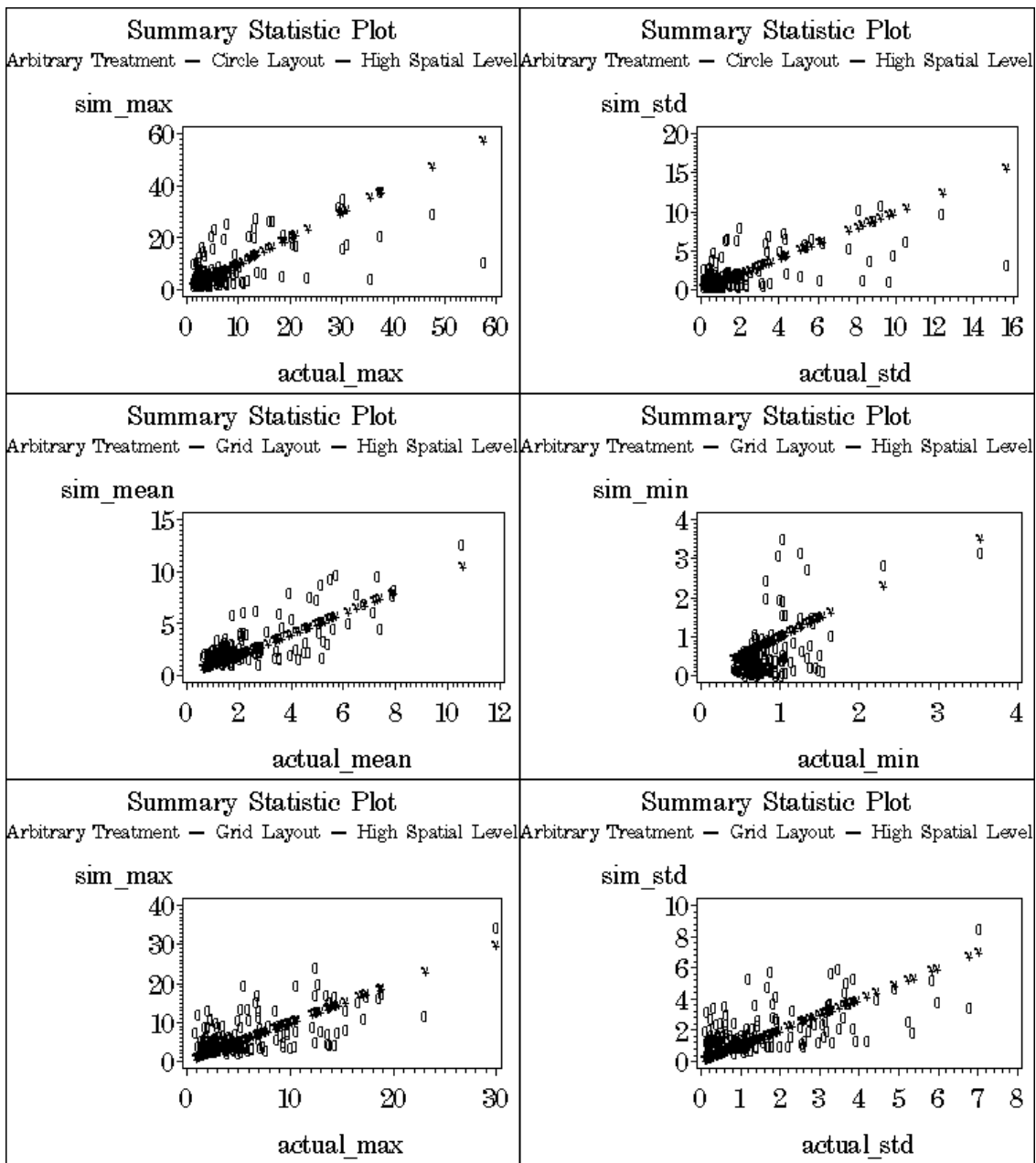


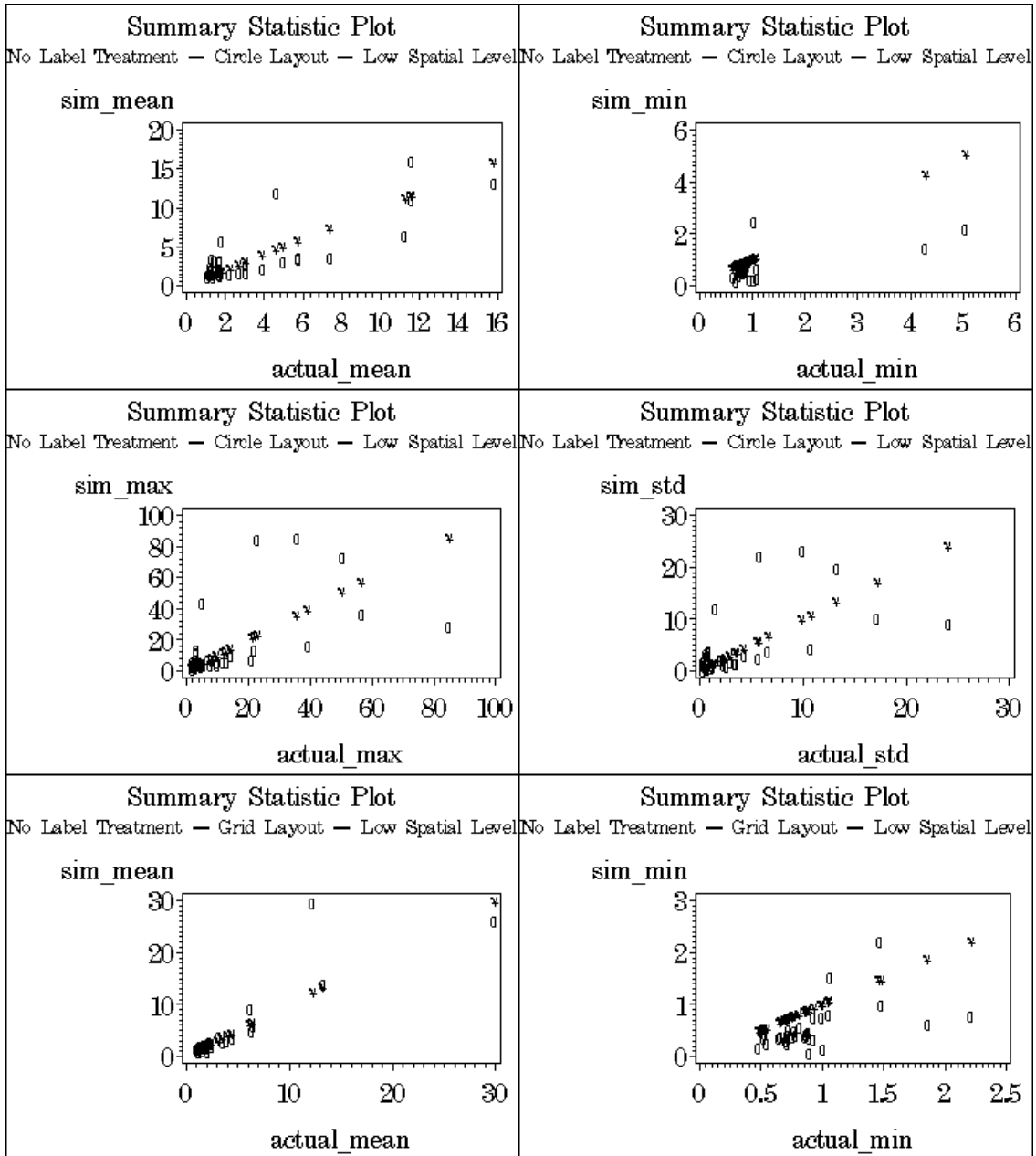


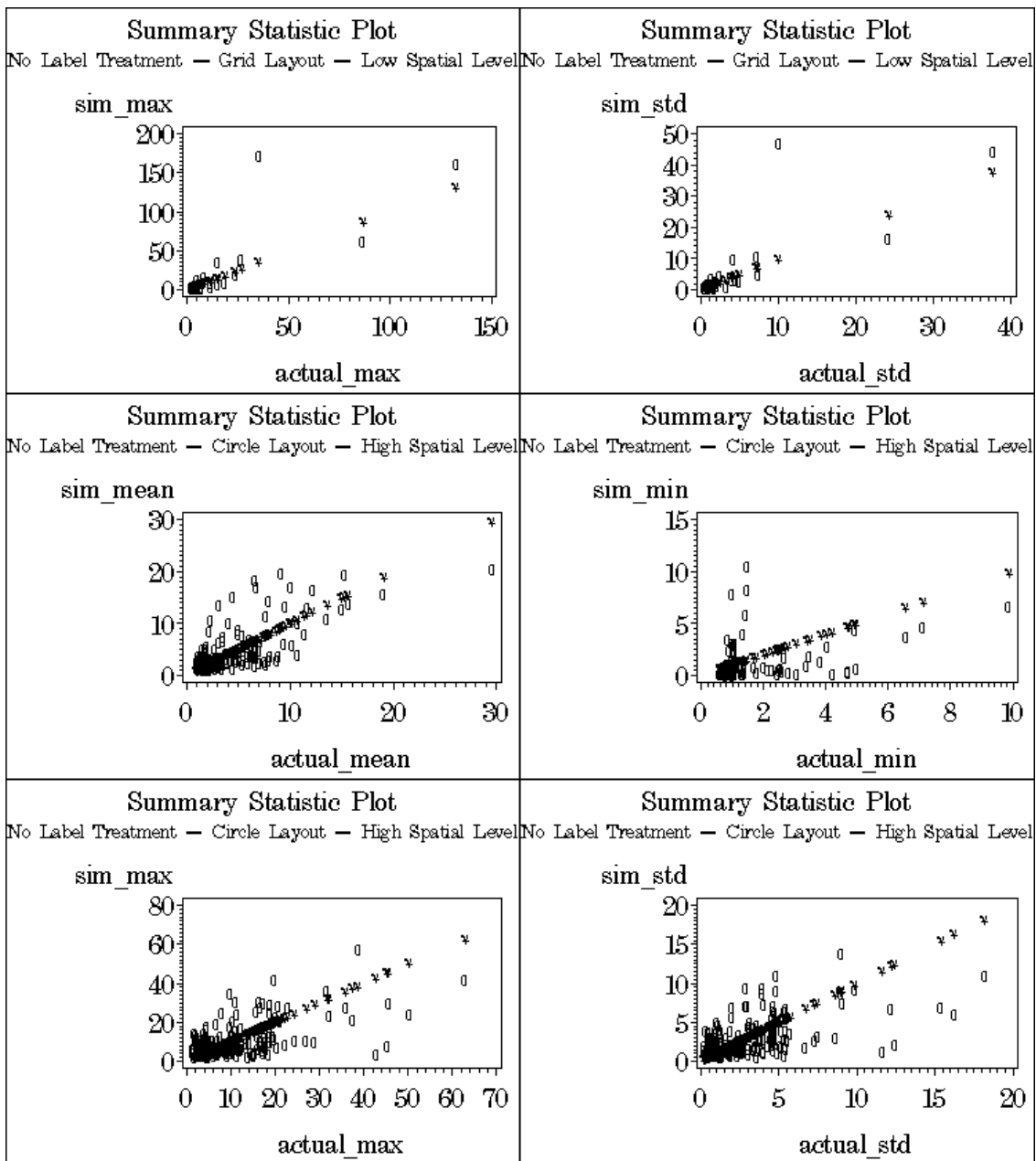


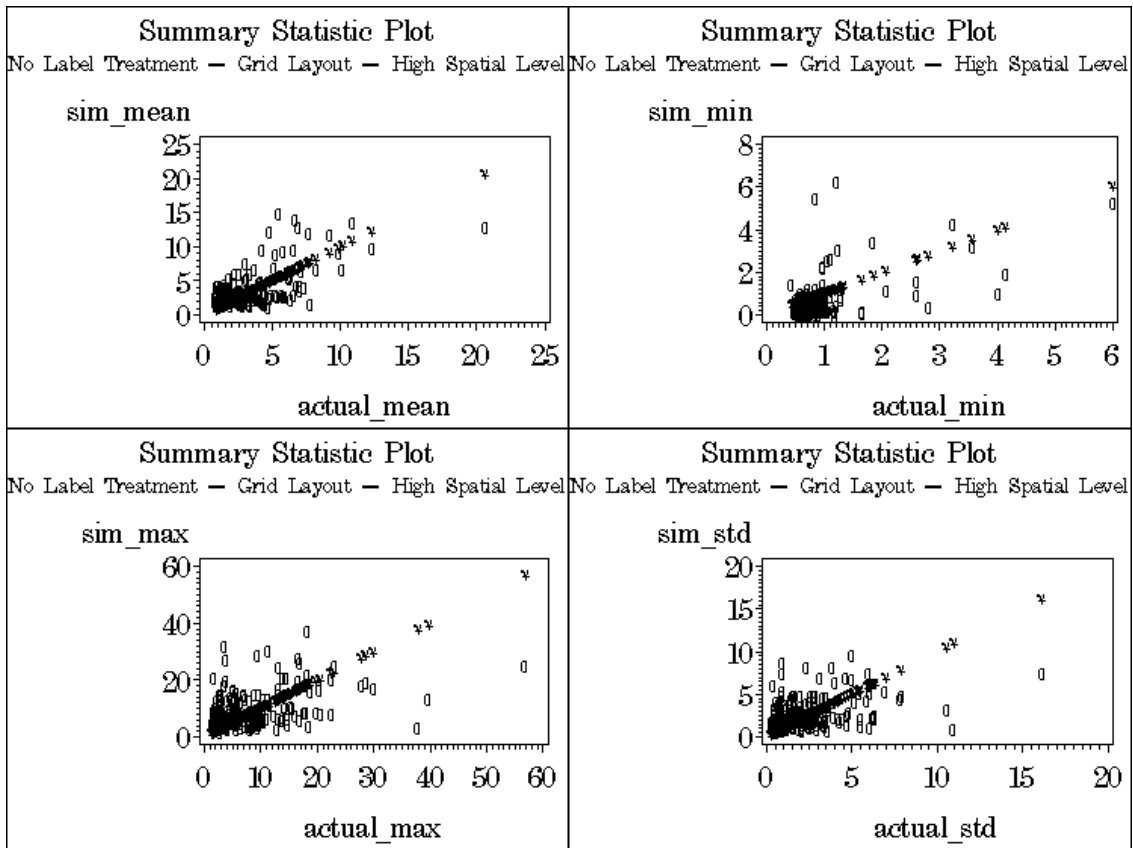






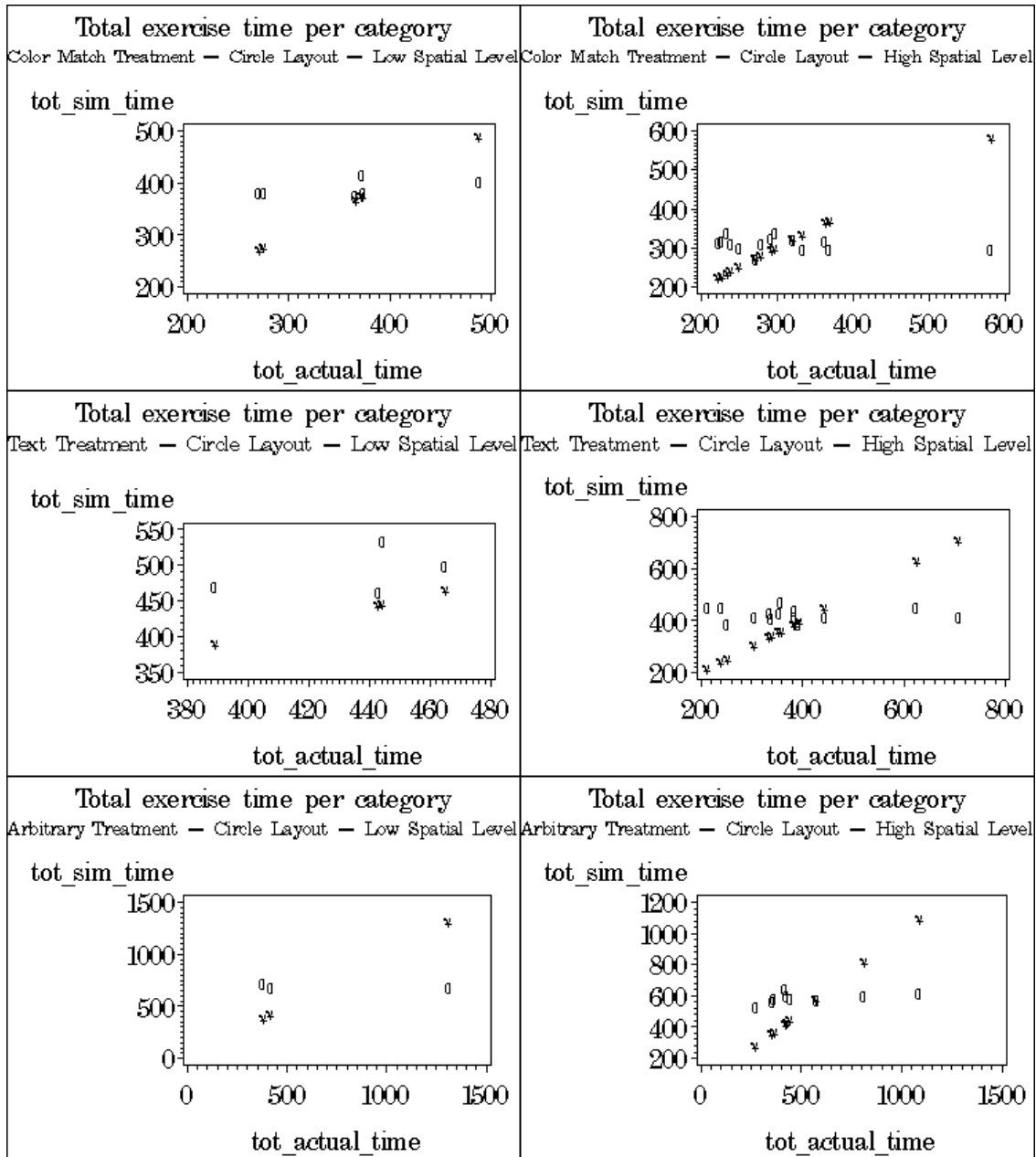


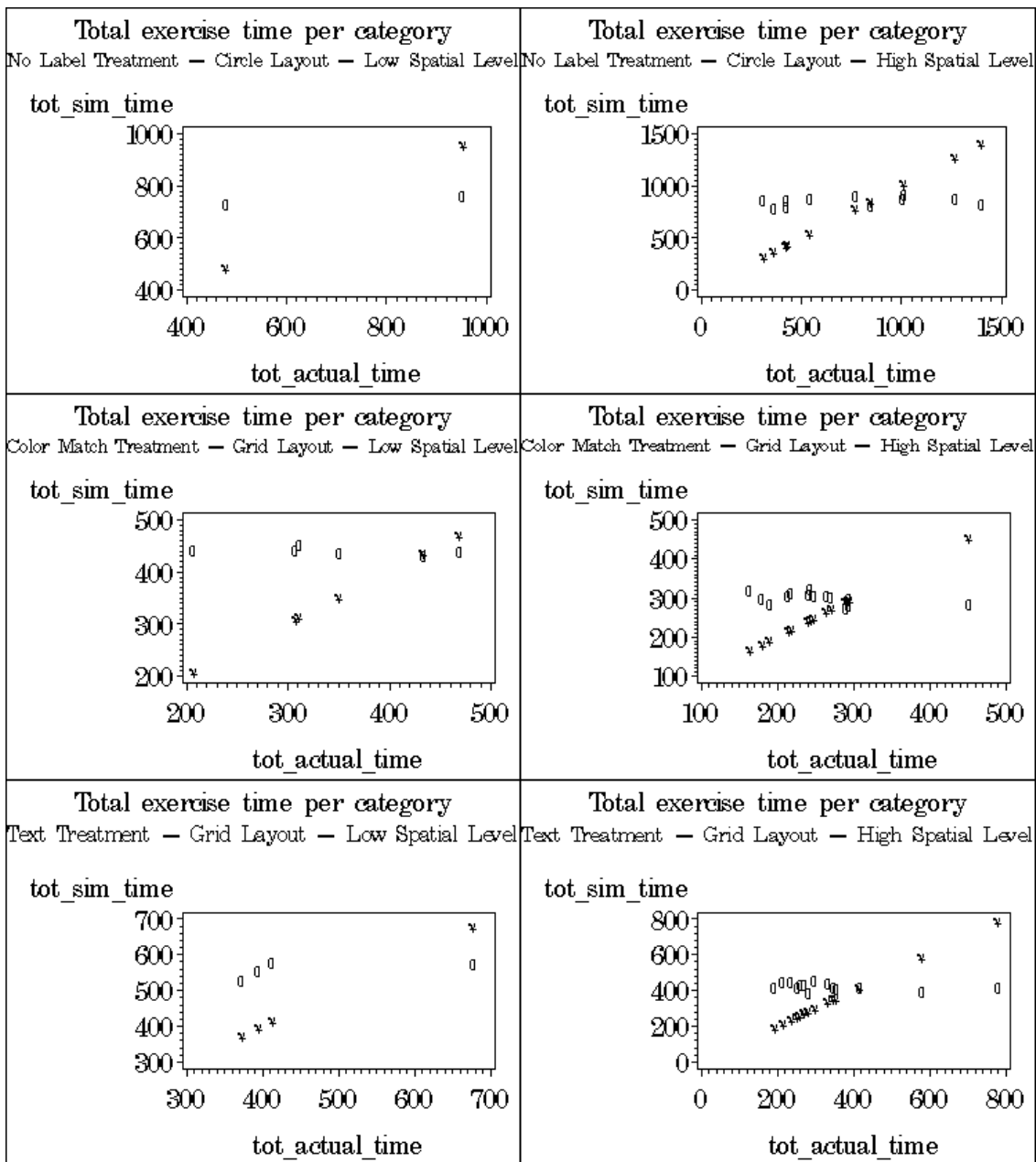


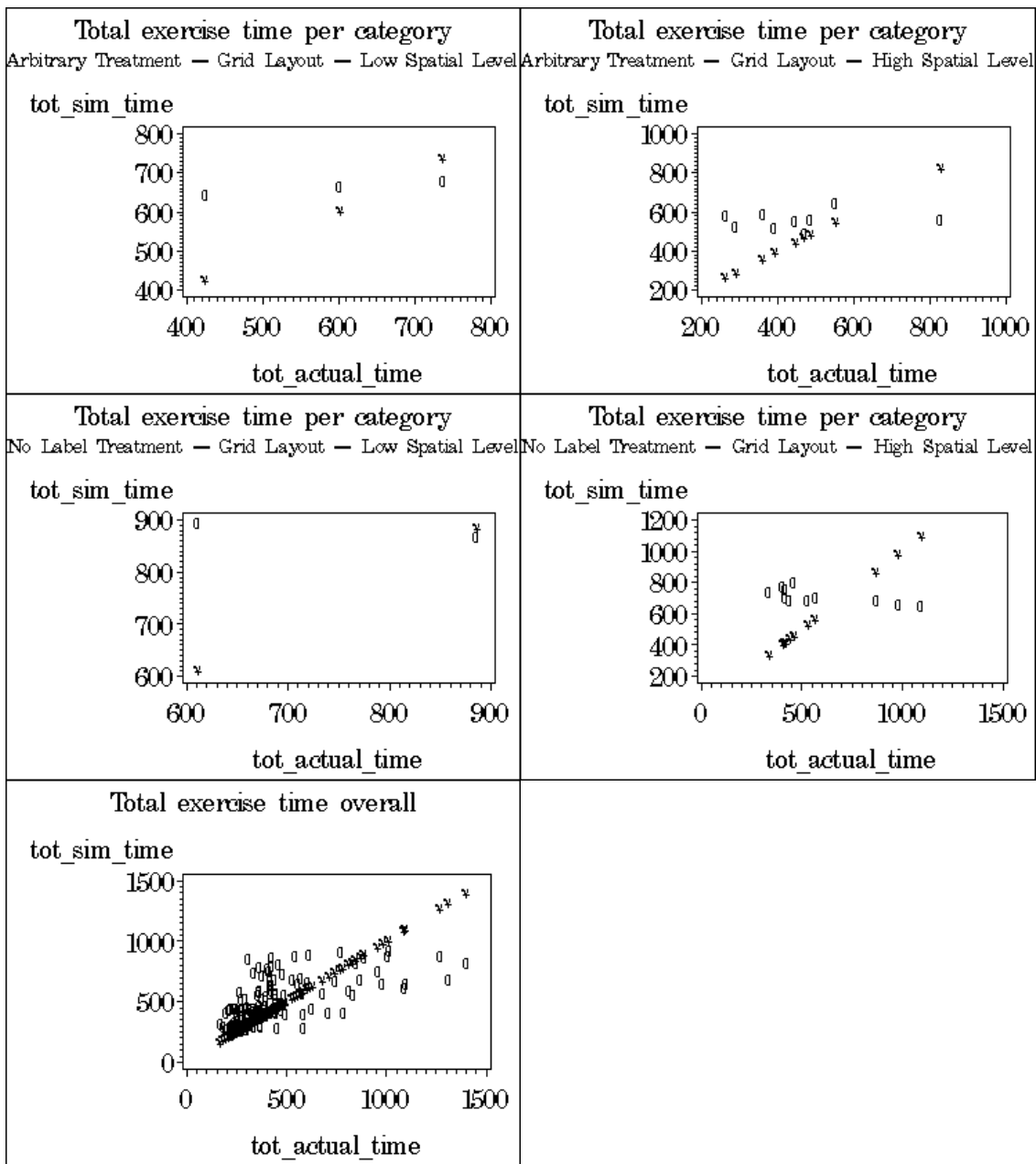


C3. Actual versus simulation exercise total time plots

Below is the plotting of exercise level performance time between actual and simulation for different spatial groups, layout and label treatments.







BIBLIOGRAPHY

- Budiu, Raluca. (07 June 2008). ACT-R: About. ACT-R Research Group Department of Psychology, Carnegie Melon University: 2002-2008. Retrieved 07 June 2008, from <http://act-r.psy.cmu.edu/about>.
- Byrne, M. D. (2003). Cognitive architecture. In J. Jacko & A. Sears (Eds.), *The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications* (pp. 97-117). Mahwah, NJ: Lawrence Erlbaum.
- Bothell, D. (2007). ACT-R 6.0 Reference Manual. Retrieved 07 June 2008, from: <http://act-r.psy.cmu.edu/actr6/reference-manual.pdf>.
- Dye, H. A. (2007). A diagrammatic reasoning: Route planning on maps with ACT-R. In proceedings of the 8th International Conference on Cognitive Modeling. Ann Arbor, Michigan, USA.
- Ehret, B. D. (2000). Learning where to look: The acquisition of location knowledge in display-based interaction. *Dissertation Abstracts International: Section B: the Sciences & Engineering*, 60(10-B), 5239.
- Ehret, B. D. (2002). Learning where to look: location learning in graphical user interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Changing Our World, Changing Ourselves* (Minneapolis, Minnesota, USA, April 20 - 25, 2002). CHI '02. ACM Press, New York, NY, 211-218.
- Ekstrom, R. B., French, J. W., Harman, H. H., & Dermen, D. (1976). *Manual for Kit of Factor-Referenced Cognitive Tests*. Princeton, NJ: Educational Testing Service.
- Gunzelmann, G. & Lyon, D. R. (in press). Mechanisms of human spatial competence. In T. Barkowsky, C. Freksa, M. Knauff, B. Kried-Bruckner, and B. Nevel (Eds.) *Lecture notes in artificial intelligence: Proceedings of Spatial Cognition (V) 2006*. Berlin, Germany: Springer-Verlag.
- Gugerty, L. & Rodes, W. (in press, publication expected fall 2007). A cognitive model of strategies for cardinal direction judgments. *Spatial Cognition and Computation*.
- Hirtle, S. & Jonides, J.: Evidence of Hierarchies in Cognitive Maps. *Memory & Cognition* 13 (1985) 208-217.
- Kitajima, Muneo. (23 October 2001). The LICAI Model. Retrieved 07 June 2008, from [http://staff.aist.go.jp/kitajima.muneo/English/Project\(E\)/CognitiveModeling\(E\)/LICAI\(E\)/LICAI\(E\).html](http://staff.aist.go.jp/kitajima.muneo/English/Project(E)/CognitiveModeling(E)/LICAI(E)/LICAI(E).html) .

- Kitajima, Muneo. (28 January 2002). CoLiDes Model. Retrieved 07 June 2008, from <http://staff.aist.go.jp/kitajima.muneo/CognitiveModeling/WebNavigationDemo/CoLiDeSTopPage.html>.
- D. Kimura, Sex and Cognition, First Edition, MIT Press, Cambridge, Mass, 1999.
- Lebiere, Christian. The dynamics of Cognition: An Act-r Model of Cognitive Arithmetic. Doctoral dissertation, Computer Science Department, Carnegie Mellon University: 1998.
- Maki, R. H. (1981). Categorization and distance effects with spatial linear orders. *Journal of Experimental Psychology. Human Learning and Memory*, 7, 15-32.
- Newell, A. (1990). Unified theories of cognition. Cambridge, MA: Harvard University Press.
- Rusch, M. L. Relationships Between User Performance and Spatial Ability in Using Map-Based Software on Mobile Devices. Doctoral dissertation, Computer Science Department, Iowa State University: in press.
- Salvucci, D. D. (2001b). Predicting the effects of in-car interface use on driver performance: An integrated model approach. *International Journal of Human-Computer Studies*, 55, 85-107.
- SAS.com. (30 May 2008). Statistical Software | SAS. Retrieved 30 May 2008, from <http://www.sas.com/technologies/analytics/statistics/index.html>.
- Sitemaker.umich.edu. (n.d.). Soar: Home. Retrieved 07 June 2008, from <http://sitemaker.umich.edu/soar/home>.
- Shelton, A. & Mcnamara, T.: Systems of Spatial Reference in Human Memory. *Cognitive Psychology* 43 (2001) 274-310.
- Stevens, A., & Coupe, P. (1978). Distortions in judged spatial relations. *Cognitive Psychology*, 10, 422-437.
- Umich.edu. 12 June 2002. Brain, Cognition, and Action Laboratory. Retrieved 07 June 2008, from <http://www.umich.edu/~bcalab/epic.html>.
- Wilton, R. (1979). Knowledge of spatial relations: The specification of the information used in making inferences. *Quarterly Journal of Experimental Psychology*, 31, 133-146.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Leslie Miller and Dr. Sarah Nusser for their guidance, patience and support throughout this research and the writing of this thesis. Second, I would also like to thank my committee members for their efforts and contributions to this work: Dr. Sarah Nusser and Dr. Simanta Mitra. I would additionally like to thank Michelle Rusch for her guidance throughout the stages of my graduate career and her motivations to always help me. I would like to thank Jason Legg to provide me with some statistics knowledge.