DEVELOPMENT, TEST, AND EVALUATION OF AUTONOMOUS UNMANNED AERIAL SYSTEMS IN A SIMULATED WIDE AREA SEARCH SCENARIO: AN IMPLEMENTATION OF THE AUTONOMOUS SYSTEMS REFERENCE ARCHITECTURE

THESIS

Katherine E. Cheney
2d Lt, USAF

David D. King
2d Lt, USAF

AFIT-ENV-MS-20-M-220

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENV-MS-20-M-220

DEVELOPMENT, TEST, AND EVALUATION OF AUTONOMOUS UNMANNED AERIAL SYSTEMS IN A SIMULATED WIDE AREA SEARCH SCENARIO: AN IMPLEMENTATION OF THE AUTONOMOUS SYSTEMS REFERENCE ARCHITECTURE

THESIS

Presented to the Faculty

Department of Systems Engineering and Management

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Systems Engineering

Katherine E. Cheney
2d Lt, USAF

David D. King
2d Lt, USAF

March 2020

AFIT-ENV-MS-20-M-220

DEVELOPMENT, TEST, AND EVALUATION OF AUTONOMOUS UNMANNED AERIAL SYSTEMS IN A SIMULATED WIDE AREA SEARCH SCENARIO: AN IMPLEMENTATION OF THE AUTONOMOUS SYSTEMS REFERENCE ARCHITECTURE

THESIS

Katherine E. Cheney
2d Lt, USAF

David D. King
2d Lt, USAF

Committee Membership:

David R. Jacques, Ph.D.
Chair

John M. Colombi, Ph.D.
Member

Trevor J. Bihl, Ph.D.
Member

Mr Jeremy P. Gray, MS.
Member

Abstract

The implementation and testing of autonomous and cooperative unmanned systems is challenging due to the inherent design complexity, infinite test spaces, and lack of autonomy specific measures. These challenges are limiting the USAFs ability to deploy and take advantage of tactical and strategic advantages offered by these systems. This research instantiates an Autonomous System Reference Architecture (ASRA) on a Wide Area Search (WAS) scenario as a test bed for rapid prototyping and evaluation of autonomous and cooperative systems. This research aims to provide a framework to evaluate the systems ability to achieve mission and autonomy objectives, develop reusable autonomous behaviors, and develop reusable cooperative decision making algorithms. For this research and application to the WAS mission, metrics of autonomy were derived from literature requirements for autonomous systems implementing reactive architectures and control: responsiveness, robustness, and perception accuracy. Autonomous behaviors, to include more complex behaviors combining simple (atomic) behaviors were developed, and a variety of cooperative decision rules were defined. The subsequent evaluation implemented a face centered cubic design of experiments over four scenarios including a single vehicle, and three levels of cooperation between two vehicles. Following a rigorous test plan, the tests were conducted in simulation implementing automated testing and expedited analysis. The test results were used to create a response surface model to characterize the system and conduct multiple response optimization to determine an optimal configuration that maximizes area searched, percent detected, and perception accuracy in a given target density.

Acknowledgements

We would like to express our sincerest appreciation to our committee members for their guidance through this process. Dr. Jacques and Dr. Colombi, we thank you for your expertise and constant encouragement you each provided. All that we have accomplished together has only been made possible by your perceptive decision to pair us on this project and we thank you for the experience. Mr. Jeremy Gray, your previous work building ASRA laid the foundation for this thesis; we thank you for your willingness to teach and help us through our many technical problems throughout this process. We have learned so much from this experience and were able to create something larger than any individual effort due to the committee's commitment to our success.

Katie Cheney and Dave King

Table of Contents

Page

Page

List of Figures

List of Tables

Table Page

List of Abbreviations

Abbreviation		Page
DoD	Department of Defense	1
SUAS	Small Unmanned Aerial System	1
OSD	Office of the Secretary of Defense	3
STAT COE	Scientific Test and Analysis Techniques in Test & Evaluation Center of Excellence	3
ASRA	Autonomous Systems Reference Architecture	6
ANT	Autonomy and Navigation Technology Center	6
AFIT	Air Force Institute of Technology	6
WAS	wide area search	6
HAMR	Hybrid Architecture for Multiple Robots	17
UBF	Unified Behavior Framework	18
DBHG	Dynamic Behavior Hierarchy Generation	20
ASRA	Autonomous System Reference Architecture	21
MBSE	model based systems engineering	22
LCM	Lightweight Communication and Marshaling	23
SITL	Software in the Loop	23
ANOVA	analysis of variance	28
RSM	response surface methodology	33
DoE	design of experiments	35
GSD	Ground Sample Distance	43
RTL	return to launch	44
OP	objective plan	46
SITL	Software in the Loop	47

Abbreviation		Page
FOV	field of view	58
FCCD	face center central composite design	59

DEVELOPMENT, TEST, AND EVALUATION OF AUTONOMOUS UNMANNED AERIAL SYSTEMS IN A SIMULATED WIDE AREA SEARCH SCENARIO: AN IMPLEMENTATION OF THE AUTONOMOUS SYSTEMS REFERENCE ARCHITECTURE

I. Introduction

1.1 General Issue

The value of autonomous systems stems from the ability to extend and complement human ability. Autonomous systems can help limit human exposure to life threatening environments as well as reduce the cognitive load on operators. These systems have been implemented in aerial, ground, maritime, and space systems and have proven valuable in Department of Defense (DoD) operations, saving lives and extending human capabilities (Defense Science Board 2012, Zacharias 2019).

A major technology that has spread rapidly in both the consumer and defense industries is small unmanned aircraft with basic autonomous capabilities. Small Unmanned Aerial System (SUAS) have been implemented in areas such as surveillance, agriculture, photography, and consumer hobbies. These systems often have basic automated features such as failsafe modes, waypoint following, auto land and take-off, and a ground control station interface. With development, these features can be expanded to more advanced functions such as target detection, identification, and tracking, decision making, data collection and analysis, and vehicle cooperation. SUAS have the capability to perform many military missions including reconnaissance, search and rescue, damage assessment, surveillance, command and control,

and assisting manned aircraft missions. Furthermore, capabilities can be combined through a network of SUAS. Distributing capabilities allows for a more robust and resilient military solution because compromising one UAS does not eliminate the total capability. Additionally, due to the smaller size and lower cost, SUAS are more attritable than traditional aircraft. This, in addition to the lack of a human pilot onboard, means SUAS can be sent to high risk areas and decrease the inherent danger of many military operations. These benefits and uses of SUAS present great potential to combine UAS and autonomy.

There is a great potential in extending SUAS autonomy. SUAS with higher levels of autonomy require a much lower level of operator input, allowing an expansion of human capability and multitasking levels. By combining these technologies, a host of new applications become available. However, the merging of these technologies brings a host of new concerns regarding the uncertainties of autonomous behavior.

A common concern with the growing complexity of autonomous systems is the lack of trust these systems naturally invoke in humans. Zacharias (2019) gives two major reasons for this lack of trust. Fist, humans trust when they know they have a common understanding how the autonomy works and how to interface with the system. This is difficult to establish due to the fundamental difference of operation between autonomous systems and humans. The second factor is the degree to which the reasoning and actions of those systems are obvious and predictable to the human. This predictability becomes more difficult as the autonomy grows in complexity, due to the increased probability of unintended or unknown behaviors emerging. This further complicates the explanation of the autonomy's behavior, affecting the level of trust it harbors from humans. The challenge of developing trust between users and autonomous systems can be addressed through a testing approach tailored to autonomous systems (Zacharias, 2019). Autonomous applications implemented re-

sponsibly and appropriately can address many of these unknowns, but these concerns should not halt autonomy development as doing so would establish a military capability deficiency compared to our adversaries.

Despite these concerns, autonomous systems have been fielded for military use to gain a strategic advantage over adversaries (Defense Science Board, 2012). Following fielding, multiple publications from DoD leadership have been released (AFRL, 2014), (Defense Science Board, 2012),(Ahner and Parson, 2016), identifying steps forward in the development of autonomous technology. One recent Air Force report (AFRL, 2014) discusses the unique challenges of testing autonomy. Since autonomous systems react to environmental stimuli, there are near infinite decision spaces. These systems are implemented in an unpredictable world with system faults and failures, human error, weather effects with humans that have varying intentions, especially in war zones. As a result, there are an infinite number of environments a system can be subjected to.

Testing all possible states and all ranges of inputs to the system is infeasible, making autonomous systems a challenge (AFRL, 2014). As a result of this challenge, the Defense Science Board task force report (2012) calls for test "techniques that focus on the unique challenges of autonomy." Areas of interest include robust simulation to capture test environments and methods to confirm autonomous systems perform as intended (Defense Science Board, 2012). In response, many studies and reports have been conducted that identified gaps and challenges in testing autonomous systems. In 2015, the Office of the Secretary of Defense (OSD) Scientific Test and Analysis Techniques in Test & Evaluation Center of Excellence (STAT COE) hosted a study on testing of autonomous systems and released a report with research areas for the DoD (Ahner and Parson, 2016). The challenge areas identified include:

- Requirements and measures

- Test infrastructure and personnel
- Design for test
- Test adequacy and integration
- Testing continuum
- Safety and cyber security for autonomous systems
- Testing of human system teaming
- Post acceptance testing

Overcoming these challenge areas is crucial to the future development of autonomous systems. The *2018 National Defense Strategy* identifies advanced autonomous systems as one of their key investments: "The Department will invest broadly in military application of autonomy, artificial intelligence, machine learning, including rapid application of commercial breakthroughs, to gain competitive military advantages" (Mattis, 2018). However, "extensive verification, validation, test, and evaluation are required before fielding autonomous weapon systems" (David and Nielsen, 2016). As a result, research efforts to develop requirements and measures for autonomous systems is a first step to gaining a competitive military advantage through autonomy.

The DoD is not the only stakeholders in testing autonomous systems. Commercial applications of autonomous systems also require rigorous testing and are facing challenges in this area. One example is the production of driver assist technologies and driverless cars. The RAND Cooperation released a report analyzing how many driving miles it would require to demonstrate autonomous vehicle reliability, highlighting the testing challenge (Karla and Paddock, 2016). The report states, "Autonomous vehicles would have to be driven hundreds of millions of miles and sometimes hundreds

of billions of miles to demonstrate their reliability in terms of fatalities and injuries." This statement points to the complex decision spaces autonomous systems are subjected to which make testing autonomy a challenge. Testing these spaces fully would take tens and sometimes hundreds of years to accomplish. As a result, "developers of this technology and third-party testers will need to develop innovative methods of demonstrating safety and reliability" in order to test adequately and affordably (Karla and Paddock, 2016).

In addition to test, directives about the development of autonomous systems have been released. Autonomous systems share common behaviors regardless of the underlying technical application. Zacharias (2019) outlines some common behaviors for all autonomous systems. As a result of these common behaviors, architectural approaches can be implemented to combine efforts across domains. In his recommendations, he mentions the need for one or more common autonomous system architectures that combine frameworks used across autonomy communities. These architectures should be fully functional, allowing users to extend the capabilities for one application and reuse them for later projects. Zacharias also discusses a useful development process for autonomous systems. These processes should support "innovation, rapid prototyping, and iterative requirement development to support rapid [Autonomous System] development and fielding." However, the software burden of autonomy presents a challenge to rapidly prototype secure autonomous systems (Zacharias, 2019). To account for this, the commonality of behaviors across domains of autonomy can be leveraged to decrease the amount of development time required.

To achieve the benefits and overcome the challenges above, the streamlining of the autonomous system development process must be achieved. Evolutionary development and test of autonomous systems can be accelerated through a modular development framework. This reusable approach can minimize rework between ap-

plications through the sharing of software components. This research evaluates a framework for this development and includes an exploration of relevant autonomy testing methods and metrics.

1.2 Scope

This research was primarily an implementation of the Autonomous Systems Reference Architecture (ASRA) developed by the Autonomy and Navigation Technology Center (ANT) Center at the Air Force Institute of Technology (AFIT). The reference architecture offers a flexible platform that enables autonomy researchers to rapidly step through evolutionary autonomy development. A large effort of this research focused on implementing the framework and building up new software components to add to the module library. The build up of the software component library in the framework allows for module reuse which is crucial to rapid evolutionary development.

ASRA was studied by implementing a wide area search (WAS) problem similar to what is presented in Decker and Jacques (2007). This WAS mission served as the test bed to evaluate the process of using ASRA for new research problems. The WAS model is based on a distribution of stationary real and false targets utilizing a probability draw to simulate sensor behavior. A confusion matrix between encountered and detected targets determines Type I and Type II sensor error which can be fine tuned to accurately simulate real world sensor performance (Decker and Jacques, 2007).

Multiple agents were implemented in order to present significant autonomy complexity and a relevant WAS scenario. This research explores variations in the level of cooperation among small multi-rotor UAS to study their effect on mission performance. A major driver of cooperative behavior are rules based on decision algorithms similar to those studied in Gillen (2003). These rules take into account various agent

and environmental factors to arrive at a decision governing the agent's cooperative behavior.

At this point, it is important to define autonomy and distinguish the difference with automation. Bihl et al. (2018) describes automation as a system that "functions with little or no human operator involvement; however, the system performance is limited to the specific actions it has been designed to do." In contrast, an autonomous system "has a set of intelligence-based capabilities that allow it to respond to situations that were not preprogrammed or anticipated in the design...[and] has a degree of self-directed behavior" (Bihl et al., 2018). However, to the layman, a system with automation is often associated with automated manufacturing instead of systems that have the ability to cooperate and weigh possible actions. Under the definitions listed above, both are considered automation. This reality indicates some sort of spectrum within automation that eventually approaches autonomy at some contested point. In light of the automation spectrum and the multi-disciplinary approach of systems engineering, a broader definition is given that aligns with the common connotation of autonomy.

In this research, autonomy is defined as: *the ability to make decisions using sensory information without human interaction*, adapted from MahmoudZadeh et al. (2019).

This definition was chosen to enable rapid prototyping of reusable behaviors and to gain access to information required for testing. The work in this research is extendable to artificial intelligence and machine learning. ASRA can be extended to include machine learning algorithms and test methods can be further developed to accommodate these systems. However, implementing automation according to Bihl

et al. (2018) allows observability and explainability of the system while developing test methods and metrics for autonomous systems. This provides a starting point for metrics and methods that can be expanded to the autonomous systems described in (Bihl et al., 2018).

In order to test the performance of the autonomy, the reference architecture required new development of software modules tailored to evaluating autonomy. In this research, autonomy is defined as the ability of an agent to make decisions according to predefined decision rules. To make these decisions, an agent perceives the world around it and uses this information to apply the rules. This research utilized simulation to vary sensor, agent, and environmental parameters over which to evaluate the ability of the agent to correctly implement decision rules. Simulation allows data to be collected over many conditions in a short amount of time and can track truth and agent perception and decisions to produce measures of autonomy and effectiveness in simulation. Test methods of autonomy is a new module in ASRA, allowing reuse and modularity.

1.3 Research Objectives and Questions

The research objectives are:

1. Further define and prototype the Autonomous Systems Reference Architecture (ASRA).

2. Develop test methods and metrics for autonomous systems.

The research questions are:

1. What additions to ASRA need to be made to implement a new WAS application?

2. How does ASRA enable reuse of the similarities that exist in autonomous and cooperative systems?

3. How does ASRA support the variations of autonomous and cooperative systems?

4. What are effective and efficient test methods for autonomy?

5. How should the test space be limited given a specific mission space?

6. What are valid and useful measures of autonomous systems?

1.4 Assumptions and Limitations

- A multi-rotor platform will be utilized for the Wide Area Search mission because of its simple flight dynamics and control.
- The WAS scenario will be multiple vehicle with multiple targets with no prior knowledge of target location.
- Inter agent communication is nominal.
- Targets are static and uniformly distributed throughout the search area
- Sensor performance can be accurately modeled with a confusion matrix.
- Vehicle operation will be nominal with the exception of a return to launch state driven by a low fuel condition.
- Errors are independently and identically distributed.
- Canonical analysis is not required.
- Optimization of response is not interested in tuning of weights or desirabilities.

- A sensor failure causing a return to launch command can be modeled with a low fuel command.
- Responsiveness to objective plan is approximately that to external stimuli.
- No decision is made on whether Type I error or Type II error is more preferable, each are weighed equally.
- The sparsity of effects principle can be applied.

1.5 Preview

Chapter I presented the general issue, listed the research goals, provided the scope and general approach of this research, and listed assumptions made. Chapter II provides a background on the WAS problem, details the use of cooperative control, discusses autonomy architectures, and presents existing research on verification and validation of autonomous systems. Chapter III introduces the specific WAS scenario and ASRA implemented in this research as well as an overview of the test design and chosen metrics. The implementation of a WAS agent in ASRA and the selected testing measures are given in Chapter IV. Finally, Chapter V gives conclusions and insights on the reference architecture development and test results. This research extends ASRA to provide an development environment to expedite future autonomous system research as well as test measures and metrics. Statistical models are created to predict system performance, including autonomous performance. These responses can be optimized for a given mission environment, demonstrating statistical models as tools for requirements prioritization, optimal vehicle configuration, and simulated model extrapolation.

II. Literature Review

2.1 Overview

The literature review details information needed to understand this research. Information on the wide area search scenario, cooperative autonomous control, autonomy architectures and the unified behavior framework are given. Previous work on the autonomous system reference architecture expanded in this research is shown and discussed. The knowledge needed to understand the test methods, models, and optimization are detailed, including mathematics and assumptions. The literature base for metrics developed in this research are over viewed. This chapter should be referenced to guide understanding of the prerequisite topics used in this presentation.

2.2 Wide Area Search Scenario

One of the many applications of UAS is wide area search and detection. With applications in search and rescue, target surveillance, and attack, these scenarios require efficient search of a large area to detect and identify targets in an unknown environment. The mundane task of flying search patterns is well suited to the abilities of autonomous aircraft making their application in this area of research importance. Additionally, the potential for increased efficiency with multiple vehicles makes this scenario an ideal test bed for autonomous operation and cooperative control development.

The scenario presented in Jacques and Pachter (2004) equips modeled agents with an ordinance and studies the trade-offs and outcomes of agent decisions to attack or continue searching for other agents. When the agents themselves are the munitions, the decision to attack terminates the agent and has a greater effect on mission success than a scenario of only search and confirm activities.

Two metrics used to evaluate wide area search and attack scenarios are area coverage rate and false target attack rate. For WAS missions, excluding munitions, false target attack rate can be roughly equated to false target detection rate which is driven by the quality of the agent's sensor. Gillen (2003) found that false target attack rate has a major impact on mission performance. To improve this false target attack rate, additional target confirmations can be performed, either by the detecting agent or another agent, at the expense of area coverage rate.

The wide area search scenario has multiple characteristics that can be adjusted to alter the simulation. The search area can be sized to match the vehicle's range or fuel resources; and in conjunction with the number of targets and distribution type, define target density. Targets can be stationary or moving with the latter enabling the targets to evade the searching agents. There are many different search patterns agents can follow such as spiral, random walk, and lawnmower patterns. Additionally, multi-agent search pattern directions, either approaching or separating from other agents, can affect the probability of detecting targets.

The WAS problem assumes on-board sensors to detect and classify the targets. If the sensor's performance characteristics are not the focus of the research, a basic sensor can be modeled with a confusion matrix as described in Jacques and Pachter (2004) or the tree approach of Ross et al. (2019) where sensor performance can be modeled for multiple target types sensed under various conditions such as lighting or orientation. A binary confusion matrix shown in Table 1 characterizes a single sensor and target type combination but the matrix can be expanded for multiple sensor and target types. For the binary confusion matrix case, the designer defines the sensor's probability of true target recognition, P_{TR}, and probability of false target recognition, P_{FTR}. Values of 1 on the diagonal describe a perfect sensor and one minus

these values is the probability of incorrect target recognition of either a true or false target. Nominal starting values are around $P_{TR} = .9$ and $P_{FTR} = .8$.

2.3 Cooperative Autonomous Control

The WAS problem's main limitation of resources can be addressed through the application of multiple agents to the same mission area. There are varying levels of this teamwork that can be implemented as well as many ways to distribute control throughout the system of agents. Martinez (2008) describes multiple types of cooperative controllers with different levels of distributed control. A centralized control architecture receives all agent information and assigns tasks and roles to each agent. Decentralized control exists when agents share goals and information allowing agents to individually make decisions that support global utility. Decentralized control allows for flexible inter-agent operation, allowing the system to adapt to new agents and to degrade gracefully if any agents fall offline. This requires an arbitration scheme between agents as no single agent makes group level decisions, but arbitration requires increased information flow and therefore more agent communication bandwidth. Decentralized control is also more robust to communication loss because agents can continue to perform useful tasks without global information, whereas losing communication under centralized control can leave agents in an unproductive state, waiting for new tasks that never arrive.

Table 1. Binary confusion matrix describing sensor performance

		Encountered	
		True	False
Detected	True	P_{TR}	1-P_{FTR}
	False	1-P_{TR}	P_{FTR}

The effect of different levels of cooperation have been reported in Dunkel (2002) and Park (2002). While both of these works analyzed cooperation in a search and attack scenario where the agents themselves were munitions, some overall conclusions may still apply to other WAS scenarios such as search and confirm. Dunkel determined that cooperation does not always yield improved results and it must be applied strategically to improve mission success. He notes that cooperative attack can easily degrade system performance because a falsely classified target can waste resources as more agents are expended to attack a false target. Cooperative classification showed more potential because the requirement to confirm with multiple agents decreased the chances of falsely classifying the target. The benefit of cooperation was greatest when sensor quality was poor as multiple agents were still able to correctly classify targets. When the result of mis-classifying a target is losing an agent and striking a non target, cooperation's effect on decreasing false target classifications becomes very valuable. For a search and confirm mission, cooperative classification can minimize wasted resources by minimizing the chances of monitoring a mis-classified target. Park (2002) determined that the number of deployed agents must be matched to the density of the target distribution. A higher target density may be best suited for more agents but only to a point, as the efficiency of agents drops off when deploying too many agents to a relatively small area.

In Gillen (2003), a cooperative decision algorithm is presented that determines the criteria an agent uses to determine if it should engage a target. This formula was designed to encourage participation in cooperative engagement when engaging the target appeared to be an efficient use of resources, such as when the agent is low on fuel and less likely to find another target through continued search. These normalized values were then weighted, summed together, and compared to a threshold. The author found that this cooperation function did improve the probability of

target detection for low quality sensors under certain mission environment conditions but tuning the cooperation function for a wide range of mission characteristics was difficult.

Dunkel (2002) chose three levels of cooperation to test cooperation performance on, a case with no cooperation, an extreme cooperation case where any agent can attack any target, and a third case of moderated cooperative classification and attack involving multiple, independent target classifications before attacking. A wide variance in cooperation effectiveness was found among these three scenarios with the moderated cooperative case benefiting most by cooperation even with decreasing sensor performance.

2.4 Autonomy Architecture

Layered Architectures.

Autonomous system architectures have progressed since the mid-eighties from deliberate "sense, plan, act loops" to reactive architectures capable of faster execution, but lacking in higher level planning (Murphy, 2000). The hierarchical control method slowly steps through sensing the environment to build a world model, determining a plan to execute a given goal, and then executing that goal in the act phase. The method is effective at achieving its goal in near-static environments that change slowly relative to the loop execution time such as space applications. If this loop is too slow, the environment has changed after sensing is complete and the execution may no longer be appropriate. Furthermore, the plan and act stages leave the agent unaware of the current environment state.

To address these limitations, reactive architectures remove the time intensive plan phase, allowing the sense and act phases to achieve fast adaptation to a changing environment. Braitenberg (1986) studied how this can be achieved by simply linking

sensory inputs directly to motor outputs. This yields a highly specific application for a given autonomous agent; to make the agent more task flexible, multiple sensors can be linked to motors with each sensor's sense-act loop running in parallel. This enables the agent to respond to specific inputs but still lacks overall planning or state maintaining capabilities enabled by the deliberative approach.

While each of these architectures are individually limited, Gat (1998) proposes combining them into a 3 layer architecture with a deliberator, sequencer, and controller. The deliberator, a high level planner ultimately guides the controller, a low level reactive mechanism. To link these layers, the sequencer translates abstract goals into appropriate reactive controller operations to achieve the goal. Running these layers concurrently in separate threads or processes frees the deliberator to model and plan at its own pace while the controller can step through the sense-act loop at a faster rate that can react to a dynamic environment.

The deliberator layer is able to slowly read the world model to drive high level planning such as path planning algorithms or mode changes. Additionally, the deliberator considers requests from the lower, sequencer layer which can notify when a goal is completed or if the sequencer was unable to determine a set of behaviors to accomplish a goal. The deliberator can then use this information in its planning process and adapt. Because the execution speed of the deliberator does not effect low level function, Gat (1998) suggests the deliberator's logic and algorithms should not be constrained, and thus could range from simple state machines to more dynamic models that learn from the environment.

The controller can be thought of as behaviors or a set of transfer functions that each convert sensor inputs to motor outputs in a unique way. Gat (1998) points out that these behaviors should "fail cognizantly" or recognize their failure and notify the

rest of the system. This allows for a fault aware system that can react to unintended performance of the controller layer.

The sequencer is then tasked with executing the goal given by the deliberator by selecting the appropriate controller behavior, or sequence of behaviors that achieves the current goal. The sequencer monitors the controller layer to determine when goals are achieved or if the deliberator's goal cannot be achieved with the agent's available behaviors. Additionally, the sequencer supplies any parameters the behavior may need for its operation such as a goal position in the case of a waypoint achieving behavior.

While the three layer architecture enables a single agent to perform complex functions in a dynamic environment, it does not provide for multi-robot interaction. Hooper and Peterson (2009) suggest that adding a fourth layer above the deliberator to act as a coordinator between agents enables high functioning multi-agent integration. This Hybrid Architecture for Multiple Robots (HAMR) assigns the coordinator the tasks of inter-agent communication and translation of important global information down to the deliberator. The coordinator maintains the state of other agents and their impact on the world model such as agents' positions, state history, and current tasks, and communicates any important internal changes to the rest of the agents. The coordinator can then operate on this data, determining the utility of tasks, and pass that information on to the deliberator to aid in decision making in light of the overall group of agents. To address the arbitration of tasks between agents and ensure the agent with the highest utility performs the task, the author suggests all agent deliberators determine their own utility for the task and send that out to all other agent coordinators so that each agent can determine if they have the highest utility to perform that task based on what they received from their coordi-

nator. Because the coordinator is unaware of all agents' capabilities, agents cannot perform this arbitration individually.

Unified Behavior Framework.

The controller layer's behaviors that form the transfer functions from sensors to motors are often not simply a single behavior but are combinations of multiple low level behaviors. Braitenberg (1986) suggests that complex behaviors observed in natural beings are simply, or at least can be modeled by, the combination of many low level behaviors. The difficulty of designing a highly capable controller layer is the organization and combination of the many behaviors required to reach the goal behavior. The Unified Behavior Framework (UBF) presented in Woolley and Peterson (2009) aims to address these issues by standardizing behaviors through encapsulation and offering a flexible environment to adjust behavior structures during execution.

To allow for the flexible use and reuse of behaviors, their external interfaces should be standardized. The UBF specifies that the controller sends sensor information in a standardized perceived state to the behavior which then returns its output as an action output. The UBF also allows for multiple behavior control structures to be implemented, such as subsumption which allows multiple behaviors to run simultaneously, or motor schema which combines behavior output vectors into a single output motor control. Traditionally the controller was limited to a single type of behavior control architecture which may not be as appropriate for all goals the agent may have.

The UBF enables behaviors to be combined into composite behaviors. This allows for software reuse, utilizing a set of atomic leaf behaviors in any number of composite behaviors that achieve a new goal that the individual behaviors could not achieve on their own. Figure 1 shows the class diagram of UBF behaviors where composite and

leaf behaviors are two types of behaviors and composite behaviors are composed of an arbiter and two or more behaviors of any type. This composite pattern builds a behavior tree that allows leaf and composite behaviors to be handled in the same way and for requests to flow smoothly down the tree (Gamma et al., 1995). Composite behaviors must include an arbiter to determine how to combine the action outputs of its behaviors. Some types of arbiters are winner take all, vector sum, and priority fusion. These take in each behavior's action vote and weight to determine a single action output for the composite behavior.

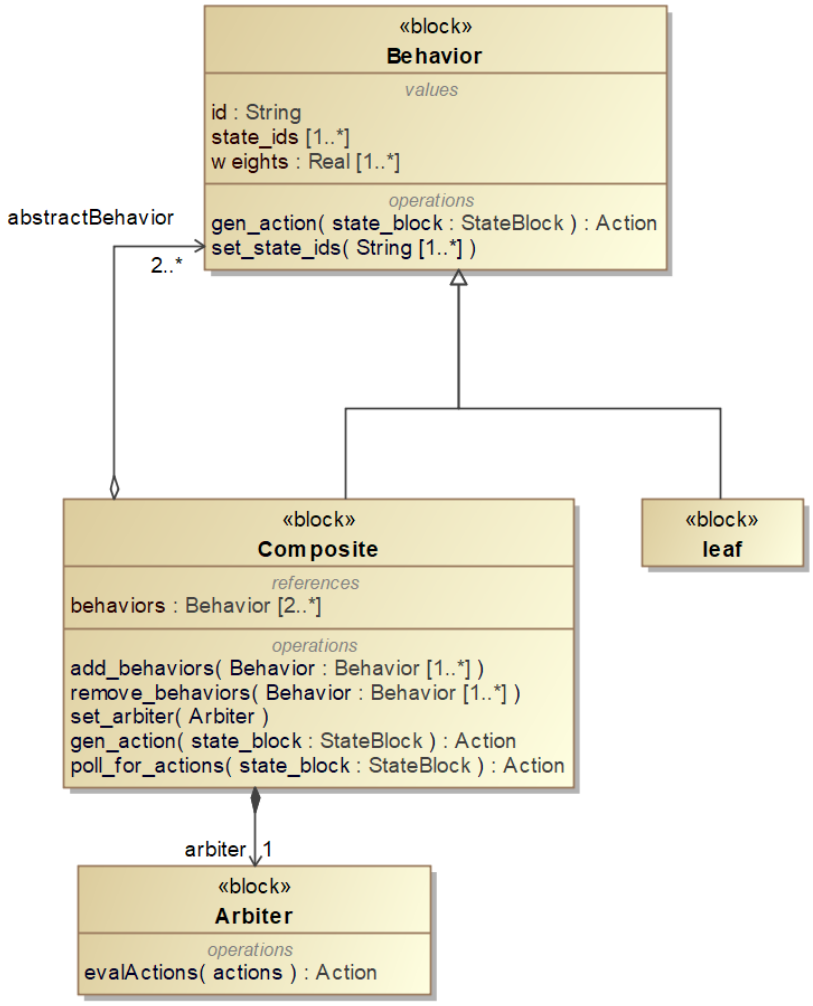


Figure 1. Class diagram of the Unified Behavior Framework showing composite and leaf types of behaviors and a composite behavior composed of an arbiter and behaviors

In the same way that the UBF standardizes behavior interfaces, Peterson et al. (2011) suggest a standard interface between the sequencer and controller that enables the sequencer to automatically build up behavior libraries instead of requiring predetermined sets of behaviors. Traditionally, the system designer must anticipate all goals and build up complex behaviors that achieve them. The sequencer then accesses this list to determine what to choose to achieve a given goal. Not only does this require extensive forethought, any new additions to the behavior library must be manually integrated into the behavior hierarchy. The Dynamic Behavior Hierarchy Generation (DBHG) attempts to standardize this link through the use of activation paths to describe leaf behaviors and enable the sequencer to build up behavior hierarchies on its own. Table 2 shows the elements that make up the activation paths.

Table 2. Description of the Activation Path which acts as a standard representation of behaviors to define the interface between sequencer and controller layers.

Characteristic	Definition
Initial Conditions	When true, generates an action vote
Post Conditions	Environment effects the behavior intends to achieve
Required Data	Sensor data required for behavior to function
Goals	Deliberator goals the behavior intends to achieve
Control Settings	Motor outputs the behavior generates
Behavior Vote	User-defined weight of the behavior when in an acting state

The sequence diagram of the DBHG is shown in Figure 2. Using activation paths as abstract representation of behaviors, the dynamic sequencer is able to translate the sequenced and prioritized goals in objective plans sent from the deliberator, which can contain any number of tasks to be completed, into an arbitrated hierarchy of behaviors to accomplish those goals. When building up a behavior hierarchy, the dynamic sequencer refers to a resource manager that monitors the agent's available resources and returns only viable behaviors to consider using. The activation path's post conditions tell the sequencer those conditions that indicate when the hierarchy has completed its current task and when the next hierarchy should be sent.

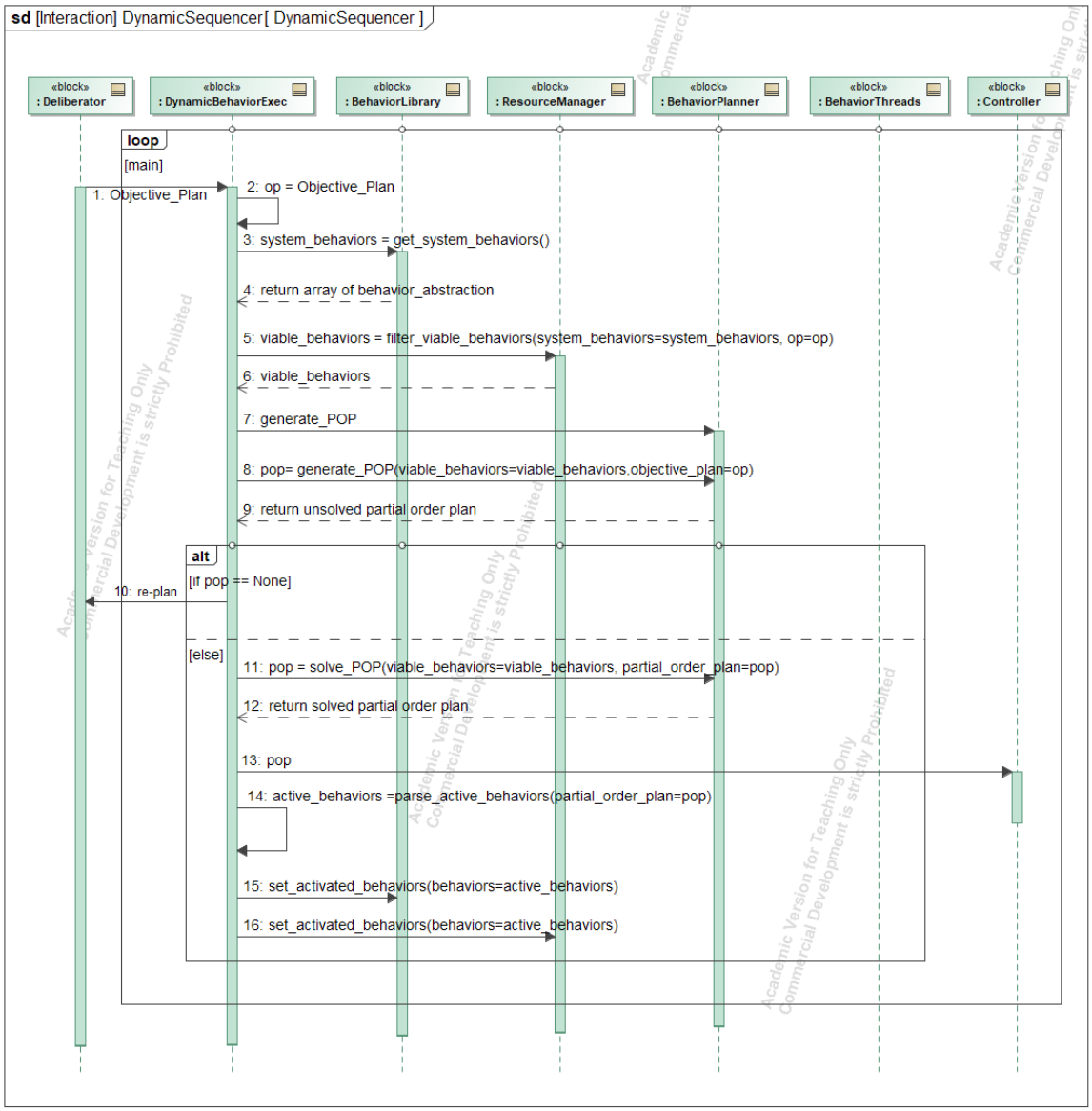


Figure 2. The sequence diagram of the Dynamic Sequencer shows task plan generation from objective plan to arbitrated behavior hierarchy.

Autonomous System Reference Architecture.

The Autonomous System Reference Architecture (ASRA), presented in Gray and Jacques (2019), aims to provide an environment for autonomy researchers to quickly spin up complex autonomous systems in a variety of domains using reusable and modular components. The reference architecture is modeled in SysML using the model

based systems engineering (MBSE) tool, Cameo System Modeler. This approach provides multiple levels of abstractions and bring out details on the architecture, interfaces, and concepts. MBSE and the Cameo tool help new users learn the architecture and experienced researchers extend and document the architecture with many different views into the system. This model provides a platform to develop implementation models which can aid in development and act as a digital twin to evaluate system performance. An ASRA implementation has been developed using the Python programming language.

The architecture thoroughly models the autonomous system in its environment as well as its interactions with other agents. Figure 3 shows a high level abstraction of an autonomous system where an agent interfaces with its environment through its action outputs, communication with other agents, and environmental precepts. This view shows how the reference architecture can model multiple agents in a system.

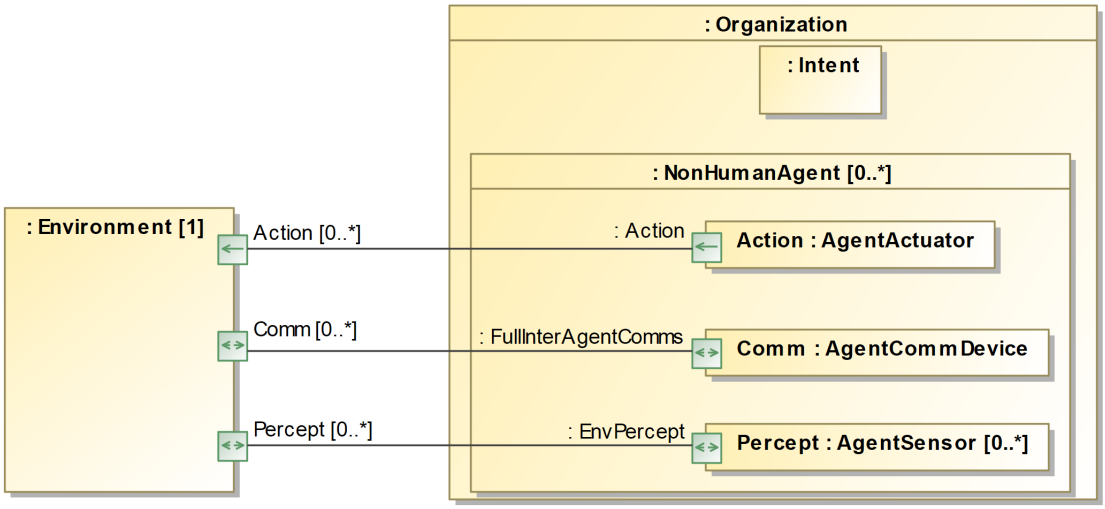


Figure 3. The agent interfaces with the environment through action outputs, a communication interface with other agents for example, and environmental precepts.

Figure 4 models the three levels of an embodied agent. The autonomy layer consists of an agent core where the autonomy architecture resides and a data marshalling

service which handles the information flow to and from the agent core. The autonomy layer interfaces with the hardware layer through the hardware interface layer. This interface layer moderates their interactions with a standardized messaging structure which allows for modularity of hardware and autonomy layers. The interface also acts as a filter, only relaying information meant for that particular agent core. This interface layer is currently implemented using Lightweight Communication and Marshaling (LCM) but other communications methods can be used as well. For the hardware layer, ASRA has existing modules to interface with the Ardupilot Software in the Loop (SITL) autopilot and a point mass simulator. By swapping out the hardware layer, researchers can go from a simulated environment to a real world environment. This is the case with SITL for example. Because SITL is a software representation of the Ardupilot autopilot in a simulated world environment, transitioning from simulation to flight testing without any major changes to their software agent. Additionally, this provides a digital twin capability, where the autonomy is run on a real and simulated agent simultaneously, with the difference being what is running in this hardware layer.

Figure 5 shows an instantiation of the agent core as the four layer HAMR architecture. This is one of many possible agent core architectures such as a simple reactive controller, or a reinforcement learning implementation. This view also shows how perceptors interface with the layers, taking in sensor information, processing it and providing state information to the rest of the agent core. These states update the agent core state block which each layer references to update its own state block, which contains only the states it needs to monitor.

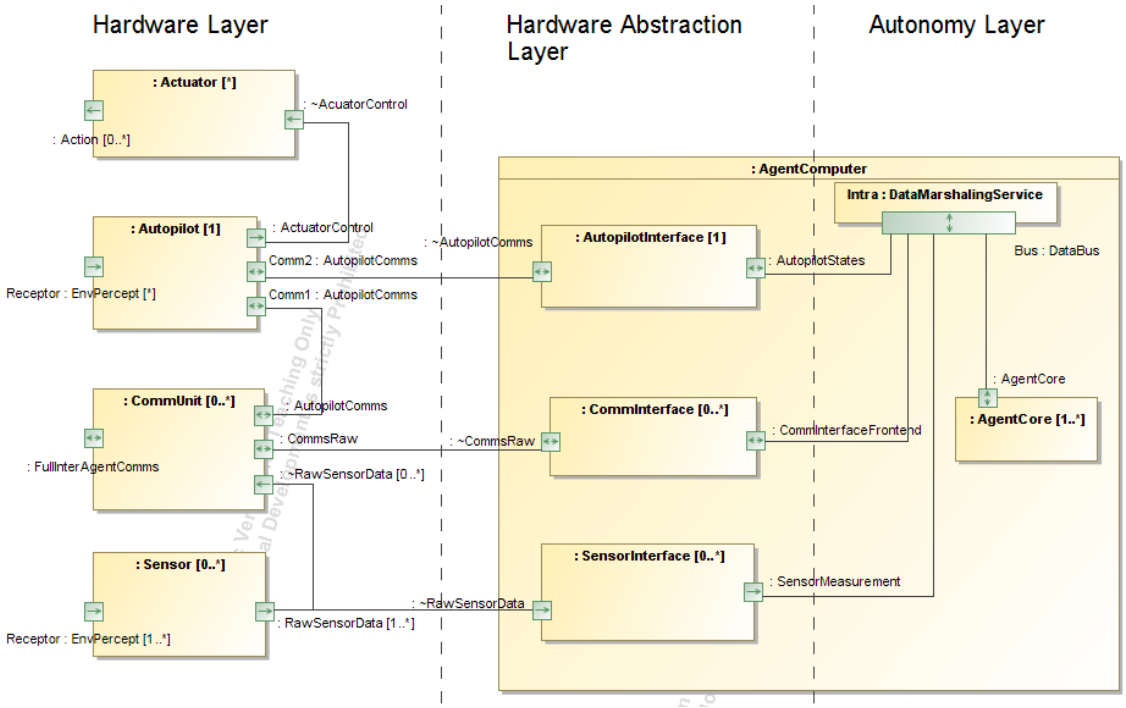


Figure 4. Model of agent architecture with the Hardware Interface Layer handling interface between Hardware and Autonomy Layers.

2.5 Statistical Models: Linear Regression

Statistical models are used to evaluate the autonomous WAS mission through experiments of the ASRA. Experiments allow researchers to observe phenomena under experimental conditions. Models are theoretical explanations of experimental observations expressed in one or more mathematical equations. These mathematical equations can be used under model assumptions to predict a response given input parameters. In test and evaluation, statistical methods are used to characterize the capability of a system with a statistical model. Statistical methods are applied to gather data in carefully designed experiments in order to asses the degree of uncertainty in results. Statistical methods fall into three categories: descriptive statistics, inferential statistics, and model building. Descriptive states allow analytic and graphical descriptions of data sets. Inferential statistics are the methods by which conclusions can be drawn about large groups from observing only a small subset of

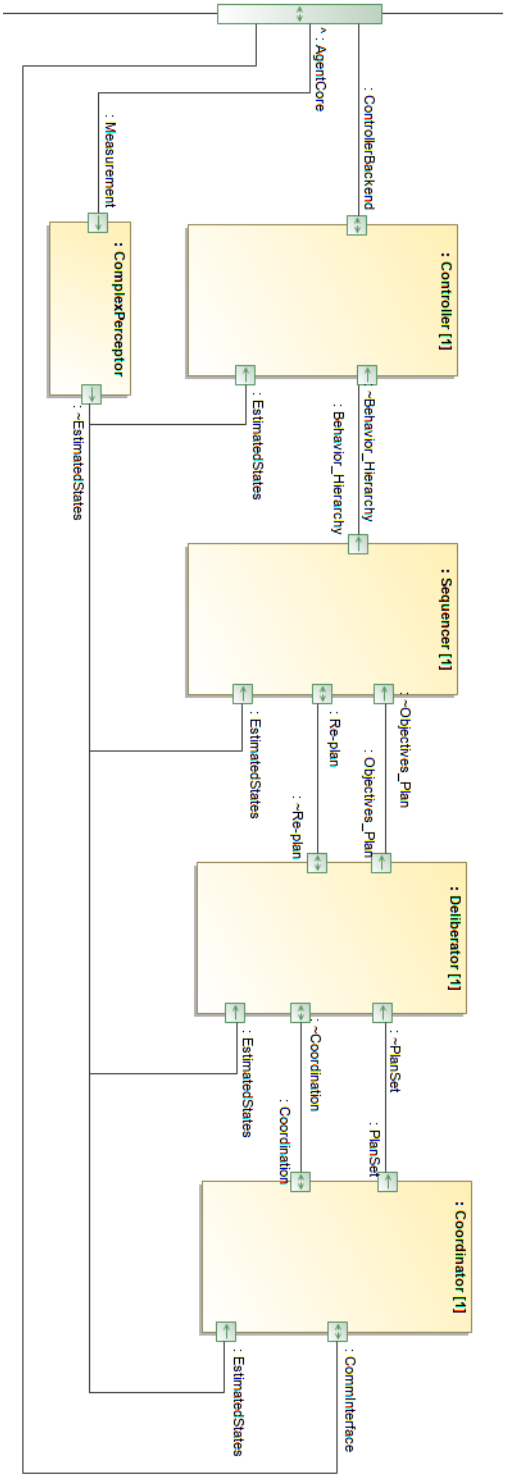


Figure 5. Model of agent core consisting of the four layer HAMR architecture and all communication routing through the data marshalling service.

the large group. In statistics, conclusions are made about a the population which include all members of a group. Only a subset of the population, referred to as a sample, is observed to draw conclusions. This concept is shown visually in Figure 6. In experiments, a sample is taken to develop prediction equations from experimental data. Prediction equations are statistical models which allow prediction of behavior from a complex system with an associated probability of error (Milton, 2003).

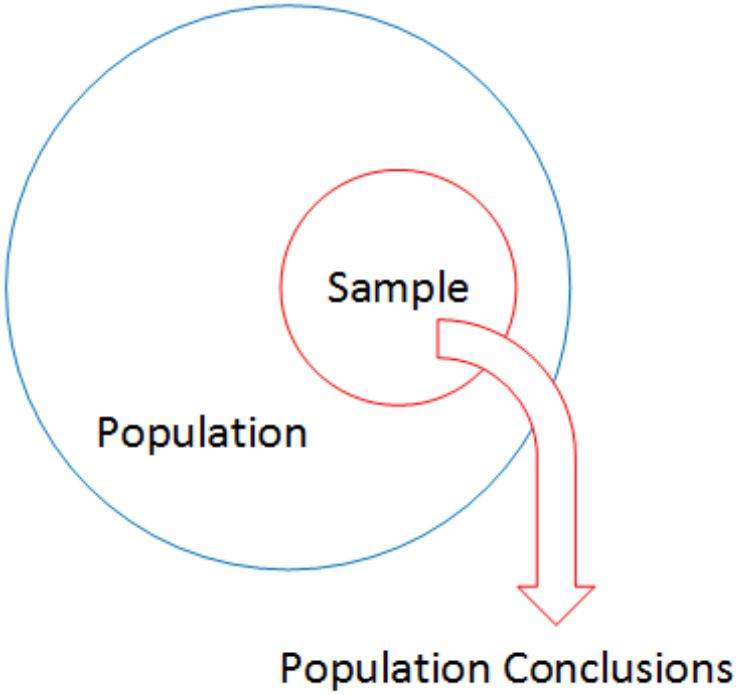


Figure 6. Graphical depiction of samples and population groups. Statistical methods allow researchers to make conclusions about a population group from the sample group, saving time and money.

In basic algebra, the equation $y = mx + b$ is used to express a linear relationship where m is the slope and b is the y-intercept. In an experiment, a response can be expressed as a linear equation with some sort of unique true error, E_i. The equation of the true relationship is given in Equation 1 where β_0 is the y-intercept and β_1 is the slope of the line.

$$Y_i = \beta_0 + \beta_1 x_1 + E_i \tag{1}$$

In simple linear regression, the true relationship in Equation 1 is modeled by Equation 2 where b_0 is the estimate of β_0, the y-intercept and b_1 is the estimate of β_1, the slope of the line, and ϵ is the estimate of true error. The response $\bar{y}$ is the mean response for input x_1.

$$\bar{y} = b_0 + b_1 x_1 + \epsilon \tag{2}$$

The method of least squares is applied to find estimates of β_0 and β_1, solving for b_0 and b_1 respectively. The least squares estimation method estimates the parameters by minimizing the square distance of the estimated error, ϵ. Using the method of least squares, the estimates for b_0 and b_1 are given in Equation 3 and Equation 4. In Equation 4, n is the total number of observations.

$$b_1 = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}) \tag{3}$$

$$b_0 = \bar{y} - b_1 x_1 \tag{4}$$

In order to test hypotheses about results, the assumptions given in Montgomery (2012) are applied:

1. The random variables Y_i and E_i are independently and normally distributed
2. Error has a mean of zero and constant but unknown variance, σ^2
3. Linearity and homoscedasticity
4. No auto–correlation or multicollinearity

These assumptions are common to both regression and analysis of variance (ANOVA). The satisfaction these assumptions allows one to assert $\epsilon \approx 0$, simplifying Equation 2 to Equation 5 where $\bar{y}$ is the mean response. Equation 5 can be expressed in matrix form, given in Equation 6. Matrices $\mathbf{X}$ and $\mathbf{b}$ are defined in Equation 7.

$$\bar{y} = b_0 + b_1 x_1 \tag{5}$$

$$\bar{y} = \mathbf{X}\mathbf{b} \tag{6}$$

$$\mathbf{X} = [1, x_1] \ , \ \mathbf{b} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \tag{7}$$

Regression analysis allows investigation and modeling of relationships between variables. The simplest type of regression is simple linear regression. A simple linear regression is an equation that predicts one response in terms of one input variable, as shown in Equation 5. The response is linear and therefore resembles the simple line equation where the intercept is b_0 and the slope is b_1. When one wishes to investigate a response with multiple independent variables, multiple linear regression models can be used in the same fashion. A multiple linear regression can be solved with a matrix approach, Equation 8.

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{21} & \dots & x_{k1} \\ 1 & x_{12} & x_{22} & \dots & x_{k2} \\ \vdots & & & & \\ 1 & x_{1n} & x_{2n} & \dots & x_{kn} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix} + \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \tag{8}$$

Equation 8 is the matrix representation of Equation 1 in multiple linear regression. Applying ANOVA assumptions simplifies the expression to Equation 9 in the form of $\bar{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{b}$.

$$\begin{bmatrix} \bar{y}_1 \\ \bar{y}_2 \\ \vdots \\ \bar{y}_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{21} & \dots & x_{k1} \\ 1 & x_{12} & x_{22} & \dots & x_{k2} \\ \vdots & & & & \\ 1 & x_{1n} & x_{2n} & \dots & x_{kn} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_k \end{bmatrix} \tag{9}$$

To find the least squares estimate for $\boldsymbol{\beta}$, $\hat{\boldsymbol{\beta}}$ is calculated using Equation 10.

$$\hat{\boldsymbol{\beta}} = \mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} \tag{10}$$

Matrix calculation of multiple linear regression is most easily done with computers. Examples of computer programs that can easily calculate regression coefficients include: MATLAB, Python, and R (Montgomery, 2012).

Once a statistical model is created, the researcher determines the statistical significance of the model. To do this, total variability is separated into its components. The total corrected sums of squares, SS_T, is obtained by Equation 11 where $\bar{y}_{...}$ is the average of all measurements for a given response variable in an experiment with a treatments and n observations of the ith treatment. The total number of response observations, N, is equal to an. A treatment is defined as a unique setting of a single factor.

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{n}(y_{ij} - y_{...})^2 \tag{11}$$

$$i = 1, 2, \dots, a \text{ treatments}$$

$j = 1, 2, \ldots, n$ observations of treatment

$N = an$ total response observations

The total sums of squares can be broken down into components of error (SS_E) and treatment (SS_{Tr}). Sums of squares error can be further broken down into components of pure error (SS_{PE}) and lack of fit (SS_{LOF}). These breakdowns are given by Equations 12 and Equation 13.

$$SS_T = SS_E + SS_{Tr} \tag{12}$$

$$SS_E = SS_{PE} + SS_{LoF} \tag{13}$$

Replicates are required to separate SS_E into its components. SS_{PE} gives an unbiased estimate of experimental error, σ^2. Without replicates, $SS_{PE} \approx SS_E$ is assumed. The SS_E is calculated by Equation 14. The SS_{Tr} is determined by simple subtraction, given in Equation 15 (Montgomery, 2017)..

$$SS_E = \sum_{i=1}^{a} \sum_{j=1}^{n} (y_{ij} - \bar{y}_{i.})^2 \tag{14}$$

$$SS_{Tr} = SS_T - SS_E \tag{15}$$

Determining statistical significance is achieved through hypothesis testing under a given decision criterion, the probability one is willing to reject a null hypothesis. To test significance of regression the following hypotheses are tested in Equation 16 and Equation 17.

$$H_0 : \beta_0 = \beta_1 = \ldots = \beta_n = 0 \tag{16}$$

$$H_1 : \text{ at least one } \beta_i \neq 0 \tag{17}$$

The hypotheses are tested using the appropriate test statistic, F_0. The test statistic, F_0 is calculated using Equation 18.

$$F_0 = \frac{SS_{Tr}/(a-1)}{SS_E/(N-a)} \tag{18}$$

To assess F_0, it is compared to the $F_{\alpha,a-1,N-a}$ where α is the decision criterion, typically 0.05. The decision criterion, α, is also referred to as Type I error and statistical level of significance. Type I error is defined as the probability of rejecting a null hypothesis (H_0) when it is actually true (false positive). It's opposite, β is called Type II error and is the probability of failing to reject the null alternative hypothesis (H_0) when it is actually false (false negative). One should reject H_0 and conclude the regression is insignificant if:

$$F_0 > F_{\alpha,a-1,N-a}$$

$F_{\alpha,a-1,N-a}$ can be determined using a F-statistic table.

Once a significant regression model is created, the analyst will need to determine model adequacy by checking the assumptions made in ANOVA. The response variables should be roughly normal with a single peak and decaying tails. This can be checked by creating a distribution of responses. Next, studentized residuals are plotted against row. Desirable studentized residuals are ones that reside in a horizontal band with no apparent correlations, shown in Figure 7a. If no pattern appears, this indicates that the errors are independently distributed. Examples of acceptable and problematic studentized residuals are shown in Figure 7. If variance of observations increases with observation, this is evidence of nonconstant variance, a violation

of ANOVA. Checking model adequacy is an important step in creating a statistical model so that the conclusions drawn from the model are valid.

Statistical models with polynomials and interactions are also used with regression and response surface methodology. The sample principles are applied in these cases. Interactions and higher order terms are included if there is evidence of significance or need. An analyst would include interactions if there is a desire to investigate the significance of factor interactions. Higher order responses are used if there is

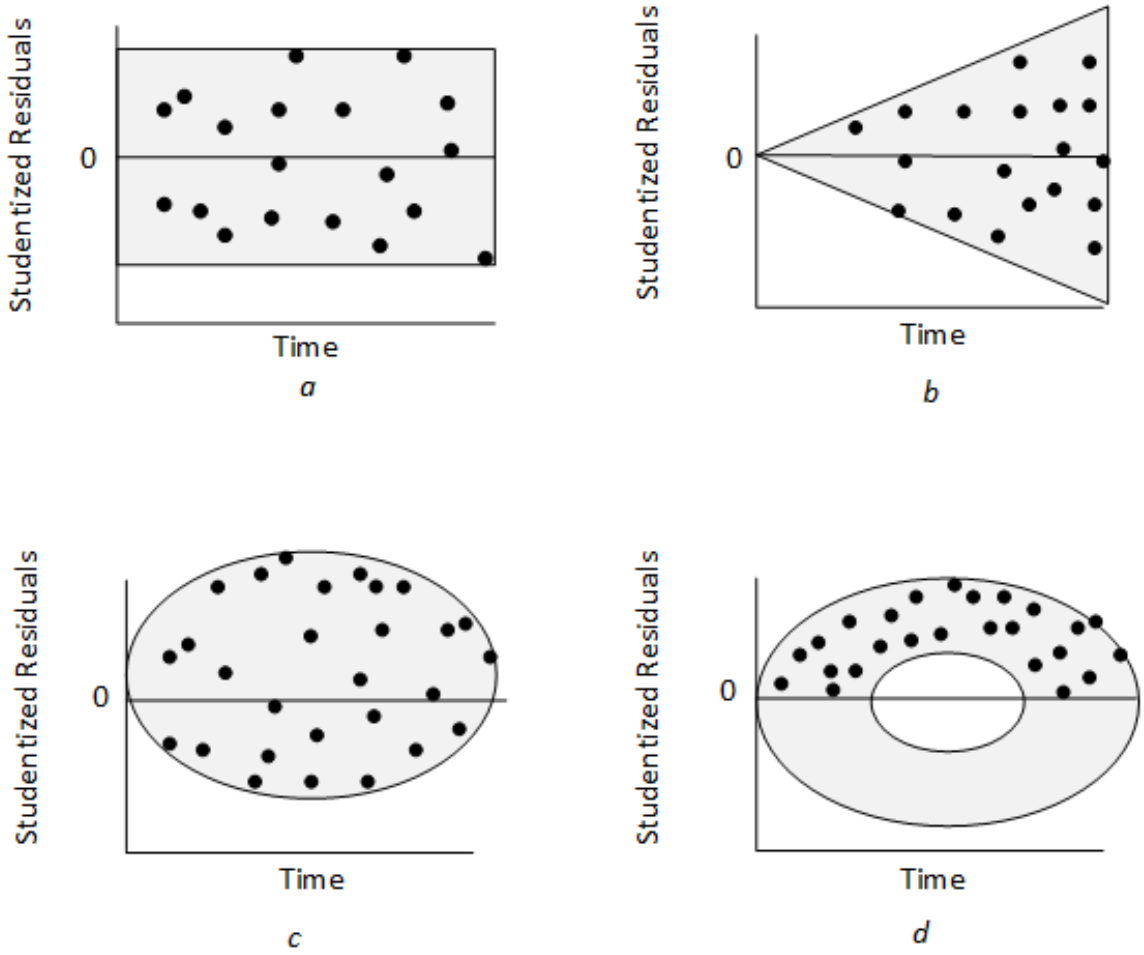


Figure 7. Checking studentized residuals for model adequacy: a). shows a horizontal band of studentized residuals, acceptable b). shows funneling, problematic c). double bow, problematic d). non-linear, problematic. Acceptable studentized residuals implies assumption of independently distributed errors is met. Problematic studentized residuals implies assumption on error is not met.Valid assumptions are needed to make valid inferences from statistical models

knowledge of second order responses from system experts or numerically if higher order model terms are indicated by a lack of fit test and curvature test. A lack of fit test indicates there is a lack of fit in the model. This can indicate a need for including interaction terms, second order effects, or some other missing effect. To show that a second or higher order model is needed, a test for curvature can be conducted. If the statistical test indicates curvature, a second or higher order model is necessary (Montgomery, 2012). Statistical software such as JMP, SPSS, and Minitab can be used to quickly calculate these statistical models and tests. Bihl (2017) gives a guide on utilizing JMP for this purpose.

In response surface methodology (RSM), a statistical model based upon linear regression above is used to create a response surface. Response surfaces are used to characterize a system and optimize multiple responses. There are multiple methods for optimizing multiple responses in a system. Desirability functions is one way to simultaneously optimize multiple responses. To do so, the analyst creates a desirability function, d_i, for each response which varies over the range $0 \leq d_i \leq 1$. If the objective is at the target, $d_i = 1$. If the response is outside of an acceptable region, $d_i = 0$. The optimal configuration is the one which maximizes overall desirability. Each objectives is given a weight w_i to account for objectives that are more important than others. Weights w_i sum to 1. In additive form, the desirability score for m objectives is given in Equation 19.

$$D = \sum_{i=1}^{m} w_i d_i \tag{19}$$

The desirability function is applied to points on the response surface. The point which gives maximum desirability indicates the optimal response based on the weights given (Myers, 2016).

2.6 Test Planning

The purpose of test planning is to ensure that the time and money put into testing yield useful and informative results. The steps in test planning help guide the analyst in creating the right test for the right reasons. Each step in the test planning process should be traceable to system requirements. The test should address the problem statement and answer the test objectives. It is very possible that one step in the process might inform earlier steps. In these cases, it is useful to interate again through the process to create the most useful and effective test plan. This feedback loop increases the amount of learning done before testing and helps to mitigate risk of test on budget and schedule. Using the guidance from the STAT COE, a summary of 10 steps have been compiled (Cortes, 2014):

1. Draft a problem statement that addresses scope and the type of problem to be investigated by the test plan
2. Create a system decomposition, often a work breakdown structure (WBS)
3. Write clear, concise, testable, traceable, and measurable test objectives
4. Identify evaluation measures (response variables)
5. Identify required data for evaluation of responses
6. Identify sources of variation that could affect responses
7. Identify and understand all potential factors that could affect the responses
8. Select region of interest, factors to vary, and factor levels
9. Select experimental design based upon the above and how many runs can be afforded by the test infrastructure, timeline, and budget.
10. Trace above to problem statement and test requirements

2.7 Design of Experiments

An important step in experimentation is the experimental design chosen (test planning step 9) using design of experiments (DoE). The results and conclusions made from an experiment are largely dependant on the methods used to collect data. Generally, experiments study processes or systems. Processes or systems consist of inputs and outputs which are impacted by factors that can either be controllable or uncontrollable and include a number of factors from design parameters to environmental and operating conditions (Montgomery, 2017). Figure 8 shows a simple graphic of this process.

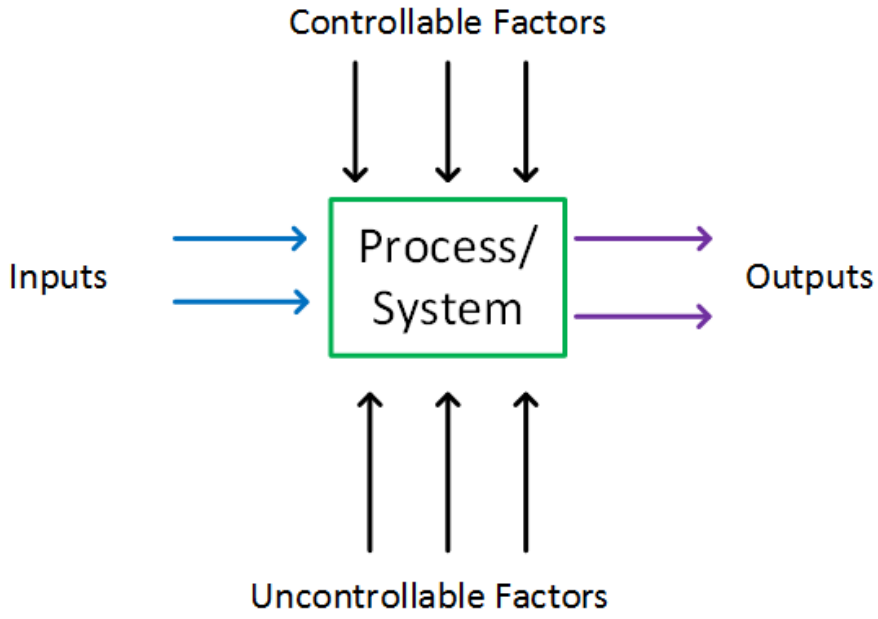


Figure 8. Graphical depiction if a process. An important initial step in experimentation is identifying inputs, outputs, controllable and uncontrollable factors. Based upon resources and time, the number of final factors, inputs, and outputs to be tested will be selected.

In experimental design, these factors are identified as either potential design factors or nuisance factors (test planning step 6-7). A design factor is one that an experimenter desires and has the ability to vary in an experiment. A nuisance factor

is one that has an effect on the process but is not of particular interest of the experiment. Nuisance factors can be classified as either controllable, uncontrollable, or noise. A controllable nuisance factor is one the experimenter can manipulate while an uncontrollable nuisance factor cannot. A noise nuisance factor is one that varies naturally but can be controlled for purposes of an experiment. Experimental design is the process of identifying these factors and utilizing randomization, replication, blocking, and the factorial principle (Montgomery, 2017).

Randomization.

Randomization is important in experimental design because statistical methods require observations to be independently distributed random variables. Randomizing the allocation of experimental material and order of runs properly allows this assumption to be valid. Additionally, proper randomization of an experiments assists in averaging the effects of extraneous factors present due to time such as learning or wear (Montgomery, 2017).

Replication.

Replication or repeats of the same experimental conditions allows the experimenter to obtain an estimate of experimental error. This has two purposes: to determine if the observed differences in data are statistically different and to more accurately estimate the mean response. Increasing the number of replicates allows the experimenter to make a more precise estimate of the true mean response rather than experimental error. Replicates reflect sources of variability both between runs and possibly within runs (Montgomery, 2017).

Blocking.

Another experimental technique is blocking. Blocking allows the experimenter to reduce or eliminate variability from nuisance factors. Generally, a block is a set of homogeneous experimental conditions where each level (setting of a factor) of conditions could potentially be a block. For example, an experiment may require multiple batches of raw material for all required runs. However, there could be differences between batches due to supplier variability. For a given experiment, the batch number may not be a factor of interest. Instead, this is a nuisance factor and each batch would be a block in the experiment to take out the effect of supplier batch from experimental error (Montgomery, 2017).

Factorial Principle.

Lastly, the factorial principle determines experimental testing of all or a fraction of combinations and interactions of factors. In design of experiments (DoE), the setting at which each factor is set is a level. If L levels of k factors are selected and a full factorial is applied, the number of runs required is L^k.

$$\text{N Runs} = L^k$$

Observing responses at selected levels and their interactions allows the experimenter to characterize the bounds of a system in a systematic way. However, selecting all factors at all levels may create a number of runs unaffordable with time and/or money. Therefore, the sparsity of effects principle is applied. The sparsity of effects principle says most systems are dominated by some main effects and low order interactions. Typically, higher order interactions are negligible and a system can be explained with a few factors and low order interactions. Classically, factorials are

selected with $L = 2$ and augmented with center points if testing for curvature (2nd order model) is of interest (Montgomery, 2017).

Factor screening can be used to keep the levels of factors selected low. In factor screening, many factors over two levels are used to determine the most influential factors for a system over a relatively large region of interest (wide breadth of experimental conditions). Designs have been created specifically for screening which give inaccurate model coefficients but allow many factors to be tested over fewer runs to determine factors with the strongest effects. Factor screening takes place is step 8 of the test planning process.

Once the number of factors are determined, a design is chosen (test planning step 9). There are many things to consider when selecting a design. When finalizing the design, the selection should be able to give the required information to satisfy the test objectives from the test plan. Decisions in test design are trade offs. Typically, by gaining one thing, another is lost. For example, having replicates allows the analyst to determine experimental error but require more runs. Additionally, adding center points to the design allows testing of curvature and assessment of lack of fit to use a quadratic model. However, adding center points decreases the variance optimally of the model in exchange for a model that is closer to the true population response. When making trade off decisions the analyst should always reference the test problem statement and objectives to make the best decision possible.

Full factorial designs include observations at all conditions and their interactions. When full factorials are not permissible due to time or money, fractional factorial designs can be implemented. Fractional designs lack most interactions of factors but have a portion included. Fractional designs can be projected by running the other portions of the fraction(s) to get closer, or eventually become a full factorial design.

Center points and replicates can be added as discussed above to give estimates of pure error, lack of fit, and assessment of curvature.

Factorial designs with three factors each with two levels are often depicted as a cube. Factors set at a high level are signified by a +1 and factors set at a low level are signified with a -1. Figure 9 shows a cube depiction of all the design points given in Table 3. A factorial design can be augmented with center points and interactions with main effects (Montgomery, 2017). Figure 10 shows a graphical depiction of the entire design given in Table 4.

Run	Factor 1	Factor 2
1	-1	-1
2	1	1
3	-1	1
4	1	1

Table 3. Factorial design: two factors, two levels

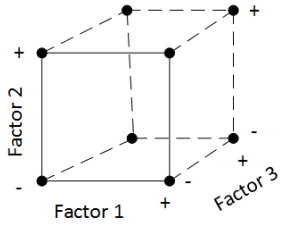


Figure 9. Factorial design cube

Run	Factor 1	Factor 2
1	-1	-1
2	1	1
3	-1	1
4	1	1
5	0	0
6	0	0
7	-1	0
8	1	0
9	0	-1
10	0	1
11	0	0
12	0	0

Table 4. Face centered cubic design

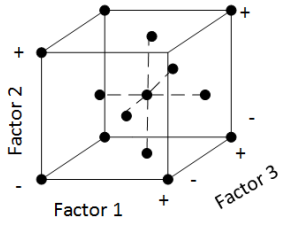


Figure 10. Factorial centered cubic design cube

2.8 Testing Autonomy

Testing autonomous systems is founded in the same principles as non-autonomous systems. Test planning, design of experiments, and statistical models are implemented. As shown in test planning, traceability to requirements is important and referenced throughout the test process. In the STAT COE workshop report, Ahner and Parson (2016) calls for requirements and measures that address the systems ability to complete critical tasks as well as autonomous decision making capabilities. In test, the ability of the system to achieve its required tasks is the primary test metric (OAS, 2010). There are a variety of metrics that have been used to describe a system with autonomy: fuel usage to measure efficient search (Berthold et al., 2019), percent detected in underwater search (Roberts et al., 2018), and response time in flocking formations of UAVs (Hauert et al., 2011). To begin creating metrics for autonomy specifically, one can research what is needed for an autonomous system to be both effective and safe and leverage these requirements to develop test metrics.

Literature addresses requirements of autonomous systems. In Woolley and Peterson (2009), autonomous systems that implement reactive architectures prescribe to requirements of reactive planning. These requirements state that an autonomous system shall be: responsive, robust, and modular (Woolley and Peterson, 2009).

1. Responsive: a responsive autonomous system allows timely planning and reaction to its environment, allowing safe operation in a dynamic environment

2. Robust: a robust autonomous system allows performance in unanticipated circumstances and sensor failures

3. Modular: a modular autonomous system allows incremental development

These requirements allow reactive architecture systems to function safely and effectively in unpredictable environments (Woolley and Peterson, 2009).

Brooks (1986) identifies 4 requirements for an intelligent autonomous mobile robot:

1. The system shall achieve multiple goals
2. The system shall have multiple sensors
3. The system shall be robust
4. The system shall be extensible

The requirement to have multiple sensors is derived from the first. According to Brooks, multiple goals are necessary for useful implementation of an autonomous system. Multiple goals are achieved by perceiving an environment with multiple sensors. The requirement of using "multiple sensors" to achieve "multiple goals" has two consequences: a requirement to be responsive and the reality of making decisions under uncertainty. According to Brooks, the control system "must be responsive to high priority goals, while still servicing necessary 'low level' goals" in order to achieve multiple goals efficiently. These goals are achieved by an agent perceiving its environment with multiple sensors and making appropriate decisions. However, each sensor provides information with an associated error. Since the agent uses this information to make decisions, autonomous systems make decisions in conditions with error (Brooks, 1986). This reality brings into question the quality of decisions made under error and the impact of error in the performance of required tasks.

The third requirement, robustness, addresses the systems ability to adapt. An agent will experience sensor failures. In response, the system should adapt its logic to only use sensors currently reliable to achieve remaining functionality.

The last requirement listed is extensible. As more capabilities are added to the autonomous system, more processing power will be needed. If the agent is not ex-

tensible, adding more capabilities will impair the robots processing speed, hurting operational functions due to slow processing time (Brooks, 1986).

These requirements can be implemented in autonomous systems using reactive or active architectures and utilized to create test objectives to enter into the test planning steps. The methods of test planning, design of experiments, and statistical models can be used once metrics are identified to measure the extent to which these requirements are met.

2.9 Summary

In literature review, information on the wide area search scenario, cooperative autonomous control, autonomy architectures and the unified behavior framework are given. Previous work on the autonomous system reference architecture expanded in this research is presented and the knowledge needed to understand the test methods, models, optimization and origin of autonomy metrics are detailed. This information will be implemented in the research implementation and analysis.

III. Methodology

3.1 Overview

Chapter III provides a discussion of the selected scenario and test methods performed in this research. Details of the selected wide area search scenario are given along with the four levels of cooperation to be tested. Next, the chosen approach to implement this WAS scenario in ASRA is discussed. For test methods, the steps used to plan the test, including metrics are presented. The details of factor selection, automated testing and optimization are presented.

3.2 WAS Scenario Design

A wide area search and confirm mission was selected as the application for this research. This differs slightly from some of the existing WAS research as no munitions are involved and the targets are not attacked, simply revisited to confrim their classification. The main objective was to both find all targets in search mode to get an initial classification, and refine that classification with a confirmation at a lower altitude. The confirming could be either the original searching agent in a single agent case or another agent in the case of a cooperating case. This provides a mission with competing goals: to efficiently find as many targets as possible in a given search area, and to accurately classify them.

All agents had the same sensor model which utilized a binary confusion matrix and a circular field of view. The binary confusion matrix was degraded based on a chosen best case Ground Sample Distance (GSD) or minimum altitude, and only the first reading of a target was recorded. This resulted in an altitude where flying higher increased the area coverage rate but degraded the sensor image quality. Flying lower than the minimum altitude did not improve the sensor any further than the best case

performance. The minimum altitude was selected as the agents' confirm altitude, so the higher search altitude diminished the sensor quality by some factor.

The agents searched in a "lawnmower" pattern with spacing determined by the sensor field of view and search altitude to give maximum coverage of the search area in the minimum number of passes. This put classification accuracy and area coverage rate at odds as both could not be maximized for the same mission parameters. Additionally, each agent had a fuel usage model based on its velocity. This limited endurance and would trigger a return to launch (RTL) condition to ensure that the vehicle returned home with a 20 percent fuel reserve. The fuel consumption rate was designed to simulate mid-sized consumer multi-rotors and, together with the search area size, limited system performance in some cases.

Real and false targets were implemented to test the sensor's false alarm and false positive error. These targets were uniformly distributed across a given search area and kept static throughout the simulation. When a target came into view, a number was pulled randomly from a uniform distribution and compared to the confusion matrix to make a determination on how the target was sensed. If the number was lower than the appropriate confusion matrix diagonal value, the target was sensed correctly, otherwise it was an incorrect classification and a Type I or Type II error was assigned to that target instead of a correct classification. Targets were sensed only once per pass so targets were ignored in sequential frames.

The scenario was run at four levels of cooperation to study their effects on system performance. First, the single agent case was run as a baseline to test the basic autonomy and ASRA performance. In this case, a single agent searched the entire area and then transitioned to confirm any targets it detected as real. Second, a basic cooperation case was implemented where two agents split the search area initially, then searched and confirmed their half without communicating with the other agent.

Third, the extreme cooperation case consisted of two agents splitting the search area and sharing their target information. As soon as an agent found a target, the other agent would immediately break from search, confirm the target, and return to their search pattern. Finally, the moderate cooperation case implemented a utility function to determine when an agent should break from search to confirm any target.

An additional case was run on the final, moderate cooperation case that specifically tested the WAS system's ability to compensate for an agent falling offline. To test this, one agent was initialized with a low fuel capacity, causing it to RTL early in the mission. The remaining agent would then continue searching the total area, taking over the area left by the other agent. The RTL agent could then return to the mission but could only confirm already found targets.

Tested Cooperation Levels:

1. Single Agent Case - one agent searches and confirms entire area.

2. Basic Cooperation Case - Two agents split search area and individually search their half.

3. Extreme Cooperation Case - Two agents split search area, search, and immediately confirm any target other agent finds.

4. Moderate Cooperation Case - Two agents split search area, search, and use a utility function to determine value of confirming any target or continuing search.

These scenarios were designed to tax the system in multiple ways, providing a means to analyze how an advanced implementation of autonomy with cooperative agent interactions can alter mission effectiveness. Many elements could have been added to the scenario such as target priorities, persistent surveillance, more agents, moving targets, or a more advanced simulation environment but were not implemented due to time constraints.

3.3 Software Design

This research implemented ASRA with a three layer architecture for the single agent case initially, and was expanded to the four layer HAMR architecture to provide the additional coordinator layer needed for the cooperation cases. To achieve this, the existing state of ASRA had to be extended which consisted of additions to the sequencer and the inter-module messaging structure as well as completely new deliberator, coordinator, and perceptor modules. LCM provided the communication interface between modules because it provides low-latency data transfer between discrete software parts along with logging and live inspection tools. All LCM messages had a single sender to prohibit information loss caused by two senders sending unique information at the same time. This increases the required number of LCM messages but allows layers to send data at anytime without requiring synchronization, a requirement when running software modules concurrently.

The coordinator relayed appropriate information between agents and provided the deliberator with additional information on other agents and utility values on cooperation related tasks. The deliberator layer was implemented as a state machine with transitions driven by completed behaviors or new goals such as stopping search to confirm a target. The sequencer layer was implemented as a static sequencer that selected behaviors by matching objective plan (OP) goals to the goals of the controller's predetermined behavior hierarchy. This is a simplified sequencer compared to the Dynamic Behavior Hierarchy Generation sequencer presented in Peterson et al. (2011) which builds up the behavior hierarchy dynamically at run time. The controller was an instantiation of the UBF that did not require the use of complex behaviors, but a test of complex behaviors was performed to analyze the process of building up complex behaviors. Behavior sensor inputs were not provided by the sequencer but

instead were provided by the originator of the sensor data such as the simulator for position data or the deliberator for waypoint position data.

The perceptor and simulator made up the remaining two modules. The perceptor contained the target sensor and communicated over LCM to send sensed target information and receive commands from the deliberator. A particle mass simulator and the Ardupilot Software in the Loop (SITL) simulator were two viable simulators. SITL's higher fidelity model but longer run time had to be weighed against the faster particle simulator. Identical scenarios were run in both simulators to determine the appropriate simulator to use for all experiment runs.

3.4 Test Definition

Test definition is an important part of test planning. Test definition includes problem statements, system decomposition, and test objectives. Test objectives map to problem statements and elements of the system breakdown. The test plan is given in Appendix Q.

Creation of the problem statement/questions provide what the test needs to address or answer and drives the following steps in the test plan.The problem statement includes the scope of study and indicates the type of problem to be investigated.

Problem Statements/Questions:

1. What configuration of design parameters will maximize area searched and percentage of real targets found for a multi-rotor vehicle(s) in a WAS mission?

2. What configuration of design parameters yield robustness, perception accuracy, and responsiveness for a multi-rotor vehicle(s) in a WAS mission?

Problem Type: Optimization of multiple response variables/objectives

Scope: Rotary vehicle in a WAS mission

System Decomposition: A breakdown of the system being studied allows the problem statements to be applied across the entire system (Figure 11).

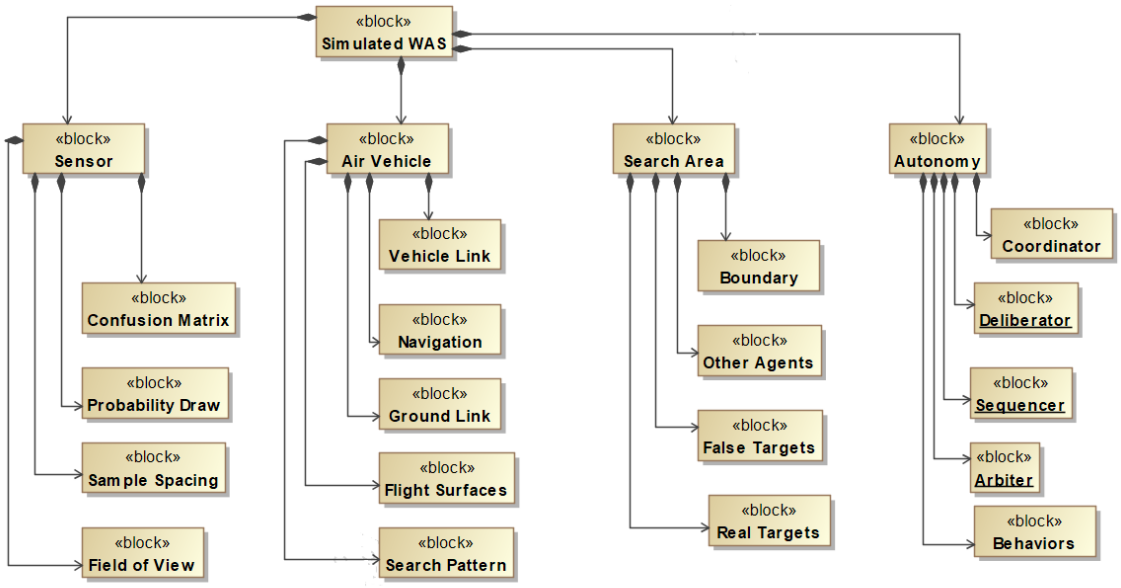


Figure 11. Component breakdown of the system

Test objectives indicate individual questions that the test should answer. Test objectives often drive response variables that will be measured in the test. The test objectives are written in question form. Test objectives are created by broadly applying the problem statements to the system breakdown given in Figure 11. Table 5 maps test objectives to system components and problem statement number.

Test Objectives:

1. What percent of targets are detected correctly?
2. What percent of targets are detected in an assigned search area?
3. What percent type I error occurs on targets out of those detected?
4. What percent type II error occurs on targets out of those detected?
5. What percent of targets are confirmed correctly out of all confirmations?

6. What percent of targets are confirmed in an assigned search area?

7. What percent type I error occurs on confirmations out of those confirmed?

8. What percent type II error occurs on confirmations out of those confirmed?

9. What percent of the assigned search area is actually searched?

10. How much time would it take to complete the mission in real time?

11. How robust is the autonomous system to sensor failure?

12. How responsive is the autonomous system to reactive planning?

13. How accurate is the perception of the autonomous system?

Table 5. Test objectives map to system components and problem statements

Objectives	WBS Element	Problem Statement Number
Percent Correct Detected	Sensor	1
Percent Detected	Sensor	1
Type I Error Detect	Sensor	1
Type II Error Detect	Sensor	1
Percent Correct Confirmed	Sensor	1
Percent Confirmed	Sensor	1
Type I Error Confirm	Sensor	1
Type II Error Confirm	Sensor	1
Percent Area Covered	Air Vehicle	1
Mission Time	Air Vehicle	1
Robustness	Autonomy	2
Responsiveness	Autonomy	2
Perception Accuracy	Autonomy	2

Table 5 shows test objectives that trace to problem statements that reflect the task and function of the system and its ability to make decisions.

3.5 Iterative Development

Methods of development and test included iterative delivery in sprints with collaboration between software development and testing. Each iteration included a test and software deliverable, demonstrating modularity and extensibility of both test and design. The required data for each test was given to the software developer in the test planning phase to allow timely testing. Collaboration between the tester and software resulted in test informed design, meaning test informed design and design informed test. Figure 12 shows the collaboration in parallel. Parallel development allowed feedback between test and development for design decisions. The sprints used in the iterative development are given below:

Sprints:

1. Deliver single vehicle with ingress, search, confirm, and land behaviors. Test included all mission and vehicle related response variables and the responsiveness autonomy metric using automated testing.

2. Deliver two vehicles with ingress, search, confirm, and land behaviors with low and high cooperation levels. Test included all mission and vehicle related response variables, responsiveness, and perception accuracy using automated testing.

3. Deliver two vehicles with ingress, search, confirm, and land behaviors with moderate corporation levels. Test included all mission and vehicle related response variables, responsiveness, perception accuracy, and robustness using automated testing.

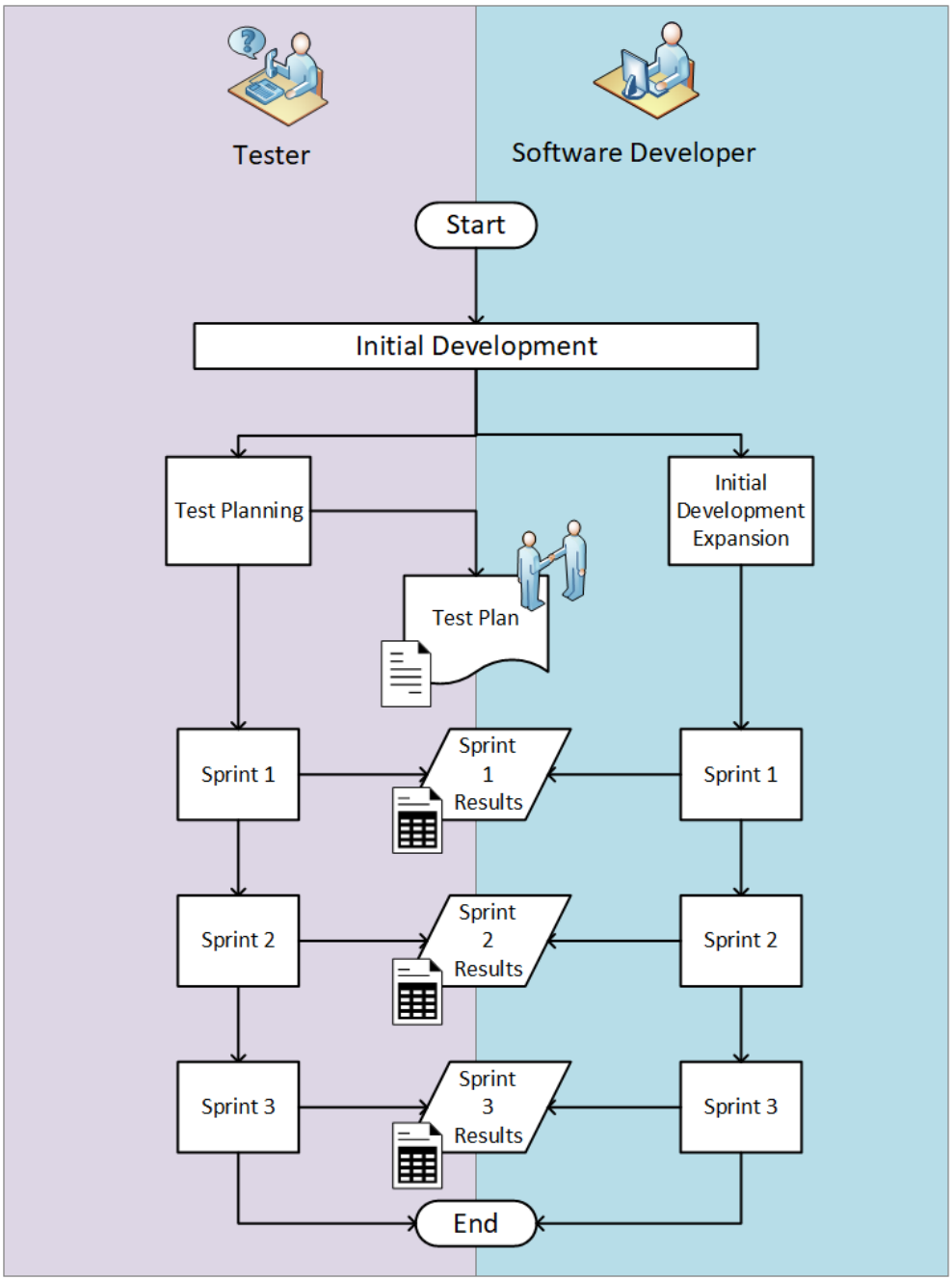


Figure 12. Graphical depiction collaboration between test and software developer

3.6 Automated Testing

The purpose of automated testing is to efficiently execute all selected factors and levels, and collect and organize all response data for analysis. The code created

followed the process shown in Figure 13. The output of the script is a spreadsheet of response variables for each factorial condition. The spreadsheet can be loaded into JMP for statistical analysis through the GUI. A guide outlining the functions of JMP in this project is outline in Appendix P. The process shown in Figure 13 is executed for each sprint.

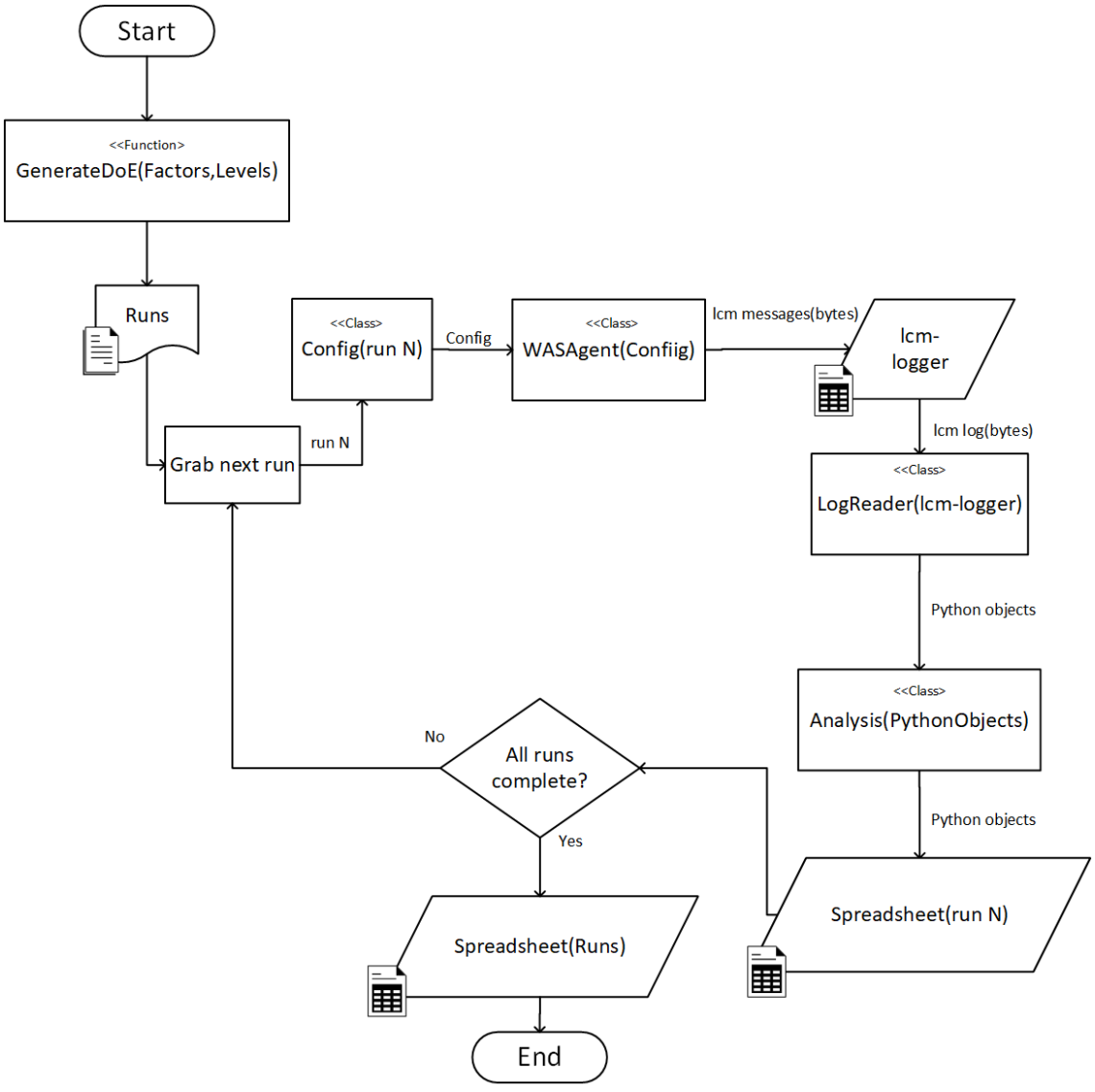


Figure 13. Graphical depiction of automated testing code

The automated testing code was run in a Python script which called each Python class shown in Figure 13. The Python script was run in a Linux terminal. The

GenerateDoE function created the experimental design using the factors and levels from inputs. The *GenerateDoE* function is from pyDOE, a Python library (Baudin et al., 2009). The resulting design is given in Table 30 and Table 31 in Appendix H.

The experimental design is saved and each row is referenced to set up the conditions for the simulation *WASAgent*. The required data for testing is found within the LCM messages passed within and/or between agent(s). The LCM messages are logged through *lcm-logger*. *LogReader* allows the lcmlog to be read and analyzed by Python in *Analysis*. Each run saves the outputs to a spreadsheet that is appended with each run to save the data in case of an error. The spreadsheet is saved as a CSV file.

Once the project progressed past sprint 1, the size of files created by *lcm-logger* grew to file sizes in the range of 200 MBs for each simulation. Additionally, 4.5 days are required to test sprint 2 entirely (320 simulations, 160 for each level of cooperation). Adjusting messages and simulation step size could potentially allow alleviation of this challenge. The simulation step size drives the precision of the simulation, measured in seconds. To alleviate file size, *lcm-logger* only subscribed to messages needed for analysis of that sprint. This resulted in faster running code and smaller file sizes. Additionally, the step size of the simulation can be adjusted in exchange for speed and smaller file sizes. As a result, tests were conducted to determine a step size that would allow faster computation without losing acceptable precision. Given the same inputs and seed number for random uniform distribution of targets, it was found that a step size of 0.3 seconds was preferable. Results are shown in Table 6. With a step size of 0.3 seconds and subscribing to only required messages, the file size for sprints 2 and 3 was brought down from 200 MB to less than 20 MB. Percent difference in Table 6 is calculated in reference to a 0.05 second step size. A negative percent difference indicates a decrease from the 0.05 reference. A

positive percent difference indicates an increase from the 0.05 reference point. In the step size test, only required messages were subscribed. In light of these findings, a step size of 0.3 is implemented in sprints two and three.

Table 6. Step size test results

Measure	0.05 Seconds	0.1 Seconds	0.3 Seconds
Percent Area Covered	62.21%	62.8% +Δ0.11%	62.83% +Δ0.99%
Mission Minutes	5.62 mins	5.67 mins +Δ0.88%	5.75 mins +Δ2.31%
File Size (MB)	13.5 MB	6.8 MB −Δ49.62%	2.3 MB −Δ82.96%

3.7 Metrics & Required Data

Using the requirements for autonomous systems listed by (Brooks, 1986) and a reactive control system as prescribed by Peterson's Unified Behavior Framework, the following metrics of autonomy were selected:

Responsiveness: The amount of time the agent requires to respond to external stimuli. This is measured as the amount of time required to actuate on an objective plan (OP). The maximum responsiveness of each run is saved and the distribution of worse case responsiveness is evaluated over all runs.

$$\text{Responsiveness} = max\,[\,t_{OP\,actuation} - t_{OP\,created}\,]$$

Robustness: The degree to which the system can continue the mission using operable vehicles after a vehicle is forced offline due to a failed sensor. Robustness is calculated as a percent difference in response Y_i for each run n, Y_{in}.

$$\text{Robustness } (Y_{in}) = 100 \left(\frac{Y_{in,\ error} - Y_{in,\ no\,error}}{Y_{in,\ no\,error}} \right) \%$$

Perception Accuracy: The impact of false perception on actions selected by the agent. This is measured by a ratio of correct plans selected by the agent against all possible plans.

$$\text{Perception Accuracy} = \frac{OP_{correct}}{OP_{Total}}$$

In order to account for all possible combinations and, the above was calculated by the following:

$$\text{Perception Accuracy} = 1 - \frac{TypeI_{Error}}{OP_{Total} - OP_{N\,Confirm} + N_{Confirm} + TypeI_{Error} + TypeII_{Error}}$$

Note that a new OP was not created and passed for each target confirmed, rather only when there was a decision to enter and exit the confirm behavior.

In addition to metrics for autonomy, the following were used to evaluate the system's mission performance:

Percent Area Searched: The percent square area evaluated by the agent for targets out of total assigned area.

$$\text{Percent Area Searched} = 100 \left(\frac{A_{searched}\, m^2}{A_{assigned}\, m^2} \right) \%$$

Mission Time: The amount of time taken by the agent(s) to finish the mission. This is the projected actual time to accomplish the mission which is proportional to the number of simulation iterations:

$$\text{Mission Minutes} = \frac{N_{sim\ iterations} N_{sim\ step\ size}}{60}$$

Percent Detected: Percent of targets detected by the agent, either as true or false. This metric captures what percent of the targets had the chance of being true or false by the sensor.

$$\text{Percent Detected} = 100 \left(\frac{N_{detected\ targets}}{N_{false\ targets} + N_{true\ targets}} \right) \%$$

Percent Correct Detected: Percent of the true targets detected by the agent.

$$\text{Percent Correct Detected} = 100 \left(\frac{N_{correct\ detection}}{N_{detected}} \right) \%$$

Type I Error Detected: The percent of real targets detected that were in truth false.

$$\text{Type I Error Detected } = 100 \left(\frac{N_{TypeI\ Error}}{N_{detected}} \right) \%$$

Type II Error Detected: The percent of false targets detected that were in truth true targets.

$$\text{Type II Error Detected } = 100 \left(\frac{N_{TypeII\ Error}}{N_{detected}} \right) \%$$

Percent Confirmed: The percent of all targets that were confirmed correctly.

$$\text{Percent Confirmed } = 100 \left(\frac{N_{correct\ confirmations}}{N_{false\ targets} + N_{true\ targets}} \right) \%$$

Percent Correct Confirmations: The percent of confirmations that were confirmed correctly out of possible confirmations.

$$\text{Percent Correct Confirmations} = 100\left(\frac{N_{correct\,confirmations}}{N_{confirmations}}\right)\%$$

Type I Error Confirm: The percent of real targets confirmed that were in truth false.

$$\text{Type I Error Confirm} = 100\left(\frac{N_{TypeI\ Error\ Confirm}}{N_{confirmed}}\right)\%$$

Type II Error Confirmed: The percent of false targets confirmed that were in truth true targets.

$$\text{Percent Type II Error Confirmed} = 100\left(\frac{N_{TypeII\ Error\ Confirm}}{N_{confirmed}}\right)\%$$

3.8 Factor and Level Selection

In theory, there are near infinite factors and levels that could be chosen for testing. However, applying a full factorial design to a large number of factors and levels makes testing unachievable. To start off, the factors in Table 7 were drafted using expert input on the WAS problem, Jacques (2019). Detect real refers to the sensor's probability of true target recognition (P_{TR}) and detect false refers to the sensor's probability of false target recognition (P_{FTR}). Using the factors as given, a DoE would require $N = 2^8 = 256$ runs without any replicates to estimate experimental error or center points to detect curvature.

.

The factors and levels in Table 7 were tested in a screening design to determine a rough estimate of factor significance. A Plackett-Burman design was implemented on

Table 7. Factors and Levels Pre-screening

Factor	High	Low
Full FOV	39°	14 °
Search Velocity	15 m/s	5 m/s
Detect Real	0.9	0.65
Detect False	0.9	0.65
Search Area	490,000 m^2	202,500 m^2
N Real Targets	19	1
N False Targets	19	1
Search Altitude	150 m	300 m

sprint 1, a nonregualr design used for screening up to 11 factors with two levels using only 12 runs. The results showed search velocity dominating over sensor configuration configuration. It was concluded that flying at 15 m/s was taxing fuel too much for the given search area, as indicated by the small percentage of assigned area covered by the single agent. The factors and levels were adjusted as given in Table 8 as a result of these findings.

Table 8. Refined Factors and Levels

Factor	High	Low
Full FOV	39°	14 °
Search Velocity	10 m/s	5 m/s
Detect Real	0.9	0.65
Detect False	0.9	0.65
Search Area	490,000 m^2	292,500 m^2
N Real Targets	19	1
N False Targets	19	1
Search Altitude	150 m	300 m

The changes made in Table 8 allowed resources to be challenged against each other. No factor level was optimal in all situations, allowing trade space evaluation. However, there is a desire to estimate experimental error and detect curvature. As a result, the test space was limited to decrease the number of runs required to meet these interests. In order to limit the test space, the field of view (FOV) and search altitude

were held constant. The full FOV diameter on the ground can be calculated using Equation 20. All possible combinations of FOV diameter on the ground are given in Table 9. The first combination is selected to hold constant across experiments.

$$\text{Ground FOV Diameter} = 2 * altitude * tan(\frac{FOV^{\circ}}{2}) \tag{20}$$

Table 9. FOV and altitude combinations

FOV	Altitude	Ground Diameter
39°	150 m	26.56 m
39°	300 m	53.18
14°	150 m	9.21
14°	300 m	18.42

The final factors and levels used in all further experiments are shown in Table 10.

Table 10. Final Factors and Levels

Factor	High	Low
FOV	39°	
Search Velocity	10 m/s	5 m/s
Detect Real	0.9	0.65
Detect False	0.9	0.65
Search Area	490,000 m^2	292,500 m^2
N Real Targets	19	1
N False Targets	19	1
Search Altitude	150 m	

Using the factors as given in Table 10, a DoE would require $N = 2^6 = 64$ runs without any replicates to estimate experimental error or center points to detect curvature. The run size is now small enough to allow replicates reasonably. Running this design showed lack of fit and significant interaction and second order effects, indicating a need to detect curvature. A final design choice of a face center central composite design (FCCD) was chosen. This design was chosen because it is a classic design for response surface methodology of second order. The design includes a factorial, center

points, and interactions between the center points and main effects to detect second order model while limiting run size. Implementing this design requires 80 runs. As a result, this model was implemented with two replicates, giving a total of 160 runs per level of cooperation or for a single vehicle. All sprints can be simulated in approximately 84 hours, with a maximum of 48 hours for sprint 2. The final experimental design is given in Table 30 and Table 31 in Appendix H.

The experimental design method for the framework is shown in Figure 14. The design takes an instantiation of the framework, WAS for a rotary vehicle, and implements the experimental design though the simulation. Data is collected for analysis of each iteration. Figure 14 shows the design in only three dimensions for readability. Since there are six factors, the test space is in six dimensions. The results of the simulation will be used to make conclusions about the instance of the framework in order to give optimal configurations for real flight using statistical methods and models.

3.9 Optimization of multiple responses

Response surfaces are created for each response variable in all sprints. Response surfaces are statistical models to predict a single response for a given set of inputs. Multiple responses can be optimized using desirability functions. Each response has a mark desirability for each response value. A more desirable response will have a higher desirability score. The range of desirability is from 0 to 1. The desirability of each response is weighted to reflect relative importance of one response over others. All weights must sum to one. The weights and individual desirabilities (d_i) are used to calculate multiple response Desirability (D_i). The optimal response is one with the maximum Desirability (D_i). The maximum Desirability configuration can be input into the response surfaces to predict performance at the optimal point given weights (w_i) and desirability (d_i) values. This is done with python classes and scripts. The

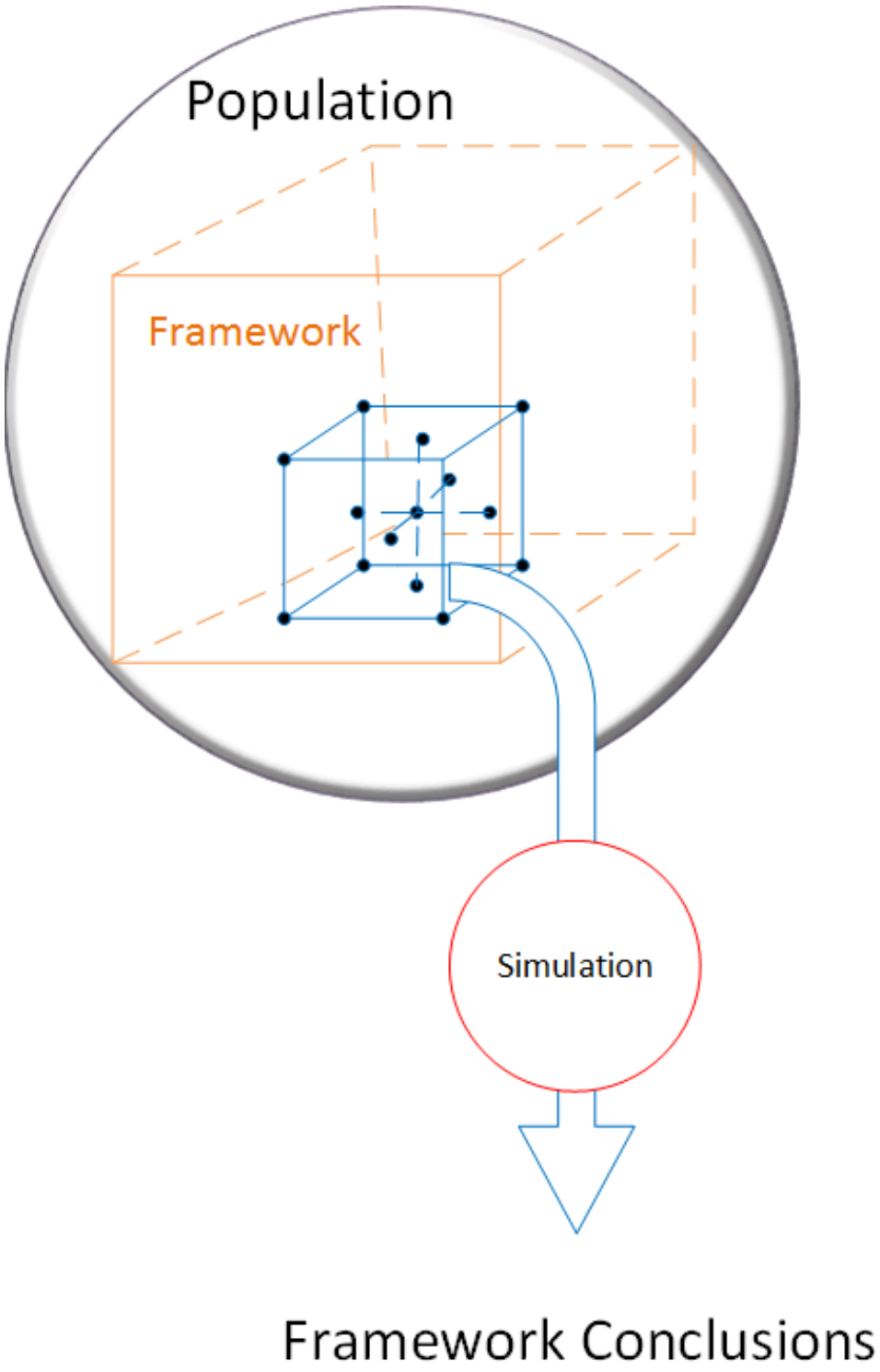


Figure 14. Graphical depiction of experimental design to test an instance of the framework

values of weights (w_i) and individual desirability (d_i) values can be changed with a configuration class.

A range of values are input into the response surface to measure Desirability. In this research, the range of values tested in experimental design are used to test all combinations. Since operational environment cannot be chosen, values of search area and number of true and false targets are defined as target sparse, moderate density, and target rich environments. Given a search area and number of true and false targets expected, the all other inputs are varied to maximize Desirability. Optimal mission parameters are selected from those that give maximum Desirability for each environment. The process for each case is shown in Figure 15. The python code created for optimization is given in Appendix O.

3.10 Summary

This chapter provided the methodology used to implement and test the WAS scenario. The search and confirm scenario to be implemented in ASRA consisted of one to two agents searching in "lawnmower" patterns to detect targets and then confirm targets at four levels of cooperation. Overall software implementation details were then given, specifying the four layer HAMR architecture and LCM messaging interface between layers. Test definition outlined the test plan implemented in this research. The iterative development strategy was presented as well as the method used to generate simulation testing using automated testing. Design and test choices such as step size selection and narrowing down of the chosen factors and levels were presented. The required data to capture the response variables were given and methods for optimization of vehicle configuration were presented. The methods shown in this chapter were implemented to generate the results discussed in the next chapter.

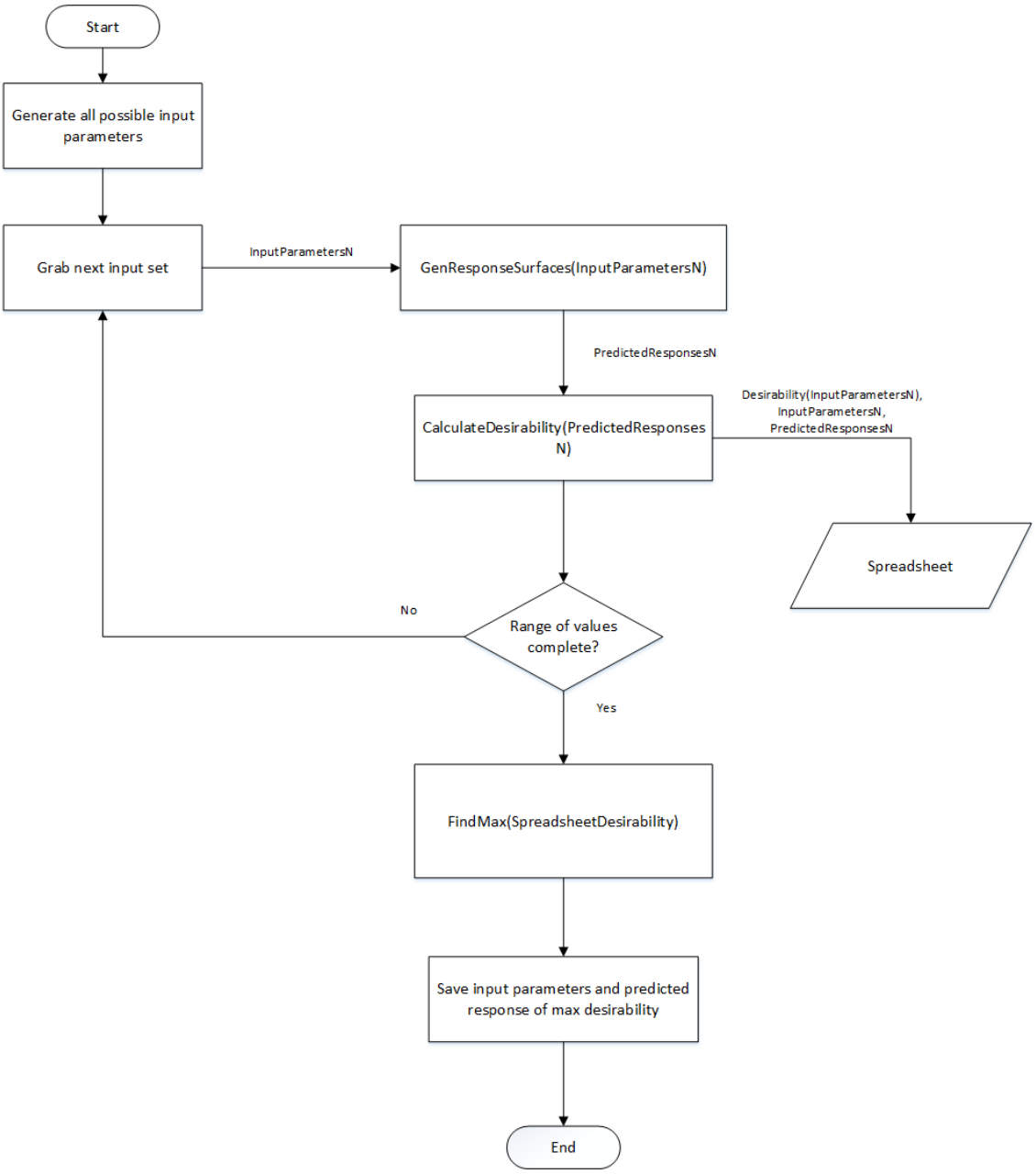


Figure 15. Graphical depiction of optimization method

IV. Analysis and Results

4.1 Overview

Chapter IV details this ASRA implementation as well as the DoE results from the experimentation runs. First, an overview of the autonomous system's architecture is given, bridging MBSE and autonomous system design methods. Next, a detailed walk through of the system's software implementation is given, starting at the controller and working up to the coordinator. The resulting mission performance at each cooperation level is presented as well as a comparison of simulators currently offered in ASRA. Finally, the results of the design of experiments are given for all sprints. An optimization of responses is applied to two vehicle operation to configure a vehicle for flight using statistical models created from design of experiments results.

4.2 ASRA Architecture Design

MBSE tools and techniques were used to design the system from both a traditional systems engineering view and the autonomy view. Both domains have unique taxonomy and system vocabulary, so a link between the two had to be made. A glossary of these terms is provided in Appendix R. Model Based Systems Engineering decomposes a system from the mission level, to tasks, and then functions. Figure 16 shows the functional decomposition for the WAS agent system. This functional decomposition is usually compared to the system's physical decomposition to allocate functions to components. This ensures a complete allocation of all functions and all components. Because the focus of the research is less on the physical instantiation of the agent and more on the software and autonomy, a physical decomposition was not created for this thesis. Instead, these system functions were traced to the autonomy functions discussed below.

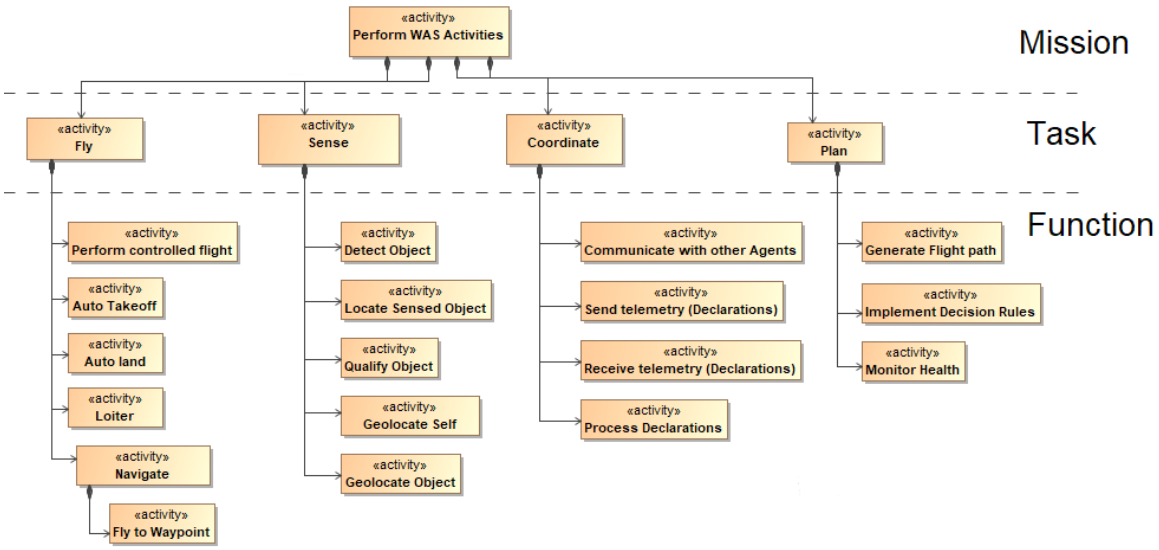


Figure 16. Functional decomposition of WAS agent system from Mission level, to task, and to function.

When decomposing autonomous systems using a behavior controller, a list of low level functions that accomplish the mission must be made. Usually this is done by decomposing the mission to objectives, to tasks, then tasks to any behaviors that accomplish that task. For this research, tasks and behaviors were mapped one to one so there was only one behavior that accomplishes a task. For this reason, the decomposition of the autonomy to behaviors shown in Figure 17 does not display a task layer. Additionally, the perceptor, hardware, deliberator, and coordinator provide functionality to the autonomous system which is displayed in Figure 18.

This implementation's four-layer HAMR architecture, shown with communication links in Figure 19, utilizes a coordinator modeled after the HAMR architecture with some simplifications, a finite state machine for the deliberator, a static sequencer, and a UBF controller. For the single agent case, the coordinator was not run so the architecture followed the three-layer architecture in this case. The coordinator receives information from other agents and sends cooperation information to the deliberator. The deliberator generates objective plans which contain one or more

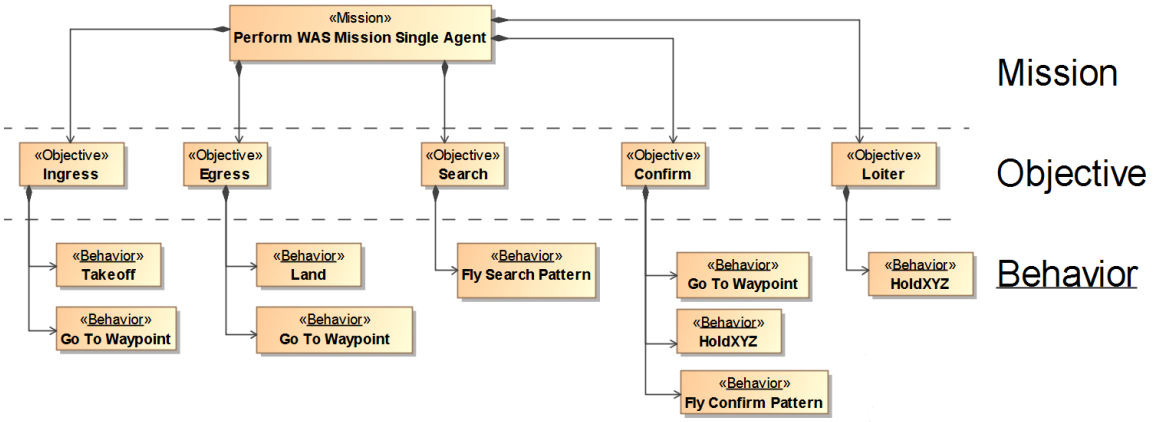


Figure 17. Behavior decomposition of WAS agent system from Mission level, to objective, and to behavior.

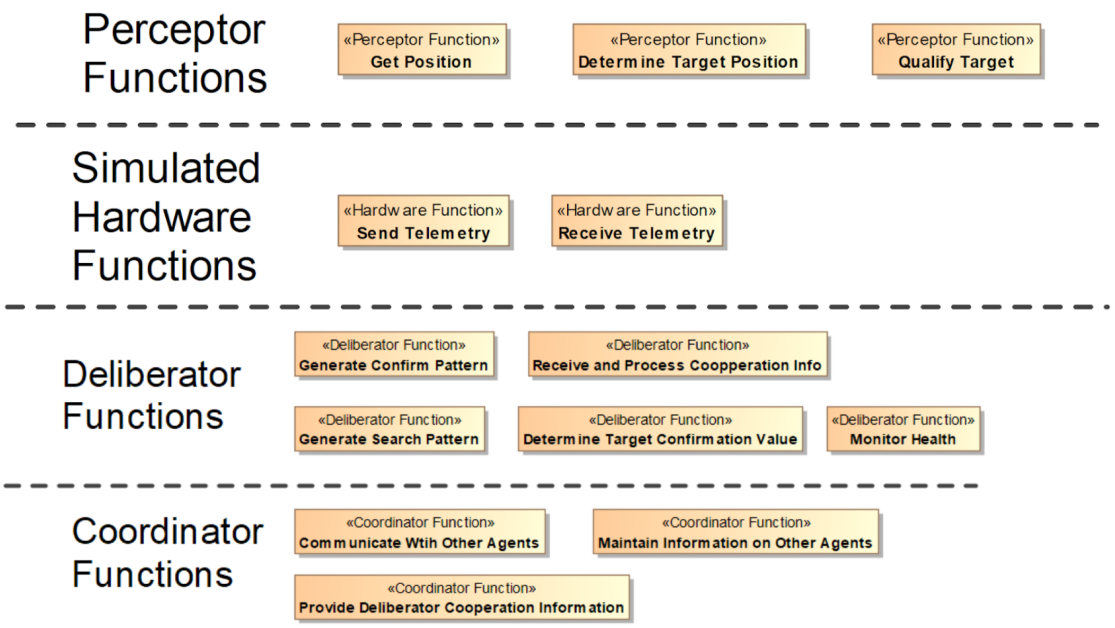


Figure 18. Additional autonomy components provide required system functionality. Perceptors provide sensor information, hardware performs hardware tasks, and logic in the deliberator and coordinator provide functionality not captured in behaviors.

goals. The sequencer receives the objective plan and converts it to sequenced tasks. These tasks must match behavior goals given in each behavior's activation paths for a viable task plan to be made by the sequencer. The sequencer then sends the individual tasks of the task plan to the controller. The controller receives the task, finds the matching behavior in the manually generated behavior hierarchy and executes the

behavior. This execution of the behaviors' motor commands is done in the lightweight particle simulator that runs in the controller.

While running the particle simulator in the controller does not strictly adhere to ASRA's embodied agent model shown in Figure 4, this design was chosen due to the simplicity and efficiency of a direct interface between the particle simulator and controller. Because the particle simulator is likely to be used mainly for development, this efficiency was determined to be worth the diminished modularity. This design still allows other simulators running in the hardware layer of Figure 4 to be implemented by selecting their interface as the behaviors' execute action in the controller, as is done when running the Ardupilot SITL simulator.

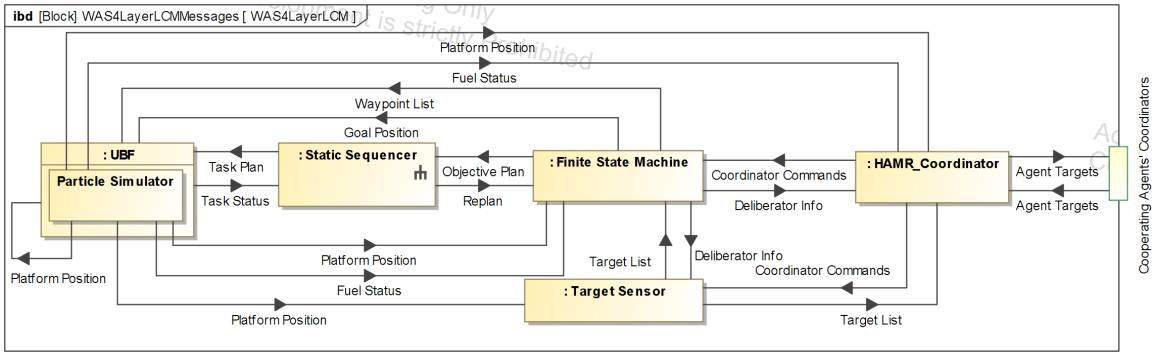


Figure 19. The four layer architecture required 12 LCM message types were used to communicate between its 5 discrete software modules in each agent as well as between agent coordinators.

4.3 ASRA Software Implementation

This section will detail this software instantiation of ASRA. First, it provides details of the overall software configuration and interaction between modules. Second, it provides a detailed explanation of the development of each module.

Software Structure.

This software instantiation of ASRA is built on top of an existing, basic ASRA implementation written in Python and developed by the AFIT Autonomy and Navigation Technology Center. The initial Python codebase included a behavior library, a particle simulator, an Ardupilot SITL interface, a UBF controller, and a static sequencer. The main modules that had to be added for this research were new behaviors, additional sequencer features, a deliberator, and a coordinator.

The interface between modules was handled by LCM. This messaging system uses UDP multicast in a publish-subscribe model where messages are sent to all clients and clients only listen to messages they are subscribed to. This method does not require a central hub as clients communicate directly. Each layer or module acted as a client and thus had a direct connection with all other layers. Each module subscribed to its required messages and all other messages were ignored by that module. This architecture used 12 LCM message types between modules as shown in Figure 19. These messages are listed in Appendix A with a description of their contents.

These modules were all designed to be able to run sequentially or concurrently where the only interface between modules after initialization is through LCM. Running these modules in parallel adheres to the layered hybrid architecture and allows them to each run at their own pace. Running in parallel allows the controller to run its reactive loop fast enough to keep up with its environment while the deliberator slowly monitors and plans.

Throughout the first half of the software development, Python threads were used to provide this parallelism, but this lead to unreliable LCM message updates. This was likely due to Python's Global Interpreter Lock which requires all threads to share the same Python interpreter. Accessing the interpreter one at time is not true concurrency. Python's multiprocessing module allows one Python program to create

additional Python processes, each with their own interpreter and allocated memory. Combining this with LCM should allow the layers to truly run concurrently, but in practice, the Python LCM library would immediately return an obscure error when running in multiple processes that were spun off using the multiprocess module. For these reasons, the main function of all layers was executed sequentially starting at the highest layer, either the coordinator or deliberator, depending on the scenario.

While running the autonomy layers sequentially somewhat defeats the purpose of the layered architecture, the limited processing and planning required of the deliberator meant this did not have a noticeable effect on the reactiveness of the controller. If the deliberator execution time for each iteration was substantial, the controller's ability to react appropriately to a dynamic environment could be reduced. When using the particle simulator, a slower deliberator would not degrade the controller's reactiveness because the particle simulator steps once each iteration, in sequence with the other layers. When using Ardupilot SITL as the simulator, which runs at a rate independent of the autonomy layers, there is a possibility of a slow deliberator hindering the controller's reactiveness. SITL's limited execution speed of around five times real time meant the controller was still reactive to the changing environment in SITL despite running sequentially with the deliberator.

A single setup script was used to initialize the WAS scenario, generating the targets and their locations, and setting up each of the layers with all necessary info. This script referenced a configuration file that defined WAS, sensor, cooperation, and simulation parameters as well as LCM channel names. For each layer, it created the state blocks that contained a unique LCM instance as well as a state class that was updated by the LCM instance when LCM's *handle* function was called. Each individual layer called *LCM.handle()* at the beginning of the iteration which updated the associated state in the state block with any new information from the LCM bus.

LCM.handle() did not update all states with a single call so it was run in a while loop until it returned 0, indicating that there were no new messages to update.

Once the layers were set up, this script iterated through all of the layers' main functions until it received notification from the deliberator that the scenario was complete. Additionally, a separate process was started to plot live agent positions in a separate window which was essential when debugging the interactions of multiple agents. Finally, the setup script would present an analysis of mission performance such as target detections and confirmations.

Controller.

The controller was structured after the UBF and consisted of a behavior hierarchy, an executive, and behavior controller. The behavior hierarchy was manually created in the setup script as a single layer of leaf behaviors contained in the behavior library because it did not require composite behaviors to provide the necessary functionality. While not used in the experiment runs, composite behaviors were experimented with to study the process of building a multi-level behavior hierarchy.

The leaf behaviors used for this simulation are shown in Table 11 and sample behavior code is given in Appendix B. These behaviors take in position data, determine where the agent is and where it needs to go, and determine the appropriate motor controls required to get there. The behavior then outputs an action which includes a weight, an action complete boolean, and an actuator command containing motor controls such as 3D velocity. Each behavior has an associated activation path that describes its attributes such as abstract goal and sensor requirements. These activation paths are standardized representations of behaviors used by the sequencer when selecting the appropriate behavior to accomplish an objective. The basic sequencer

used in this research only required the name and abstract goal fields of the activation path to determine the appropriate behavior to perform a task.

Table 11. Behaviors used in this simulation and their required sensor inputs.

Behavior	Required Sensor Input
Takeoff	Agent Position, Goal Position
Land	Agent Position, Goal Position
FlySearchPattern	Agent Position, Waypoint List
FlyConfirmPattern	Agent Position, Waypoint List
GoToWaypoint	Agent Position, Goal Position
HoldXYZ	Agent Position, Goal Position

To gain experience building composite behaviors in ASRA, ConfirmOrbit and SearchAvoid composite behaviors shown in Figure 20 were constructed but not implemented in the simulation. Composite behaviors are the combination of two or more leaf behaviors. The outputs of these behaviors must be combined in some way to convert each of their action outputs into a single action output of the composite behaviors. Arbiters accomplish this synthesis and can take many forms. A priority arbiter selects the action output of the behavior with a higher priority while a vector sum arbiter performs vector addition to return a single action output motor command vector.

ConfirmOrbit was implemented to add a surveillance orbit around the target once confirmed. This behavior was composed of OrbitRevs and FlyConfirmPattern leaf behaviors and used a priority arbiter. The behaviors set their own priorities based on when they should be active. OrbitRevs has a priority of 0 until it comes into proximity of a target. The FlyConfirmPattern priority was always 0.5 and was active until the agent arrived at a target and confirmed its target type. At that point, the OrbitRevs behavior would set its priority from 0 to 1, causing the arbiter to select its action until one orbit revolution was completed. Next, OrbitRevs set its weight to 0

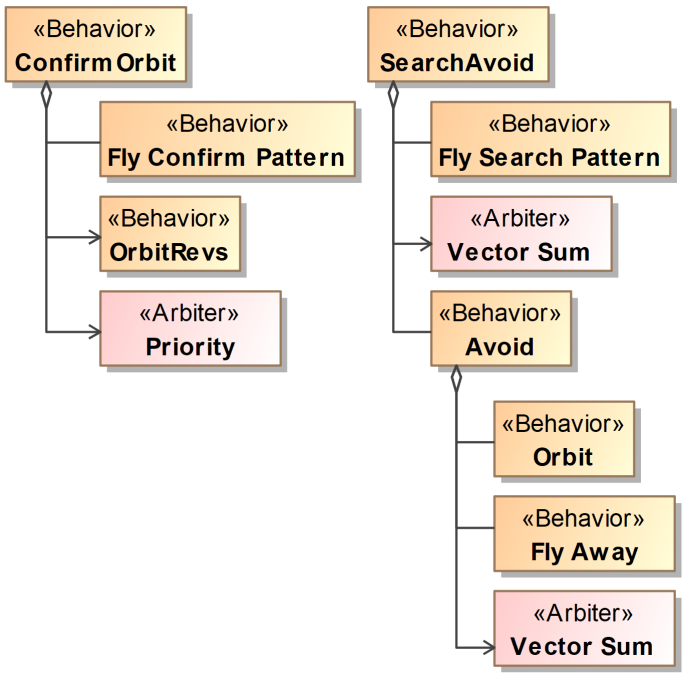


Figure 20. Additional composite behaviors were not used in the simulation but were implemented to test the process of implementing composite.

and the arbiter selects the FlyConfirmPattern action output due to its weight of 0.5, causing the agent to continue towards the next target.

The priority arbiter selecting a single action based on its priority is similar to leaf behaviors acting individually and sequentially. By combining behaviors into a composite behavior, the task of switching behaviors at the right time is handled by the behaviors themselves instead of by the sequencer. This packages the behaviors together, frees sequencer resources, avoids unnecessary communication between layers, and should yield more responsive control.

The SearchAvoid composite behavior was made of the FlySearchPattern leaf behavior and the Avoid composite behavior. This behavior attempts to fly the search pattern while staying away from an avoid location as shown in Figure 21. The Avoid composite behavior was made of Orbit and FlyAway leaf behaviors. Both of these composite behaviors use the vector sum arbiter which simply performs vector addi-

tion on the 3D velocity vectors in each behavior's actuator output. The Orbit and FlyAway leaf behaviors were designed to only provide a non-zero actuator motor velocity when within a certain range of the avoid location. The Avoid composite behavior's actuator velocity vector pointing away from the avoid location would then be combined with FlySearchPattern's velocity vector pointing at the next target to smoothly route around the avoid location.

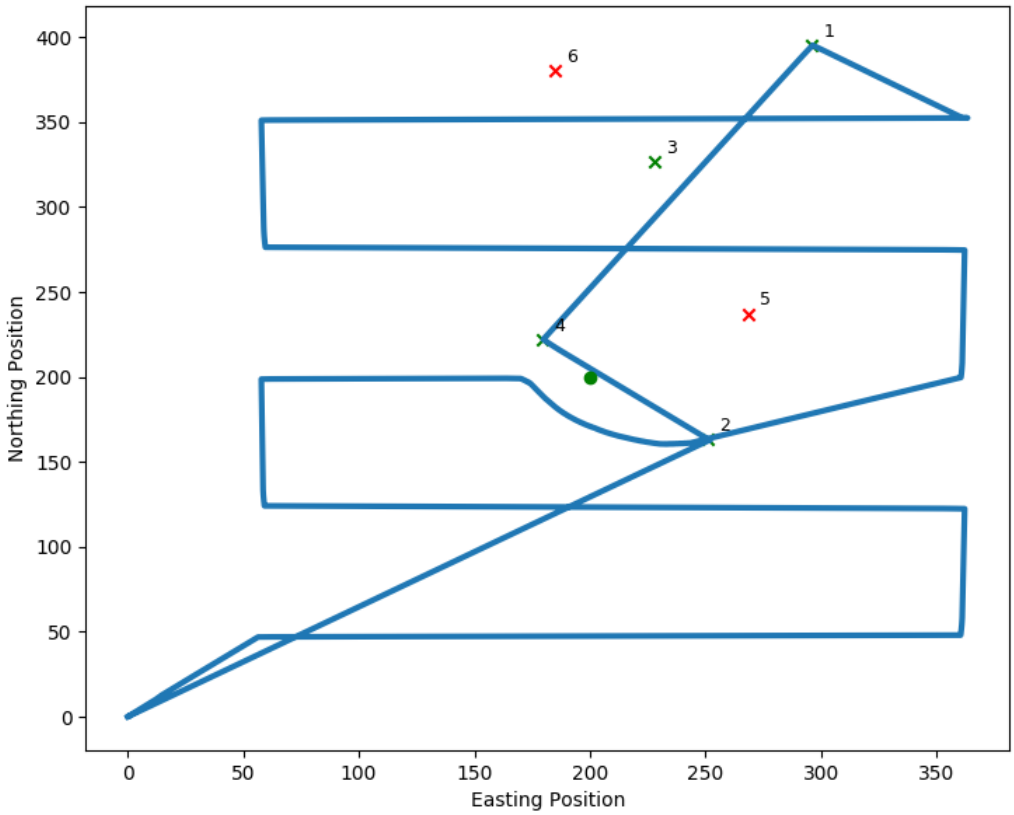


Figure 21. The SearchAvoid composite behavior causes the agent to fly the search pattern while avoiding a specified location, shown here as the green dot.

The process for creating these composite behaviors in ASRA was intuitive. For each composite behavior, the leaf behaviors must be designated and the composite behavior's sensor inputs must be set to include all sensor inputs required by the leaf behaviors. Next, an activation path is created for the composite behavior. When setting up the behavior hierarchy in the setup script, the weights of each leaf be-

haviors as well as any parameters unique to that leaf behavior such as tuning gains can be specified if different from default values. Once the composite behavior is created, the composite structure allows both composite and leaf behaviors to be handled identically by the controller.

For new ASRA implementations, this behavior and activation path generation are the two main adaptations that must be made. The behavior library simplifies this process by providing behaviors that can be used as is, modified slightly, or used as a template to create new behaviors. Additionally, the example composite behaviors in the library demonstrate the slightly more complex process of building up composite behaviors.

The second piece of the controller layer was the executive which contained the layer's main function. This main function, given in Appendix C, first receives updated state information over LCM such as positions or new tasks from the sequencer and updates its stateblock. If it receives a new task plan, the executive finds the appropriate root behavior by matching the current task goal to the behavior's abstract goal in its activation path. The executive would then set the task status message to "in progress". The executive then calls the behavior controller to generate an action by handing the current root behavior a stateblock with the necessary sensor inputs. If that root behavior is a composite behavior, the root behavior transfers the stateblock to the leaf behaviors, arbitrates their action outputs, and returns a single, arbitrated action. If that action returns as complete, which usually takes a few iterations, the executive updates the task status message to "complete" and sends the message to the sequencer. This message along with all other LCM messages used by the controller are shown in Table 12.

Behaviors were designed such that if they are completed and no new task plan was received, the behavior would continue to hold the current state. For example, when

Table 12. The controller required two LCM messages for two way communication with the sequencer, three messages with position information for the behaviors, and the particle simulator sent the vehicle position and fuel status.

Message Name	Sender	Recipient	Description
TaskPlan	Sequencer	Controller	Task list and current task number
TaskStatus	Controller	Sequencer	Current task name and status
PlatformPos	Simulator	Controller	Agent position
GoalPos	Deliberator	Controller	Current goal position for behavior to achieve
WaypointPos	Deliberator	Controller	List of waypoints forming search or confirm pattern
FuelStatus	Simulator	Deliberator, Coordinator	Agent fuel remaining

a waypoint was achieved, the behavior would simply continue to achieve the same waypoint and effectively hover at the point instead of stopping all motor outputs. This meant designing behaviors to "go to a waypoint" instead of just "go forward". This was determined to be the safest design as defaulting to a hover is usually the safest action for multi-rotors.

The particle simulator and Ardupilot SITL were interchangeable and interfaced with the behavior controller. The controller's execute action callback determines what software module handles the behaviors' action outputs in the execute action method. The simulator of choice was set as this execute action callback. When using Ardupilot SITL, the callback is set as an interface that would convert the action output's motor commands to control messages understood by SITL. When using the particle simulator, the callback was set as the simulator's update position function. This function takes in the action output's motor commands, steps forward one simulator iteration which updates the agent position, and updates the agent position LCM message. This particle simulator was selected for the experiment runs and a comparison to SITL is provided later in this chapter.

Sequencer.

The sequencer handles the conversion of objective plans sent from the deliberator to individual tasks sent to the controller. For this research, tasks and behaviors were mapped one to one so objectives were only broken down to tasks, and did not need broken down further to separate behaviors. The objective plans could include one to many objectives, each with a sequence number and activation priority. The sequencer takes the objectives and creates a list of tasks in order of the objective's sequence number. If objectives shared a sequence number, the associated task is ordered based on the activation priority. Table 13 shows the four messages the sequencer used for two way communication with the deliberator and controller.

Table 13. The sequencer required four LCM messages for two way communication with the deliberator and controller.

Message Name	Sender	Recipient	Description
TaskPlan	Sequencer	Controller	Task list and current task number
TaskStatus	Controller	Sequencer	Current task name and completion status
ObjectivePlan	Deliberator	Sequencer	List of objectives
RePlan	Sequencer	Deliberator	Current Objective Plan name, completion or failed solution status

The sequencer's main function is given in Appendix D. This starts by updating its state block with any new messages from LCM. If a new objective plan was received, it attempts to generate a task plan for the objective plan. To do this, it searches all activation paths for goals that match the current objectives. This effectively matches objectives with behaviors by way of the activation path interface. If no activation paths are found with matching goals, the sequencer notifies the deliberator of a failed task plan generation through the RePlan message. Multiple checks could be included in this process, such as only picking viable behaviors that have all the necessary sensor information available to them, but this was not implemented in this research. The

sequencer then sends the new task plan and sequence number, indicating which task in the plan to complete immediately, to the controller to execute.

If the objective plan was not new, the sequencer handles the current plan by checking if the TaskStatus message indicates a change in the controller, such as the new task has started or the current task has completed. When the task is completed, the sequencer increases the task status sequence number to tell the controller to move to the next task in the plan. When the controller finishes the final task, the sequencer notifies the deliberator that the objective plan is completed through the RePlan message and the controller continues executing the current behavior until a new task is given.

ASRA has areas in the sequencer that can be expanded such as building in a resource manager to return only viable behaviors, or utilizing the activation path's initial and post conditions to determine when a task completed. These were not implemented in this research as they were not necessary on this initial ASRA implementation. The sequencer requires little modification to the specific autonomy application. It behaves like a transfer function, converting objective plans to task plans. As long as these are specified correctly, new implementations of ASRA should not need to make many changes to this layer.

Deliberator.

The deliberator provides the high level decision making for the WAS agent and was implemented as a finite state machine. There were two versions of these state machines; one for the single agent mission consisting of the six states shown in Figure 22, and one for the cooperative mission consisting of the seven states shown in Figure 23. The deliberator's nine messages, more than any other layer or module, are

shown in Table 14. These messages provide communication with every other software module.

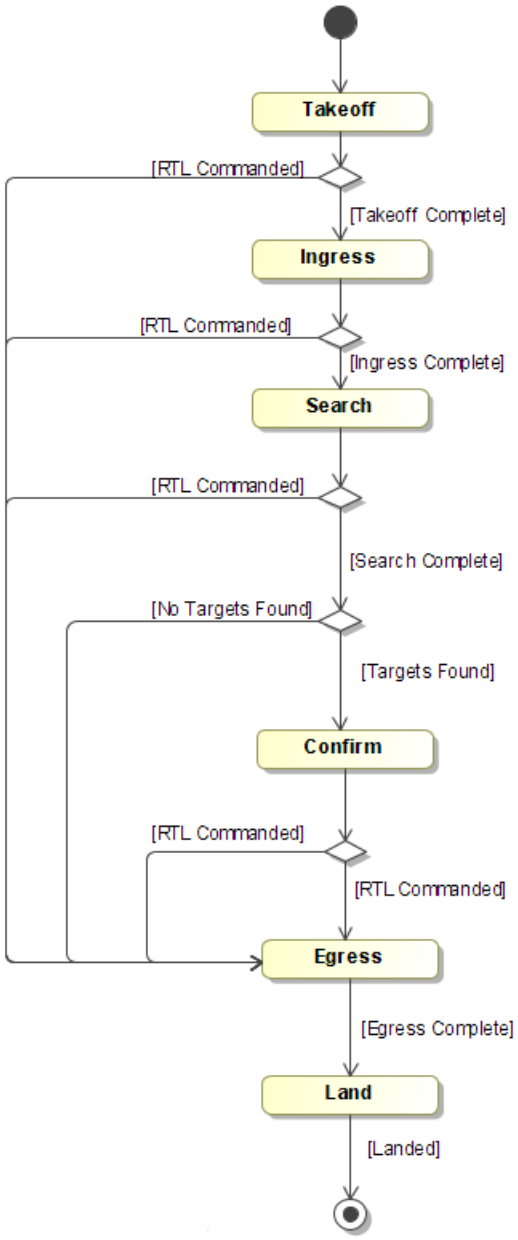


Figure 22. The state diagram for single agent has a mostly linear flow except for the return to launch condition triggered by low fuel at any time.

The perceptor, discussed in the next section, contains the target sensor and was told what mode to put the sensor in by the Deliberator in the DelInfo message. The

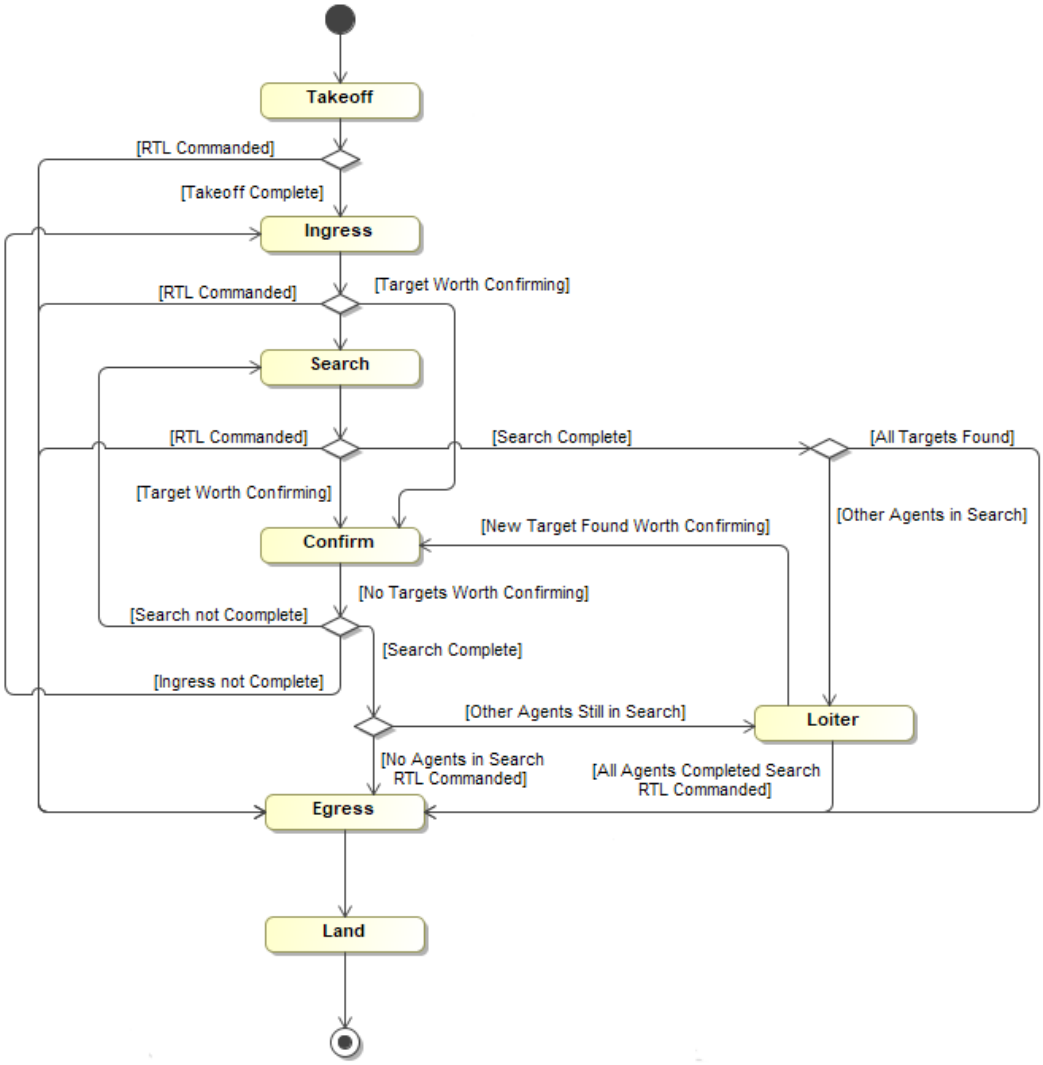


Figure 23. The state diagram for a cooperative agent has more forks because *search* can be exited early to confirm a target and must be returned to if area still needs searched.

Table 14. The deliberator required nine LCM messages for communication. Additional recipients of messages not sent from the deliberator are excluded.

Message Name	Sender	Recipient	Description
ObjectivePlan	Deliberator	Sequencer, Coordinator	List of objectives to achieve current goal
RePlan	Sequencer	Deliberator	Current Objective Plan name, completion or failed solution status
DelInfo	Deliberator	Perceptor, Coordinator	Commands to control perceptor and agent information for Coordinator
TargetList	Perceptor	Deliberator	List of targets perceptor has found
CoordCmds	Coordinator	Deliberator	Information on other agents, loiter value, offline agents
GoalPos	Deliberator	Controller	Current goal position
WaypointPos	Deliberator	Controller	List of waypoints forming flight pattern
PlatformPos	Simulator	Deliberator	Agent position
FuelStatus	Simulator	Deliberator	Agent fuel remaining

perceptor then provided its list of targets to the deliberator in the TargetList message when its list changed, such as when a new target was found. The DelInfo message was also received by the coordinator to provide it with the state of the deliberator's state machine, current state, search area completion status, and the targets currently being confirmed. The coordinator then used this information to calculate cooperation utilities or to share with other agents.

The deliberator's main script starts by updating its stateblock with any new LCM messages. Then, only on the first iteration, the deliberator checks to see if the coordinator has specified a portion of the search area to search. If not, the deliberator reads the full search area out of the configuration file and generates a search pattern. The state is then set to *takeoff* and the takeoff objective plan is then sent to the sequencer in the ObjectivePlan message. Additional information needed to accomplish any goal in the objective plan is then sent over the WaypointPos and GoalPos messages to the controller. This code is given in Appendix E

On every iteration of the main function, the deliberator checks for a RePlan message from the sequencer indicating that the current objective plan is complete. This triggers the deliberator to move to the next state such as *Ingress* to *Search* or *Confirm* to *Egress*. This logic was sufficient for the single agent case where no new decisions, besides an RTL command, would interrupt the standard flow of events. The cooperation cases required additional state change logic to break out of the current state to confirm another agent's targets for example. The deliberator also updates its target list based on the perceptor's target list or the coordinator's list of targets, compiled from other agents.

Additional deliberator functions specific to the WAS scenario:

- Fuel Status Monitoring

  The deliberator incorporated a RTL feature that would automatically cause the agent to return to the takeoff location with a fuel buffer specified in the configuration file, usually 20%. This prediction was accurate because the deliberator knew the flight profile that would be used to return home from the current location and it used the same fuel usage rate calculation used by the simulator which was only a function of velocity. This prediction was performed every iteration of the simulation and if it calculated that the fuel capacity spent to return to launch from the current location left the vehicle with less than the specified buffer, the agent performed the RTL.

- Search Pattern Generation

  The search pattern for each agent was generated during initialization in the deliberator based on the search boundaries received from the coordinator or configuration file for the single agent case. The horizontal pass "lawnmower" pattern was generated using the sensors predicted ground field of view radius in

the search flight profile, calculated from the search altitude and sensor angular field of view given in the configuration file. This radius was scaled down slightly to provide a slight overlap between passes to ensure any targets centered in between passes were not skipped over due to the simulator's step size. The scaled down radius formed the spacing off the edges of the search area and twice this radius was used as the pass spacing. This provided the minimum number of passes required to fully search the area.

- Search Area Redistribution

  In the moderate cooperation case, the agents were given the ability to redistribute the search area if an agent fell offline due to a RTL. When the search pattern gets split by the coordinator during initialization, it provides the deliberator with a list of all agents' waypoint lists. When an agent falls offline, its last recorded search position is compared with its waypoint list to determine the last horizontal pass it fully completed. A remaining agent then extends its search pattern to completely search this horizontal pass. This means that the maximum overlap between the two agent's search areas is at most one horizontal pass. The final pass of the remaining agent could have been shortened such that there was no search pattern overlap but it was simplest to limit waypoints to those given in the initial waypoints lists by the coordinator.

- Cooperative Utility Function The utility function that determined cooperation value on a per-target basis ran in the deliberator. The overall design of this cooperative ability followed the three key points for decentralized, cooperative asset management given in Malhotra et al. (2017). First, each agent's deliberator is initialized with the same mission goals so that "each agent knows the mission(s)." Second, nominal operation and information sharing between agents is assumed in this research and all necessary information is shared such that

"each asset has a nearly-common operational picture of the environment." Finally, each agent runs the same utility function so that "each asset knows the capabilities and control algorithms used by the other agents" (Malhotra et al., 2017).

This leads to a utility function that received information about all other agents and calculated each of their utility for every target that needed confirmed and was not currently in another agent's confirm list. If the current agent had the greatest utility that was above a certain threshold, that agent would stop searching and confirm that target. Often it was determined that multiple targets were worth confirming. In this case, the target positions were all given in a waypoint list message to the controller. The confirm behavior would then route through all the target locations given in this message.

Some of these parameters were developed when testing with three agents and are less applicable to the two agent case. These parameters were included in the two agent experiment runs for consistency if three or more agents are run in the future. All parameters are scaled between 0 and 1 and summed to form a utility value for a given target for a given agent that must be above a threshold for that agent to confirm that target. Through qualitative testing, a threshold value of 3.2 and the parameters and functions shown below were chosen for the experimental runs because they yielded reasonable performance across all scenarios.

- Distance to Target: high target value if low distance to target

  Targets close to the current agent should be valued higher than further targets. This horizontal distance was run through the function in Figure 24 to return a value between 0 and 1. This function likely should be scaled with the search area size, but was held constant for the experiment runs.

For example, the distance value of a target 100 meters away should be more in a large search area than in a smaller search area. The gains that define this curve were tuned over multiple test runs to achieve resonable behavior.

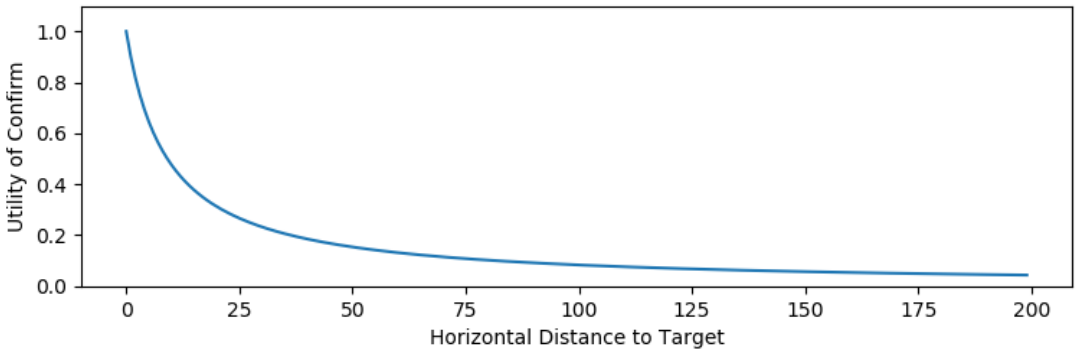


Figure 24. The closer an agent is to a target, the more valuable it is to go confirm it.

– Fuel Status: low confirm value if low fuel

  It is not optimal for an agent with low fuel to stop searching as the time spent moving to the target is of no value and the chances of running out of fuel on the way to and from the target is increased. With a limited amount of fuel, agents are more valuable continuing search because every unit of distance traveled in search is valuable and that value is guaranteed. This value used the function in Figure 25 to return a value between 0 and 1.

– Number of Targets Found: high confirm value if many targets already found

  This describes the number of targets each agent has already found. This parameter values continuing to search when the agent has found only a few targets and values confirming if the agent has already found many targets. With a uniform distribution of targets, if an agent comes across many targets already, their probability of finding more targets within their

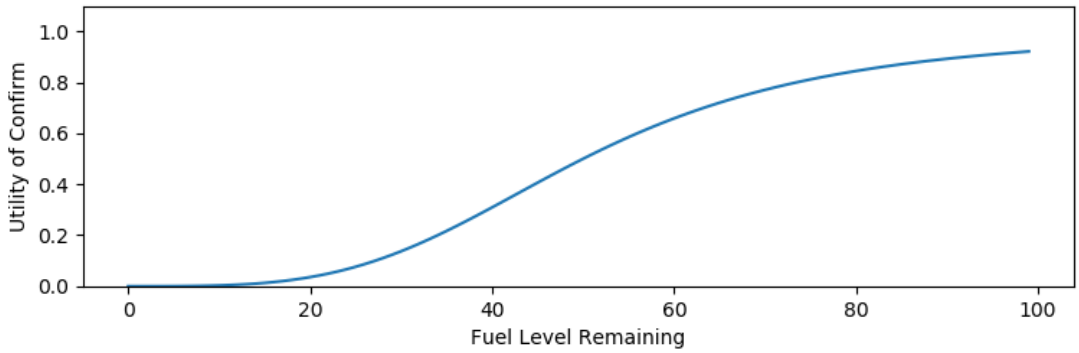


Figure 25. The more fuel an agent has, the more valuable it is to go confirm instead of continuing to search. If an agent has low fuel, it is better to continue searching to not risk running out of fuel on the way to or from the target.

search area decreases, so confirming becomes more useful for that agent. This value used the function in Figure 26 to return a value between 0 and 1. This value was set to .5 if the agent had already finished *search*, as the number of targets it found once completing search becomes irrelevant.

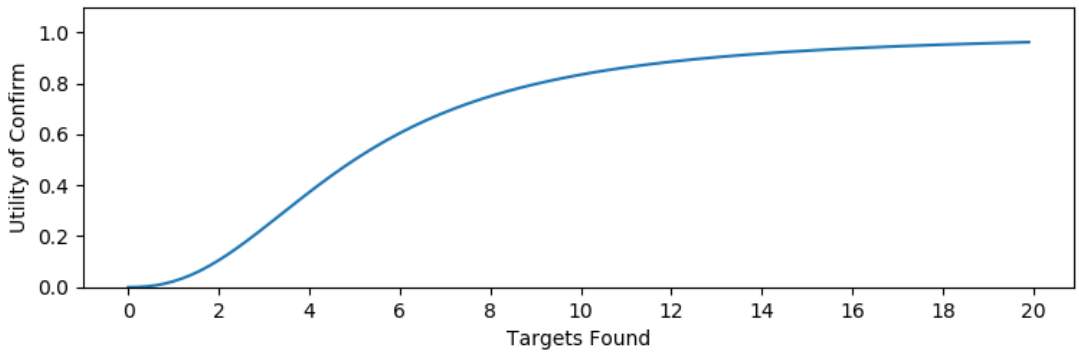


Figure 26. The more targets an agent found, the more valuable it is for that agent to stop searching and confirm a given target.

– Search Complete: high confirm value if the search pattern is complete

Figure 27 shows that an agent that has completed *search* is assigned a higher utility than one who still has area to search. This promotes finishing *search* before confirming but allows agents to stop searching to confirm a high value target. This also prioritizes the agent who has finished *search*

over one still searching. This value is set to 1 if the agent has finished *search*, and 0 otherwise.

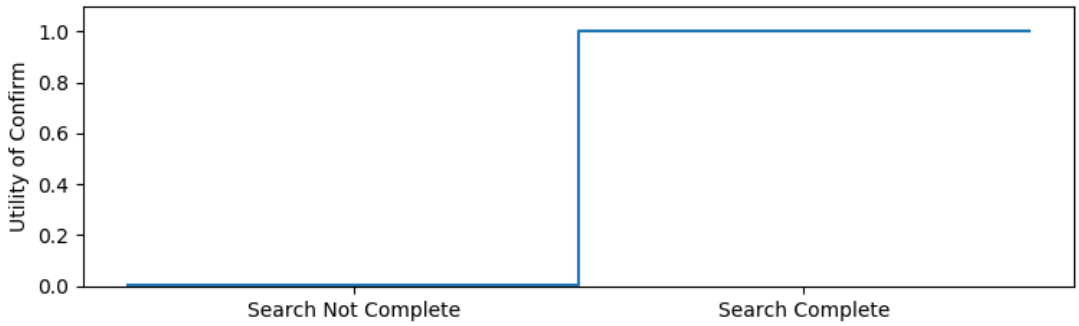


Figure 27. An agent who has finished search values confirming any target more than an agent still searching.

– In Loiter State: high confirm value if the agent in loiter

Figure 28 shows that a loitering agent generates a high confirm value as confirming is their only remaining task. This parameter works in concert with the search complete parameter as this will be true only if *search* is complete. This parameter goes a step further in assigning more value to loitering agents who have completed *search*, rather than confirming agents who have finished *search*. This was included because in testing with three or more agents, when two agents loitered, one of the loitering agents would confirm all targets because it happened to be closer to many and the remaining agent continued to loiter. This value is set to 1 if the agent is in *loiter*, and 0 otherwise.

– Number of Agents Finished Search: low confirm value if many agents finished search

If many agents have finished *search*, confirming any target should be weighted less. It is not necessary to know if the current agent is one who has finished *search* or not because the search complete weight brings that information

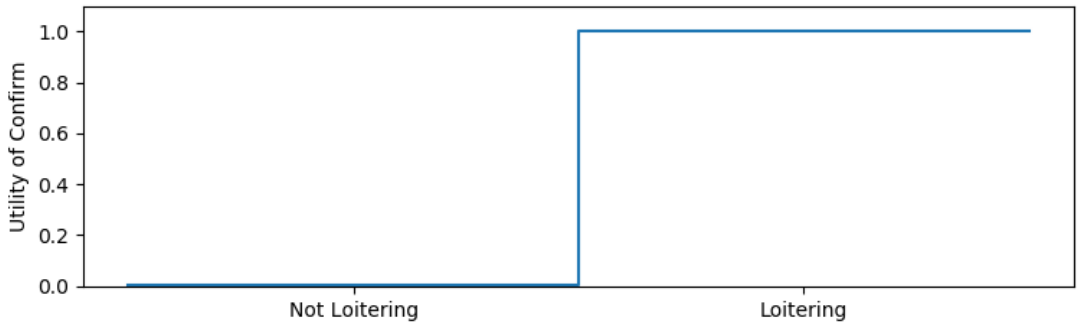


Figure 28. An agent who is loitering values confirming any target more than an agent not loitering. This helps distribute targets between loitering agents so that no agent remains loitering while the other confirms all remaining targets.

into the utility function. While this parameter is the same for all agents, it was included because it can determine if the minimum value threshold is achieved or not. This value is set to one minus the percentage of agents who have finished search as shown in Figure 29.

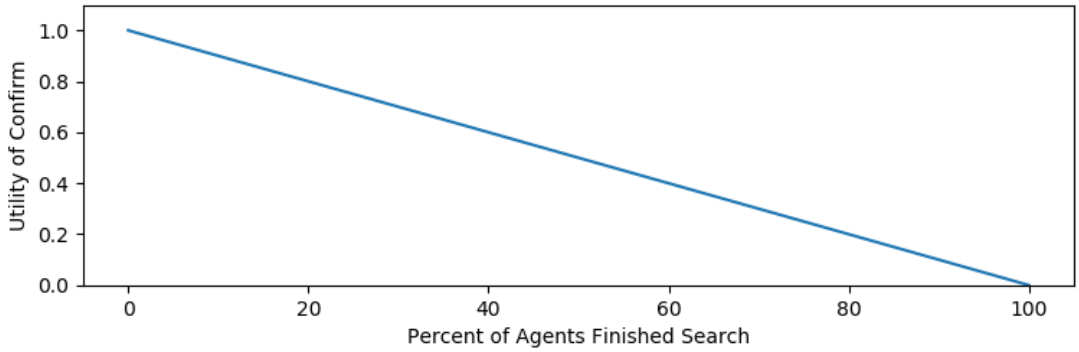


Figure 29. Agents who have finished *search* should confirm before an agent still in *search*, so the value to stop *search* and go confirm a target should decrease as more agents finish *search*. The agents who have finished *search* make up for this utility loss in the Search Complete parameter.

- Confirm pattern generation

  When the deliberator determined multiple targets required confirmation at once, it would generate a confirm pattern that visited the targets in order of proximity to the previous target visited. So if three targets needed confirmed, the agent

would first visit the nearest target to the agent, then the target nearest that second target, and then the final target. While this is in no way a truly optimal route, it did prove to generate a fairly sensible flight path through the targets. A more advanced optimal path algorithm could have been implemented but this method was implemented quickly and proved sufficient for the purposes of this research.

- State Transitions

  The deliberator state machine for the single agent and minimal cooperation cases followed the main flow *ingress*, *search*, *confirm*, and *egress* with only an RTL command causing a diversion from this nominal flow. These main state transitions were triggered by the completion of the associated objective plan. In this case, the deliberator ran the fuel check each iteration and if it did not return an RTL command, the deliberator would check if the sequencer indicated that the current objective plan was completed. If it was completed, the deliberator would move to the next state in the nominal flow and send the associated objective plan.

  The extreme and moderate cooperation cases introduced new transitions triggered by target confirmation decisions. To extend the existing single agent state machine, a new function was added to handle these additional transitions. This function ran every iteration and checked the status of the confirm target list generated by the target utility function. If the list was empty when the vehicle was in *confirm*, this function would handle the state transition out of *confirm*. If the list contained targets when the vehicle was not confirming, the state was changed to *confirm* and necessary actions were taken such as sending the correct objective plan and waypoint message to the controller.

Deliberator outputs, such as objective plans or waypoint lists, occurred during state transitions and the sequencer and controller performed the operations contained in the states, such as the sequencer advancing tasks or behaviors flying the search pattern. This meant the deliberator state machine design was centered on the transitions, and not the activities within the state. For this reason, the deliberator has no procedures assigned to certain states. It's functions run regardless of the current state because the majority of its functionality is determining when a new state change or new objective plan is required. For example, the RTL calculation, target utility function, and agent offline check were run every iteration, regardless of the state and only their execution varied slightly depending on the state. This approach allowed for the resuse of the single agent deliberator in the multi-agent deliberator.

Perceptor.

The perceptor was implemented as a separate software module, similar to the HAMR layers. It used the four LCM messages shown in Table 15 to communicate with the deliberator and coordinator. The perceptor module was the simplest of the five, as it only needed to run a target sensor algorithm upon the deliberator's request and return a list of targets and their attributes. This sense algorithm is capable of running in a separate thread but was simplified to run once per iteration like the rest of the modules without affecting performance.

The perceptor was initialized with a target list from the configuration file that contained true target attributes such as the true target type and true position. The *target_generation* function was run in the setup script and accepted the search area boundary and the number of real and false targets from the configuration file. It would then randomly pick target x and y locations from a uniform distribution for

Table 15. The perceptor required four LCM messages for communication with the deliberator, coordinator, and simulator.

Message Name	Sender	Recipient	Description
DelInfo	Deliberator	Perceptor	Agent information and controls commands for perceptor
TargetList	Perceptor	Deliberator	List of targets perceptor has found
CoordCmds	Coordinator	Perceptor	Target list compiled from all agents
PlatformPos	Simulator	Perceptor	Agent position

the desired number of real and false targets. The seed could be specified for this distribution in cases where constant target locations were desired over multiple runs.

When the perceptor determined one of these targets was in view, it would run the sense algorithm on the target and assign it sensed parameters containing error, such as sensed position and sensed target type. The perceptor's main script first updated its state block with any new LCM messages, and then updated its target list with any new target's found or confirmed in the coordinator's list of targets compiled from the other agents. The perceptor then ran the appropriate *sense* function based on the perceptor command received from the deliberator, either search or confirm, updating the appropriate target attributes shown in Table 16, and sending it to the deliberator and coordinator. This *sense* function is given in Appendix F.

Table 16. Target attributes.

Attribute	Description
search agent	ID of search detecting agent
confirm agent	ID of confirming agent
real	Boolean of target type, confusion target or real target
true_position	True 3D target position
search_position	Sensed position in search, contains error when error is enabled
confirm_position	Sensed position in confirm, contains error when error is enabled
search_detected	Boolean of if target has been detected
confirm_detected	Boolean of if target has been detected
search_type	Detection type: TypeI, TypeII, correct, not_detected
confirm_type	Detection type: TypeI, TypeII, correct, not_detected

The perceptors sensor was modeled as a circular field of view with degrading sensor performance as altitude increased above a minimum altitude. The field of view on the ground was a function of altitude and sensor angular field of view. To keep the sensor simple, only the first reading of a target as it came into frame was evaluated and subsequent readings were ignored. The *sense* algorithm was called each simulation iteration when in *search* or *confirm* states. *Confirm* and *search* detections were handled with slightly different *sense* functions as each ignored a target if it had been sensed in that flight regime already.

When the *sense* algorithm is called, it degrades the sensor quality by scaling down the diagonal confusion matrix probabilities, based on how the current GSD compares to the best case GSD. If the GSD was calculated as less than the best case, the detection parameters were scaled down using an exponential decay function, resulting in a worse performing sensor. Using GSD more accurately degraded the sensor by including the effects of altitude and field of view. This is shown in the code given in Appendix F.

The sensor then loops through the target list, checking if any new targets' true positions are inside the current ground field of view radius. Because the sensor only sensed and confirmed a target once, targets that had already been found in *search* or *confirm* were ignored when in the same mode. When a target was new and in the field of view, a number was pulled from a uniform, random distribution between zero and one. If this value was below the potentially degraded diagonal value of the confusion matrix, the target reading was recorded correctly, if not, the sensor incorrectly classified the target and a typeI or typeII error was assigned to the target for the given detection mode. If the configuration file specified a location error for the sensor, a random, uniform radius and angle around the true target position was

selected as the target's sensed position. The perceptor then sent the updated target list to the coordinator and deliberator.

Coordinator.

The coordinator formed the fourth autonomy layer and was not used in the single agent case as the deliberator received no value from a coordinator that had no communication with other agents. An attempt was made to design the coordinator to accommodate any number of agents but the task of splitting the search area beyond three agents became a challenge not necessary to address for this research. Table 17 lists the messages sent and received by the coordinator communicate with the deliberator, perceptor, and simulator.

Table 17. The coordinator required five LCM messages for communication with the other layers as well as an additional message for communication with any additional agent.

Message Name	Sender	Recipient	Description
DelInfo	Deliberator	Coordinator	Commands to control perceptor and agent information for Coordinator
TargetList	Perceptor	Deliberator	List of targets perceptor has found
CoordCmds	Coordinator	Deliberator	Information on other agents, loiter value, offline agents
FuelStatus	Simulator	Coordinator	Agent fuel remaining
AgentTargs	Coordinator	Other Coordinators	Agent position, state, compiled target list, fuel status

Using LCM with a dynamic number of agents presents a challenge because agents must first know the agent's channel name and subscribe to it before receiving LCM message from that channel. One way to address this is to leave a channel open for new agents to announce their presence. Besides the startup sequence, the odds of two agents sending on this channel simultaneously is rather low. Strategies could be implemented to address this, such as each agent sending at a constant but random

interval until all agents confirm each agents' presence. Because this problem was not important to the scope of the research, the decision was made to simply initialize the agents with full knowledge of cooperating agents so they could be subscribed to during initialization.

The coordinator's main function, given in Appendix G, first updates its stateblock with any new LCM messages. Then if the agent has not been initialized, it splits the overall search area and sends its portion to the deliberator to generate a search pattern. Each iteration, the coordinator also calculates the utility of loitering after finishing *search* to wait for new targets to be found and need confirmed. This decision is based on the state of other agents, mainly if all have finished their search pattern yet. Next, the coordinator updates and sends its global agent message to other agents shown in Table 18. Finally, the coordinator updates its target list with any new information from other agents, detects if any agents fell offline by a lack of heartbeat message, generates a list of targets currently being confirmed, and sends that information with the loiter utility to the deliberator through the CoordCmds message.

Table 18. Global agent information message attributes.

Attribute	Description
agent_id	ID agent this message describes
unique_id	send timestamp, used as a heartbeat to detect agent loss
target_list	Agent's complete list of targets
agent_position	Agent's position
confirming_list	Targets agent is actively pursuing
state	Agent's deliberator state, used when calculating utility
search_complete	Boolean of if agent has fully searched its area

4.4 Simulator Comparison

While ASRA can be extended to utilize many simulators, support already existed for two simulator options that could have been used for development and test runs. A particle simulator was the lighter weight option, starting up and running quickly but providing a low fidelity model. This simulator simply took in velocity commands and stepped the vehicle in that direction a certain amount based on the time step and velocity magnitude. This did not involve a vehicle dynamics model so it simulates a massless particle that can move in any direction. This provided a fairly reasonable model of a small multi-rotor vehicle when driven by velocity commands because these vehicles can hover and move in any direction. This simulator in its existing form was not conducive to simulating a fixed wing aircraft due to the lack of rules limiting movement to that of fixed wing aircraft. This simulator also does not currently incorporate the effects of environmental factors such as wind or air density. These rules and features could be implemented in the particle simulator to provide advantages of this simulator to fixed wing applications. If not acceptable for final evaluation of an autonomous system, the particle simulator could still be valuable to researchers during the development of an autonomous system, when its fast startup and execution time are most valuable.

The other simulator option that was already incorporated into ASRA was the Ardupilot SITL. This simulates a UAS flight controller by running flight controller firmware on the computer in a simulated environment. This allows autonomous systems to move from simulation to flight tests with minimal changes, as the interface with Ardupilot SITL and physical flight controllers is identical. This comes at a cost of a longer startup and simulation time as well as a more complicated interface with the simulator. The Ardupilot SITL simulator is also limited by the fact that it cannot initialize a vehicle in the air. Additionally, because the flight controller firmware

is only designed to run at real time speed, Ardupilot SITL becomes unstable when running simulations faster than around five times realtime.

The Ardupilot flight controller firmware uses the MAVLink messaging protocol to communicate with other devices so this is the communication method used by Ardupilot SITL. This means that to communicate with Ardupilot SITL, the action outputs of the behaviors must be converted to MAVLink messages. ASRA provides modules to perform this conversion in the interface layer.

One way ASRA can convert behavior action outputs to MAVLink messages is by setting the behavior controller's execute action method to convert the motor commands of behaviors' actions directly to MAVLink messages and then sending them to Ardupilot SITL. This is the simpler of the two methods but is less modular as it builds in Ardupilot SITL specific functionality into the controller. The other way ASRA handles this conversion, is by providing a separate software module as an interface between the controller and simulator. The controller's execute action function is set to send the behavior's actions over LCM which are received by the Autopilot interface module. This interface then converts the standardized LCM messages into messages understood by the currently selected simulator, Ardupilot SITL in this case. In both cases, an interface module is required to provide a conversion in the opposite direction, reading in Ardupilot SITL's MAVLink messages and sending out similar LCM messages to the agent core.

Ardupilot SITL's higher fidelity model had to be tested against the more time efficient particle simulator. Identical single agent scenarios were run in both simulators and compared. It was determined that the difference in scenario outputs was negligible. To make the two runs comparable, they were run with the same target positions and with perfect sensor accuracy to ensure the same targets were revisited in both cases. The difference in area covered was within 0.13%, but the main difference was

the execution time. The particle simulator completing the simulation in 8 seconds while Ardupilot SITL took 8 minutes. The minimal difference in area covered was acceptable given the particle simulator's immense time savings and the relatively low fidelity required for this research. For these reasons, the particle simulator was chosen for the experiment runs.

If Ardupilot SITL had been chosen for the experiment runs, there would still be value in being able to use the particle simulator for development. This ASRA implementation was able to easily switch between both simulators by adjusting a single configuration file parameter. This quick switch offered the particle simulator's fast startup and execution times for development and Ardupilot SITL's higher fidelity simulation environment for analysis runs.

4.5 WAS Simulation Performance

This section provides insight on the WAS mission execution at the various levels of cooperation by presenting agent position plots to aid in understanding agent behavior. Additionally, an overview of the intricacies of designing and tuning the utility function is provided.

The first mission that was developed was the single agent mission which took around 8 seconds to run in the particle simulator. This short run time and simple flight path made viewing plotted results after the simulator completed an acceptable way to debug the autonomy. Figures 30 and 31 show a sample of this output which depicts the flight path of a single agent mission which included ingress, search, confirm, and egress.

The cooperative missions' longer run times of roughly 30 seconds and more complex, multi-vehicle flight paths necessitated a live plotting capability. This provided faster feedback to the developer as the flight path was displayed during the simula-

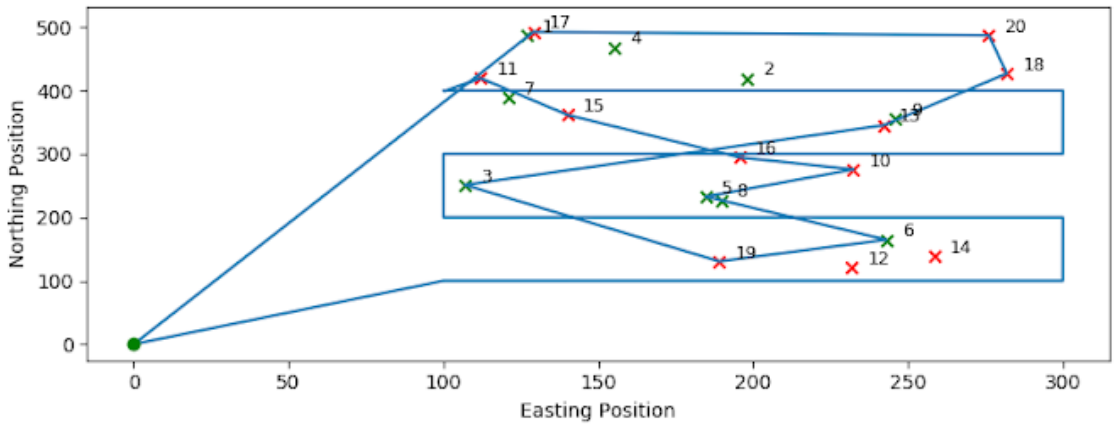


Figure 30. A single agent mission in 2D showing the search and confirm patterns.

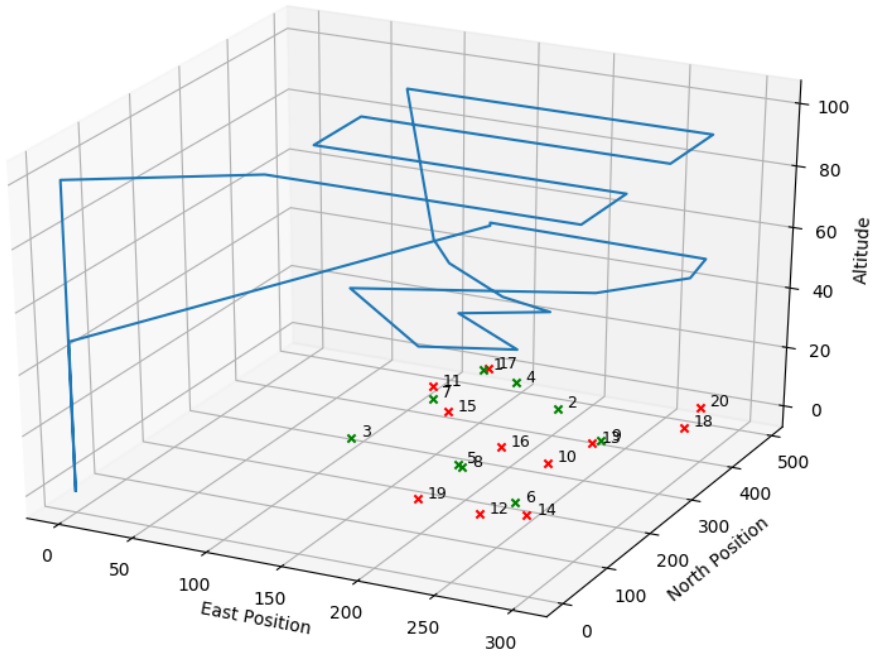


Figure 31. A single agent mission in 3D showing the search altitude above the confirm altitude and the targets below.

tion. When tuning the cooperative utility function, this live plotting was especially useful as it allowed the developer to walk through the simulation with the agent to better understand the cooperative decisions each agent was making.

Figure 32 shows the basic cooperation case scenario where the search area is split and agents search and confirm their sections individually. This basic cooperation case

was a simple extension of the single agent case, and only required an additional search area splitting algorithm run in the setup script during initialization. Figure 33 shows an example output of the extreme cooperation case where agents immediately confirm targets found by the other agent. The agent that completes search first, loiters in the center, waiting for the searching agent to find new targets. In this case with a high target density, it was common for one agent to follow the other, confirming its targets before being able to return to its own search pattern. The agents would then switch roles for the other half of the total search area. As expected, this behavior appears to be very inefficient. This case required the coordinator layer to be implemented as more information passed between agents than just the search area. The transition from basic cooperation to this case of extreme cooperation is when the majority of the inter-agent communication was developed.

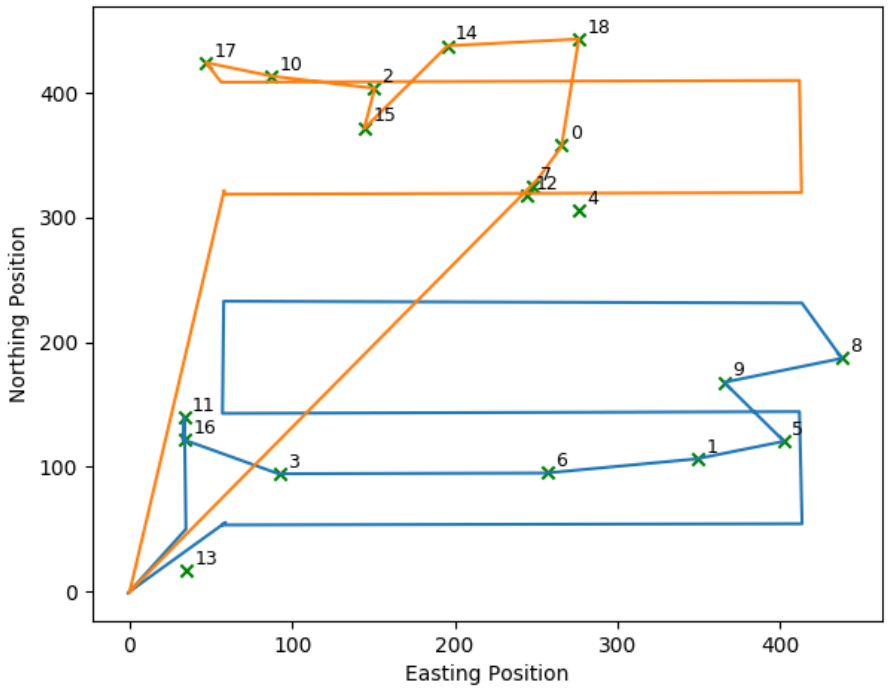


Figure 32. A 2D plot of the basic cooperation case where the search area was divided between agents and agents searched and confirmed their sections individually. Any unconfirmed targets were due to miss-classifications of the imperfect sensor.

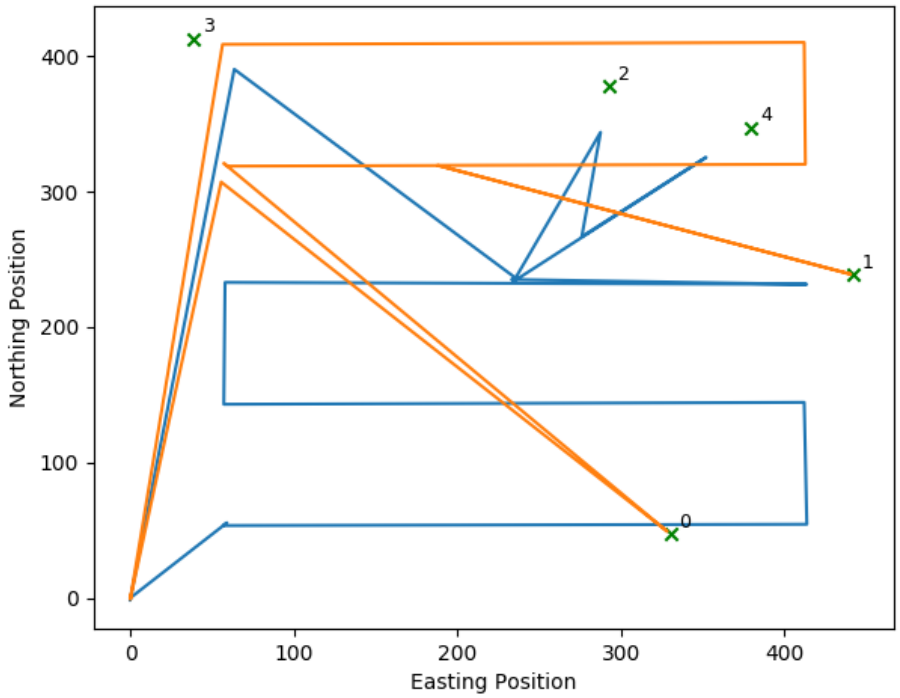


Figure 33. A 2D plot of the extreme cooperation case where agents immediately confirmed targets found by the other agent. If one agent finishes searching, it loiters in the center waiting to confirm any new targets the remaining, searching agent may find. Targets may not be completely flown over as the sensor field of view senses the target some distance away.

Tuning the utility function in the moderated cooperation case across all scenarios presented in the DOE experiment proved to be a challenge. Significant experimentation was performed to select utility function tuning parameters that yielded balanced performance across all scenarios. These tuning parameters consisted of the minimum utility threshold required to confirm a target, or the gains used in the weighting functions presented in the deliberator section above. Figure 34 shows an example of an unintended behavior of valuing target proximity with a low overall utility threshold, causing the agents to stop searching and confirm the target they just found individually. This did not exhibit the desired level of cooperative behavior for the moderated cooperation case.

Opposite to the immediate confirmations shown in Figure 34 is a refusal to leave the search pattern to confirm. This tendency to complete the search pattern before

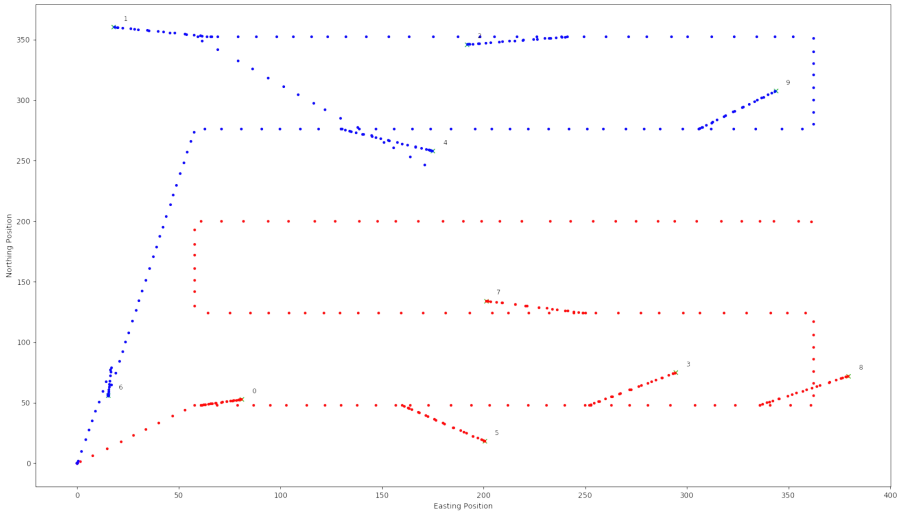


Figure 34. A truncated 2d plot of a three agent moderated cooperation case showing unintended behavior from the cooperative utility function that overly values target proximity with a low threshold, allowing agents to individually and immediately confirm a target they just found in search. Note that the marker spacing is based on the plot refresh rate and not the simulator step size.

confirming any targets can be seen in Figure 35 and is driven by a threshold set too high, keeping agents from deciding to search a target when in search. When agents completed search, the search weight parameter in the utility function was usually enough to push the utility value above the threshold. Depending the target locations, the agents may end up only confirming targets in their own section, exhibiting behavior very similar to the basic cooperation case shown in 32.

Figure 36 shows a middle ground between remaining in search until completed and immediately confirming targets that were found. The agents are willing to immediately confirm targets but also save up targets to confirm all at once such as targets 2,7, and 13. There happens to be minimal overlap between agents due to the target locations and confirmation path generation method.

The final utility function selected for the moderated cooperation case experiment runs performed similarly to that shown in Figure 35 where breaking from search to confirm was discouraged, but immediately confirming very near targets was allowed.

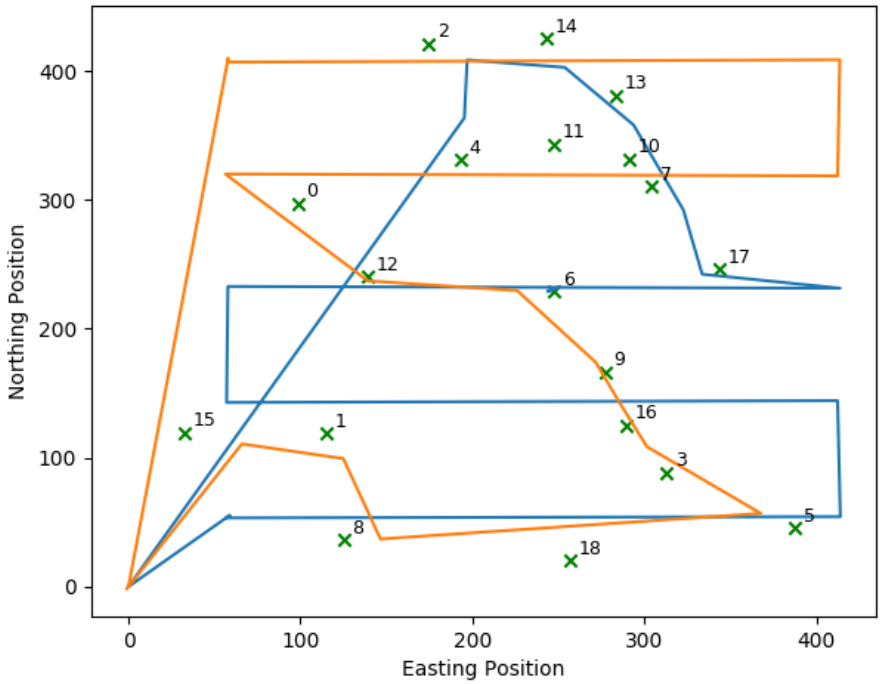


Figure 35. A 2d plot of two agents with a cooperative utility function tuned to discourage confirming during search. In this case, agents waited until search was completed to distribute and confirm targets in a somewhat efficient manner.

This minimized breaking from search early but allowed it when the target was very close. What was considered "close" likely should change based on search area, but this would create another tuning parameter. With a fixed idea of what "close" is, the agents stopped search to confirm more often in smaller search areas because the odds of targets being close to the search path is higher in this case. This set of tuning parameters was selected over the balanced utility function shown in Figure 36 due to the inefficiencies associated with breaking from search too often. These utility function parameters were selected through qualitative analysis of the cooperative performance but a rigorous experimental analysis to tune this utility function would likely improve its performance much further. Additionally, modifying the tuning parameters for each mission based on the expected characteristics of that mission, such as search area size or expected target density, would likely improve the utility function's performance across a wider range of scenarios.

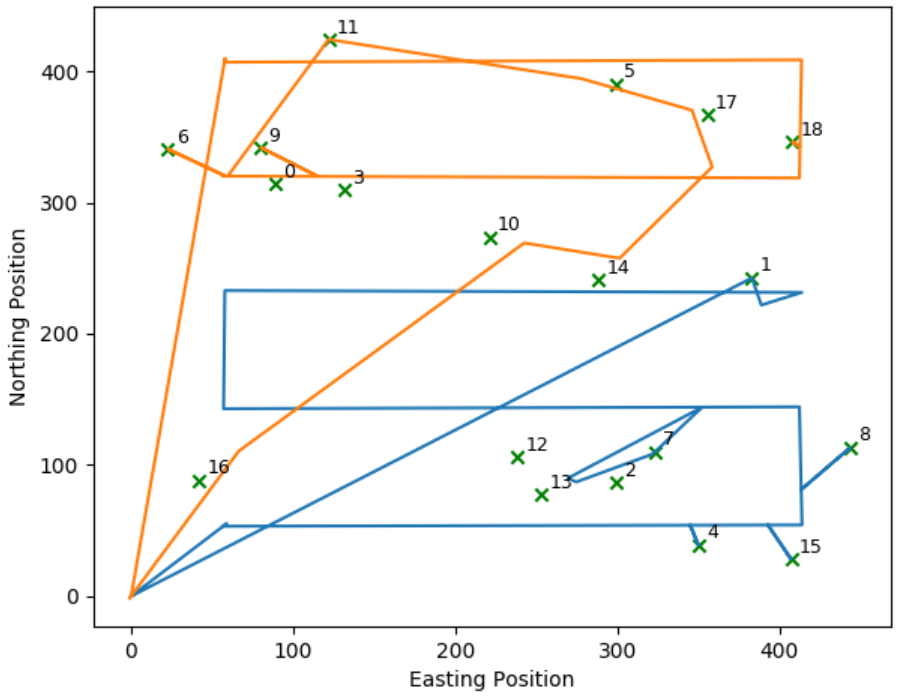


Figure 36. A 2d plot of two agents in the moderated cooperation case with a cooperative utility function tuned to balance immediate confirmations and a tendency to keep searching.

4.6 Design of Experiments Results

The design of experiments test of WAS was conducted with two replicates for both one vehicle and two vehicle simulations. This was chosen to both minimize simulation and analysis time, to estimate experimental error, and detect a lack of fit. However, to fully validate that two replicates are appropriate and more runs are not required for a valid model, a multiple comparison test was conducted to show that the two groups of replicates are statistically the same. All response variables passed the multiple comparison test, allowing the conclusion that two full runs of the FCCD design is adequate enough to capture experimental error and describe the system with a statistical model. The results are included in Appendix I and Appendix J. These results are generated in JMP and indicate that there is no statistically significant difference between replicates 0 and 1 for 80 runs, in both one vehicle operation and

two vehicle operation. This is shown by the blue point and line on the scatter plots. All response variables passed both Tukey HSD and Student's t comparison tests which test for a null hypothesis that both populations are the same. Tukey's multiple comparison test is more reliable than a single comparison student t test. Both results were given for completeness.

Responsiveness.

The responsiveness variable in one vehicle operation was recorded for each objective plan: takeoff, search, confirm, and land. The maximum for each run was saved as the responsiveness. These models and distributions are given in Appendix M. There is a statistically significant and valid model for one vehicle responsiveness as a function of search area. The studentized residuals and lack of fit tests indicate a valid model. The significant ANOVA indicates a statistically significant model. The responsiveness distribution shows a roughly normal response with responsiveness ranging from 0.355 μs - 0.707 μs. The model, distribution, and descriptive statistics are given in Appendix M.

In two vehicle operation, responsiveness was not correlated to any independent variable; parameter estimates were not statistically significant despite a significant ANOVA. In addition, the responsiveness distribution is not roughly normal. Although the residuals look acceptable and the regression is significant, this model is not suggested due to the roughly marginal significance of parameter estimates. Responsiveness ranges from 1.62 μs - 0.275 ms. The model, distribution, and descriptive statistics are given in Appendix N.

The importance of responsiveness will increase as the operational environment becomes more dynamic and constant planning and re-planning is required. In an application of urban WAS, the responsiveness metric could be more useful. The results

suggest the time required to actuate from an objective plan is essentially computation time. The results suggest that the system is responsive enough to objectives plans to assume real time planning and implementation, an important requirement of reactive architectures. The results are similar for one vehicle operation and are given in Appendix M.

One Vehicle Operation.

The FCCD design was applied to one vehicle operation. Statistical models were created to predict response variables using the input parameters given in Table 19. The models are given in Appendix K. These models can be used to predict future responses. The response *percent area covered* did not pass a lack of fit test and a model could not be found that fixed this issue. In lieu of *percent area covered*, the *percent detected* metric can be used to get an idea of how much area was covered. However, *percent area covered* will still be calculated in two vehicle operation because it is possible to detect all targets without finishing the search area assigned. All other response variables pass the lack of fit test and visual inspection of studentized residuals. As a result, model adequacy is held and valid conclusions can be made from these models. In two vehicle operation tests, the response variables will be narrowed down for optimization of multiple responses. *Perception accuracy* will capture error that impacts objective plans. As a result, Type I and Type II error for detection and confirmation will be eliminated for response optimization. *Percent detected* and *percent confirmed* can also be eliminated if a suitable model for *percent area covered* is found with the results. Mission time will remain a response of interest.

Table 19. Variables used in the model given in Appendix K

Input Parameters	Response Variables
Search Velocity	Percent Correct Detected
Detect real	Percent Correct Confirmed
Detect False	Mission Time
N True Targets	Detect Type I Error
N False Targets	Detect Type II Error
Search Area	Confirm Type I Error
	Confirm Type II Error
	Percent Detected
	Percent Confirmed

Two Vehicle Operation and Cooperation Levels.

The FCCD was applied to all three levels of cooperation: basic, moderate, and extreme. The data was compiled across all three levels and cooperation level was used as an input parameter. A statistical model was created to predict the response variables using the input parameters given in Table 20. The model results are given in Appendix L as JMP outputs. A FCCD was applied to capture the detected curvature in system response. Figure 37 shows an example of curvature in the perception accuracy response.

Table 20. Variables used in the model given in Appendix L

Input Parameters	Response Variables
Search Velocity	Percent Correct Detected
Detect real	Percent Correct Confirmed
Detect False	Percent Area Covered
N True Targets	Mission Time
N False Targets	Perception Accuracy
Search Area	
Cooperation Level	

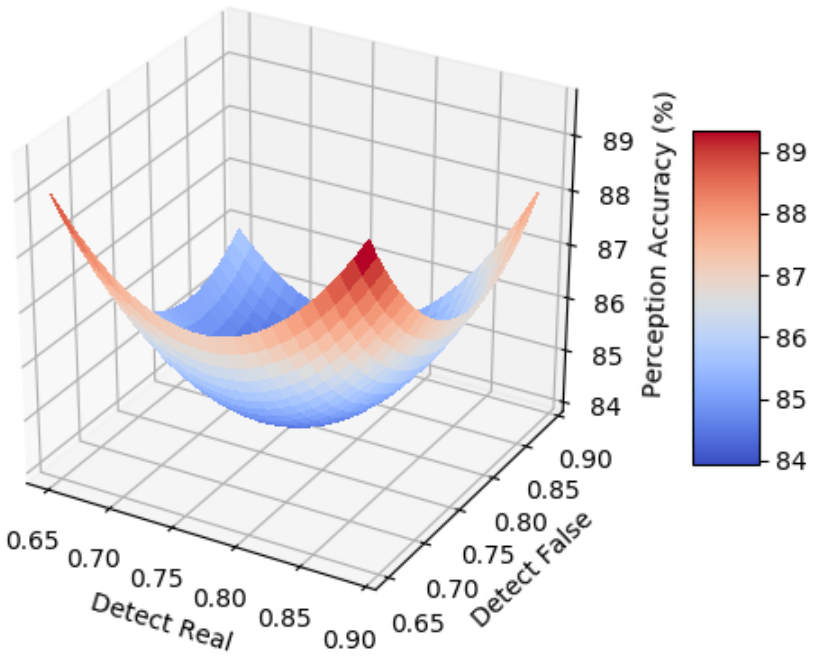


Figure 37. Response surface for perception accuracy as a function of sensor parameters at a high level of cooperation.

These models allow analysis of predicted performance. To demonstrate this ability, mission time is plotted as a surface and contour plot at three levels of cooperation in Figure 38 with vehicle parameters given in Table 21 as a function of the number of real and false targets. Analysis such as one shown in Figure 38 allows decision makers and engineers to determine operating conditions for a given mission, based upon mission and vehicle parameters such as number of true and false targets as shown. Figure 38 shows the system is more sensitive to the number of true targets than false targets in mission time response. The most optimal cooperation level to minimize mission time is the moderate case. This shows the utility function minimizes mission time for any given mission. However, extreme cooperation is counterintuitively more optimal

than basic cooperation. This could be a result of one or two things: more runs are require to adequately model extreme cooperation or there are unknown effects that need to be captured in order to more accurately inform intuition. To gain a better understanding while also factoring out the effect of target placement on extreme cooperation results, more replicates should tested. Collecting more data runs will either bring to light unknown dependencies of extreme cooperation or decouple the impact of target location on test results, making these results more explainable. Viewing various combinations of predicted response plots for all responses to determine optimal vehicle configuration would be impractical. To aid in implementing these models, multiple surface optimization through desirability functions allows quick analysis for vehicle configuration based upon user preferences.

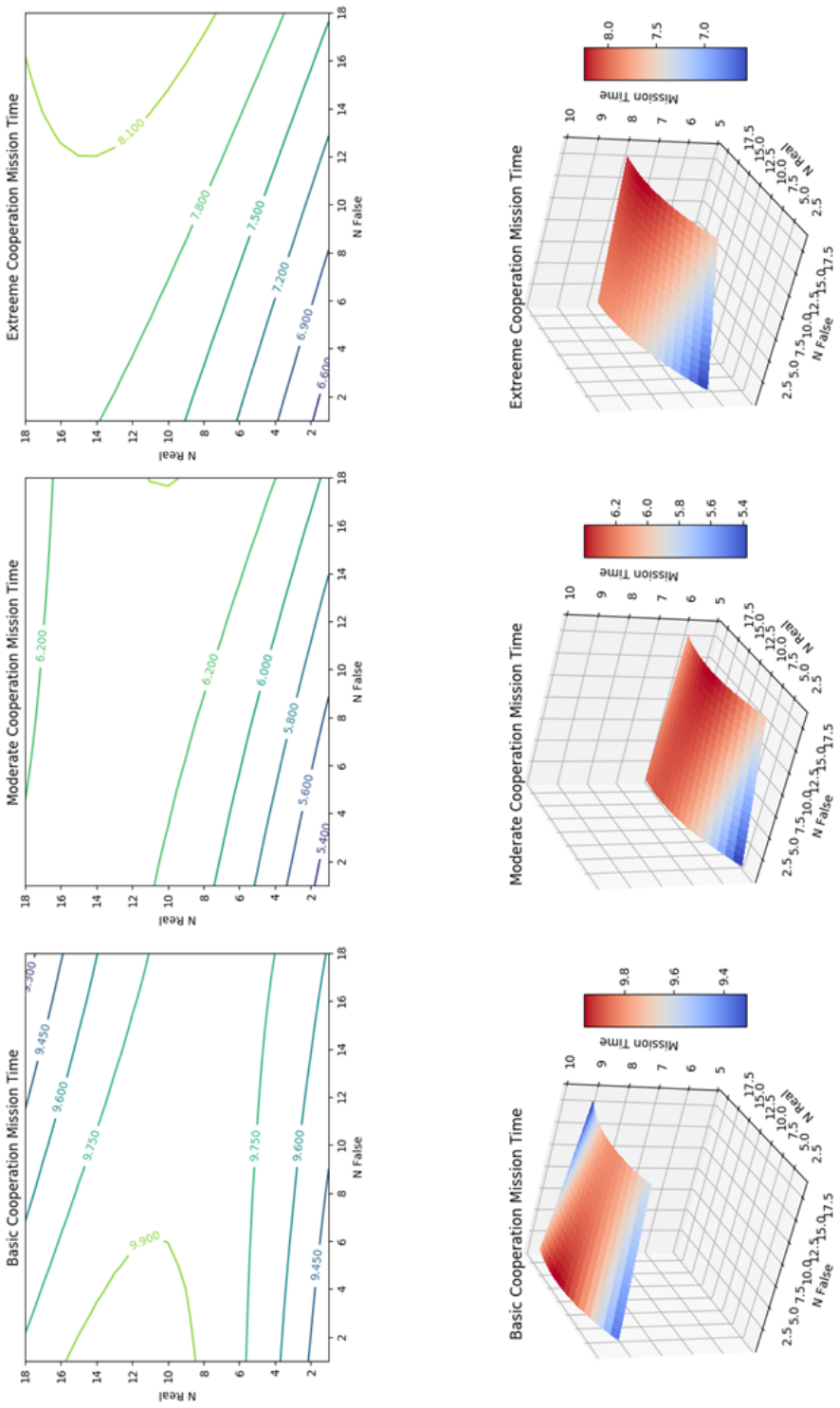


Figure 38. Response surface and contour plots of mission time for each level of cooperation by number of true and false targets.

Table 21. Constant conditions used to mission time in Figure 38

Input parameter	Setting
Search Velocity	10 m/s
Detect Real	0.80
Detect False	0.80
Search Area	450m x 450m

The response surface equations given in Appendix L are used to determine optimal configuration for given operational environment. Three example environments: target sparse, target moderate, and target rich are defined to demonstrate the ability to optimize vehicle configuration based off an expected environment. The settings used to define these conditions are given in Table 22. To determine optimal response, the range of input parameters tested in experimental design with a certain precision of change are inputted into the prediction equations for all response variables. These values are given in Table 23. The step size indicates how the values between the maximum and minimum are spaced in the range of tested values. The range of values tested for optimal response are implemented for each level of target density to produce an optimal configuration for a given expected operational environment.

Table 22. Definitions of target sparse, moderate density, and target rich environments

Environment	Search Area	N Real Targets	N False Targets
Target Sparse	450m x 450m	1	1
Moderate Density	700m x 700m	10	10
Target Rich	700m x 700m	19	19

Table 23. Range of values tested in response surface for desirability calculations

Input Parameter	Minimum	Maximum	Step Size
Search Velocity	5 m/s	10 m/s	1 m/s
Detect Real	0.65	0.9	0.01
Detect False	0.65	0.9	0.01
Cooperation Level	-1	1	1

To conduct optimization, a desirability function is calculated for each response tested. To calculate the desirability of each response, the disabilities given in Table 24 are implemented with a linear scale between bounds. Values outside specified values are given a desirability of zero. To optimize multiple responses, a weighted desirability function is implemented for each configuration with the weights given in Table 25. The weights given represent an example of multiple response prioritization. The maximum weighted desirability is chosen as the optimal configuration for a given environment. The findings for each environment are given in Table 26.

Table 24. desirability (d_i) for each response variable

Percent Correct Detected			
	Low	Medium	High
Response	60%	85%	110%
desirability, d_i	0.02	0.38	0.6
Percent Correct Confirmed			
	Low	Medium	High
Response	60%	80%	110%
desirability, d_i	0.05	0.4	0.55
Percent Area Covered			
	Low	Medium	High
Response	60%	85%	115%
desirability, d_i	0.05	0.15	0.8
Mission Time			
	Low	Medium	High
Response	5 mins	12 mins	21 mins
desirability, d_i	0.45	0.4	0.15
Perception Accuracy			
	Low	Medium	High
Response	35%	70%	105%
desirability, d_i	0.05	0.45	0.6

The calculations of desirability for optimal response was calculated using original Python classes and scripts. The code used to conduct optimization is given in Appendix O. The code allows the user to change the weights (data in Table 25), singular response desirability functions (data in Table 24), definitions of target density (data

Table 25. Weights used in the Desirability function

Response Variable	Desirability Weight, w_i
Percent Correct Detected	0.225
Percent Correct Confirmed	0.225
Percent Area Covered	0.15
Mission Time	0.15
Perception Accuracy	0.25

Table 26. Configurations that maximize desirability using information from Table 25 and 24

	Target Rich	Moderate Density	Target Sparse
Cooperation Level	Extreme	Extreme	Extreme
Search Velocity	8 m/s	7 m/s	5 m/s
Detect False	0.89	0.89	0.65
Detect Real	0.89	0.65	0.65
Desirability	0.6364	0.6549	0.6833

given in Table 22), and the range of values to evaluate desirability (data in Table 23). These results serve as an example for how response surface models can be practically used to configure vehicles for optimal performance. The configuration results can be inputted back into the response surfaces to predict performance. The response predictions for the optimal configurations are given in Table 27 for each operational environment.

Table 27. Predicted responses of optimal configuration using desirability criterion and response surfaces

Response	Target Rich	Moderate Density	Target Sparse
Correct Detected	64.70%	60.41%	60.86%
Correct Confirmed	64.80%	56.80%	35.13%
Area Covered	99.78%	104.07%	107.73%
Mission Time	11.48 mins	12.30 mins	11.80 mins
Perception Accuracy	79.42%	76.96%	96.94%

The predictions given in Table 27 were validated by testing these conditions in the simulation. The results and the associated percent difference are given in Table

28. A positive percent difference indicates an increase in simulation response from the predicted response. A negative percent difference indicates a decrease in simulation response than predicted. For the target sparse simulation, the false target was identified correctly and the true target was identified incorrectly. As a result, none of the targets were confirmed, explaining the zero percent correct confirmed and much lower mission time. The incorrect target identification is a result of Type II error. Since Type II error does not change the objective plan, the perception accuracy for the target sparse environment is 100%. Overall, one confirmation simulation for each case shows a well performing low fidelity simulation prediction model for each response, with the exception of percent correct confirmed. This demonstration implements optimization of mission and autonomy related response in order to select the best platform for a given mission and select optimal vehicle parameters. If more effort was applied to improving these models, the worst performing model, percent correct confirm, should be prioritized, followed by the mission time model.

In addition to surface confirmation runs, model adequacy was confirmed with lack of fit tests and visual confirmation of studentized residuals. Appendix L shows the studentized residuals and lack of fit test for all response variables. All response variables show adequate randomness in a horizontal band. The near-perfect randomness of residuals can be attributed to the uniform distributions used in generating target locations and sensor probability draws. The satisfactory inspection of residuals indicates the ANOVA assumptions hold. Additionally, each response lack of fit test resulted in rejection of the null hypothesis that a lack of fit in the model exists. This suggests there are no missing higher-order effects. Therefore, the results from the confirmation runs, lack of fit tests, and visual inspection of studentized residuals indicate an adequate statistical model, allowing valid conclusions from this statistical model.

Table 28. Optimal configuration confirmation simulation runs and percent difference to predicted responses

Response	Simulation Response	Percent Difference
Target Rich		
Correct Detected	71.05%	Δ +9.82%
Correct Confirmed	55.56%	Δ -14.26%
Area Covered	91.47%	Δ -8.321%
Mission Time	13.90 mins	Δ +21.08%
Perception Accuracy	80.00%	Δ +0.73%
Moderate Density		
Correct Detected	65.00%	Δ +7.60%
Correct Confirmed	20.00%	Δ -64.79%
Area Covered	104.59%	Δ +0.50%
Mission Time	13.94 mins	Δ +13.31%
Perception Accuracy	88.89%	Δ +16.32%
Target Sparse		
Correct Detected	50%	Δ -17.84%
Correct Confirmed	0%	Δ -100%
Area Covered	107.10%	Δ -0.59%
Mission Time	6.81 mins	Δ -47.635%
Perception Accuracy	100%	Δ +3.16%

Robustness.

The moderate cooperation simulation implemented a robustness test. In this test, one replicate of the FCCD design was run with a random seed of seven. This test was repeated but with error triggering one vehicle to RTL. To simulate the off-boarding of a vehicle due to sensor failure, one of the vehicles is given 60% fuel at take off, triggering RTL. The remaining vehicle is given the remaining area and replans accordingly. The down vehicle refuels and return as a confirmation agent if the mission is not yet completed. The use of consistent random seeds makes target locations identical for each run, removing the impact of target placement on response variables and making runs more directly comparable. The *mission time* and *area covered* between runs were evaluated to determine the extent to which the mission was diminished, using a percent difference. A negative value indicates degradation of performance (increased time, decreased percent area covered). A positive value indicates an improvement. Descriptive statistics of this test are given in Figure 39. The percent difference in *area covered* is at worse 4.76% degraded while mission time is at worse degraded by 7.35%. Area is degraded less than mission time because the RTL agent can return as a confirmation agent, helping to decrease time required to finish the mission alone. However, this is not the case for *percent area covered* because the RTL agent cannot enter search. When the remaining agent needs to RTL due to fuel, it is unable to refuel and search cannot continue. However, the RTL agent can continue confirming the remaining agents. The degradation percentages are rather low, indicating the system is adequately able to continue the mission of WAS.

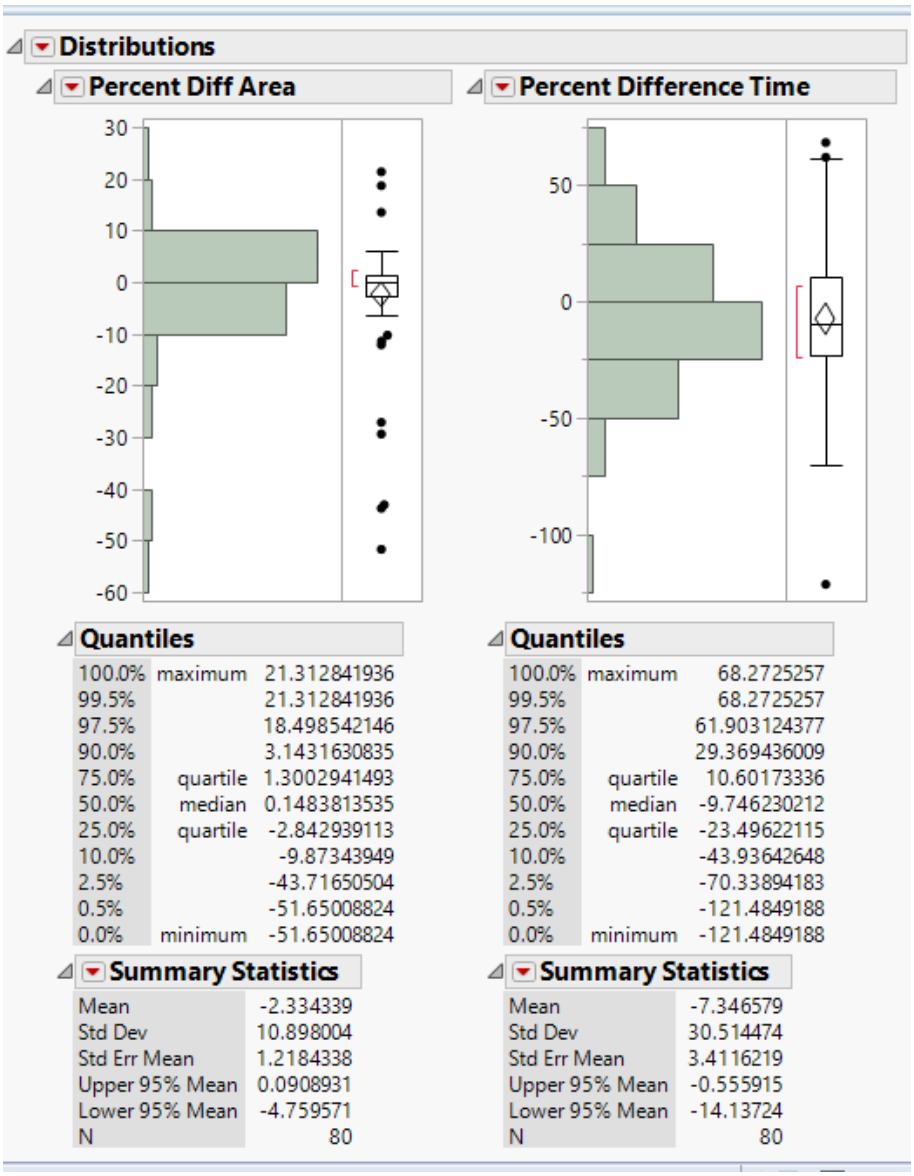


Figure 39. JMP output for percent difference in mission time and percent area covered

Application of DoE and RSM Results.

DoE and RSM allows identification of design parameters with the largest effect on system performance. Identification of these parameters allows prioritization of requirements. For example, if flying at a certain speed allows optimal system performance, the ability of the aircraft to fly at that given speed would be a high priority. Additionally, implementing DoE and RSM allows one to vary operational environ-

ments to determine an optimal system configuration for a given scenario. The optimal system configuration from statistical models for a given scenario can be implemented in real missions to achieve predicted performance.

4.7 Test Method for Autonomous Systems

This research implemented rigorous test planning, automated testing, and analysis for an autonomous system. The following steps are compiled to capture the test method utilized in this research:

1. Acquire system requirements for the system of interest. If there are no requirements for autonomy, write what is required for this autonomous system; conduct a literature review and talk to developers to determine what is required
2. Draft a problem statement from requirements that addresses scope and the type of problem to be investigated by the test plan
3. Create a system decomposition, often a work breakdown structure (WBS) and/or a context/domain model.
4. Write clear, concise, testable, traceable, and measurable test objectives that apply the problem statements to entire system, to include autonomy
5. Identify evaluation measures (response variables)
6. Identify required data for evaluation of responses
7. Identify sources of variation that could affect responses
8. Identify and understand all potential factors that could affect the responses

9. Select region of interest, factors to vary, and factor levels. If applicable, run a screening design to limit the test space and check region of interest for a full scale test.

10. Select experimental design based upon the above and how many runs can be afforded by the test infrastructure, timeline, and budget

11. Trace above to problem statement and test requirements

12. Perform the test, implementing automated testing where possible to streamline the process. Before moving on to the next step, ensure initial data allows the test objectives to be met.

13. Implement statistical methods according to the test plan to describe the system. Predictable models should be statistically significant, have no lack of fit, and pass visual inspection of residuals for ANOVA assumption verification

14. Select weights to represent the prioritization of response variables. Next, determine desirability functions for each response. The resulting Desirability equation aggregates overall value.

15. Conduct multiple-response optimization to determine operating parameters that maximum Desirability.

16. Make recommendations according to findings from test methodology.

4.8 Summary

This chapter presented the design and implementation of an autonomous WAS agent in ASRA as well as the tests and results performed on the simulation. MBSE principles aided in designing the autonomous agent and software implementation.

ASRA provided the capability to implement an autonomous system to accomplish this WAS scenario and offered existing software and modules which reduced development times. Qualitative scenario results showed the challenges associated with developing cooperative autonomous systems and tuning their behavior and interactions.

The DoE results allowed prediction for response variables, with exception of responsiveness and robustness. These prediction equations were used to optimize performance of two vehicle operation using Desirability functions. The bounds of inputs were selected for three target density levels: target sparse, medium density, and a target rich environment. The test methodology implemented in this research is presented for autonomous systems, providing test and evaluation professionals a general, domain agnostic outline of steps for testing an autonomous system.

V. Conclusion

5.1 Overview

Chapter V provides conclusions in light of the overall research that was accomplished. The research questions presented earlier are answered based on the work performed to this point. Finally, important lessons learned throughout this research development are recorded along with future work that could be done to extend this research.

5.2 Research Findings

What additions to ASRA need to be made to implement a new WAS application?

ASRA MBSE models were already developed to the point that the WAS application could be accurately modeled. The software instantiation of ASRA, used as the basis for this research, also included the majority of the functionality for a 3 layer architecture with a basic sequencer, UBF controller, and behavior library. Only slight modifications to the sequencer and controller were required, with the exception of new behaviors unique to this scenario. The basic sequencer required some additions to sequence objective plans with multiple individual objectives and hand appropriate behaviors to the controller one at a time. Additionally, communications between the sequencer and controller were limited and thus extended to provide the sequencer insight on the status of the controller. Additional messages were also added to provide two way communication between layers. These were fundamental items needed for any three layer autonomy implementation.

To implement the WAS scenario specifically, more substantial additions were also required. A new deliberator layer was implemented as a state machine to handle high

level planning and a coordinator was added to handle the communication between agents. These modules are fundamentally less universally applicable to new mission types and as implemented, are specific to the WAS mission. New WAS specific behaviors and their associated activation paths had to be developed. Many of the behaviors that existed in the behavior library could be reused with only slight modifications, while others such as the FlySearchPattern behavior required new development. As the behavior library grows, the number of needed behaviors that do not exist in the library should decrease. Finally, additional messages were required, some specific to the WAS mission, but others to improve the overall communication between layers.

How does ASRA enable reuse of the similarities that exist in autonomous and cooperative systems?

ASRA's MBSE models are designed be reusable as components can be saved in Cameo Systems Modeler and used for new designs. These models describe different levels of abstractions so new autonomous systems may share a high level model while a lower level model may address the differences between systems. As more autonomous systems are modeled with ASRA, the library of modeled components will expand and become more likely to contain the models required in new applications.

This specific software implementation is highly reusable for future three or four layer designs, but less for completely new agent core architectures. The sequencer and controller layers are more universally useful than the deliberator and coordinator. Except for new additions of more advanced features, modifying the sequencer in new applications is likely not necessary. For example, moving from the single agent to multi agent scenarios required no modifications to the sequencer. For the controller, the behavior library allows reuse of existing behaviors directly, with slight modifications, or as guides to develop new behaviors. The main contents of the deliberator and coordinator are tailored to the WAS mission specifically, making them less di-

rectly usable by new applications. The interface with other layers that exists in these higher layers is still useful as a basis when implementing these layers in new systems.

How does ASRA support the variations of autonomous and cooperative systems?

ASRA models are designed to handle a wide range of autonomous system designs thanks to the modularity of its architecture. Defined interfaces allow module contents to be swapped yet still work in concert with other modules, but these interface definitions are not fully defined. Further defining these interfaces, would improve the model's ability to handle any new type of autonomy to be accurately modeled and implemented. The majority of variations in autonomous systems can be found in the agent core within the autonomy layer. ASRA's hardware interface layer standardizes the interface between the hardware layer and the agent core, allowing compatibility with many agent core designs such as a machine learning implementation or reactive controller.

This specific software implementation of ASRA is tailored to a layered architecture with a UBF controller. By simply adjusting the behavior set, this architecture is very flexible to new implementations. If a completely new agent core is desired, the existing software for the layered architecture would be replaced with this new autonomy architecture. In this case, the existing LCM messaging system could be reused to handle the interface with this new architecture. This would provide an interface with the existing perceptor and simulator modules. Any additional perceptor or simulators could also be implemented using the existing interface.

What are effective and efficient test methods for autonomy?

Effective and efficient test methods are rooted in application of test planning and design of experiments. Effective test methods are those which implement test planning to answer relevant questions for a given system. Autonomy should heighten

performance of the overall mission. As a result, mission tasks are allocated to autonomy, and test objectives are made for these tasks. This was done in the work breakdown of the system and its use to create test objectives. This process allows for an effective test that answers relevant questions. To ensure test efficiency, design of experiments can be implemented to test a design space of interest. Design of experiments allows a systematic approach of testing a design space. Applying design of experiments with test planning allows the required questions to be answered with efficient use of resources.

How should the test space be limited given a specific mission space?

The test space can be limited by utilizing factor selection and limitation methods. In this research, potential factors were identified through the use of screening designs. These factors were limited by holding some of these factors constant throughout the experiment in order to limit the space. Limiting the space allowed a full factorial to be tested with center points and experimental replication of design. The selected factors were checked with a screening design to confirm only significant factors were selected for a final design of experiments. Limiting the factor space saved time and resources, allowing multiple trials. In this research, factor limitation was a trade for implementing a full factorial in simulation rather than a fractional factorial. Using a fractional factorial can be efficient but aliasing exists. Having aliasing means that the test is unable to separate all effects in the test. With a full factorial, the analyst has no coupling between factor effects but more runs are required than a fractional design.

What are valid and useful measures of autonomous systems?

Measures of autonomy were derived from reactive architecture requirements of autonomous systems. Useful measures are informative to the user and can be measured. Valid measures are those which are based in system requirements. In this research,

the requirements of responsiveness, robustness, and perception accuracy were quantified and tested. The extent to which these requirements were met can be used to determine the suitability of autonomy for the given system.

In this research, the *responsiveness* metric could be improved for future research. However, it has the potential to be better quantified; it is a valid metric because it is based in reactive control requirements for an autonomous system. Suggestions made in future research will allow it to be more useful to users.

Robustness gave insight into degradation of the systems ability to cover an assigned area and the time required to finish a mission. It is a valid metric because it is based in system requirements. It is useful because it measurable and gives some insight but is not predictable. The results give a range of percent difference values for *mission time* and *percent area covered*. However, the *robustness* metric is somewhat time dependent on error. This is one area for improvement of this metric.

Perception accuracy gave insight on the impact of sensor error on objective plan selection. The percent to which sensor error had no impact on objective plan was recorded in each run. This response is predictable with a significant and valid statistical model. As a result, this metric could be considered the most useful because it is informative to the user, measurable, and predictable. This metric is valid because it is tied to a consequence of using sensors. Multiple sensors is one of the requirements from (Brooks, 1986). Brooks references consequences of using multiple sensors, one of which being autonomous systems perceive and operate under error. This metric gives insight on the extent to which error does not impact objective plan selection, informing the user on the suitability of the sensor used for autonomous logic.

5.3 Lessons Learned

Cooperative Utility Function Design

Implementing a cooperative utility function and achieving reasonable results presented a challenge. Not only must parameters of the utility function be tuned, the parameters themselves must be selected from a pool of many potential aspects that affect cooperation. While qualitatively good results could be achieved for a specific set of scenario conditions, generating reasonable performance from the utility function across all scenario conditions with only qualitative testing was difficult. To address this, the utility function could be designed to include knowledge of the scenario, such as search area size or expected target density. This could provide the performance achieved when tuning to a specific scenario, across the range of scenarios the system may face. Additionally, a quantitative testing method could be implemented to aid in tuning and selecting utility function parameters. Under certain tuning parameter conditions, the utility function resulted in indecisiveness where the agent could not decide if it should confirm a target or not so over the course of making one decision and then the opposite decision, the agent would position itself directly in between the goal of each position. This behavior was tuned out but could also be avoided by requiring agents to stick to a decision.

LCM Usage

LCM's main handle function should be run until it returns 0 to ensure all new messages' callback functions are executed. Utilizing *lcm-logplayer gui* with *lcm spy* is quicker than using *logreader* code created to inspect what happened during the flight. When working on sprints two and three, file size was an issue and the log files stopped recording. After a lot of testing, it was found that subscribing to required messages and running the code in a Linux terminal solved the issue. During this process, using *lcm-logplayer gui* with *lcm spy* was key because the files were so large. Tests could not be conducted in a smaller form because the issue was dependent on file size. Running the simulation took awhile and inputting large files into *logreader*

took even longer. Additionally, running code in Pycharm at large scale only created more problems so all test runs were executed in the Linux terminal.

5.4 Future Work

WAS Scenario

This wide area search scenario could be extended to study the cooperative benefit of three or more agents that can dynamically join and leave the network. This implementation can accommodate more than two agents in many areas but due to development time, the functions dealing with search area distribution between agents was limited to two agents. Additional search patterns might also yield interesting results along with the study of moving targets and multiple sensor types distributed among agents. Extending the utility function to include information about the potential scenario could improve its performance across a wider range of missions. Finally, performing more rigorous tests on the utility function to determine optimal tuning parameters could yield improved results.

ASRA

This software implementation of ASRA utilized a static sequencer with basic functionality. This could be improved by implementing the dynamic sequencer described in Peterson et al. (2011). Initial steps in that direction could be to implement a resource monitor in the sequencer that condenses the list of viable behaviors to those that can actually be implemented with the agent's current set of sensor information and system controls. Additionally, monitoring the current task's post conditions in the sequencer would provide a more robust way to determine task completion, especially with complex behaviors.

This research sent additional required information to the behavior such as a list of waypoints to achieve from the deliberator. This limited the type and number

of objectives that an objective plan could include because the deliberator does not know the current task being performed. By sending all required information with the objective plan to the sequencer, the sequencer can then provide the required information to the correct behavior.

The particle simulator could be extended to provide support for a wind or weather effects, obstacles, or additional vehicle types such as fixed wings or ground vehicles. This would provide some of the useful features of Ardupilot SITL in the lightweight particle simulator. Finally, successfully using LCM in multiple processes would allow for future layered architectures to run concurrently.

ASRA models currently lack the component interface definitions required for complete modularity of components. Defining these interfaces would extend the model's ability to guide the creation of new software modules, such that they interface with existing components. This serves to achieve ASRA's goal of rapid prototyping by ensuring component interoperation between existing and new autonomy components.

Responsiveness

In this research, responsiveness was the least useful as currently defined. In a wide area search application, responsiveness was effectively processing time. However, in a urban environment with multiple agents, this became more important for vehicle operation. Additionally, based upon the architecture chosen, the first timestamp was taken from the creation of objective plans. However, the ability to compare actuation time to objective plans trigger would be more informative. This would allow responsiveness to external stimuli to be captured rather than responsiveness to objective plan creation.

Robustness

In this research, *robustness* was not a predictable response. Instead, descriptive statistics were used to show the range of values observed over all runs. This metric was

time dependant on the amount of fuel given to one agent. In the future, this trigger to RTL could be random with more runs to get a better sample that is not dependent on time. Another area of research is robustness to operational environment. For example, the performance of an agent configured for a target rich environment operating in a target sparse environment can be used to determine the robustness to operational environment.

Optimization

Optimization method included desirability functions which used weights. A common next step which was not included in this research is weight sensitivity analysis. If the weight is changed by a small amount, this analysis looks at how desirability changes relative to weight change. If it is sensitive to weight change, more thought and attention should be applied to selection of the weights. Since this research used optimization to demonstrate the tool and practical use of statistical models, this area was not explored. Another area of research for optimization would be confirming optimal performance settings in a higher fidelity simulation such as SITL and then in flight test. These tests would help extrapolate the model from simulation to real world to given more accurate results.

Model Based Systems Engineering (MBSE)

This research included coded elements from a systems engineering model. The next process would be to update the model to ensure consistancy between the model and coded elements of that model. Current ASRA models stop short of defining the software elements within components and could be extended to fully define the software elements used in this ASRA implementation. Implementing an accurate system model to the level of software code improves documentation, decreasing acclimation and development time.

5.5 Final Thoughts

This research implemented a cooperative wide area search and confirm scenario, building on an existing ASRA software instantiation. By combining MBSE and autonomous system design concepts, an autonomous system was fully implemented in ASRA. Through this development, ASRA's software and models were extended, easing the development of future ASRA implementations through its reusable and modular design.

This research expanded test and evaluation of autonomous systems through test planning and metrics specific to autonomy. Automated scripts tested the system's main tasks and functions, as well as the use of autonomy to complete those tasks. The results allow insight for the entire system, including autonomy. This research baselined the use of *responsiveness* and *robustness* metrics, with *perception accuracy* shown to be a predictable system response. The statistical models created from test data allows a user to optimize configuration for a given environment based upon user defined weights and desirabilities. This research enables confirmation of system configuration responses in SITL and flight test for predicted optimal performance. Lastly, the fully implemented four layer architecture along with integrated testing procedures allows for an efficient digital twin implementation for a variety of military designs.

Statistical models allow prioritization of requirements for system design and development. Requirements can be prioritized by largest effect on system performance in order to make the largest system impact with research and development time and money. This ability allows developers and managers to make more significant impacts on performance, budget, and schedule.

Finally, this research delivered a test methodology for autonomous systems. This methodology allows testers to make recommendations and conclusions about au-

tonomous systems, informing decision makers about optimal operating conditions. The ability to rapidly prototype (Zacharias, 2019), test, and evaluate (Defense Science Board, 2012) autonomous systems allows for the fielding of autonomous systems to gain a military advantage, a primary goal of the 2018 National Defense Strategy (Mattis, 2018).

Appendix A. LCM Message Descriptions

Table 29. 12 LCM messages were required to provide the necessary communication between layers.

Message Name	Sender	Recipient	Description
TaskPlan	Sequencer	Controller	Task list and current task number
TaskStatus	Controller	Sequencer	Current task name and status
PlatformPos	Simulator	Controller Deliberator Perceptor Coordinator	Agent position
GoalPos	Deliberator	Controller	Current goal position for behavior to achieve
WaypointPos	Deliberator	Controller	List of waypoints forming search or confirm pattern
FuelStatus	Simulator	Deliberator Coordinator	Agent fuel remaining
ObjectivePlan	Deliberator	Sequencer	List of objectives to achieve current goal
RePlan	Sequencer	Deliberator	Current Objective Plan name, completion or failed solution status
DelInfo	Deliberator	Perceptor Coordinator	Commands to control perceptor and agent information for Coordinator
TargetList	Perceptor	Deliberator Coordinator	List of targets perceptor has found
CoordCmds	Coordinator	Deliberator Perceptor	Information on other agents, loiter value, offline agents
AgentTargs	Coordinator	Other Coordinators	Agent position, state, compiled target list, fuel status

Appendix B. Sample Behavior Code

Sample Takeoff Behavior

```
14 class Takeoff(Behavior):
15     """
16     Takeoff to a specified target altitude at current horizontal position.
17     """
18 
19     def __init__(self, weight=1.0):
20         super().__init__(weight)
21         self.pid = PID()
22 
23     def gen_action(self, state_block):
24         """
25         :param state_block:
26         :type state_block: StateBlock
27         :return: A unit vector of the velocity in the navigation frame (NED)
   pointing in the negative down direction.
28         :rtype: Action
29         """
30 
31         # Initialize action output
32         action = MultiRotor.Move(behavior_weight=self.weight, behavior_id=
   self.id)
33 
34         # Get sensor inputs
35         platform_pos = state_block[self.state_ids[0]].position
36         target_pos = state_block[self.state_ids[1]].position
37         delta_alt = target_pos.z - platform_pos.z
38 
39         # Apply PID
40         u = self.pid.evaluate(delta_alt)
41 
42         # Scale behavior output between 0 to 1
43         if la.norm(u) > 1.0:
44             act_cmd_z = u / la.norm(u)
45         else:
46             act_cmd_z = u
47 
48         # Pack action
49         action.actuators.vertical.motion_type = 'velocity'
50         action.actuators.vertical.z = act_cmd_z
51         if round(platform_pos.z) % 25 is 0:
52             print('current alt is', platform_pos.z)
53         if abs(delta_alt) <= 3:
54             print('at target altitude in takeoff behavior')
55             action.complete = True
56         else:
57             action.complete = False
58 
59         return action
60 
```

```
61      def set_state_ids(self, platform_position, target_position):
62          self.state_ids = [platform_position, target_position]
```

Sample FlySearchPattern Behavior

```
186 class FlySearchPattern(Behavior):
187     """
188     Fly through waypoints given in waypoint list.
189     """
190 
191     def __init__(self, weight=1.0):
192         super().__init__(weight)
193         self.pid = PID()
194         self.current_wp_num = 0
195 
196     def gen_action(self, state_block):
197         """
198         :param state_block:
199         :type state_block: StateBlock
200         :return: A unit vector of the velocity in the navigation frame (NED
    ) pointing towards the next waypoint.
201         :rtype: Action
202         """
203 
204         # Initialize action output
205         action = MultiRotor.Move(behavior_weight=self.weight, behavior_id=
    self.id)
206 
207         # Get sensor inputs
208         platform_pos = state_block[self.state_ids[0]].position
209         path = state_block[self.state_ids[1]].waypoint_list
210         target_pos = path[self.current_wp_num]
211 
212         # Calculate distance from target position
213         delta_ned = np.zeros(3)
214         if isinstance(target_pos, GeodeticPosition) and isinstance(
    platform_pos, GeodeticPosition):
215             delta_ned[0] = delta_lat_to_north(target_pos.latitude -
    platform_pos.latitude,
216                                               platform_pos.latitude,
    platform_pos.altitude)
217             delta_ned[1] = delta_lon_to_east(target_pos.longitude -
    platform_pos.longitude,
218                                              platform_pos.latitude,
    platform_pos.altitude)
219             delta_ned[2] = target_pos.altitude-platform_pos.altitude
220         elif isinstance(target_pos, LocalLevelPosition) and isinstance(
    platform_pos, LocalLevelPosition):
221             delta_ned[0] = target_pos.x - platform_pos.x
222             delta_ned[1] = target_pos.y - platform_pos.y
223             delta_ned[2] = target_pos.z - platform_pos.z
224         else:
225             raise RuntimeError('Target and platform positions must be in
    same frame and same container type')
```

```
226
227         # Apply PID
228         u = self.pid.evaluate(delta_ned)
229
230         # Scale behavior output between 0 to 1
231         if la.norm(u) > 1:
232             act_cmd_ned = u / la.norm(u)
233         else:
234             act_cmd_ned = u
235
236         # Pack action
237         action.actuators.horizontal.motion_type = 'velocity'
238         action.actuators.horizontal.x = act_cmd_ned[0]
239         action.actuators.horizontal.y = act_cmd_ned[1]
240         action.actuators.vertical.motion_type = 'velocity'
241         action.actuators.vertical.z = act_cmd_ned[2]
242
243         if la.norm(delta_ned) <= 3:
244             if self.current_wp_num == len(path)-1:
245                 # all waypoints have been routed through
246                 action.complete = True
247             else:
248                 # move to next waypoint
249                 self.current_wp_num += 1
250         return action
251
252     def set_state_ids(self, platform_position, waypoint_position):
253         self.state_ids = [platform_position, waypoint_position]
```

Appendix C. Controller Main Function Code

Controller Main Function

```
67 def main(self):
68
69     while self.__state_block.lcm_update_states():
70         pass
71
72     task_plan_state = self.__state_block[self.__task_plan_state_id]  # total task plan right out of sequencer
73     task_status_state = self.__state_block[self.__task_status_state_id]  # current task
74
75     if task_plan_state.plan:  # keep doing original task even if finished
76         root_behavior = self.get_root(task_plan_state.plan[task_plan_state.sequence_no])
77         self.controller.set_root_behavior(root_behavior)
78
79         if task_plan_state.plan[task_plan_state.sequence_no].task != task_status_state.name or \
80                 task_status_state.unique_id != task_plan_state.unique_id:
81             task_status_state.name = task_plan_state.plan[task_plan_state.sequence_no].task
82             task_status_state.unique_id = task_plan_state.unique_id
83             task_status_state.set_status_in_progress()
84             self.__state_block.send_state(self.__task_status_state_id)
85
86         # Get and execute action
87         cur_action = self.controller.gen_action(self.__state_block)
88         self.controller.execute_action(cur_action)
89
90         # Set TaskStatus state
91         if cur_action.complete is not None:
92             if cur_action.complete:
93                 task_status_state.set_status_finished()  # set current task to finished
94                 task_status_state.name = task_plan_state.plan[task_plan_state.sequence_no].task
95                 self.__state_block.send_state(self.__task_status_state_id)
96             elif not cur_action.complete:
97                 task_status_state.set_status_in_progress()  # set current task to in progress
98                 task_status_state.name = task_plan_state.plan[task_plan_state.sequence_no].task
99                 self.__state_block.send_state(self.__task_status_state_id)
```

Appendix D. Sequencer Main Functions Code

```
117 def handle_plan(self):
118     """
119     Handles the current task plan by dispatching new tasks to controller and
     monitoring task state in controller.
120     """
121     task_plan_state = self.__state_block[self.__tp_state_id]
122     replan_state = self.__state_block[self.__replan_state_id]
123     task_status_state = self.__state_block[self.__task_status_state_id]
124 
125     if task_plan_state.is_plan_new() \
126             and task_status_state.name == task_plan_state.plan[
    task_plan_state.sequence_no].task:
127         task_plan_state.set_status_in_progress()
128         self.__state_block.send_state(self.__tp_state_id)
129 
130     elif task_plan_state.is_in_progress() \
131             and task_status_state.name == task_plan_state.plan[
    task_plan_state.sequence_no].task:
132         if task_status_state.is_finished():
133             task_plan_state.set_status_finished()
134             if task_plan_state.sequence_no < len(task_plan_state.plan)-1:
     # if whole task_plan is finished
135                 task_plan_state.sequence_no += 1
136 
137             self.__state_block.send_state(self.__tp_state_id)
138 
139     elif task_plan_state.is_finished() and \
140             task_status_state.name == task_plan_state.plan[task_plan_state.
    sequence_no].task and \
141             task_status_state.is_finished():
142         replan_state.set_status_op_finished()
143         replan_state.unique_id = task_plan_state.unique_id
144         replan_state.replan_reason = task_plan_state.plan[task_plan_state.
    sequence_no].task
145         self.__state_block.send_state(self.__replan_state_id)
146 
147 def main(self):
148     """
149     Main function to be called in loop, either by executive's run or setup
    script).
150     Handles new objective plans, converting to task plans or notifying
    deliberator of failed task plan generation
151     """
152     while self.__state_block.lcm_update_states():
153         pass
154 
155     task_plan_state = self.__state_block[self.__tp_state_id]
156     op_state = self.__state_block[self.__op_state_id]
157     replan_state = self.__state_block[self.__replan_state_id]
```

```
158
159     if self.old_op_id != op_state.unique_id:
160         plan = self.gen_plan()  # Generate new plan
161         if plan is not None:
162             if plan != []:
163                 task_plan_state.plan = plan
164                 task_plan_state.sequence_no = 0
165                 task_plan_state.unique_id = op_state.unique_id
166                 replan_state.unique_id = op_state.unique_id
167                 self.__state_block.send_state(self.__tp_state_id)
168
169                 replan_state.set_status_op_in_progress()
170                 replan_state.replan_reason = task_plan_state.plan[
    task_plan_state.sequence_no].task
171                 replan_state.unique_id = task_plan_state.unique_id
172                 self.__state_block.send_state(self.__replan_state_id)
173
174     if task_plan_state.plan:
175         self.handle_plan()
176     else:
177         replan_state.set_status_op_failed()
178         self.__state_block.send_state(self.__replan_state_id)
```

Appendix E. Deliberator Main Function Code

Deliberator Main Function

```
830 def main(self):
831 
832     while self.state_block.lcm_update_states():
833         pass
834 
835     # get coordination info during initialization
836     while not self.got_coordination_info:
837         if self.state_block[self.config.coordinator_commands_state_id].
    boundary_list != []:
838             self.boundary = self.state_block[self.config.
    coordinator_commands_state_id].boundary_list[self.config.agent_id]
839             self.got_coordination_info = True
840 
841     # check on fuel level and trigger egress state if necessary
842     if not self.check_low_fuel():  # returns True if out of fuel and RTLing
843         self.save_agent_last_search_location()
844         self.check_loiter_utility()
845         if 'RTL' not in self.state_history:
846             self.check_redistribute_search()
847         if not self.machine.is_RTL():
848             self.update_confirm_list()  # constantly keep an optimized
    route through targets needing confirmed
849             self.do_confirm_transition()  # update the waypoint list and
    change state to confirm if necessary
850 
851     replan_state = self.state_block[self.__replan_state_id]
852 
853     # startup state
854     if replan_state.is_op_failed():
855         if self.machine.is_initial():
856             print('Initializing state machine...')
857             self.send_waypoint_list(states.WaypointListNED(waypoint_list=
    self.gen_search_pattern(),
858                                                            channel=self.
    config.waypoint_channel))
859             self.return_to_search_location = self.state_block[self.config.
    waypoints_state_id].waypoint_list[0]
860             self.send_target_position(states.PositionNED(position=self.
    return_to_search_location))
861             self.handle_state('ingress')  # to ingress state
862 
863     # state transitions based off a finished and matching replan to
    objective plan
864     if replan_state.is_op_finished() and self.matching_replan():
865         if self.machine.is_ingress():
866             self.handle_state('search')  # to search state
867 
868         elif self.machine.is_search():
869             self.search_complete = True
```

```
870             self.send_deliberator_info()
871
872             if self.confirm_list == []:
873                 confirm_pattern = self.gen_confirm_pattern_with_alt(alt=
    self.config.confirm_alt)
874             else:
875                 confirm_pattern = self.gen_confirm_pattern_with_alt(alt=
    self.config.confirm_alt, target_list=self.confirm_list)
876
877             if confirm_pattern == []:  # if no targets to confirm
878                 self.handle_state('egress')
879             else:
880                 self.send_waypoint_list(states.WaypointListNED(
    waypoint_list=confirm_pattern,
881                                                                channel=
    self.config.waypoint_channel))
882                 self.handle_state('confirm')
883
884         elif self.machine.is_confirm() and self.search_complete:
885             self.handle_state('egress')
886
887         elif self.machine.is_coop_confirm():
888             if 'search' not in self.state_history or self.state_history[-2]
     is 'search':
889                 self.send_waypoint_list(states.WaypointListNED(
    waypoint_list=self.gen_search_pattern(),
890                                                                channel=
    self.config.waypoint_channel))
891                 self.handle_state('search')  # to search state
892
893         elif self.machine.is_loiter():  # not actually going to be reached
    since the loiter behavior never ends
894             self.handle_state('egress')
895
896         elif self.machine.is_egress():
897             self.handle_state('landed')  # to landed state
898
899         elif self.machine.is_RTL():
900             self.handle_state('landed')
```

Appendix F. Perceptor Sense Function Code

Perceptor Sense Function

```
102 def detect_target(self):
103     """
104     Determine if any targets are in current field of view and simulate a
    sensor with error
105     """
106 
107     location = self.state_block[self.config.platform_pos_state_id].position
108     target_list = self.state_block[self.config.target_list_state_id].
    targets
109 
110     # get FOV radius on ground for current altitude
111     fov_radius = (abs(location.z) * math.tan(math.radians(self.config.FOV /
     2)))
112 
113     # calc GSD for current altitude
114     GSD = fov_radius * 2 / 1080
115     if GSD > self.config.best_gsd:
116         detect_false = self.detect_false * np.exp(self.config.gsd_decay * (
    GSD - self.config.best_gsd))
117         detect_real = self.detect_real * np.exp(self.config.gsd_decay * (
    GSD - self.config.best_gsd))
118     else:
119         detect_false = self.detect_false
120         detect_real = self.detect_real
121 
122     if target_list is not None:
123         for target in target_list:  # compare every target location to the
    current agent location
124             if target.search_detected:  # only need to detect new targets
125                 continue
126 
127             # Check if the target is within the ground FOV radius and the
    agent is at the current altitude
128             if ((location.x - target.position.x) ** 2 +
129                (location.y - target.position.y) ** 2) < fov_radius ** 2
    and \
130                     abs(location.z - self.config.search_alt) < 3:
131 
132                 target.search_detected = True
133                 if target.agent is None:
134                    target.agent = self.config.agent_id
135 
136                prob = np.random.uniform()  # make a random probability d
    raw for each time target was sensed
137 
138                if prob < detect_real and target.real is True:
139                     # encounter real and detect real
140                     target.search_type = 'correct'
141 
```

```
142                 elif prob > detect_real and target.real is True:
143                     # encounter real and detect false
144                     target.search_type = 'TypeII'
145 
146                 elif prob < detect_false and target.real is False:
147                     # encounter false and detect false
148                     target.search_type = 'correct'
149 
150                 elif prob > detect_false and target.real is False:
151                     # encounter false and detect real
152                     target.search_type = 'TypeI'
153 
154                 # Get target's sensed position with error
155                 normal_diagonal = np.random.normal(0, fov_radius * self.co
    nfig.fov_error)
156                 uniform_angle = np.random.uniform(0, 360)
157                 rand_x = normal_diagonal * math.cos(math.radians(
    uniform_angle))
158                 rand_y = normal_diagonal * math.sin(math.radians(
    uniform_angle))
159                 target.search_position = LocalLevelPosition(x=target.
    position.x + rand_x,
160                                                             y=target.posi
    tion.y + rand_y,
161                                                              z=0)
162                 self.state_block.send_state(self.config.target_list_stat
    e_id)
```

Appendix G. Coordinator Main Function Code

Coordinator Main Function

```
250 def main(self):
251 
252     while self.state_block.lcm_update_states():
253         pass
254 
255     if not self.initialized:  # split search area, reset target list
256         self.state_block.states[self.config.coordinator_commands_state_id].
    boundary_list = self.split_search_area()
257         self.should_loiter()
258         self.state_block.send_state(self.config.
    coordinator_commands_state_id)
259         self.state_block[self.config.agent_targets_state_id].targets = []
260         self.state_block.send_state(self.config.agent_targets_state_id)
261         self.initialized = True
262 
263     self.update_deliberator_state()
264 
265     if self.state_history != []:
266         if 'RTL' != self.state_history[-1]:
267             self.update_global_agent_message()
268             self.check_cooperative_confirm()
269 
270     elif not self.RTL_sent:
271         self.RTL_sent = True
272         self.update_global_agent_message()
273         self.state_block[self.config.agent_targets_state_id].state = 'RTL'
274         self.state_block.send_state(self.config.agent_targets_state_id)
```

Appendix H. Face Centered Central Composite Design Matrix

Table 30. PyDOE output for experimental design of final factors and levels selected, given in Table 10

Run	Search Velocity	Detect Real	Detect False	Search Area	N Real	N False
1	5	0.65	0.65	450	1	1
2	10	0.65	0.65	450	1	1
3	5	0.9	0.65	450	1	1
4	10	0.9	0.65	450	1	1
5	5	0.65	0.9	450	1	1
6	10	0.65	0.9	450	1	1
7	5	0.9	0.9	450	1	1
8	10	0.9	0.9	450	1	1
9	5	0.65	0.65	700	1	1
10	10	0.65	0.65	700	1	1
11	5	0.9	0.65	700	1	1
12	10	0.9	0.65	700	1	1
13	5	0.65	0.9	700	1	1
14	10	0.65	0.9	700	1	1
15	5	0.9	0.9	700	1	1
16	10	0.9	0.9	700	1	1
17	5	0.65	0.65	450	19	1
18	10	0.65	0.65	450	19	1
19	5	0.9	0.65	450	19	1
20	10	0.9	0.65	450	19	1
21	5	0.65	0.9	450	19	1
22	10	0.65	0.9	450	19	1
23	5	0.9	0.9	450	19	1
24	10	0.9	0.9	450	19	1
25	5	0.65	0.65	700	19	1
26	10	0.65	0.65	700	19	1
27	5	0.9	0.65	700	19	1
28	10	0.9	0.65	700	19	1
29	5	0.65	0.9	700	19	1
30	10	0.65	0.9	700	19	1
31	5	0.9	0.9	700	19	1
32	10	0.9	0.9	700	19	1
33	5	0.65	0.65	450	1	19
34	10	0.65	0.65	450	1	19
35	5	0.9	0.65	450	19	1
36	10	0.9	0.65	450	1	19
37	5	0.65	0.9	450	1	19
38	10	0.65	0.9	450	1	19
39	5	0.9	0.9	450	1	19
40	10	0.9	0.9	450	1	19

Table 31. PyDOE output for experimental design of final factors and levels selected, given in Table 10, continued

Run	Search Velocity	Detect Real	Detect False	Search Area	N Real	N False
41	5	0.65	0.65	700	1	19
42	10	0.65	0.65	700	1	19
43	5	0.9	0.65	700	1	1
44	10	0.9	0.65	700	1	19
45	5	0.65	0.9	700	1	19
46	10	0.65	0.9	700	1	19
47	5	0.9	0.9	700	1	19
48	10	0.9	0.9	700	1	19
49	5	0.65	0.65	450	19	19
50	10	0.65	0.65	450	19	19
51	5	0.9	0.65	450	19	19
52	10	0.9	0.65	450	19	19
53	5	0.65	0.9	450	19	19
54	10	0.65	0.9	450	19	19
55	5	0.9	0.9	450	19	19
56	10	0.9	0.9	450	19	19
57	5	0.65	0.65	700	19	19
58	10	0.65	0.65	700	19	19
59	5	0.9	0.65	700	19	19
60	10	0.9	0.65	700	19	19
61	5	0.65	0.9	700	19	19
62	10	0.65	0.9	700	19	19
63	5	0.9	0.9	700	19	19
64	10	0.9	0.9	700	19	19
65	7.5	0.775	0.775	575	10	10
66	7.5	0.775	0.775	575	10	10
67	5	0.775	0.775	575	10	10
68	10	0.775	0.775	575	10	10
69	7.5	0.65	0.775	575	10	10
70	7.5	0.9	0.775	575	10	10
71	7.5	0.775	0.65	575	10	10
72	7.5	0.775	0.9	575	10	10
73	7.5	0.775	0.775	450	10	10
74	7.5	0.775	0.775	700	10	10
75	7.5	0.775	0.775	575	1	10
76	7.5	0.775	0.775	575	19	10
77	7.5	0.775	0.775	575	10	1
78	7.5	0.775	0.775	575	10	19
79	7.5	0.775	0.775	575	10	10
80	7.5	0.775	0.775	575	10	10

Appendix I. Multiple Comparison Test Results: One Vehicle Operation

Response Percent_Detected

replicate

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	93.940789	1.1732841	158	91.623445	96.258134
1	94.641447	1.1732841	158	92.324103	96.958792

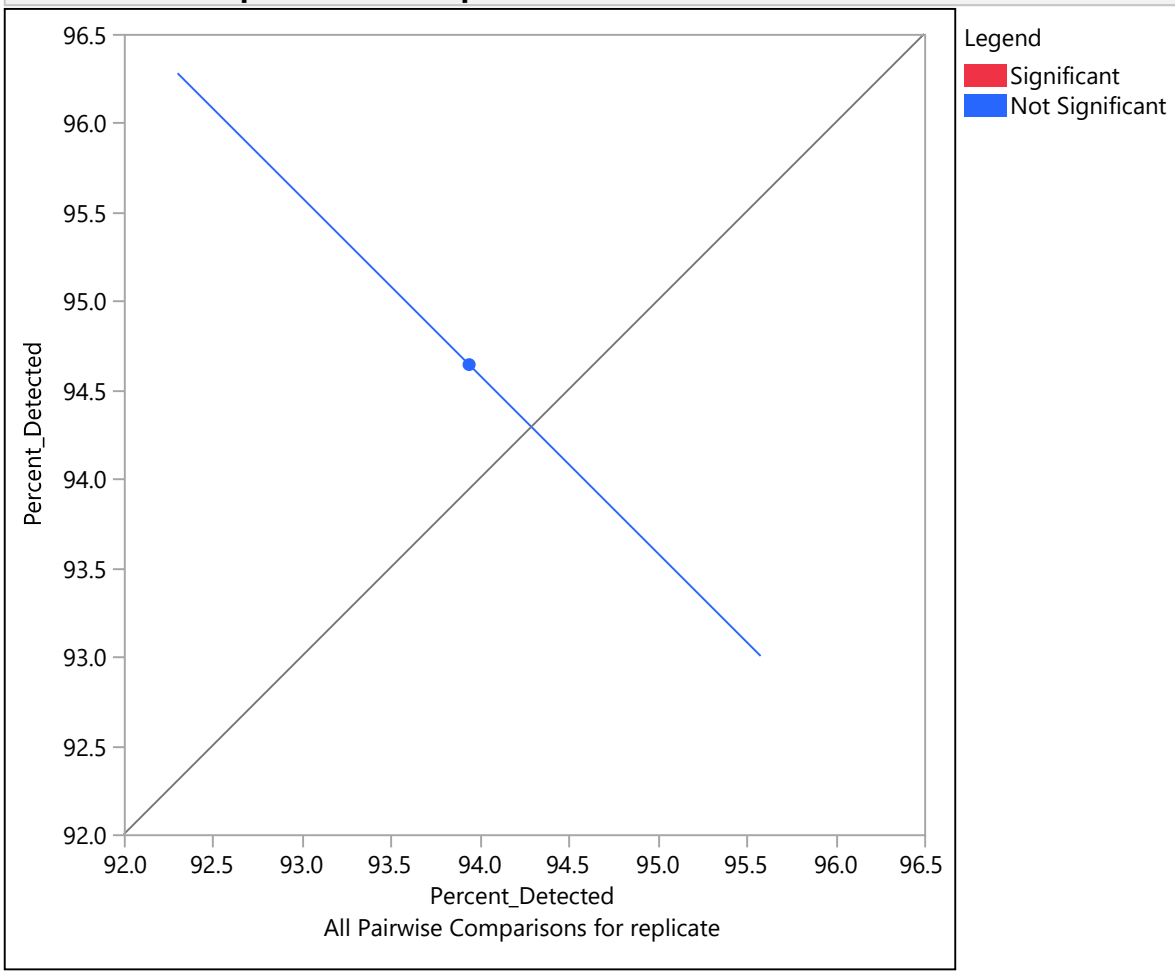
Tukey HSD All Pairwise Comparisons

Quantile = 1.97509, Adjusted DF = 158.0, Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.700658	1.659274	-0.42	0.6734	-3.97788	2.576562

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

Quantile = 1.97509, DF = 158.0

All Pairwise Differences

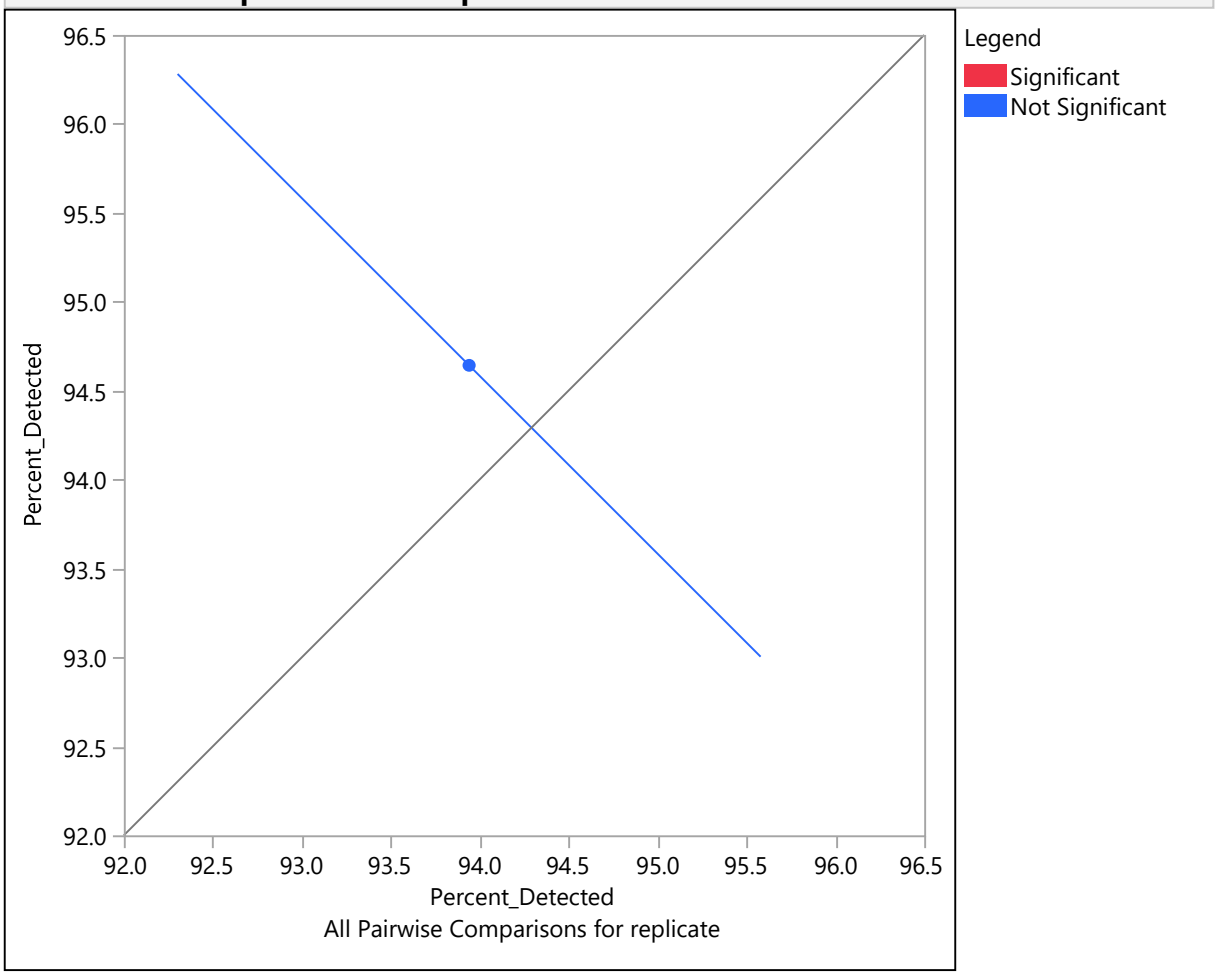
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.700658	1.659274	-0.42	0.6734	-3.97788	2.576562

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot

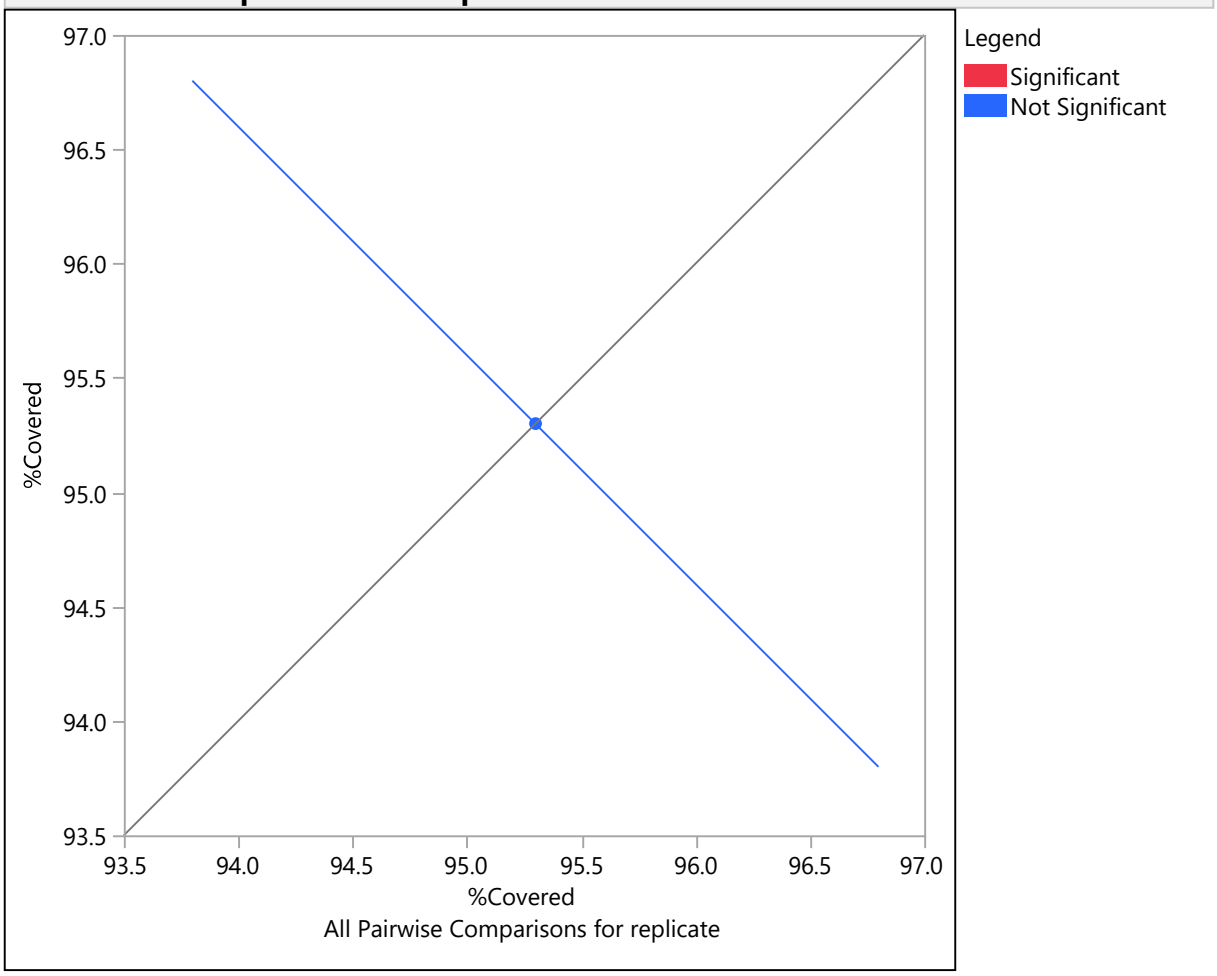


Response %Covered

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

Quantile = 1.97509 , DF = 158.0

All Pairwise Differences

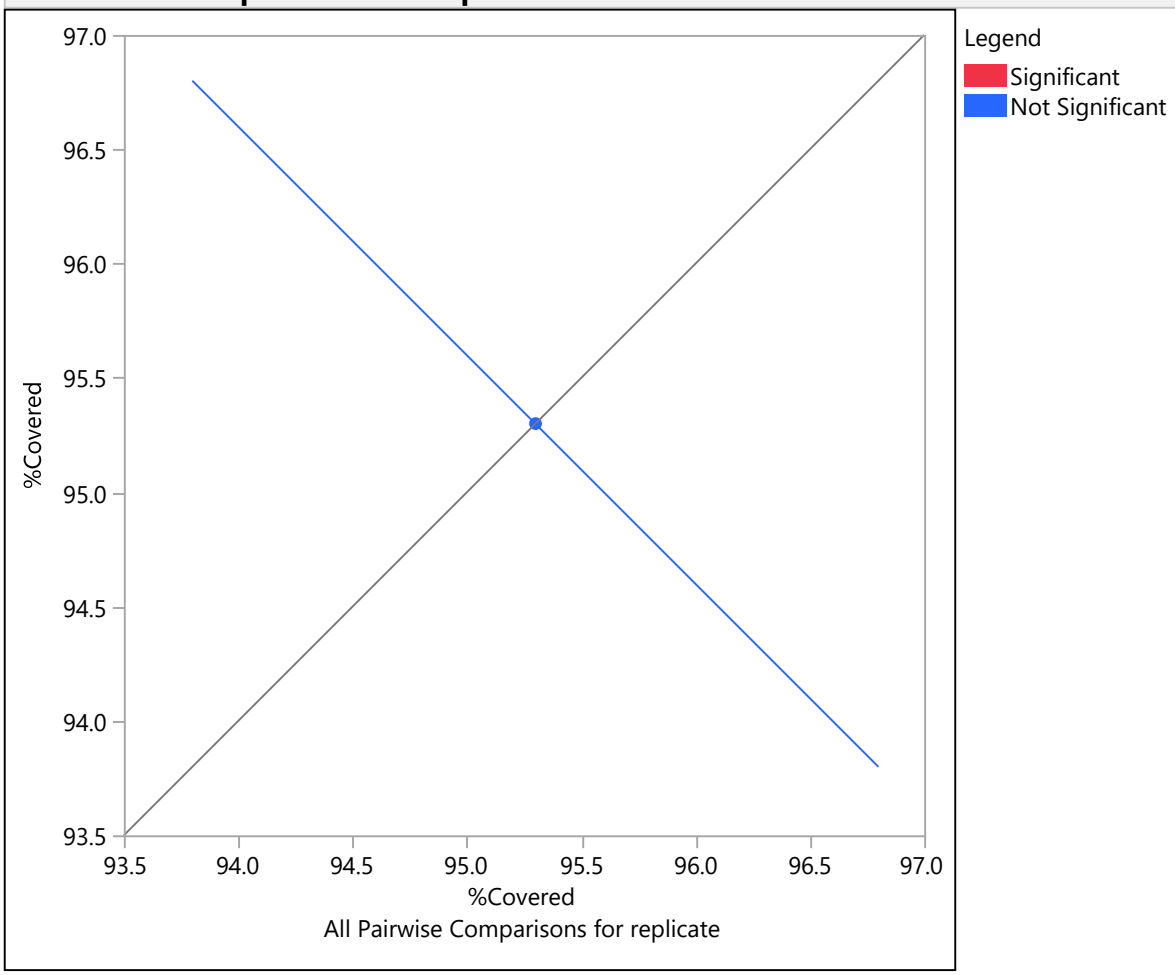
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0	1.519137	0.00	1.0000	-3.00043	3.000435

Response %Covered

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Percent_Correct_Detected

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	64.546005	2.1565791	158	60.286562	68.805447
1	61.834048	2.1565791	158	57.574605	66.093490

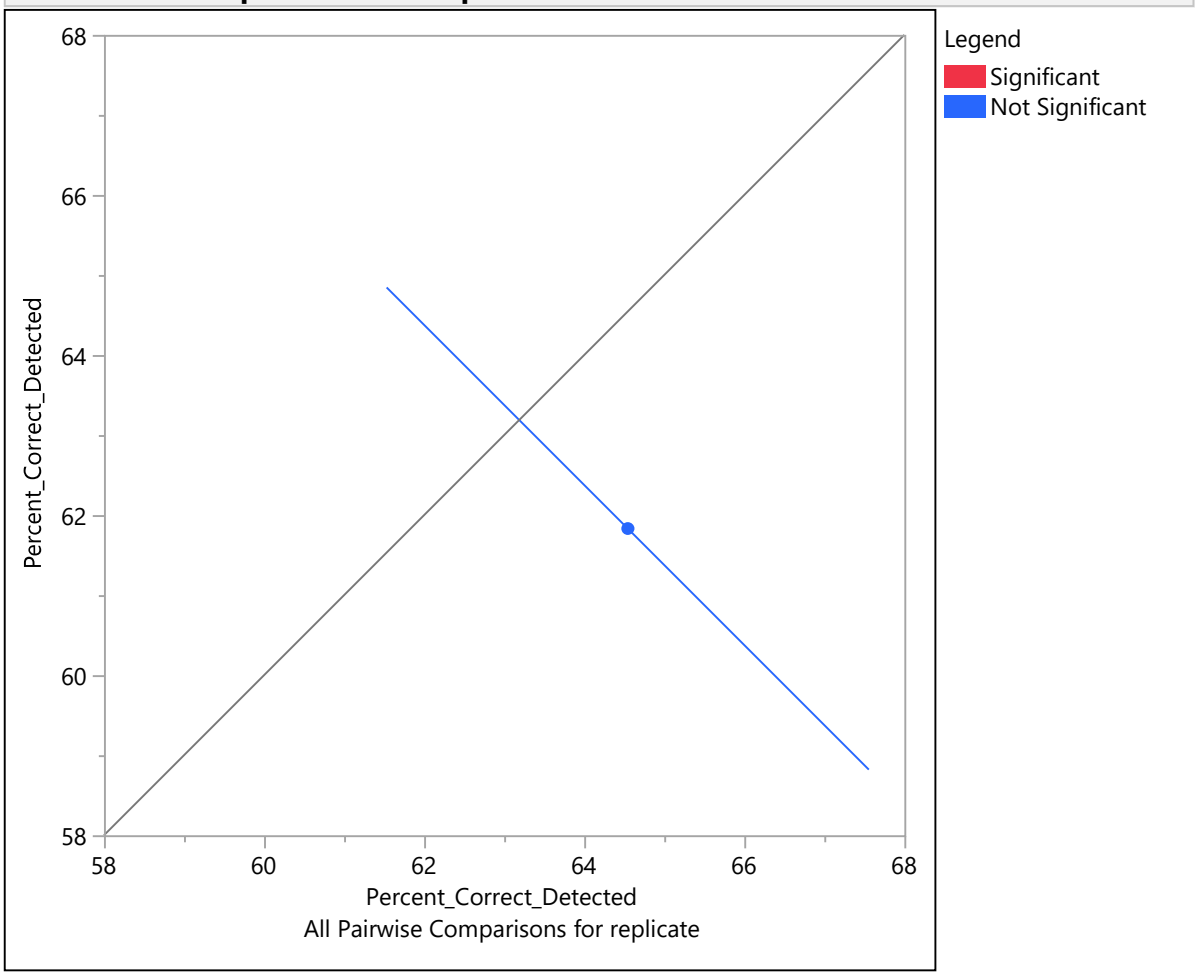
Tukey HSD All Pairwise Comparisons

Quantile = 1.97509, Adjusted DF = 158.0, Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	2.711957	3.049863	0.89	0.3752	-3.31181	8.735719

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

Quantile = 1.97509, DF = 158.0

All Pairwise Differences

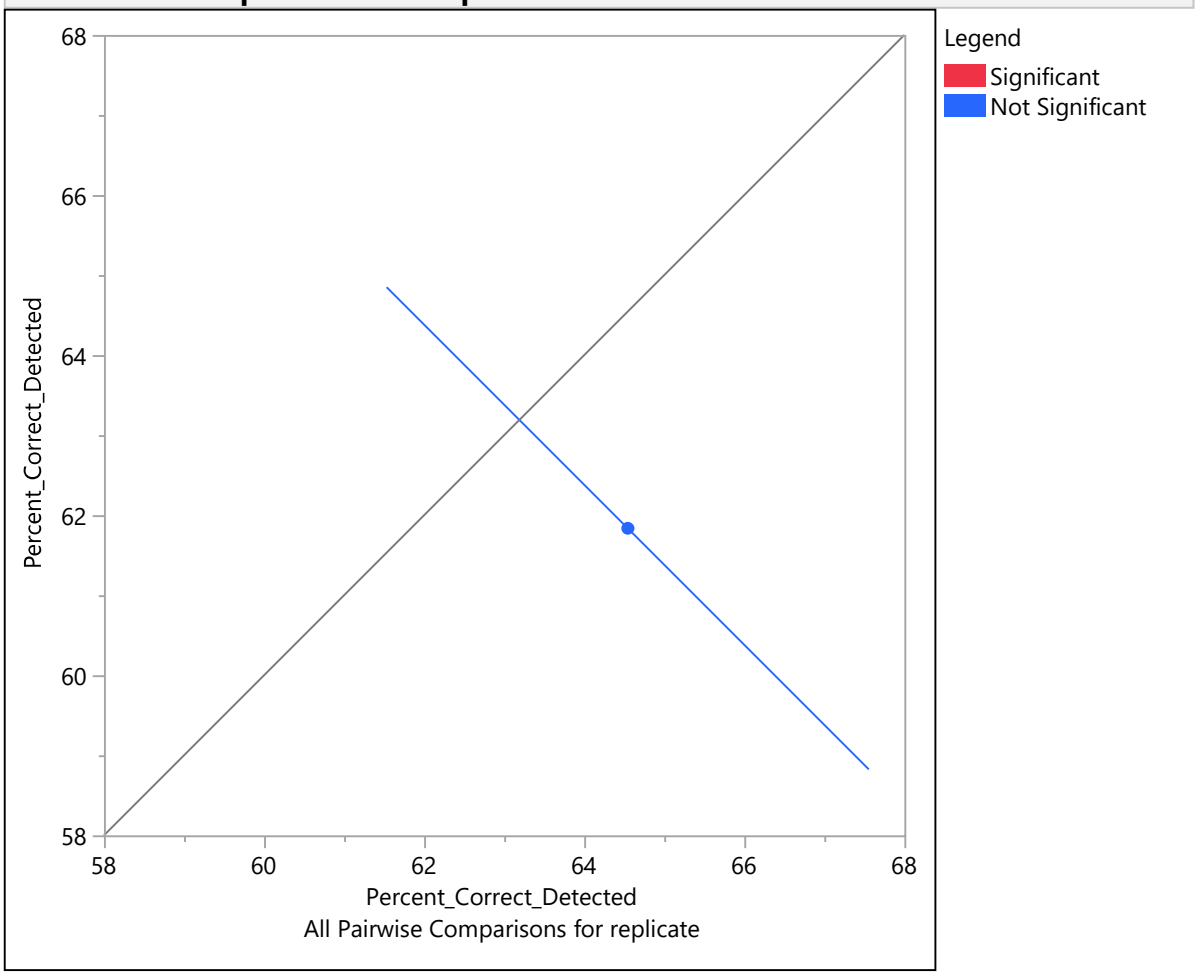
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	2.711957	3.049863	0.89	0.3752	-3.31180	8.735718

Response Percent_Correct_Detected

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot

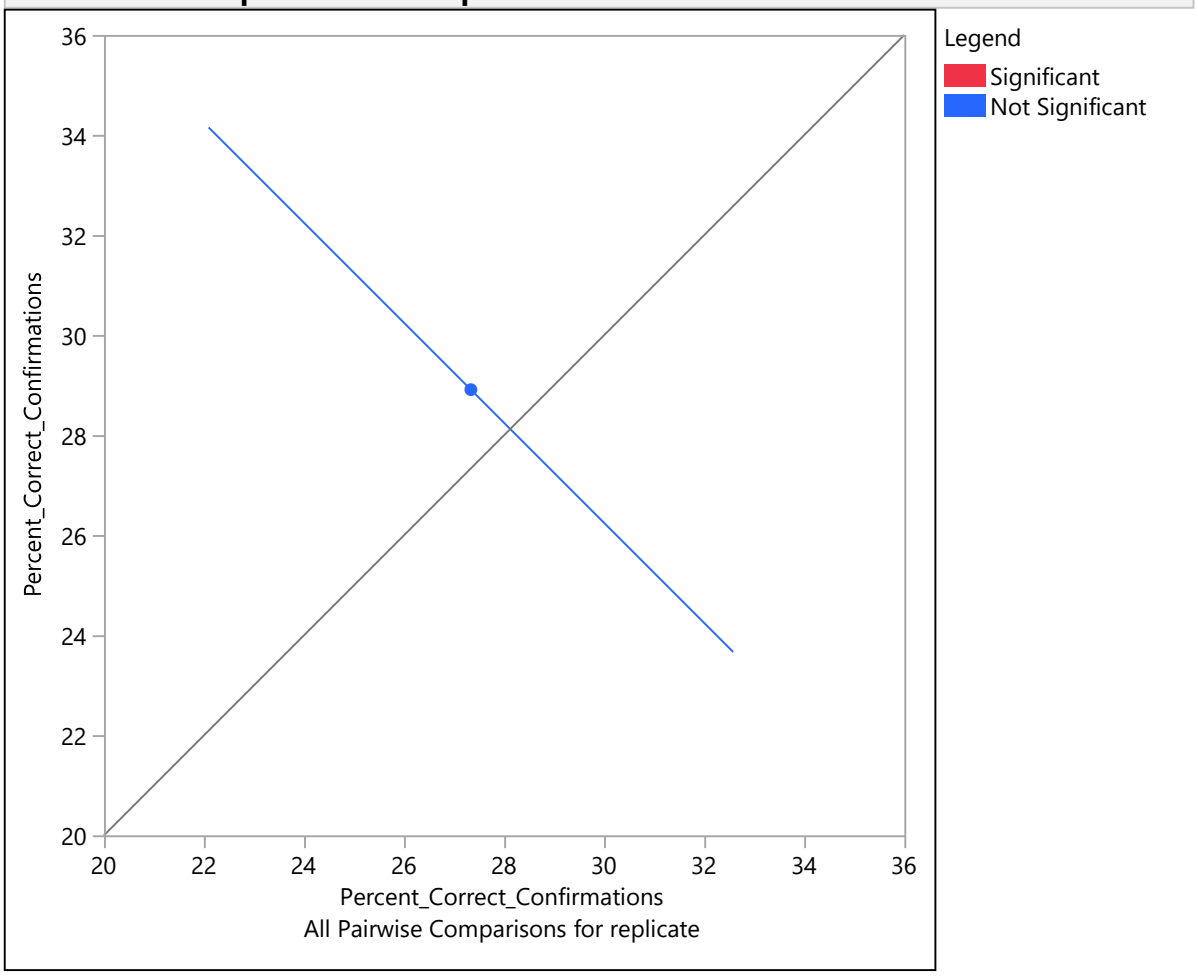


Response Percent_Correct_Confirmations

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

Quantile = 1.97509, DF = 158.0

All Pairwise Differences

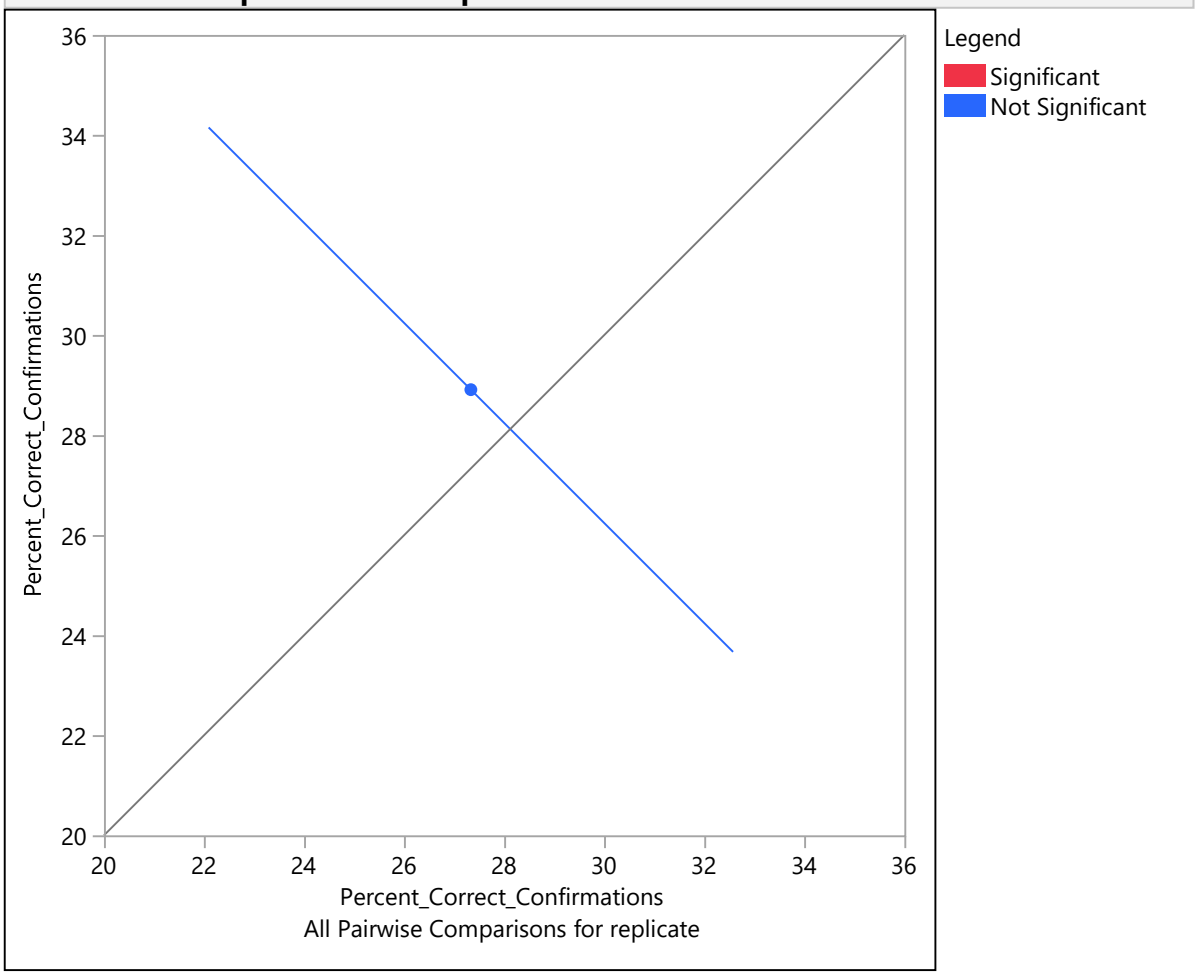
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-1.57277	5.308607	-0.30	0.7674	-12.0578	8.912218

Response Percent_Correct_Confirmations

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Percent_Confirmed

replicate

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	14.208247	2.1402996	158	9.980959	18.435536
1	15.038556	2.1402996	158	10.811267	19.265845

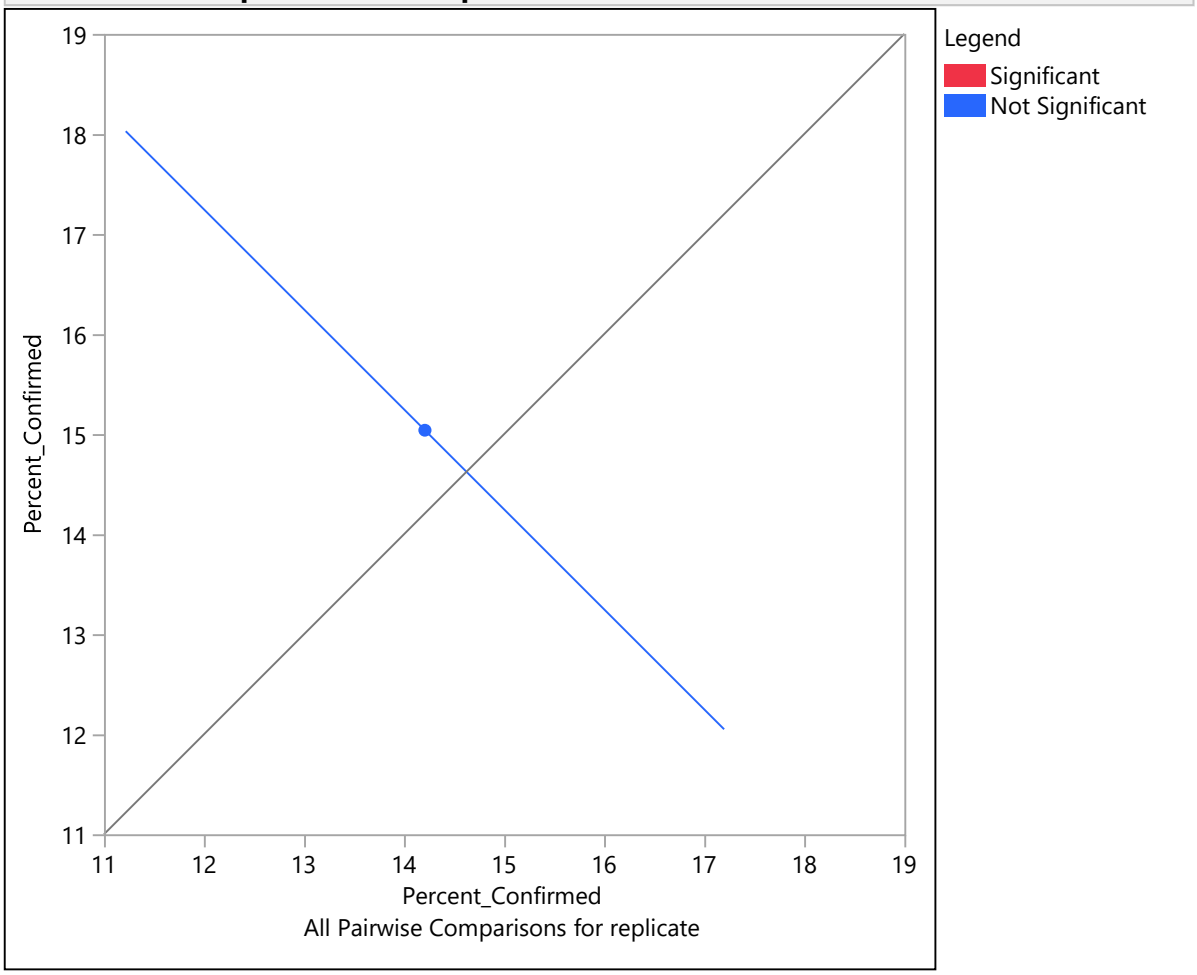
Tukey HSD All Pairwise Comparisons

Quantile = 1.97509, Adjusted DF = 158.0, Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.830309	3.026841	-0.27	0.7842	-6.80860	5.147982

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

Quantile = 1.97509, DF = 158.0

All Pairwise Differences

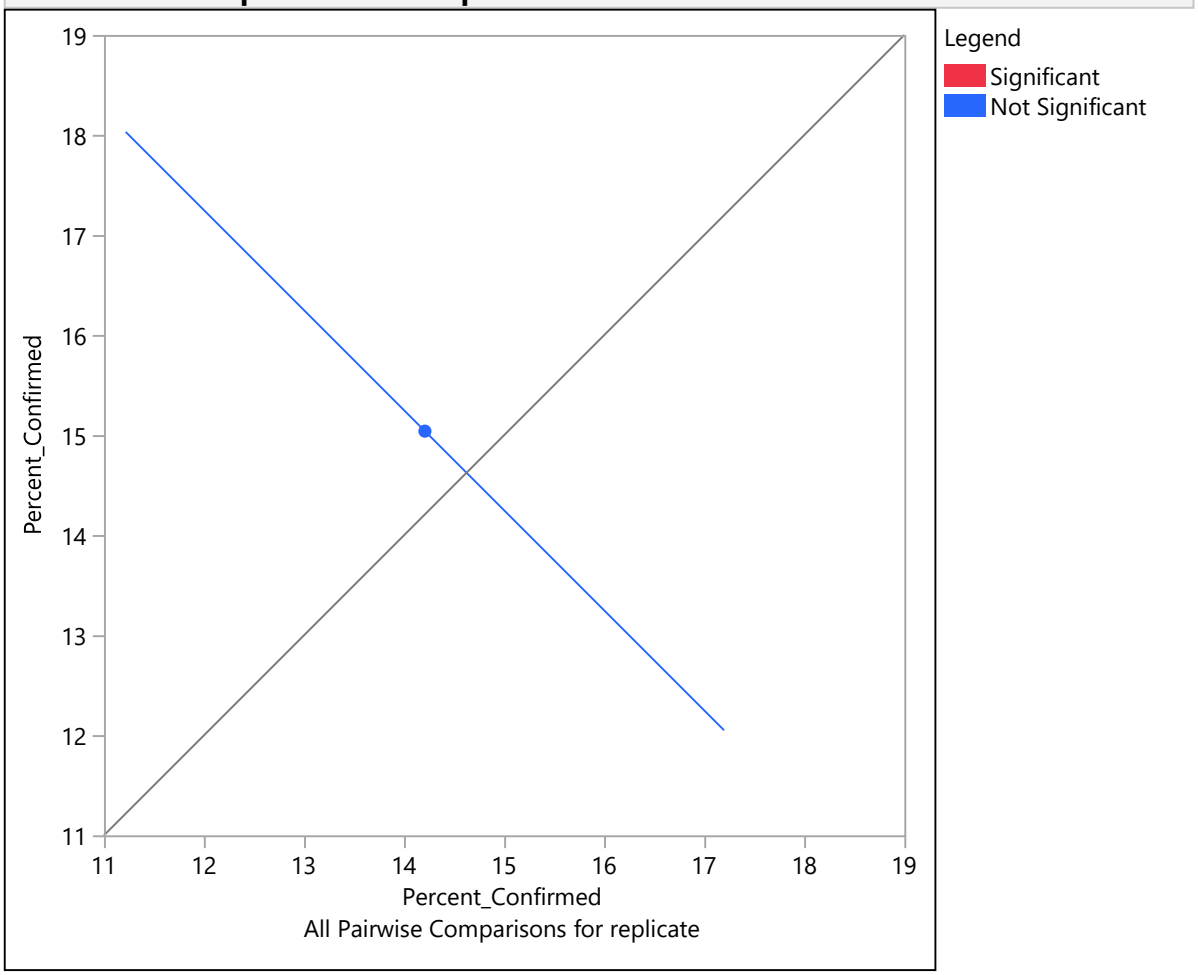
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.830309	3.026841	-0.27	0.7842	-6.80860	5.147981

Response Percent_Confirmed

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Mission Time

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	12.522854	0.55316678	158	11.430299	13.615409
1	12.703854	0.55316678	158	11.611299	13.796409

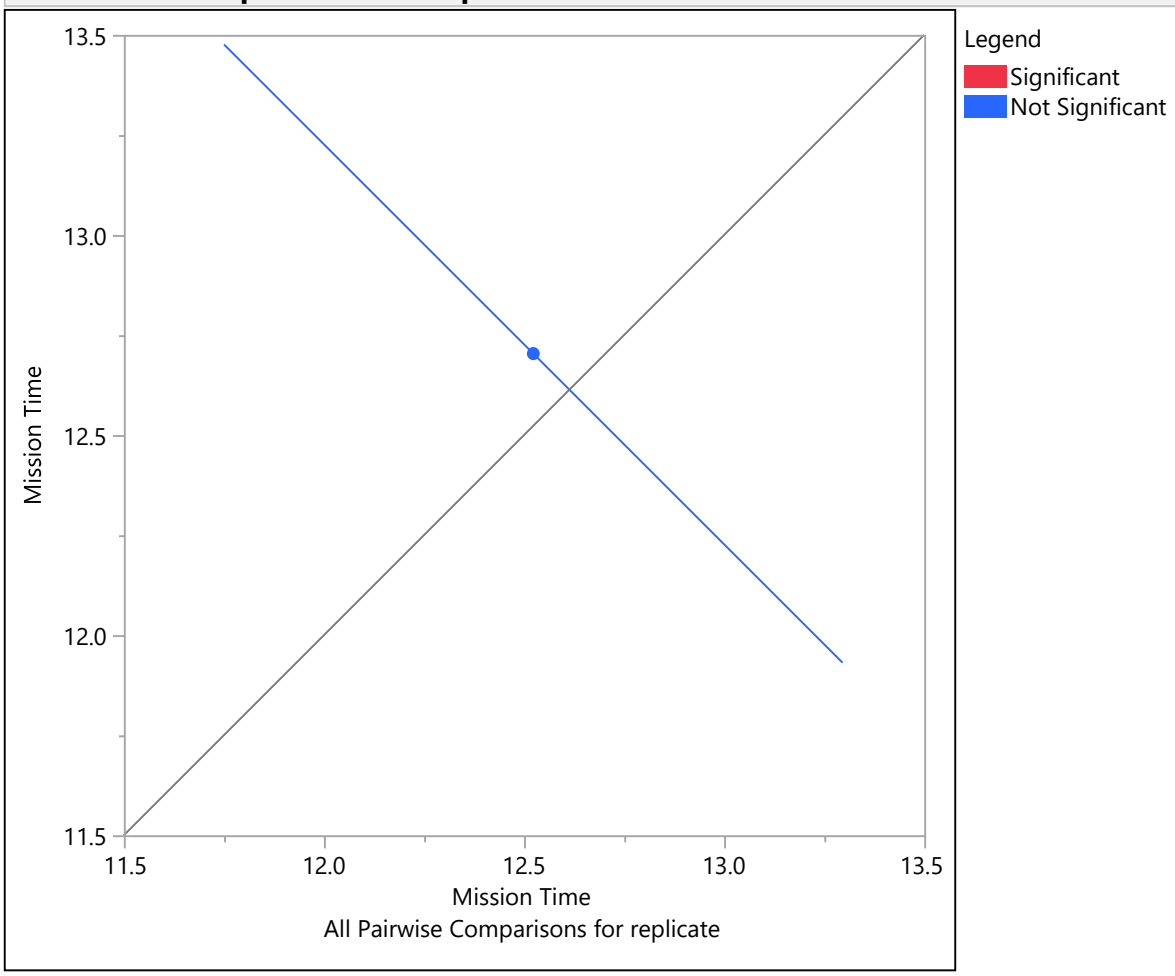
Tukey HSD All Pairwise Comparisons

Quantile = 1.97509, Adjusted DF = 158.0, Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.181000	0.7822960	-0.23	0.8173	-1.72611	1.364107

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

Quantile = 1.97509, DF = 158.0

All Pairwise Differences

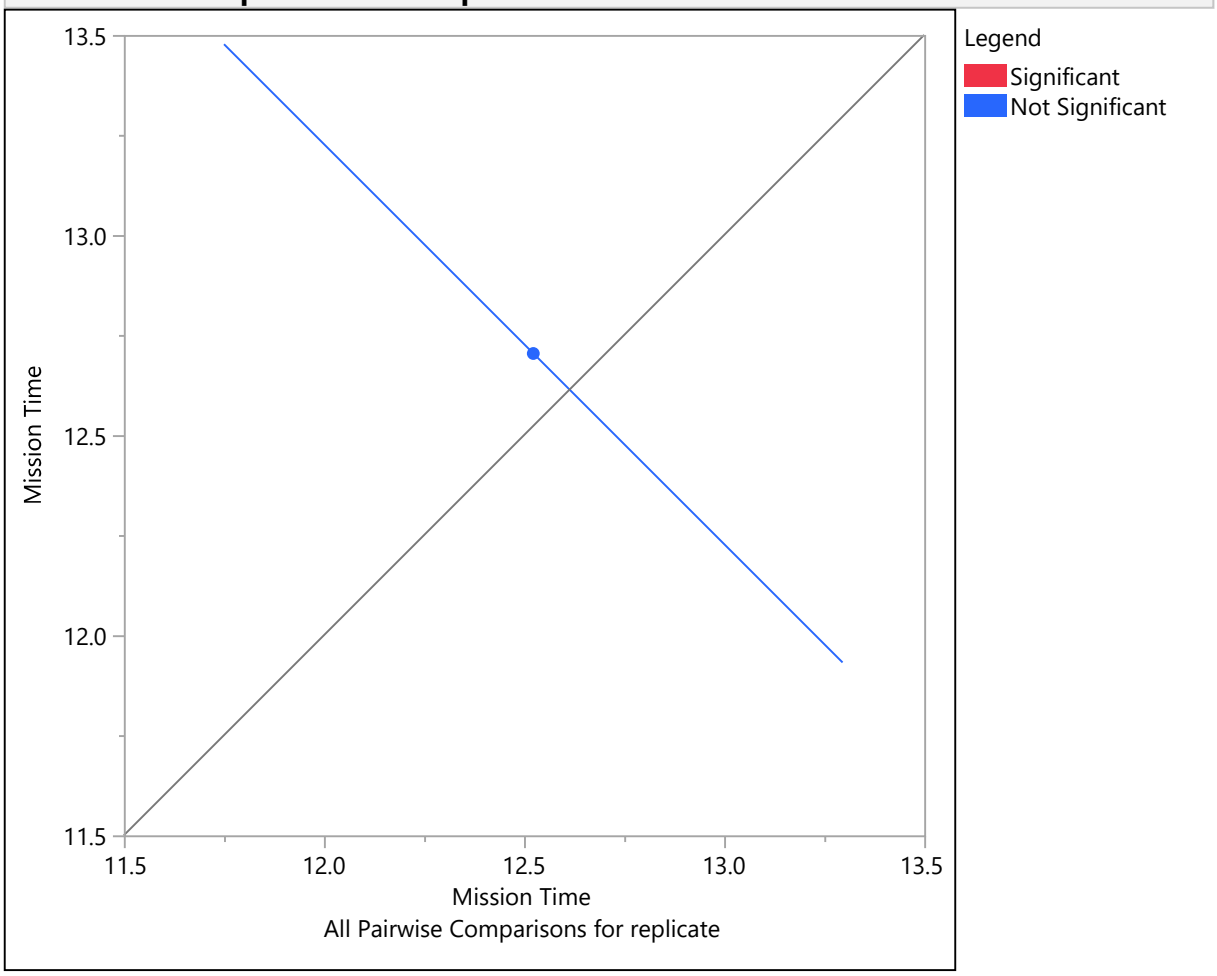
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.181000	0.7822960	-0.23	0.8173	-1.72611	1.364107

Response Mission Time

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Detect_Typel_error

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	17.400840	1.8401869	158	13.766301	21.035378
1	18.866969	1.8401869	158	15.232430	22.501507

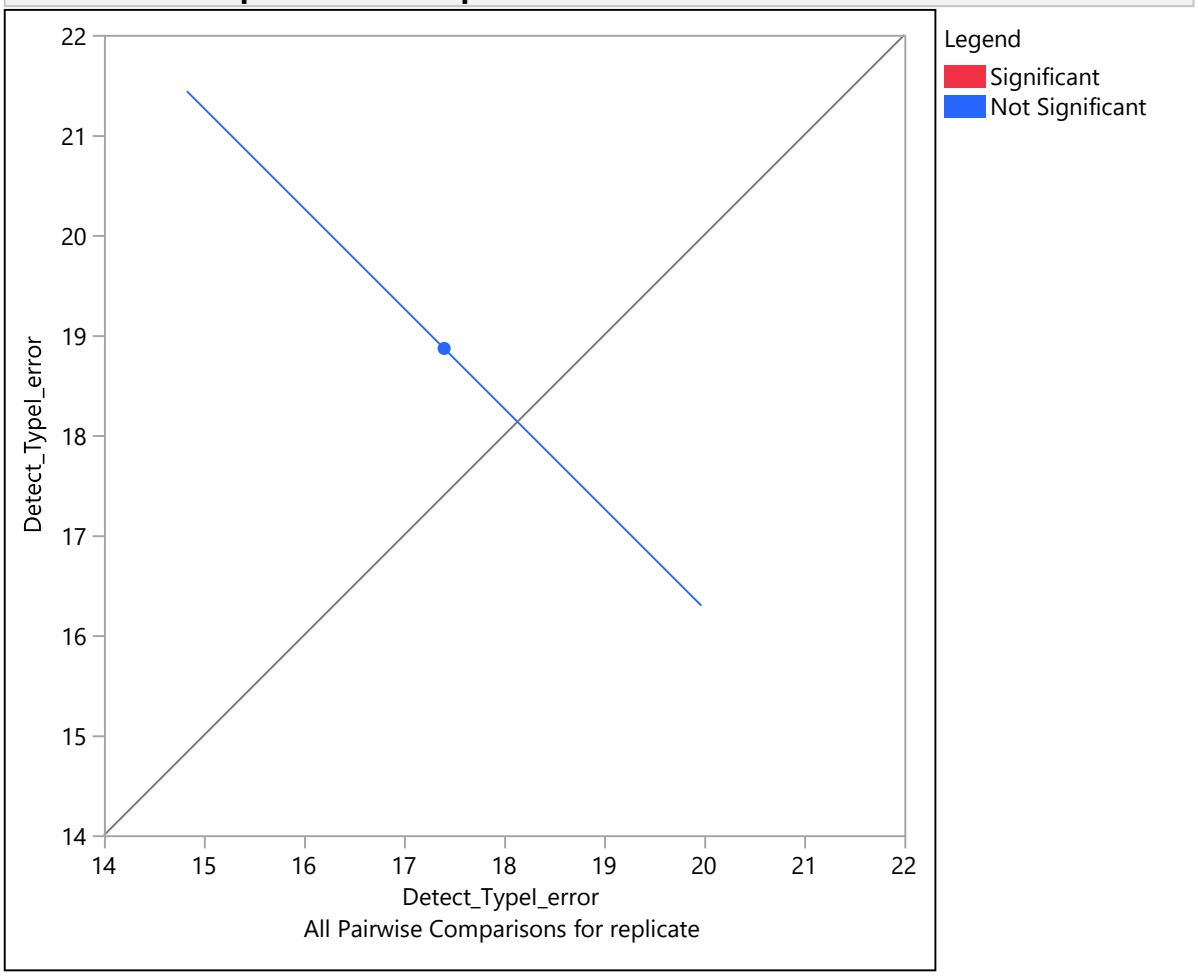
Tukey HSD All Pairwise Comparisons

Quantile = 1.97509, Adjusted DF = 158.0, Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-1.46613	2.602417	-0.56	0.5740	-6.60614	3.673885

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

Quantile = 1.97509, DF = 158.0

All Pairwise Differences

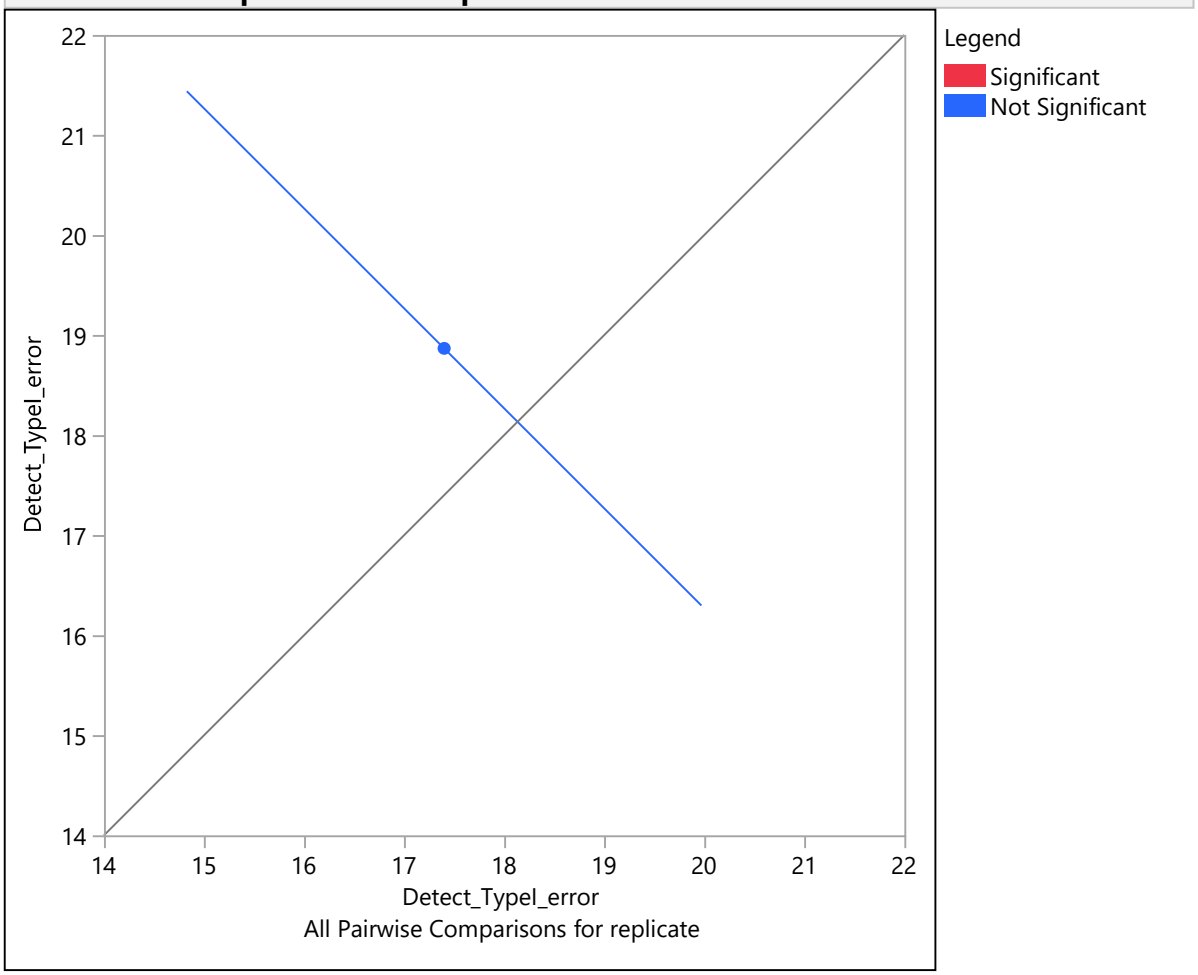
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-1.46613	2.602417	-0.56	0.5740	-6.60614	3.673884

Response Detect_TypeI_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Detect_TypeII_error

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	18.053156	2.1051321	158	13.895326	22.210985
1	19.298984	2.1051321	158	15.141154	23.456813

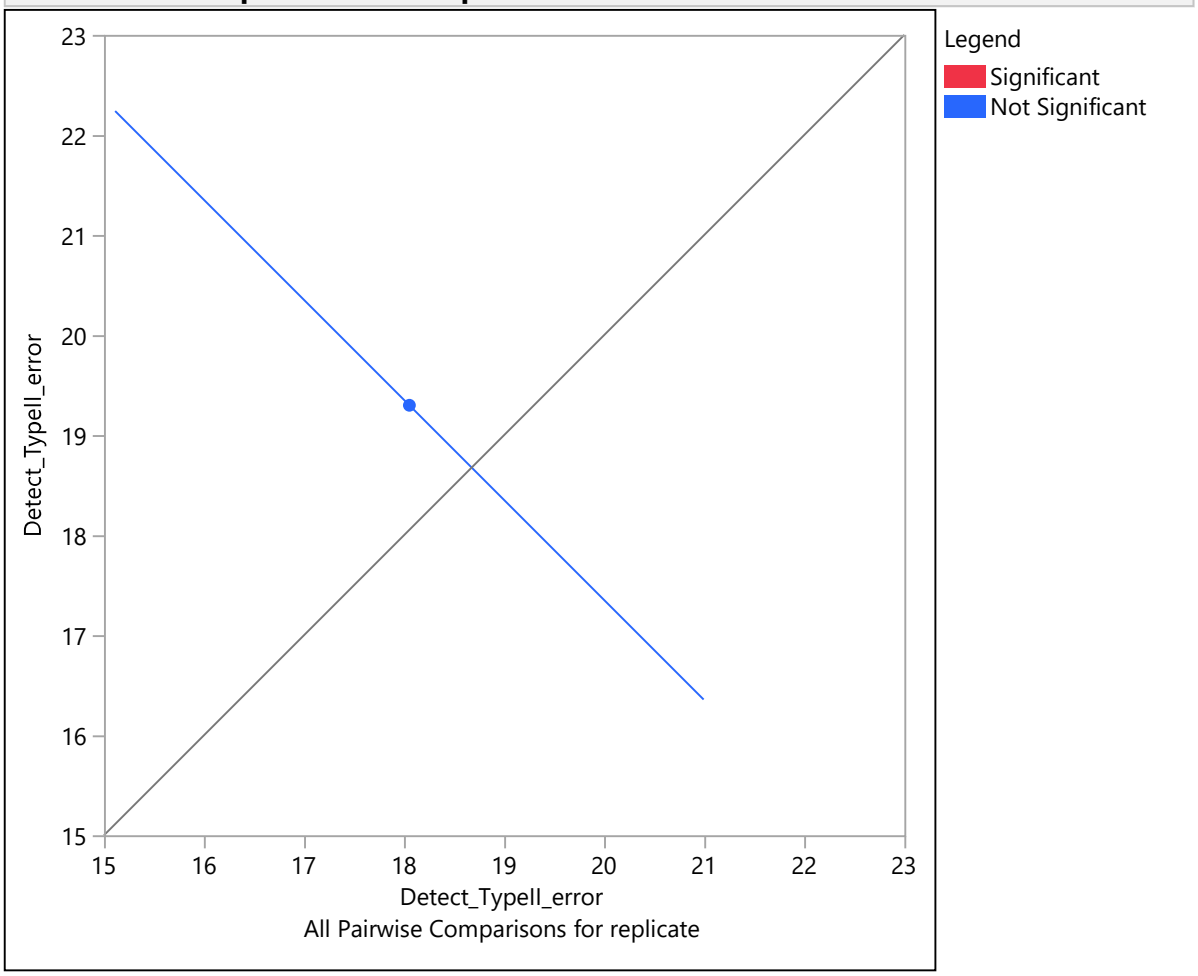
Tukey HSD All Pairwise Comparisons

Quantile = 1.97509, Adjusted DF = 158.0, Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-1.24583	2.977106	-0.42	0.6762	-7.12589	4.634232

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

Quantile = 1.97509, DF = 158.0

All Pairwise Differences

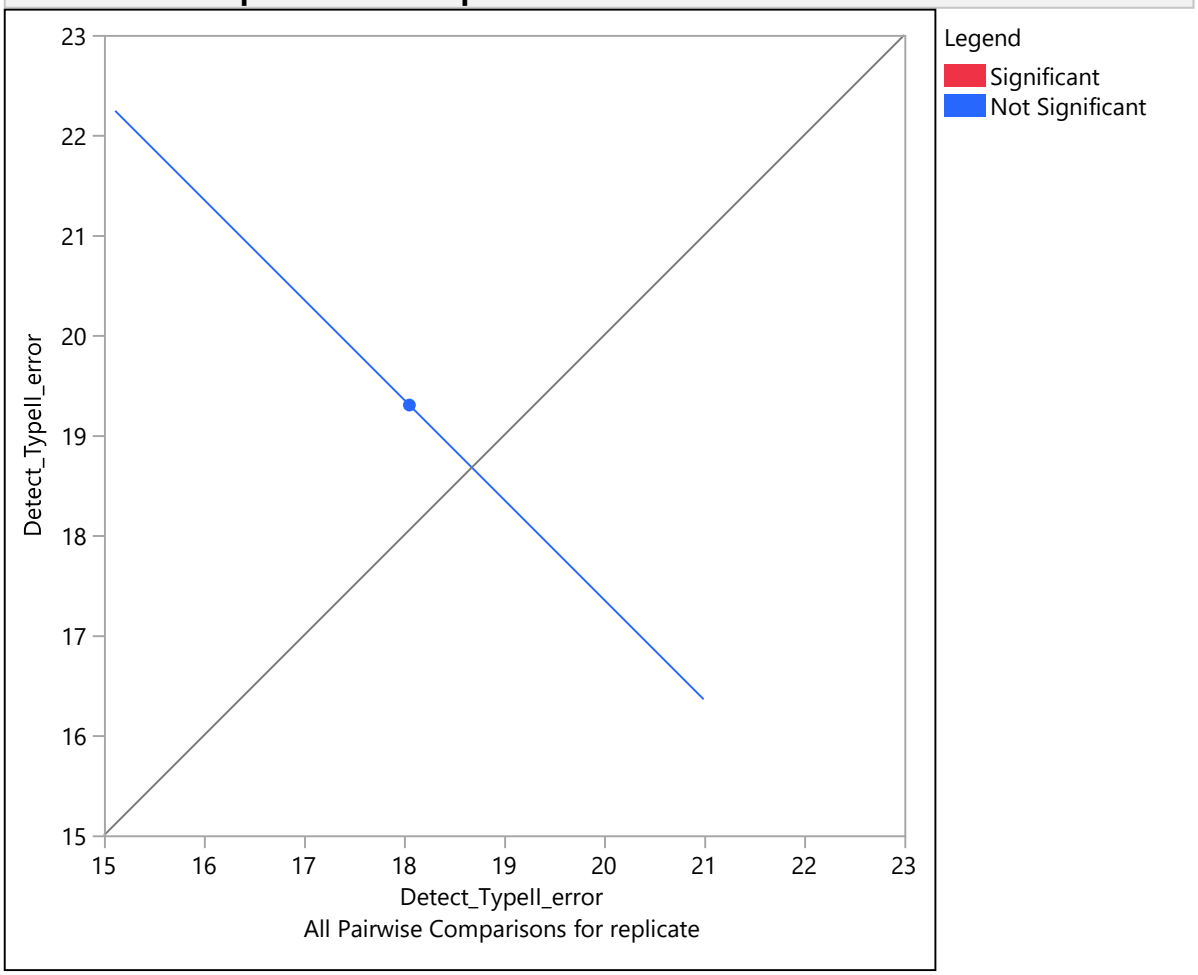
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-1.24583	2.977106	-0.42	0.6762	-7.12589	4.634231

Response Detect_TypeII_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Confirm_TypeI_error

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	5.7822009	1.3694528	158	3.0774055	8.4869964
1	4.7082643	1.3694528	158	2.0034688	7.4130597

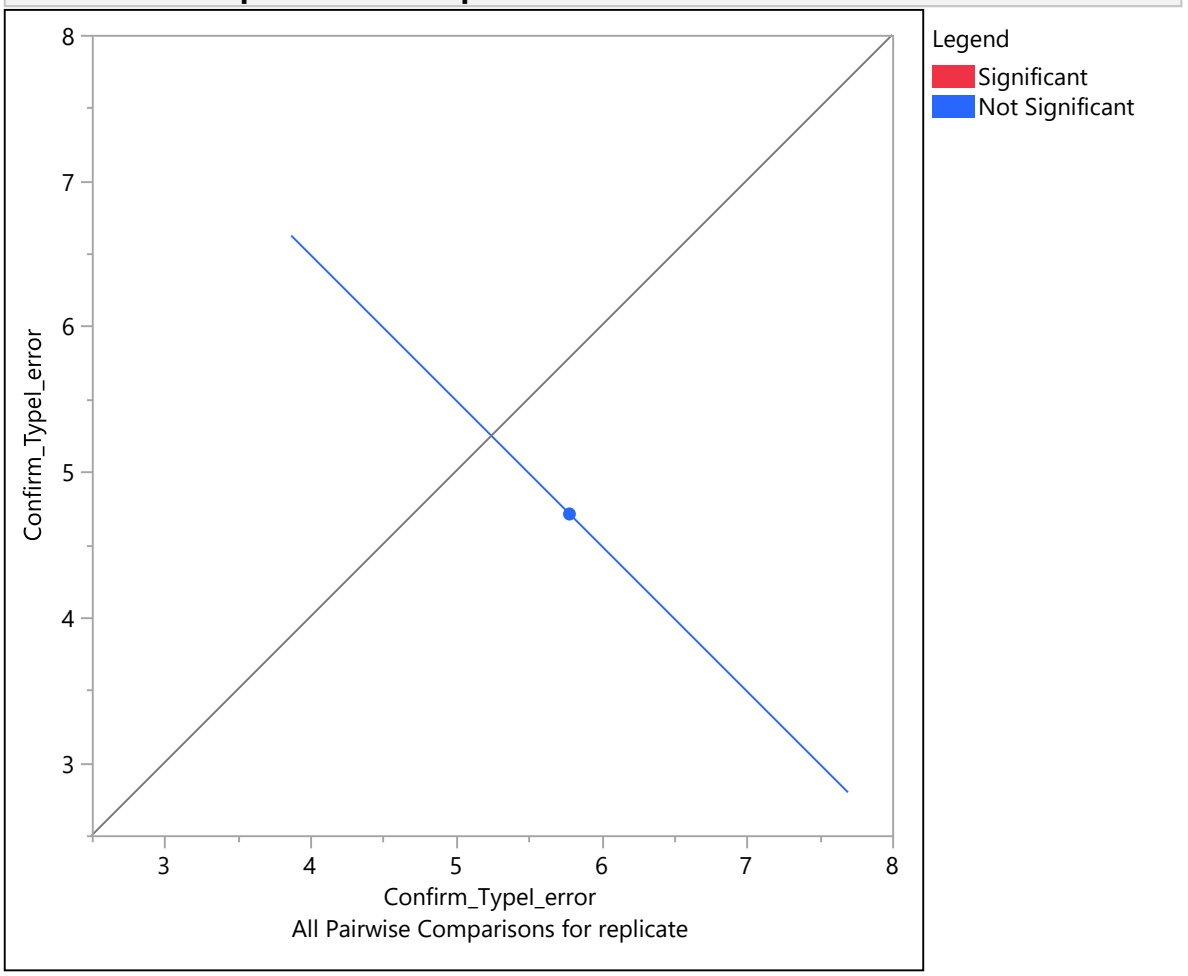
Tukey HSD All Pairwise Comparisons

Quantile = 1.97509, Adjusted DF = 158.0, Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.073937	1.936699	0.55	0.5800	-2.75122	4.899096

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

Quantile = 1.97509, DF = 158.0

All Pairwise Differences

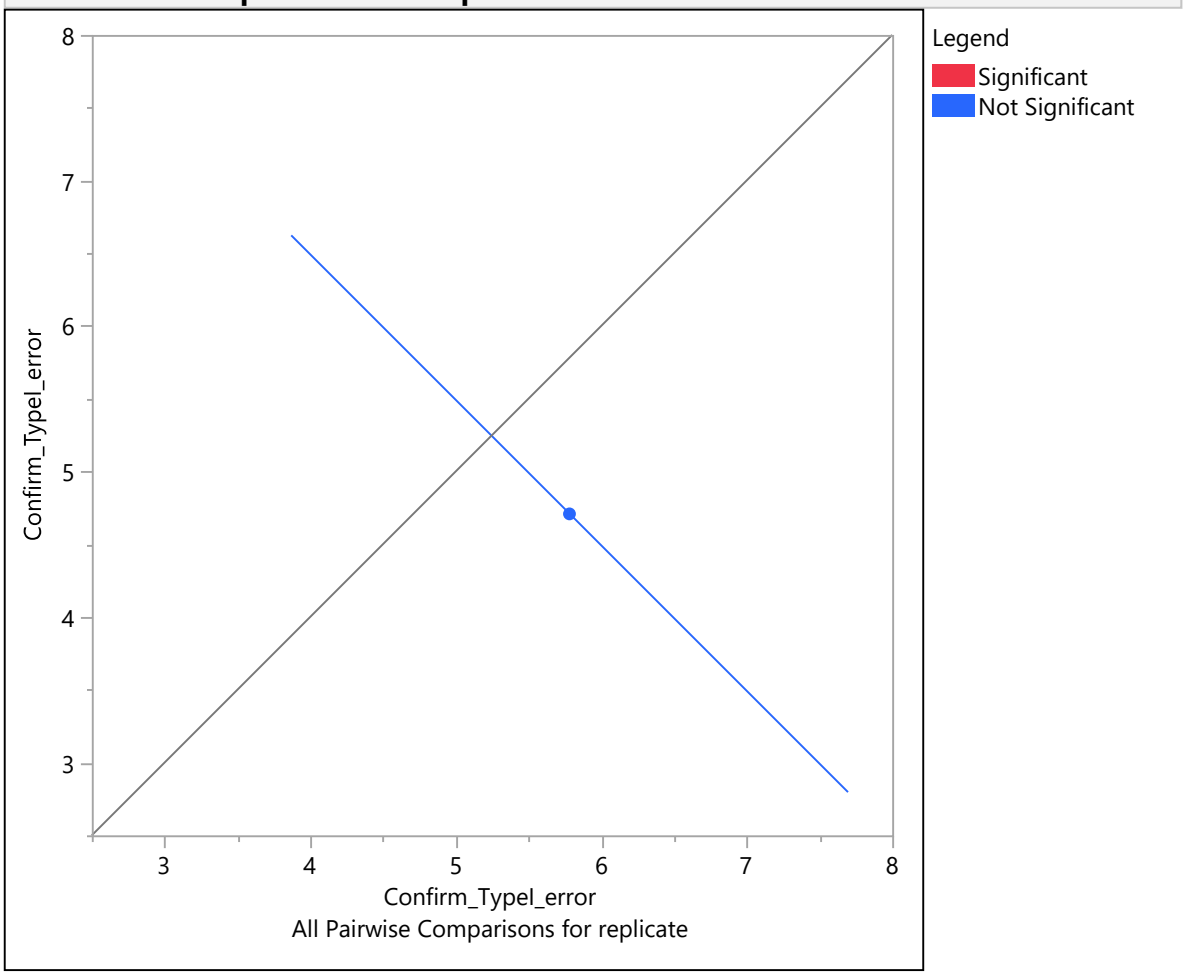
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.073937	1.936699	0.55	0.5800	-2.75122	4.899095

Response Confirm_TypeI_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Confirm_TypeII_error

replicate

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	6.7671160	1.4294746	158	3.9437721	9.5904600
1	5.5855962	1.4294746	158	2.7622522	8.4089401

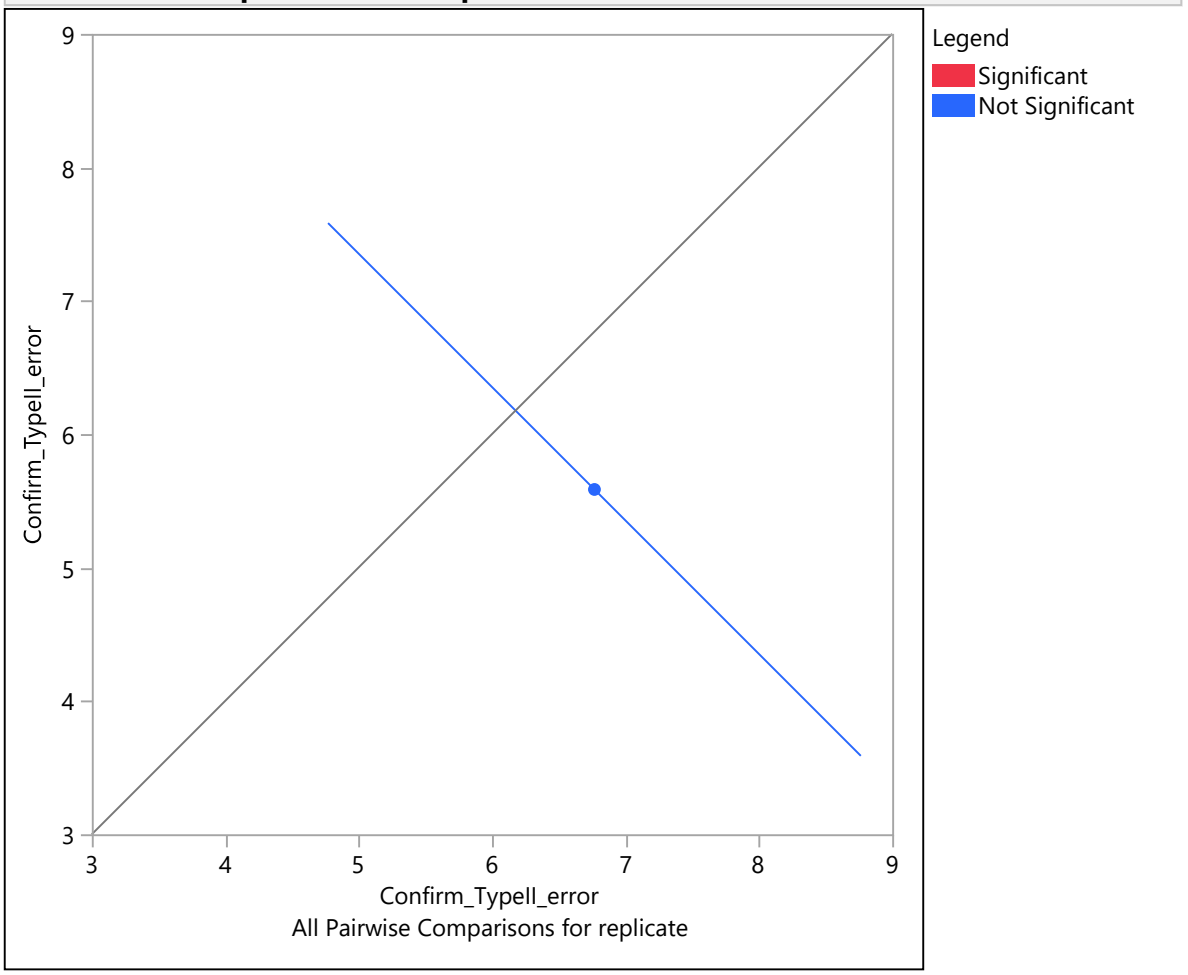
Tukey HSD All Pairwise Comparisons

Quantile = 1.97509, Adjusted DF = 158.0, Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.181520	2.021582	0.58	0.5597	-2.81129	5.174332

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

Quantile = 1.97509, DF = 158.0

All Pairwise Differences

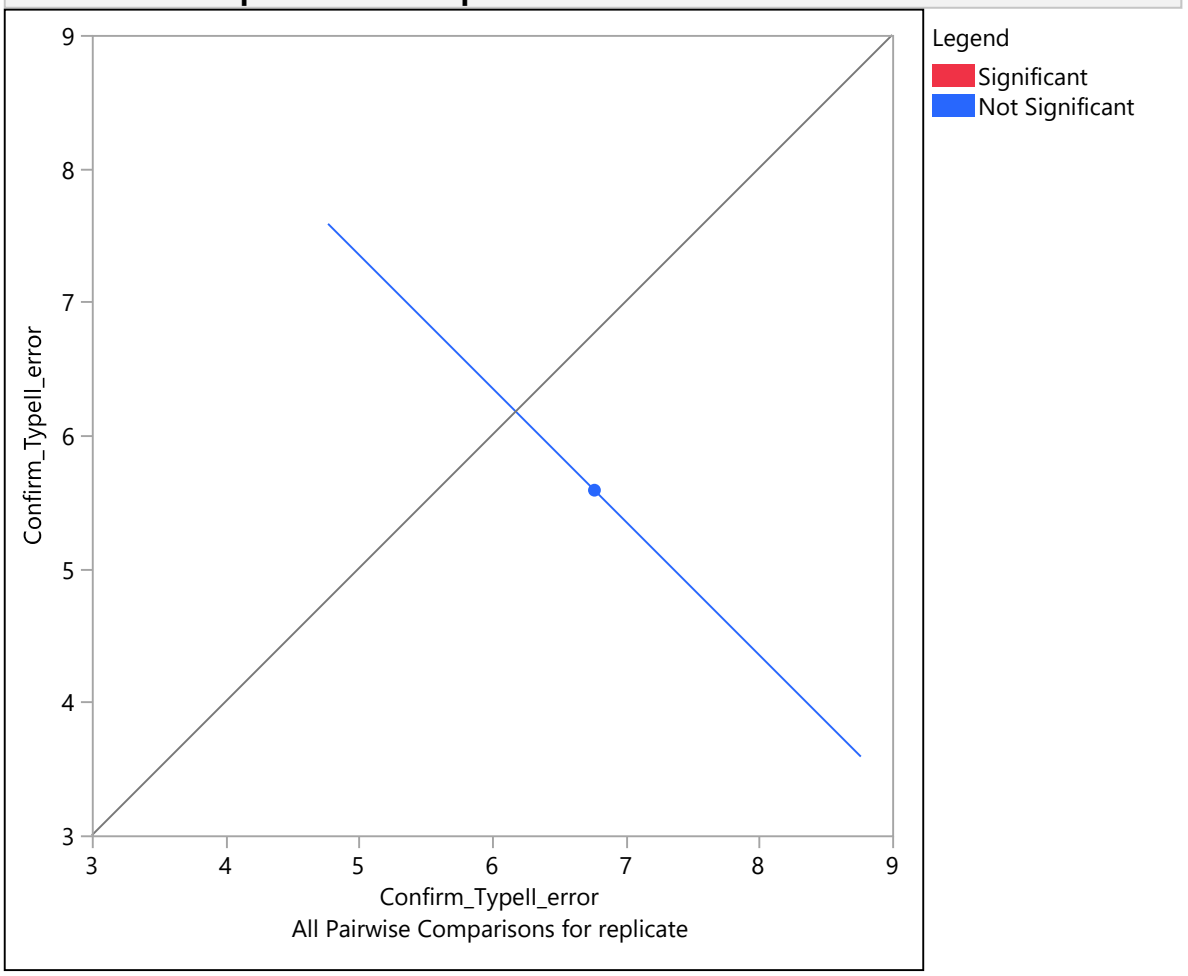
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.181520	2.021582	0.58	0.5597	-2.81129	5.174331

Response Confirm_TypeII_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Max

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	4.03725e-7	3.84262e-9	158	3.96135e-7	4.11315e-7
1	4.10287e-7	3.84262e-9	158	4.02698e-7	4.17877e-7

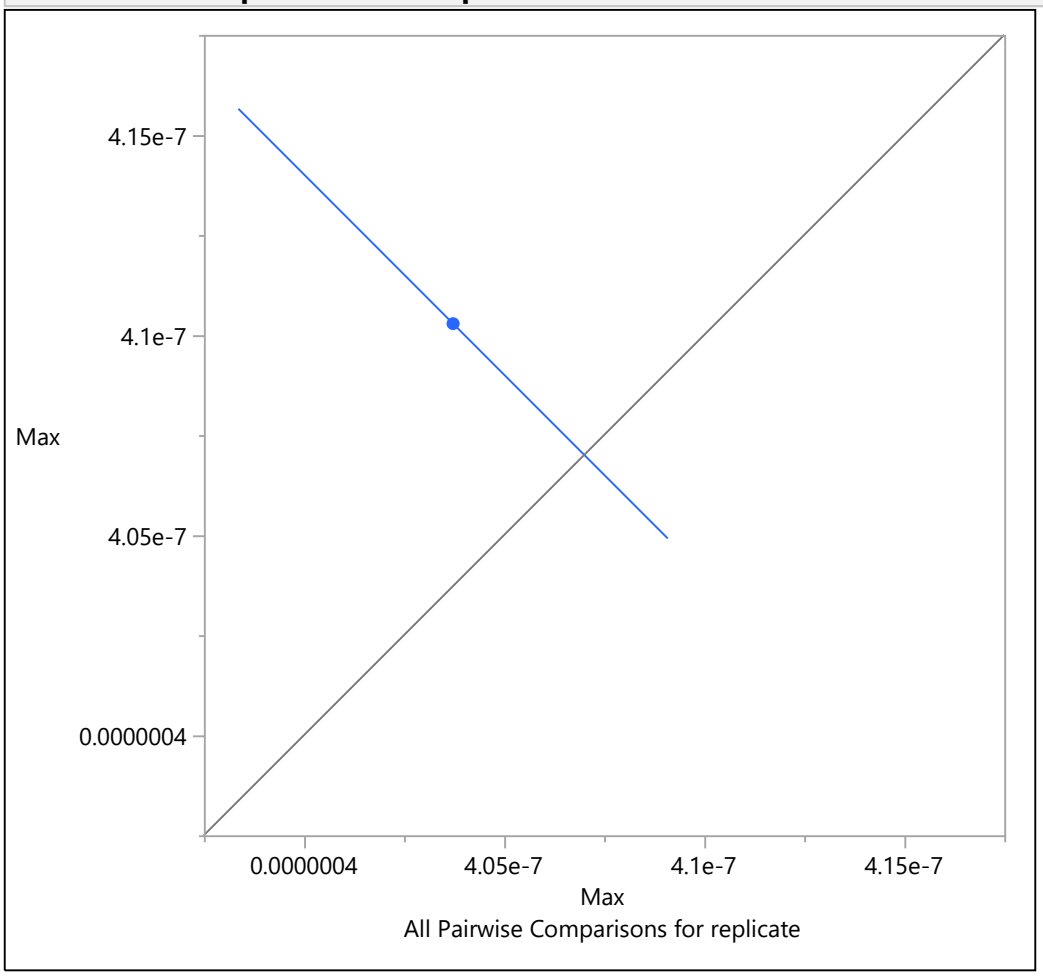
Tukey HSD All Pairwise Comparisons

Quantile = 1.97509, Adjusted DF = 158.0, Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-6.562e-9	5.4343e-9	-1.21	0.2290	-1.73e-8	4.1707e-9

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

Quantile = 1.97509, DF = 158.0

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-6.562e-9	5.4343e-9	-1.21	0.2290	-1.73e-8	4.1707e-9

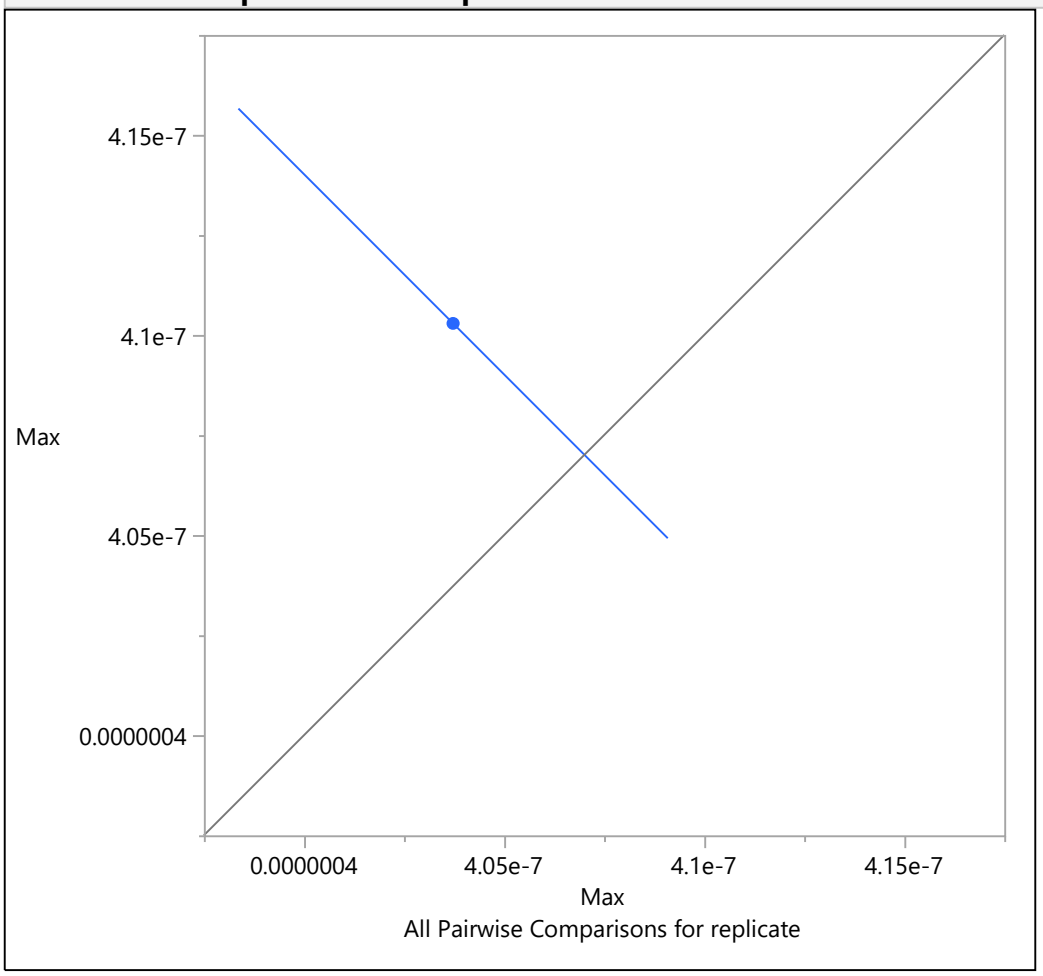
Legend
Significant
Not Significant

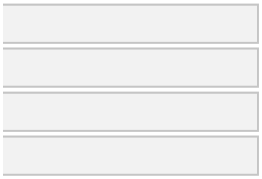
Response Max

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot





Legend

Significant

Not Significant

Appendix J. Multiple Comparison Test Results: Two Vehicle Operation

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	100.02390	0.46344967	477	99.113241	100.93455
1	100.03289	0.46344967	477	99.122239	100.94355

Tukey HSD All Pairwise Comparisons

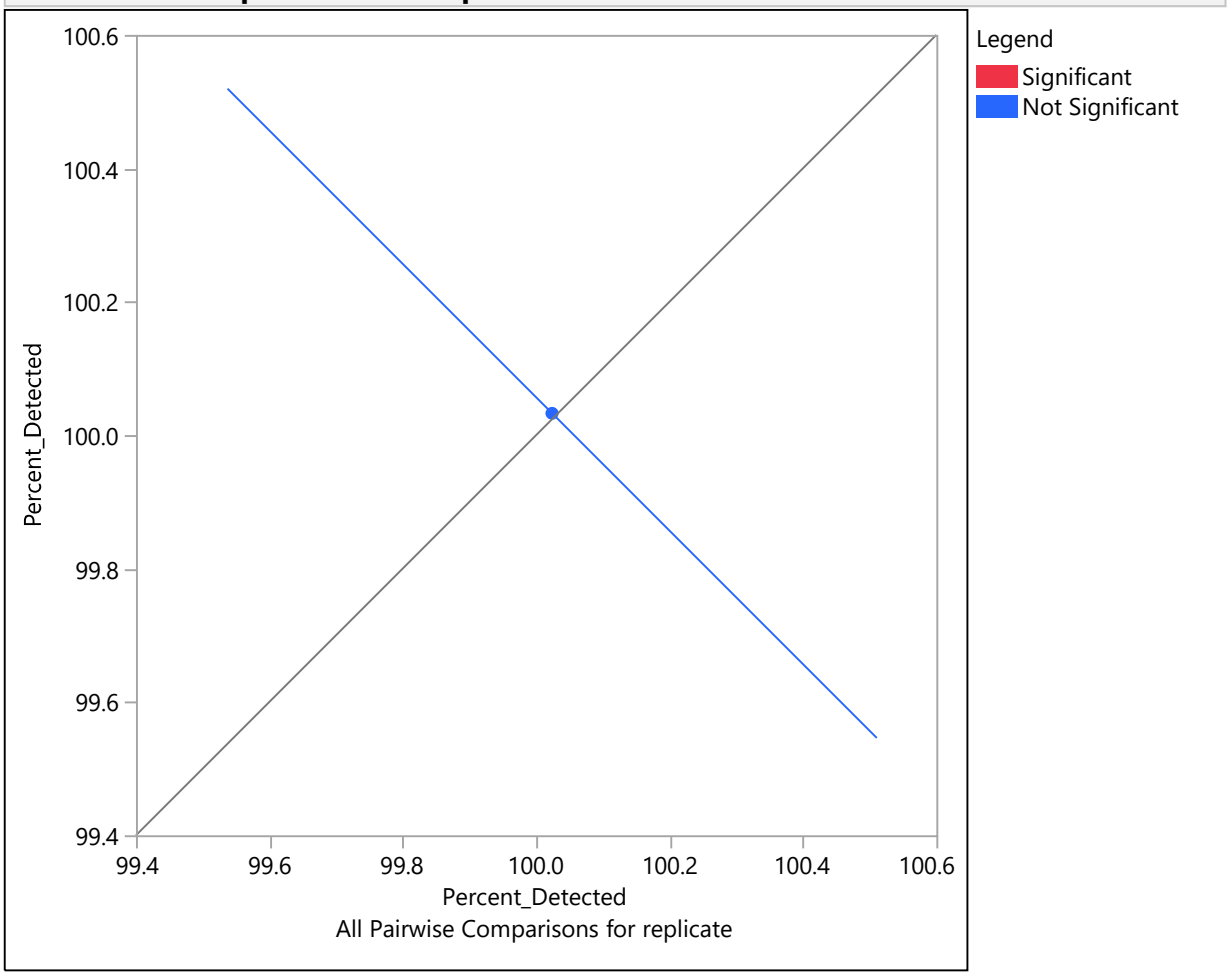
corp level = -1

Quantile = 1.96495 , Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982531	0.9645331

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

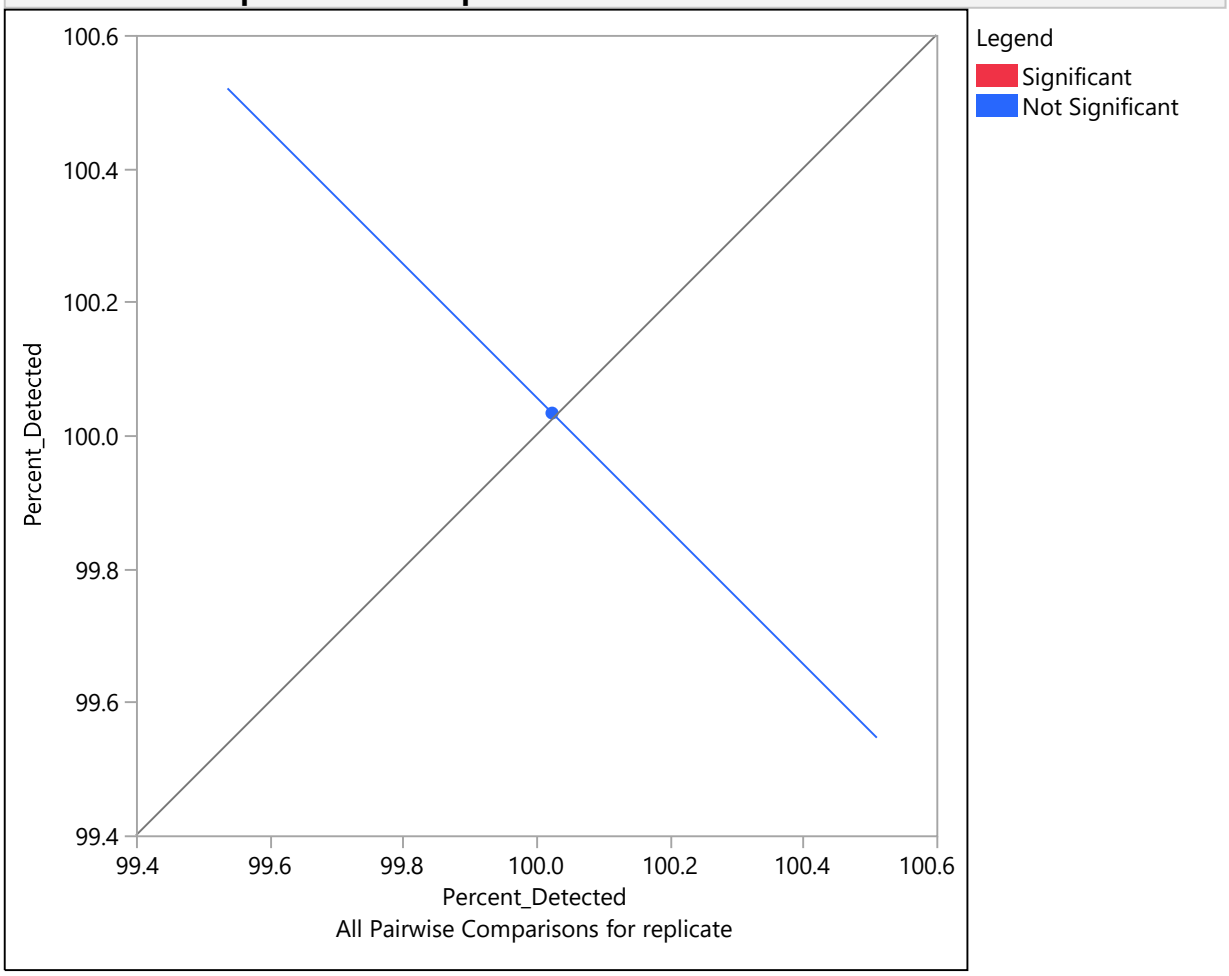
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982530	0.9645327

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	98.346264	0.35033502	477	97.657874	99.034655
1	98.355263	0.35033502	477	97.666872	99.043654

Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96495, Adjusted DF = 477.0, Adjustment = Tukey

All Pairwise Differences

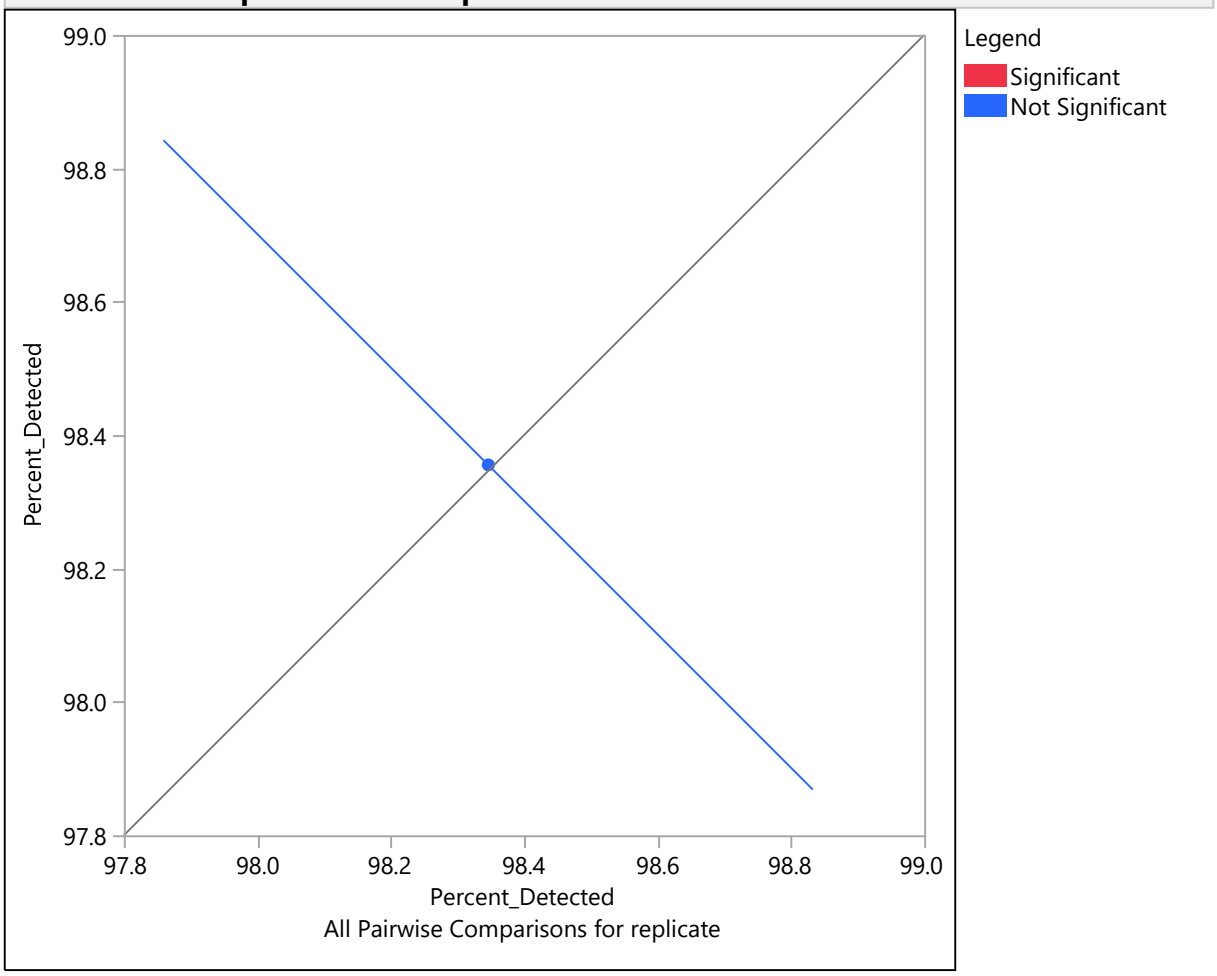
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982531	0.9645331

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 0
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

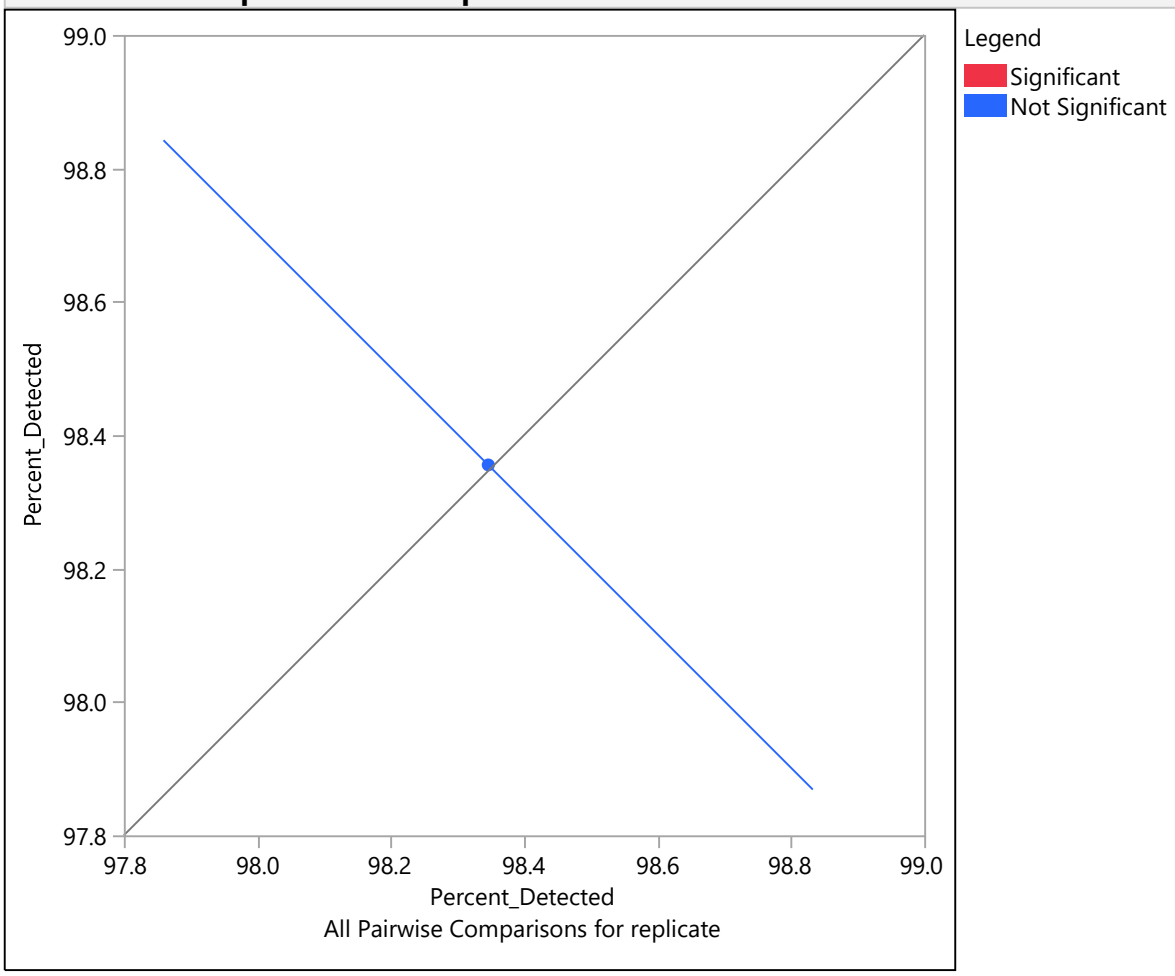
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982530	0.9645327

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	96.668633	0.46344967	477	95.757977	97.579288
1	96.677632	0.46344967	477	95.766976	97.588287

Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96495, Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

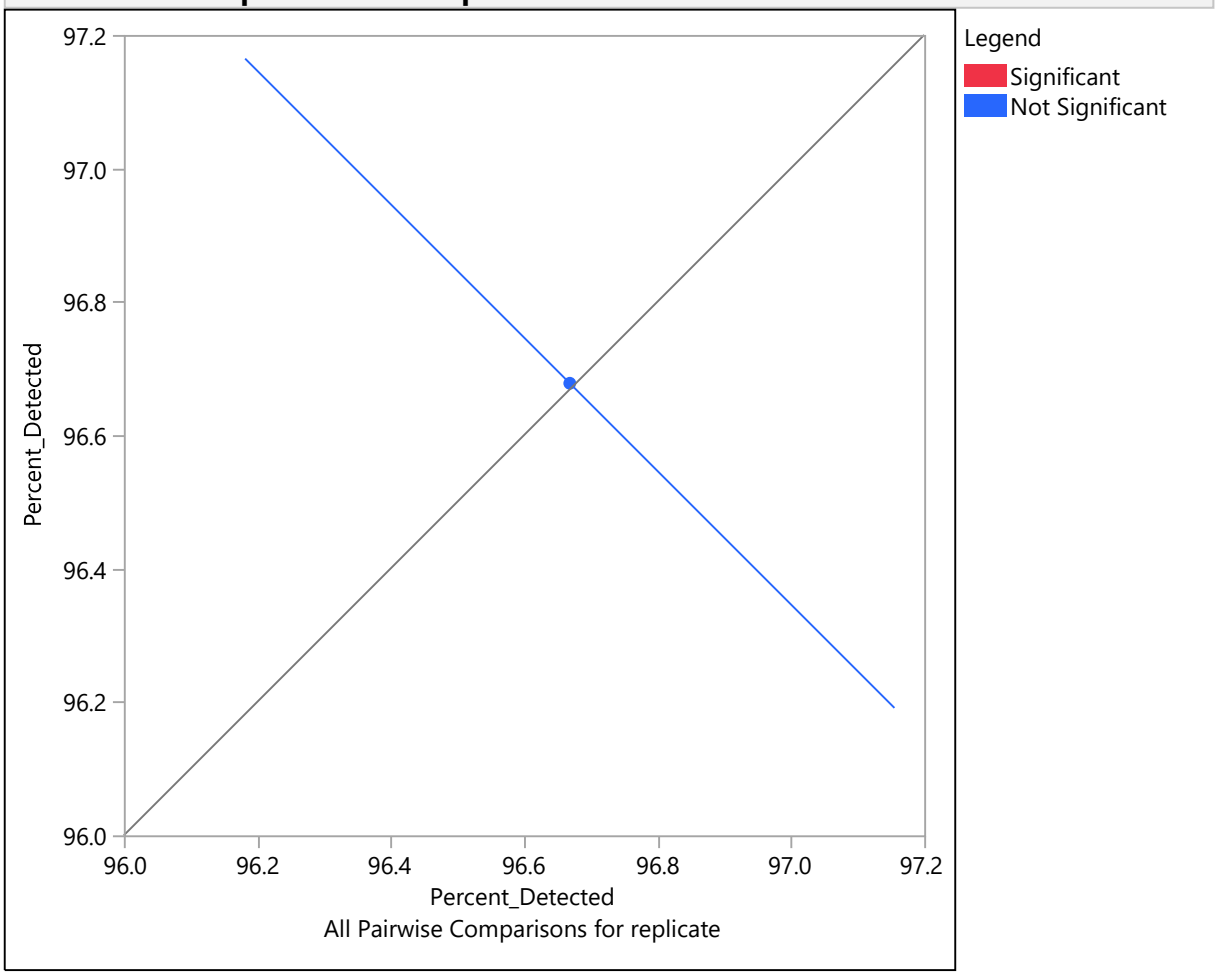
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982531	0.9645331

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 1
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

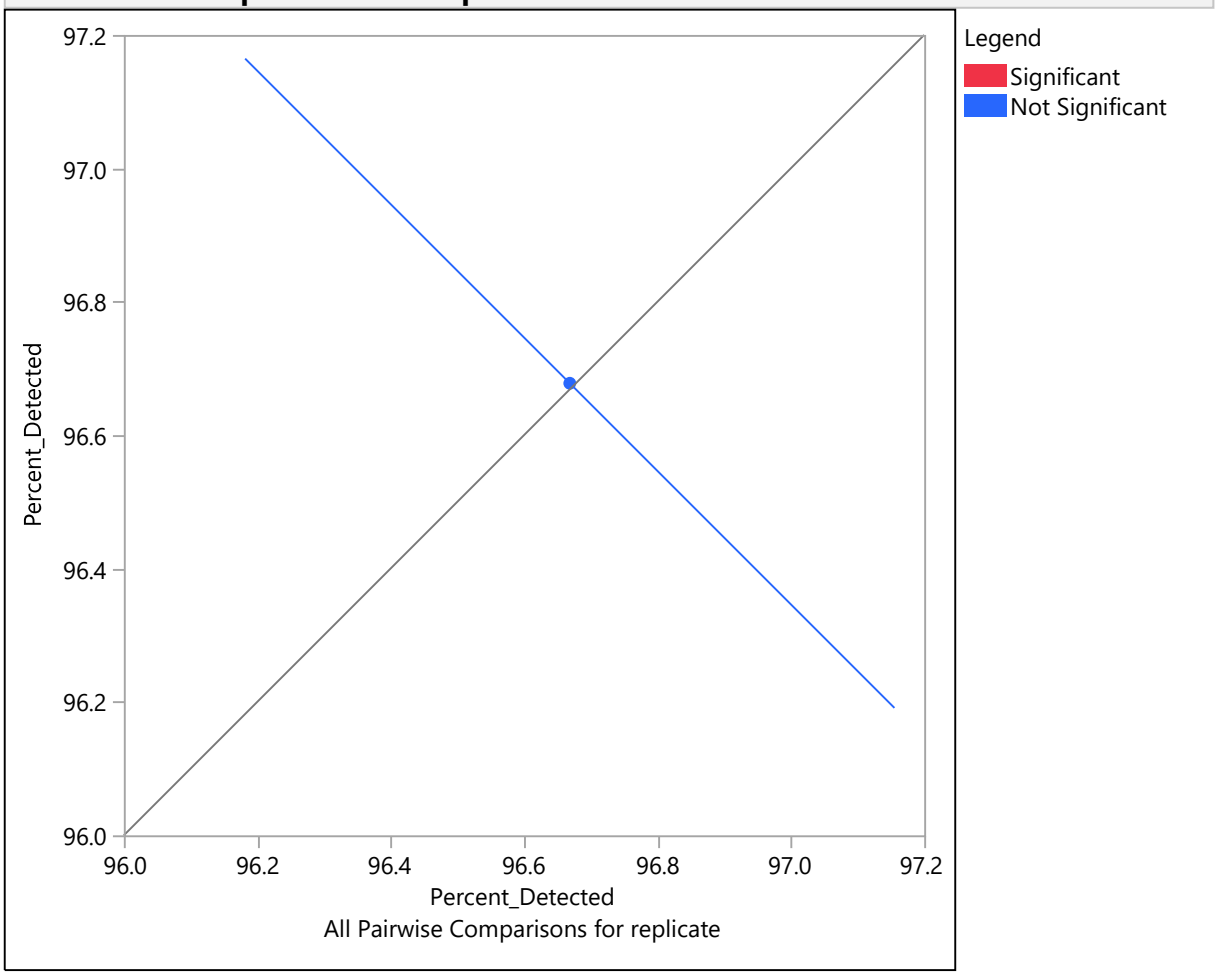
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982530	0.9645327

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	100.02390	0.46344967	477	99.113241	100.93455
1	100.03289	0.46344967	477	99.122239	100.94355

Tukey HSD All Pairwise Comparisons

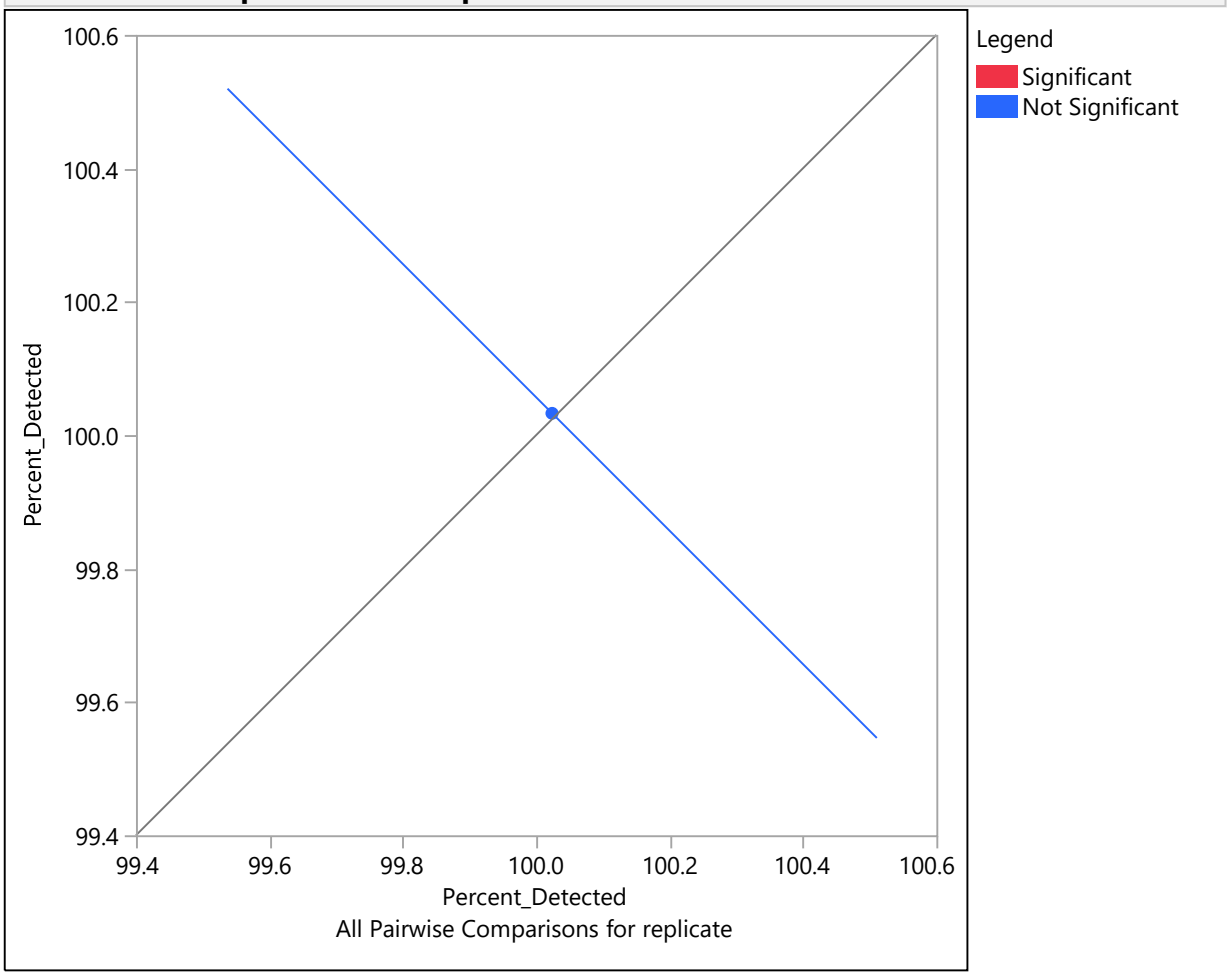
corp level = -1

Quantile = 1.96495 , Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982531	0.9645331

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

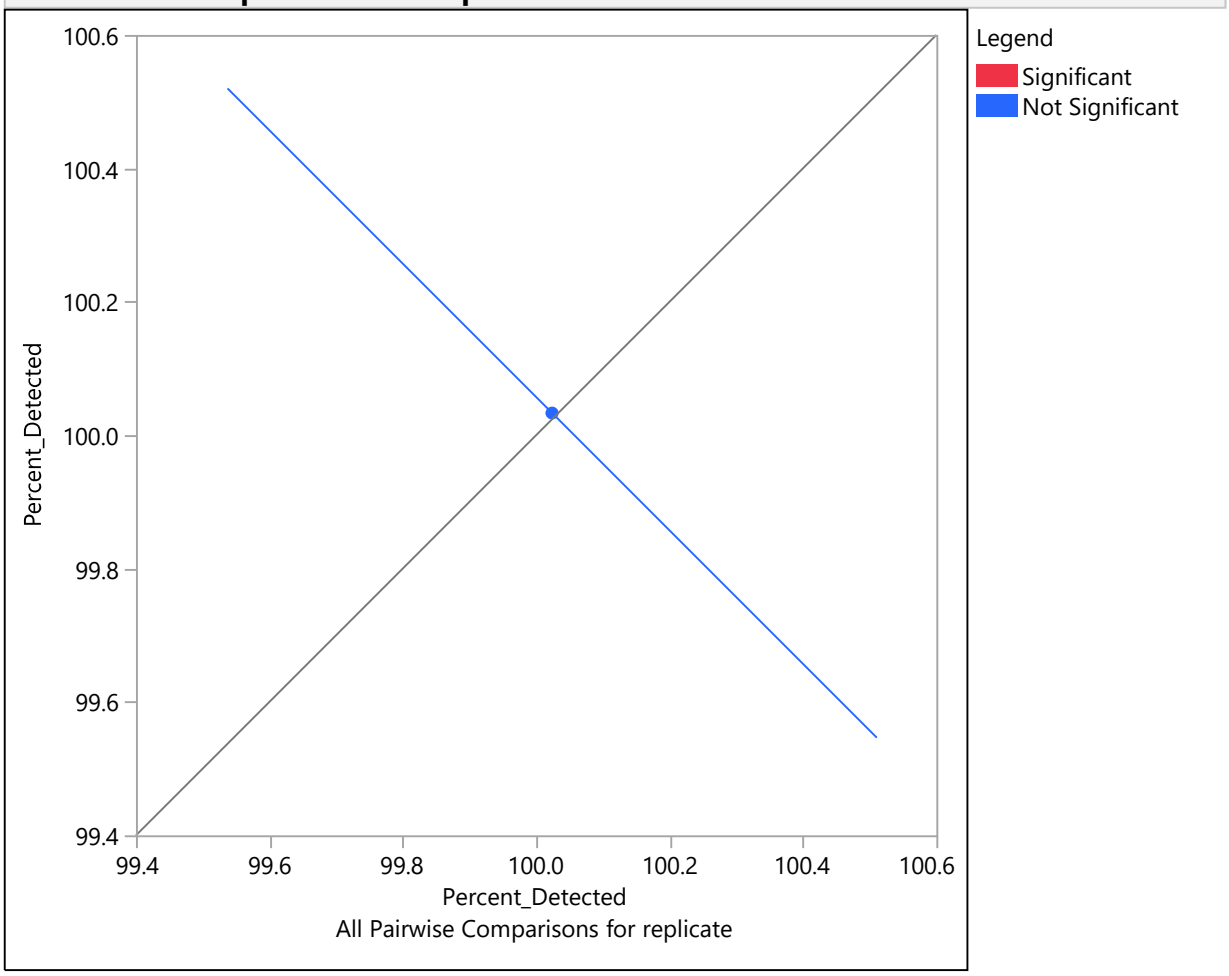
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982530	0.9645327

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	98.346264	0.35033502	477	97.657874	99.034655
1	98.355263	0.35033502	477	97.666872	99.043654

Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96495, Adjusted DF = 477.0, Adjustment = Tukey

All Pairwise Differences

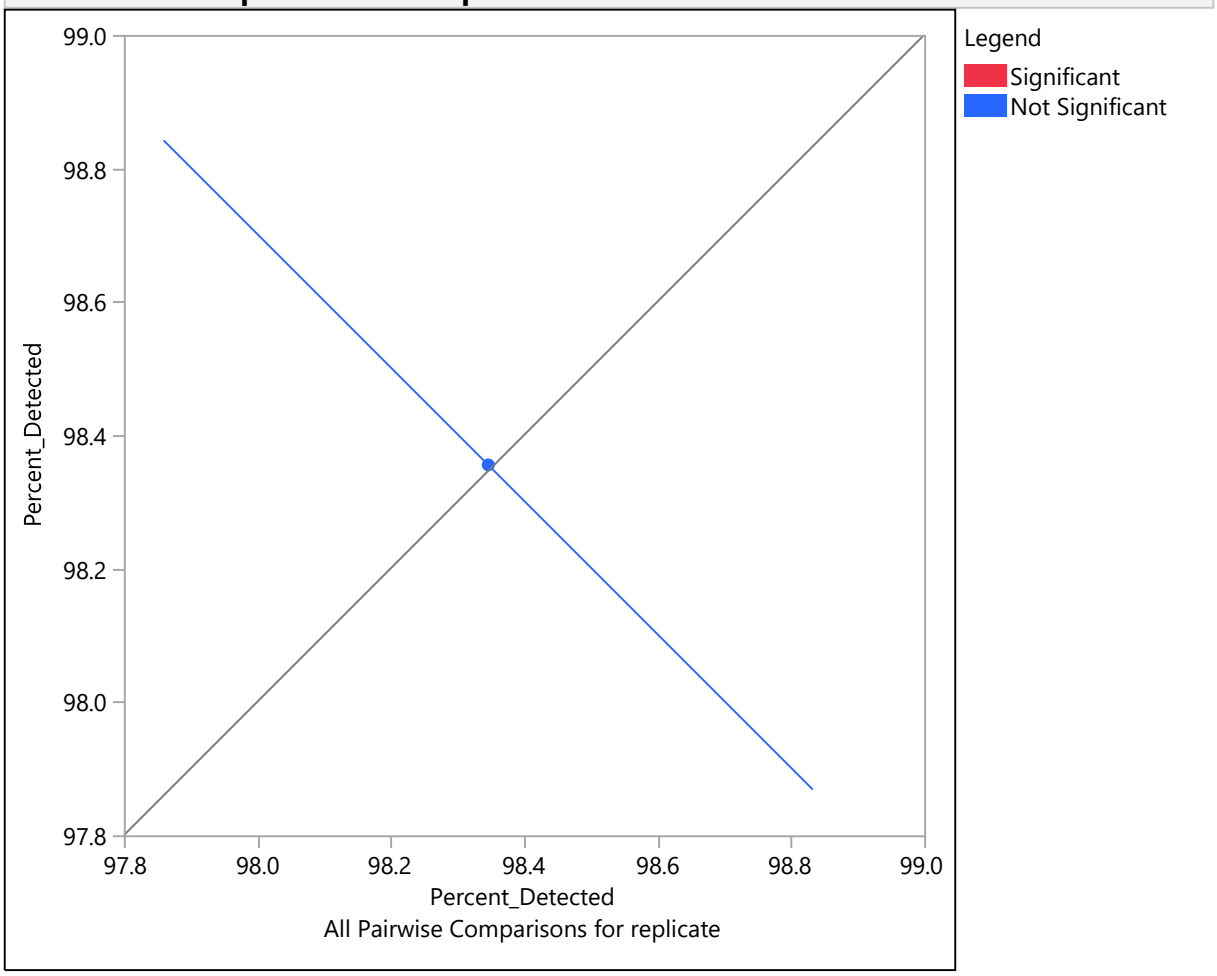
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982531	0.9645331

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 0
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

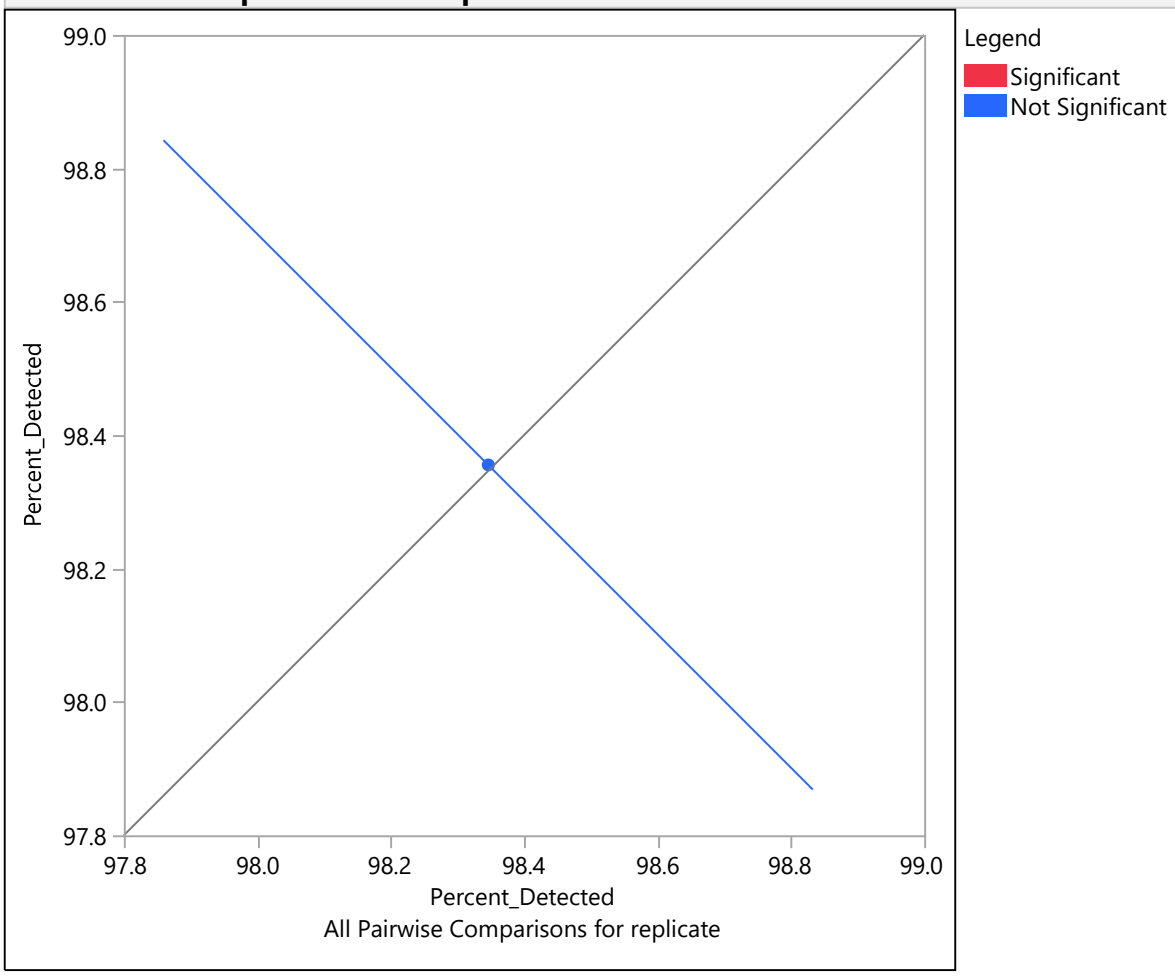
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982530	0.9645327

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	96.668633	0.46344967	477	95.757977	97.579288
1	96.677632	0.46344967	477	95.766976	97.588287

Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96495, Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

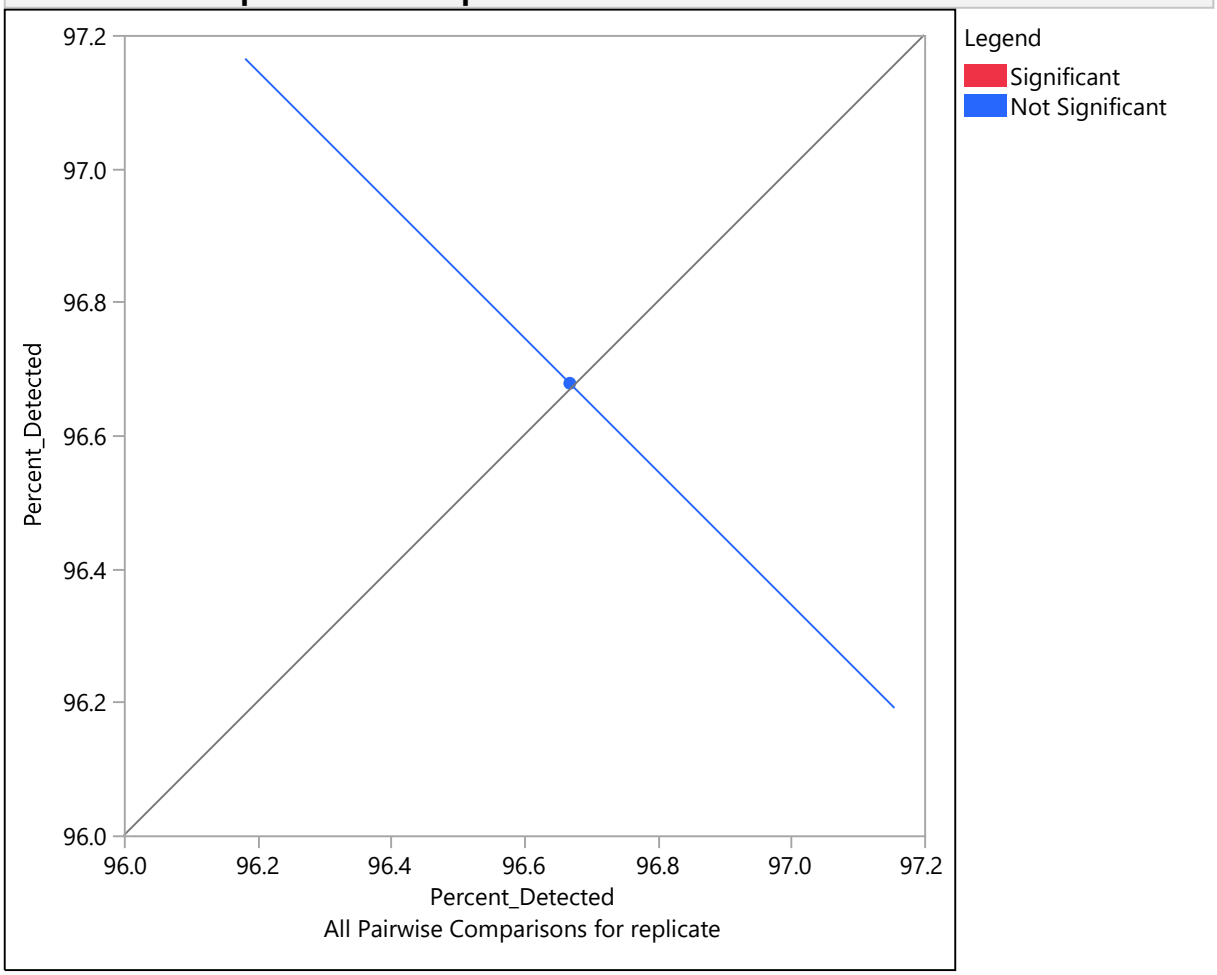
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982531	0.9645331

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 1
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

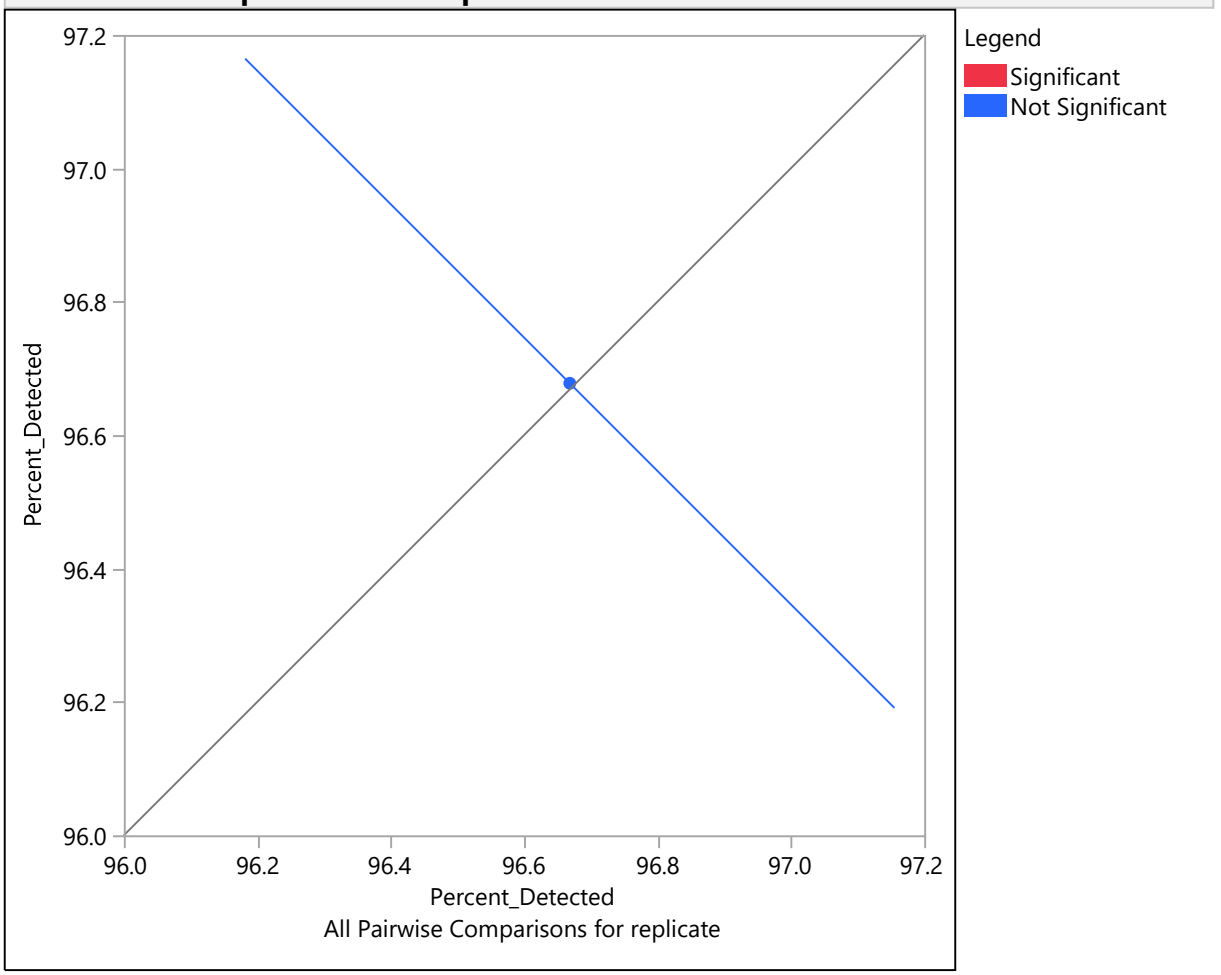
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.008999	0.4954485	-0.02	0.9855	-0.982530	0.9645327

Response Percent_Detected

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Perception_Accuracy

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	77.627898	1.4755611	477	74.728495	80.527301
1	77.785175	1.4755611	477	74.885772	80.684578

Tukey HSD All Pairwise Comparisons

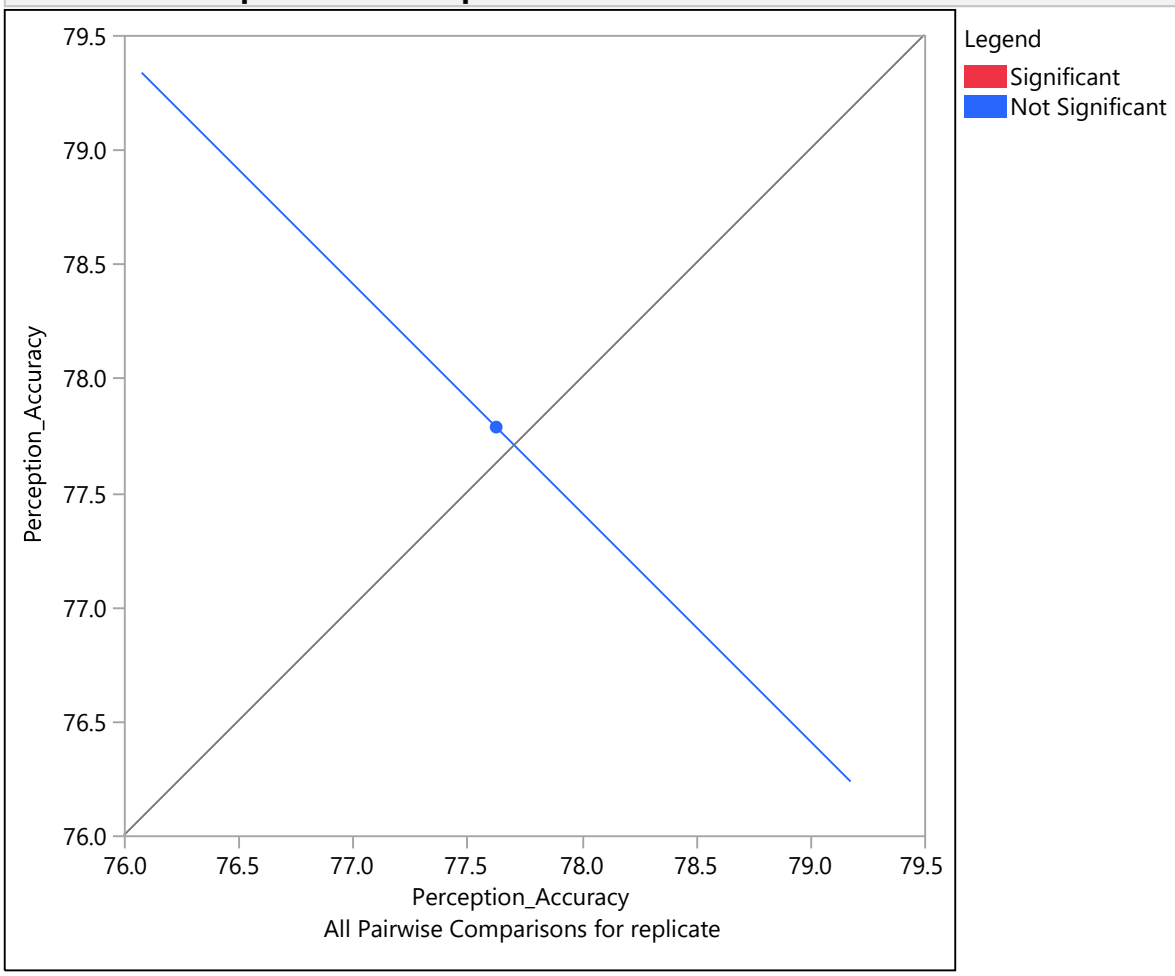
corp level = -1

Quantile = 1.96495 , Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.157277	1.577441	-0.10	0.9206	-3.25687	2.942317

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

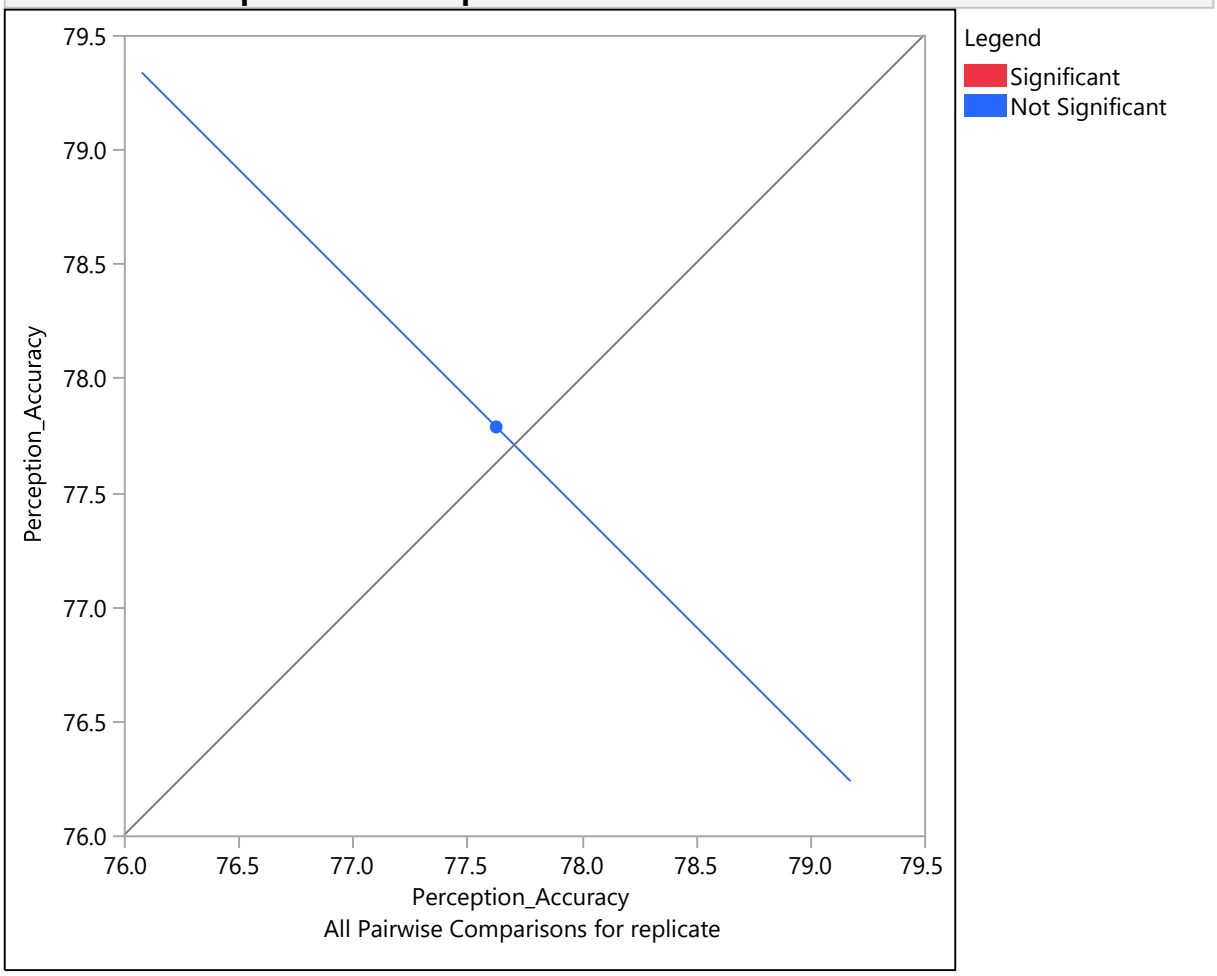
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.157277	1.577441	-0.10	0.9206	-3.25687	2.942316

Response Perception_Accuracy

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	79.741957	1.1154193	477	77.550214	81.933700
1	79.899234	1.1154193	477	77.707491	82.090977

Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey

All Pairwise Differences

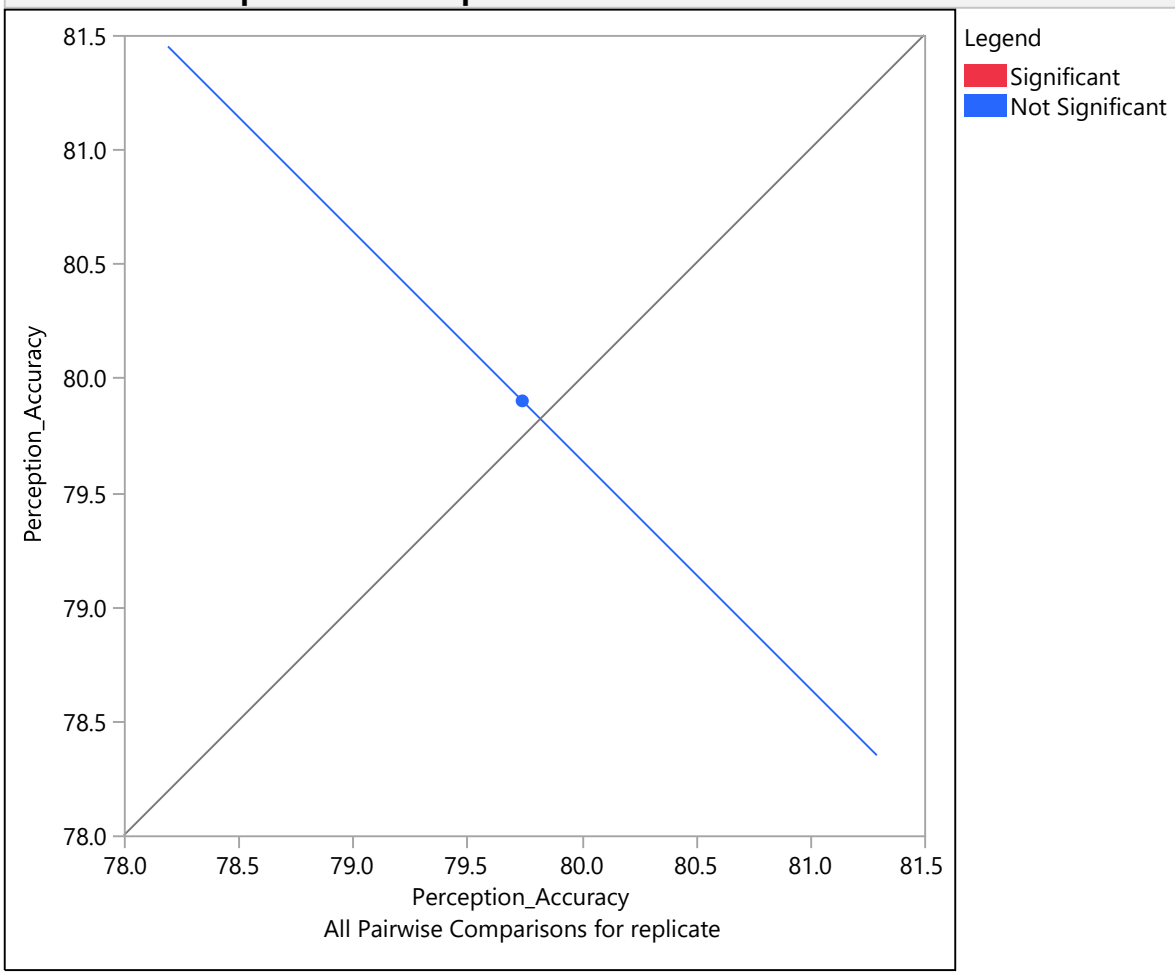
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.157277	1.577441	-0.10	0.9206	-3.25687	2.942317

Response Perception_Accuracy

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 0
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

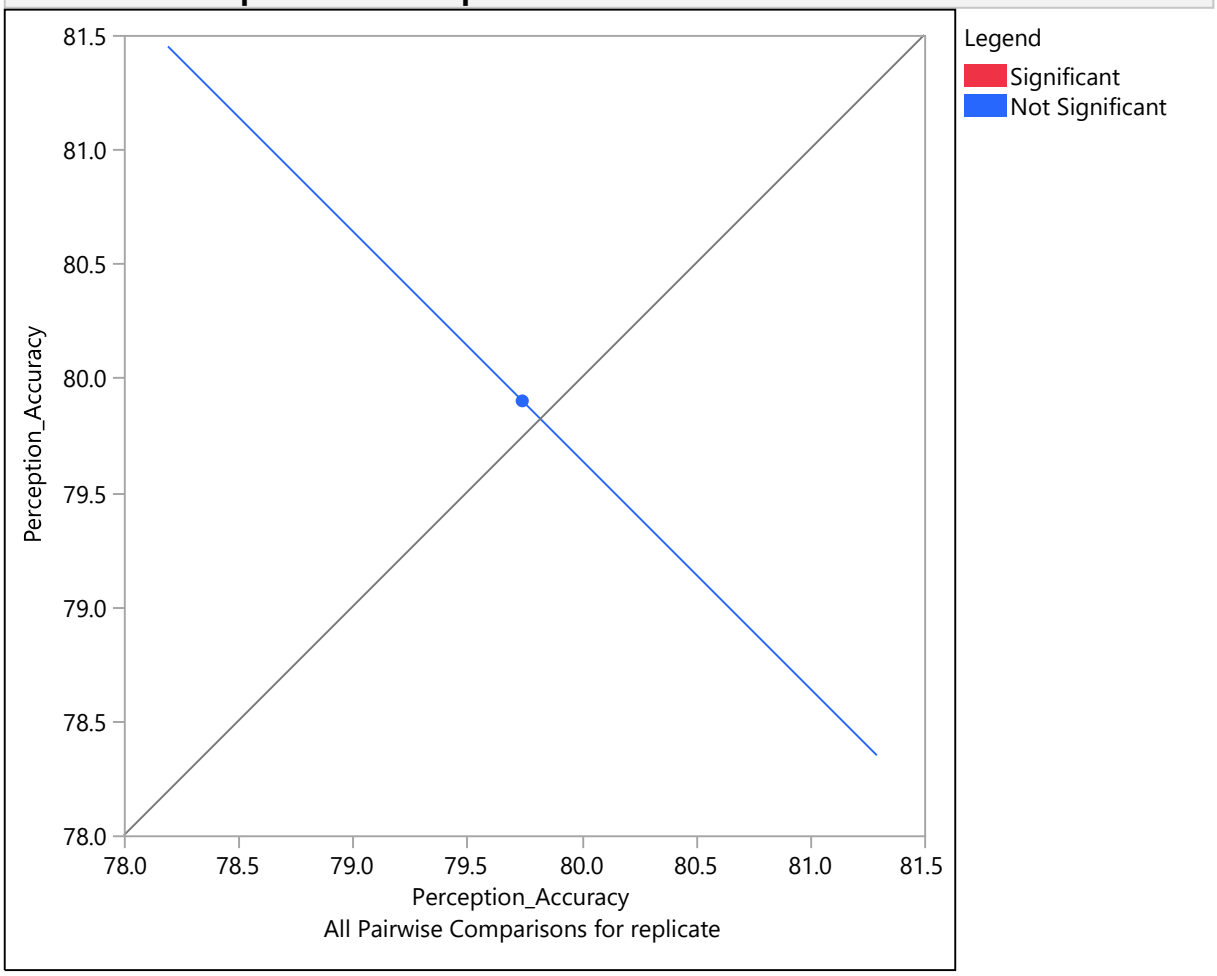
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.157277	1.577441	-0.10	0.9206	-3.25687	2.942316

Response Perception_Accuracy

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	81.856015	1.4755611	477	78.956612	84.755419
1	82.013292	1.4755611	477	79.113889	84.912696

Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey-Kramer

All Pairwise Differences

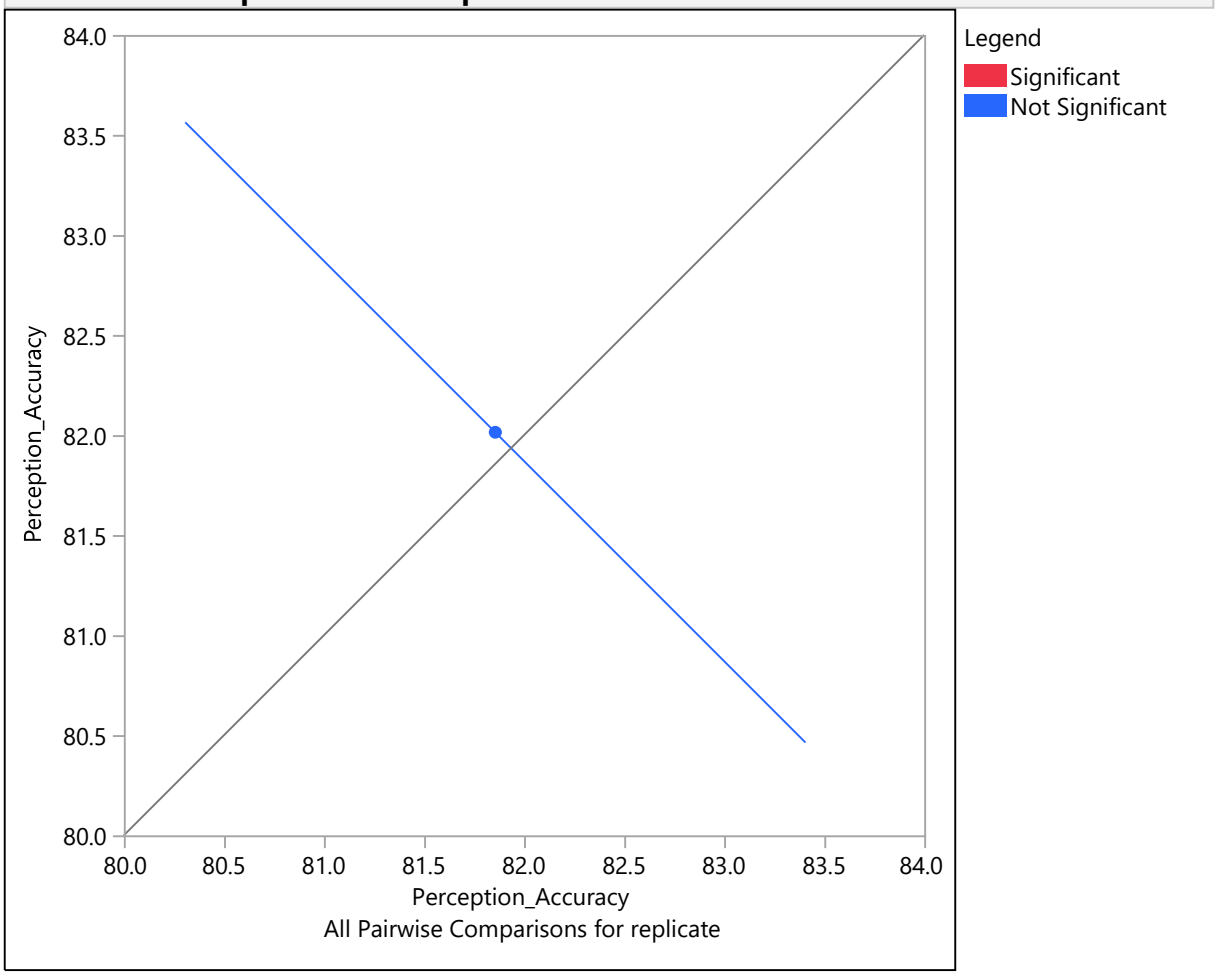
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.157277	1.577441	-0.10	0.9206	-3.25687	2.942317

Response Perception_Accuracy

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 1
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

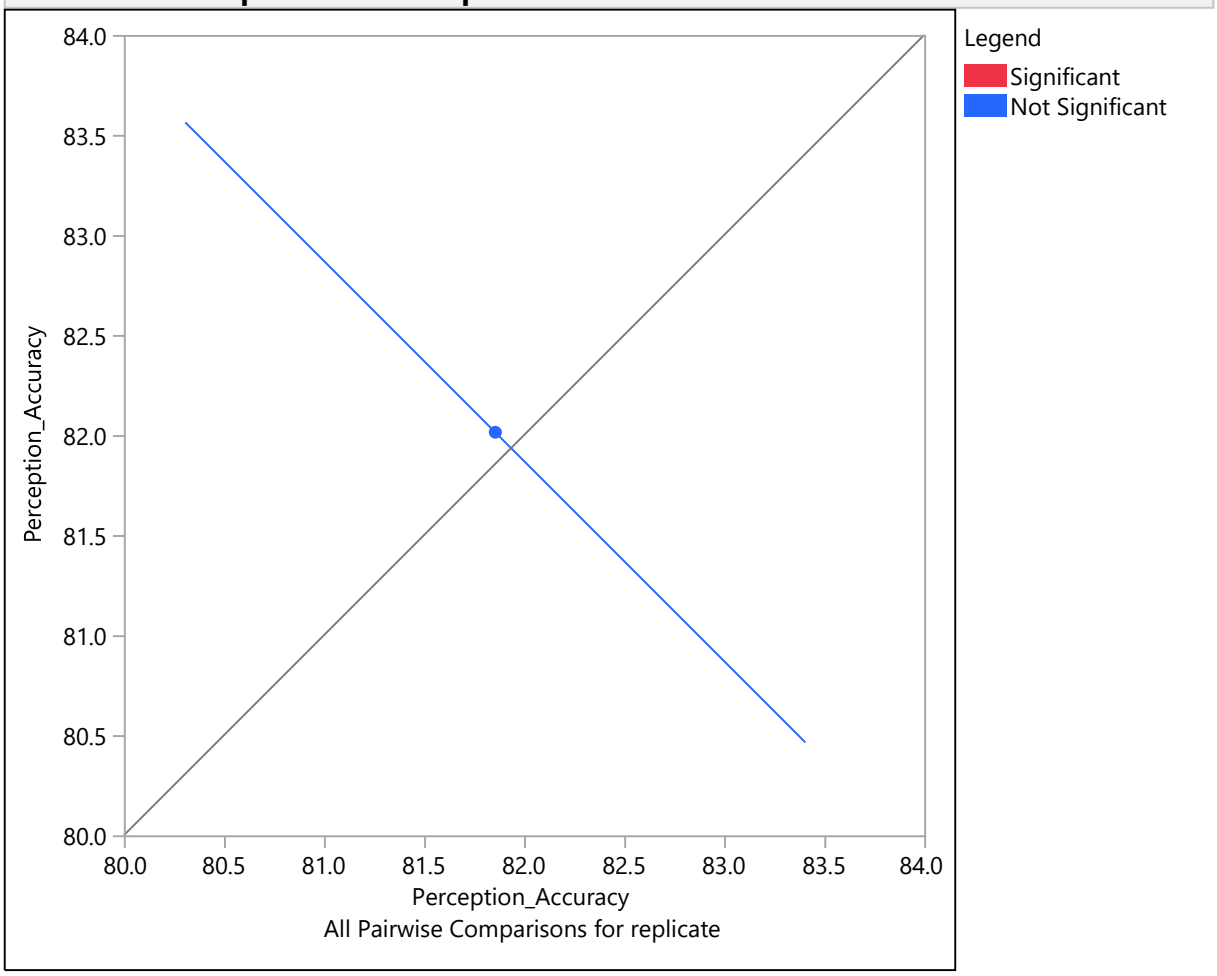
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.157277	1.577441	-0.10	0.9206	-3.25687	2.942316

Response Perception_Accuracy

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Detect_TypeII_error

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	19.547732	1.4928725	477	16.614312	22.481151
1	19.100830	1.4928725	477	16.167411	22.034250

Tukey HSD All Pairwise Comparisons

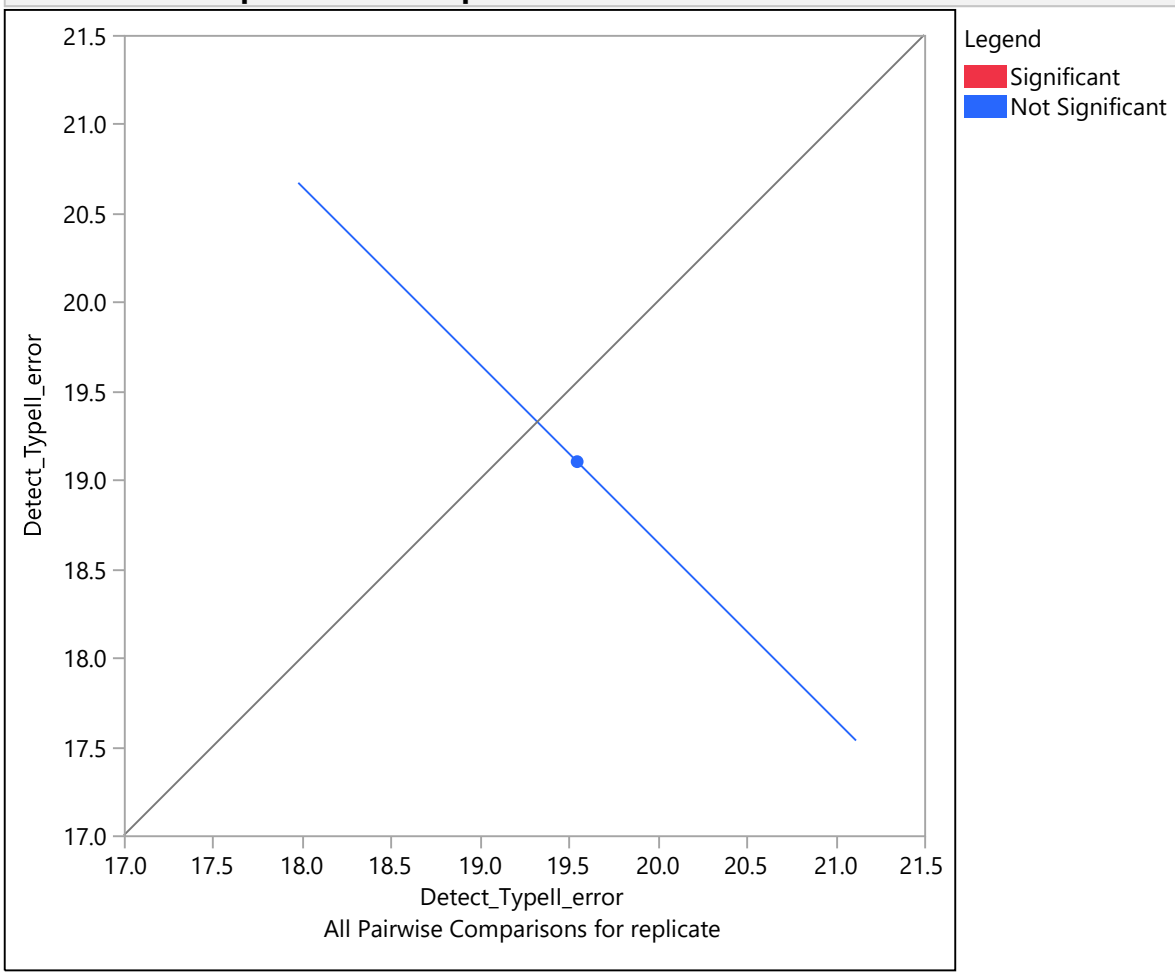
corp level = -1

Quantile = 1.96495 , Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.4469013	1.595948	0.28	0.7796	-2.68906	3.582860

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

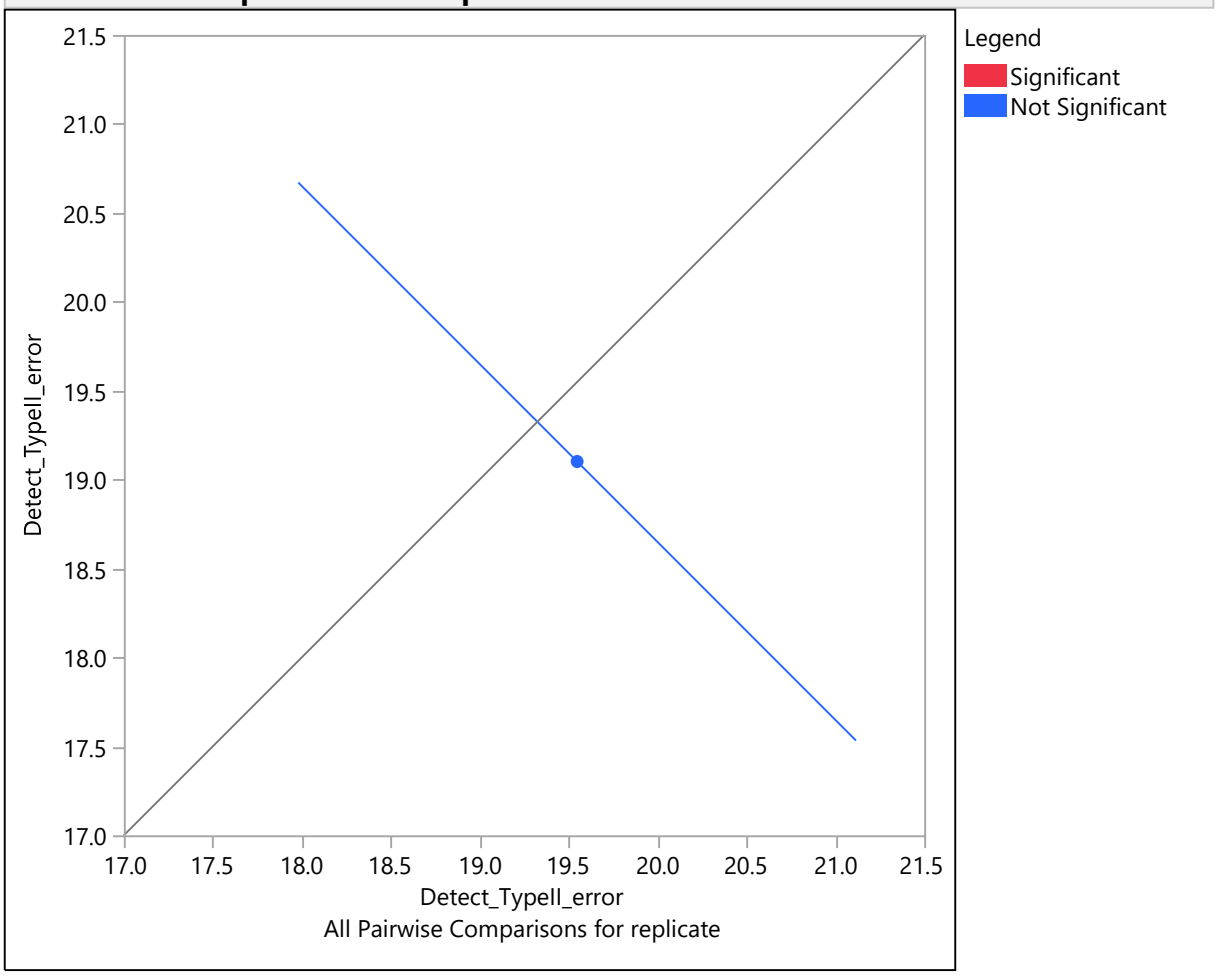
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.4469013	1.595948	0.28	0.7796	-2.68906	3.582859

Response Detect_TypeII_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	18.983199	1.1285055	477	16.765742	21.200656
1	18.536298	1.1285055	477	16.318841	20.753754

Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96495, Adjusted DF = 477.0, Adjustment = Tukey

All Pairwise Differences

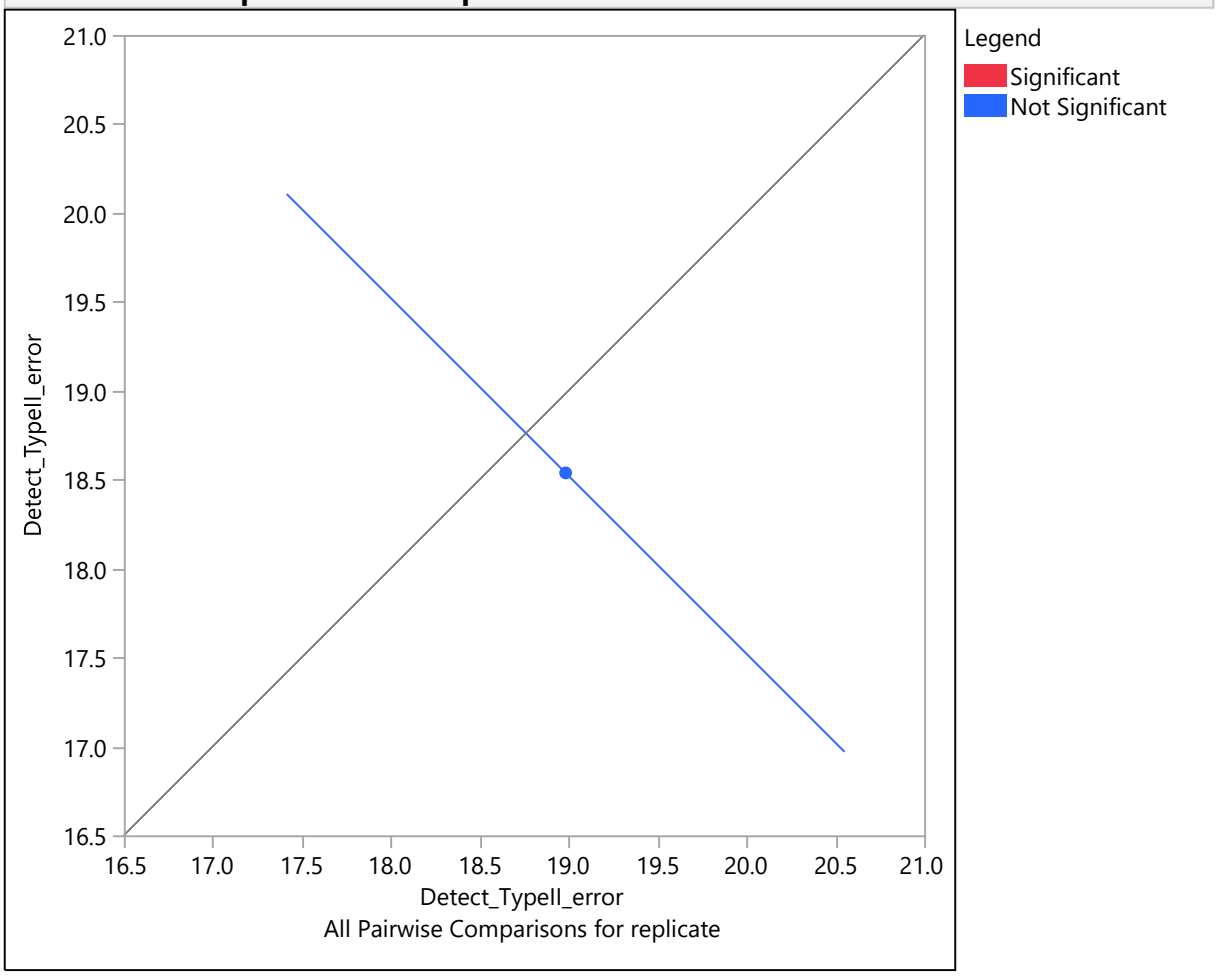
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.4469013	1.595948	0.28	0.7796	-2.68906	3.582860

Response Detect_TypeII_error

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 0

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

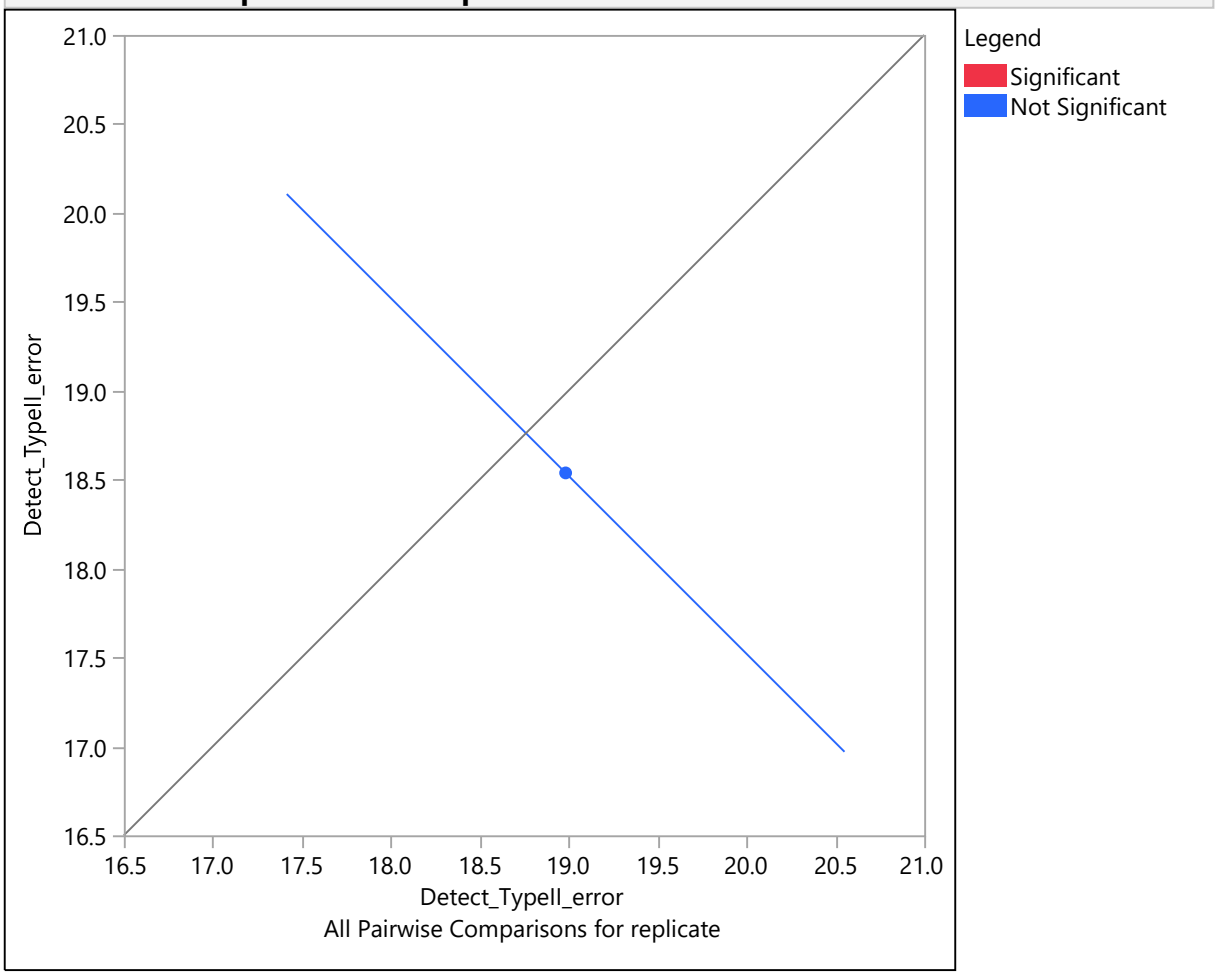
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.4469013	1.595948	0.28	0.7796	-2.68906	3.582859

Response Detect_TypeII_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	18.418666	1.4928725	477	15.485247	21.352086
1	17.971765	1.4928725	477	15.038346	20.905184

Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey-Kramer

All Pairwise Differences

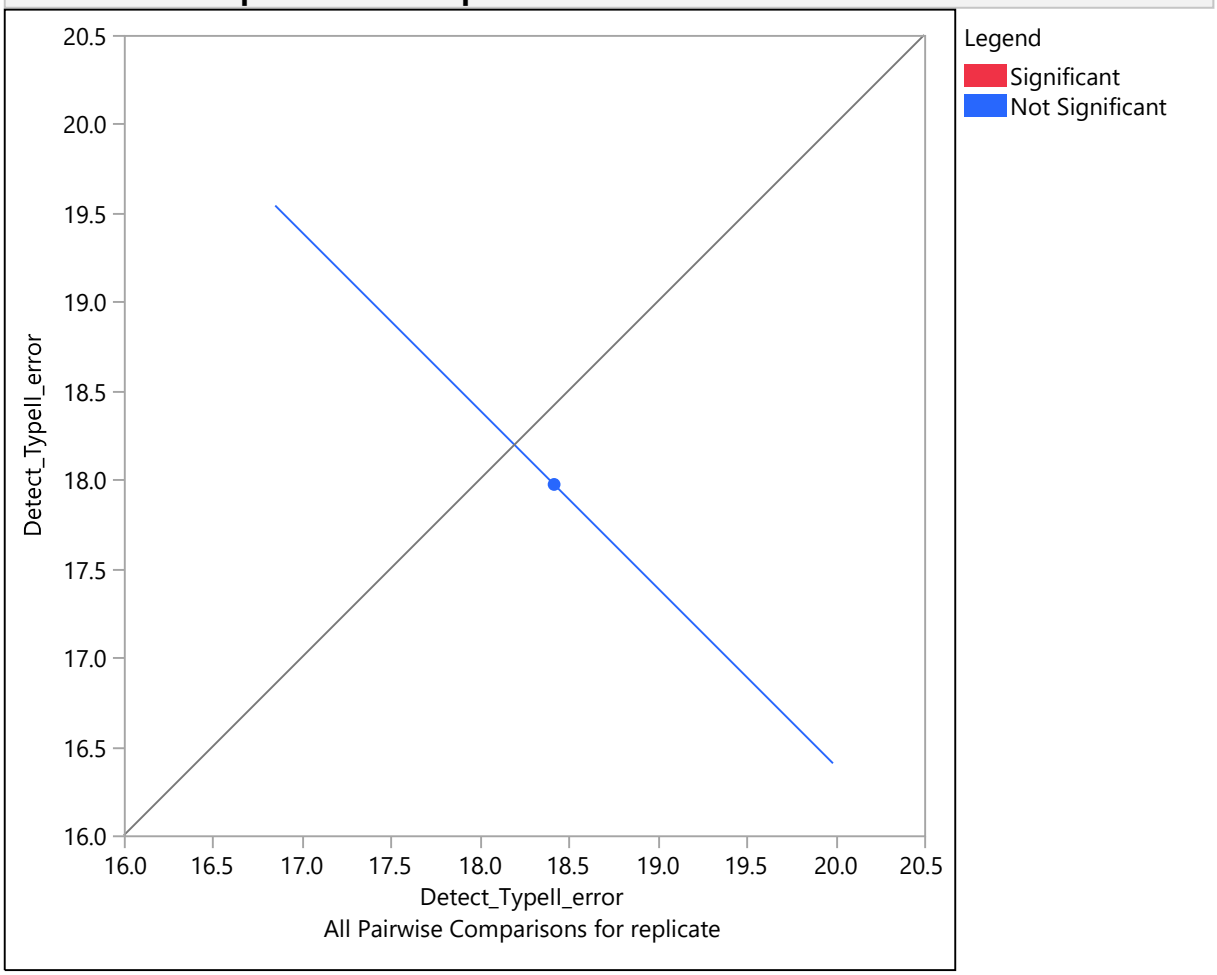
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.4469013	1.595948	0.28	0.7796	-2.68906	3.582860

Response Detect_TypeII_error

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

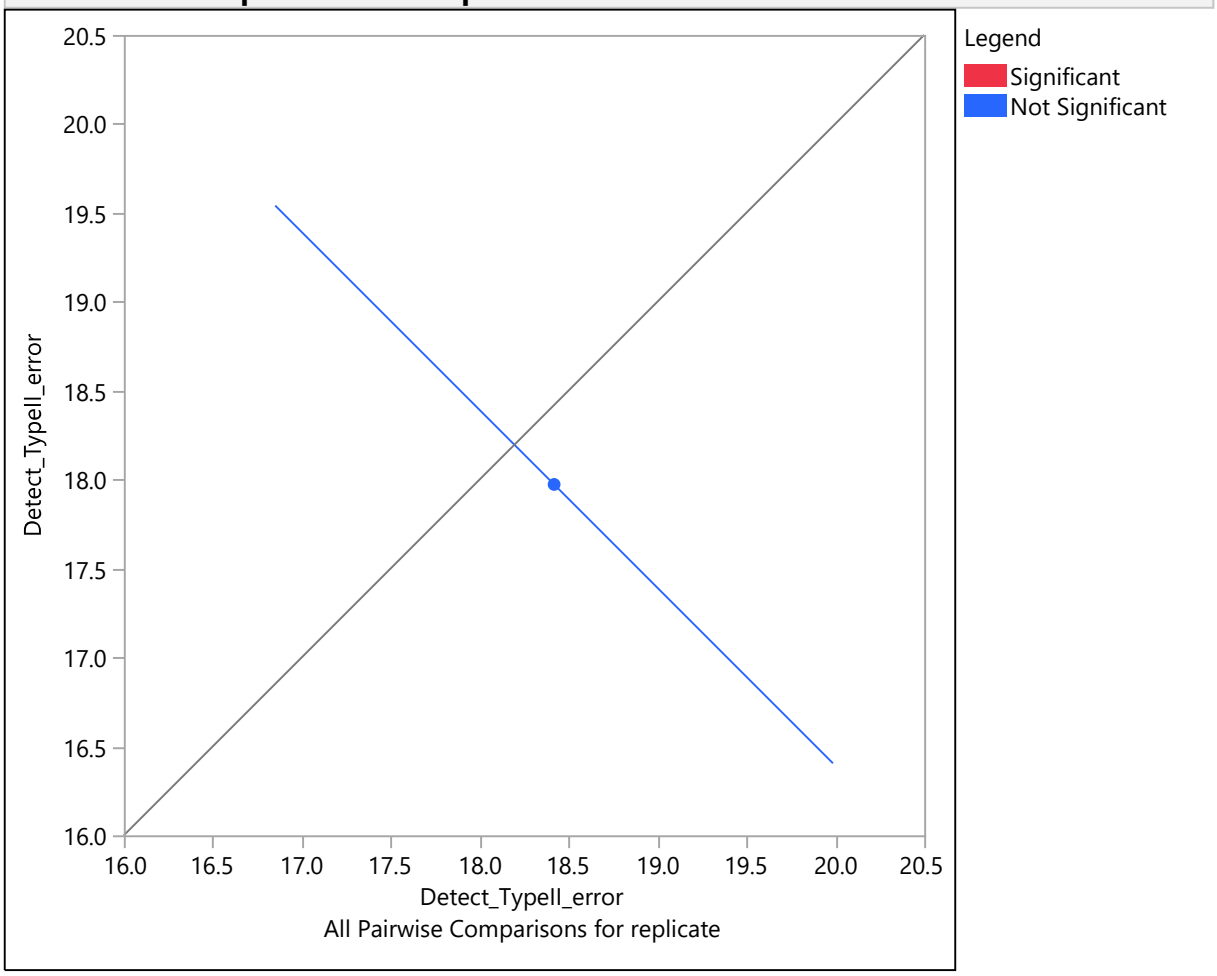
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.4469013	1.595948	0.28	0.7796	-2.68906	3.582859

Response Detect_TypeII_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Confirm_TypeII_error

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	11.042013	1.7508611	477	7.6016595	14.482367
1	12.456437	1.7508611	477	9.0160828	15.896791

Tukey HSD All Pairwise Comparisons

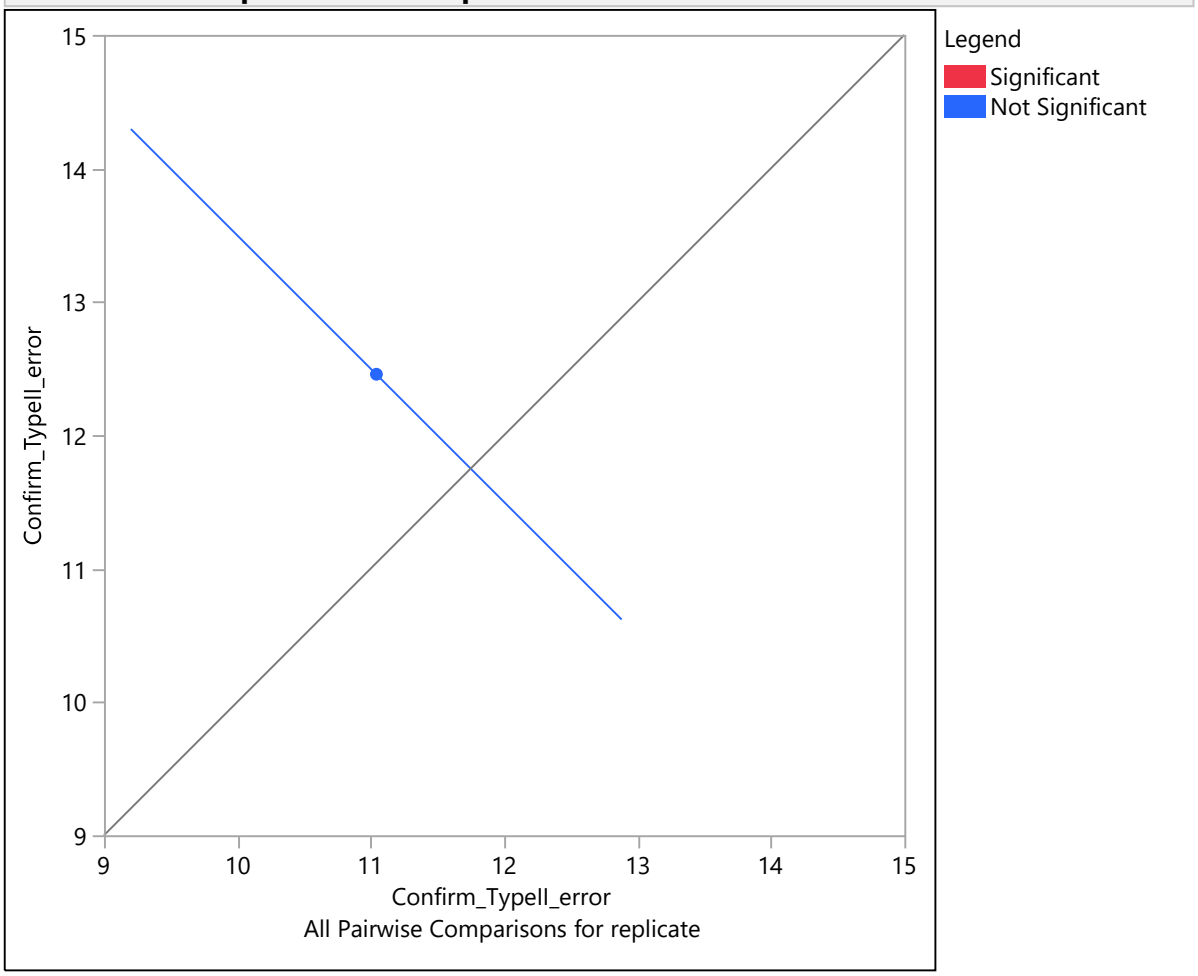
corp level = -1

Quantile = 1.96495 , Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-1.41442	1.871749	-0.76	0.4502	-5.09232	2.263471

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

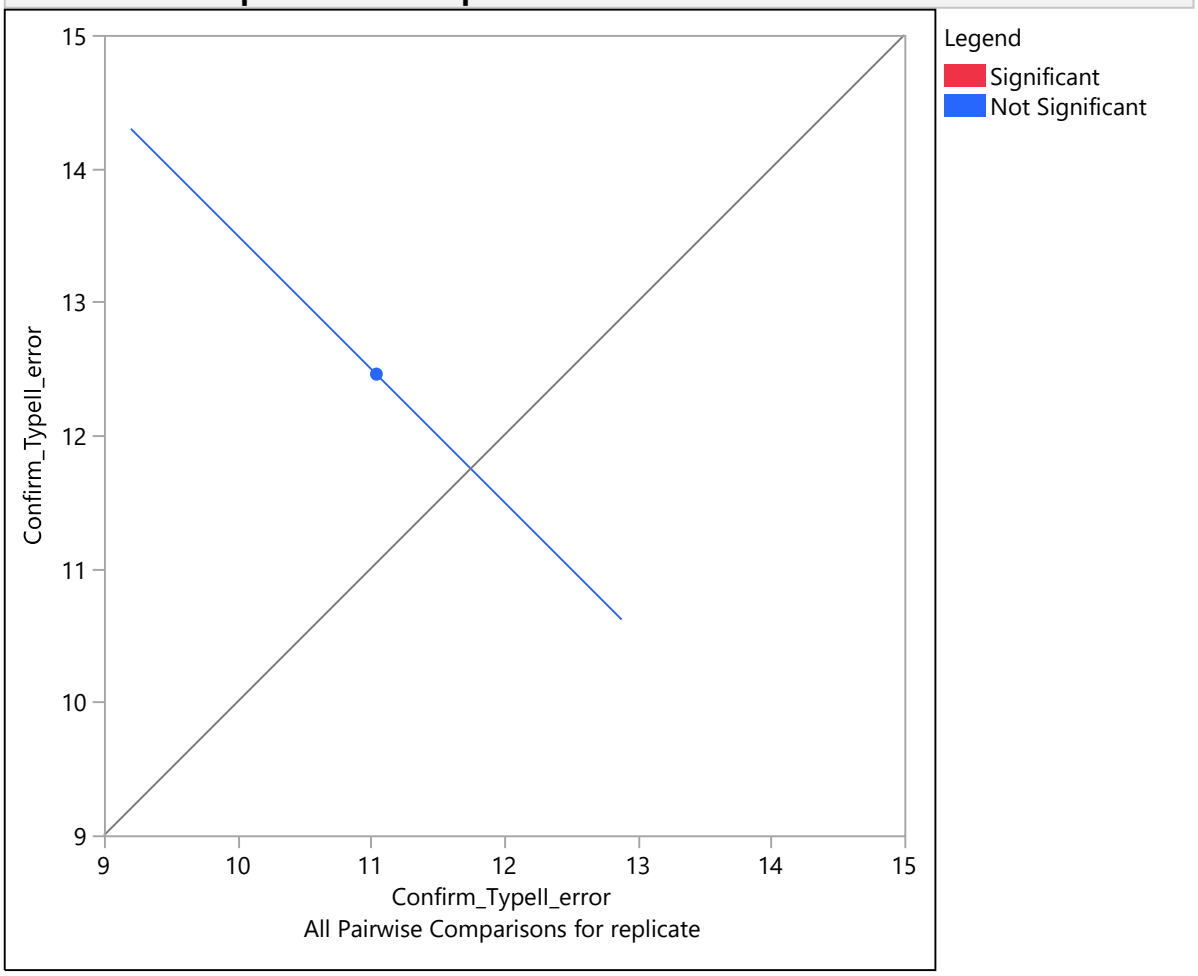
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-1.41442	1.871749	-0.76	0.4502	-5.09232	2.263470

Response Confirm_TypeII_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	13.404461	1.3235265	477	10.803798	16.005124
1	14.818885	1.3235265	477	12.218221	17.419548

Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey

All Pairwise Differences

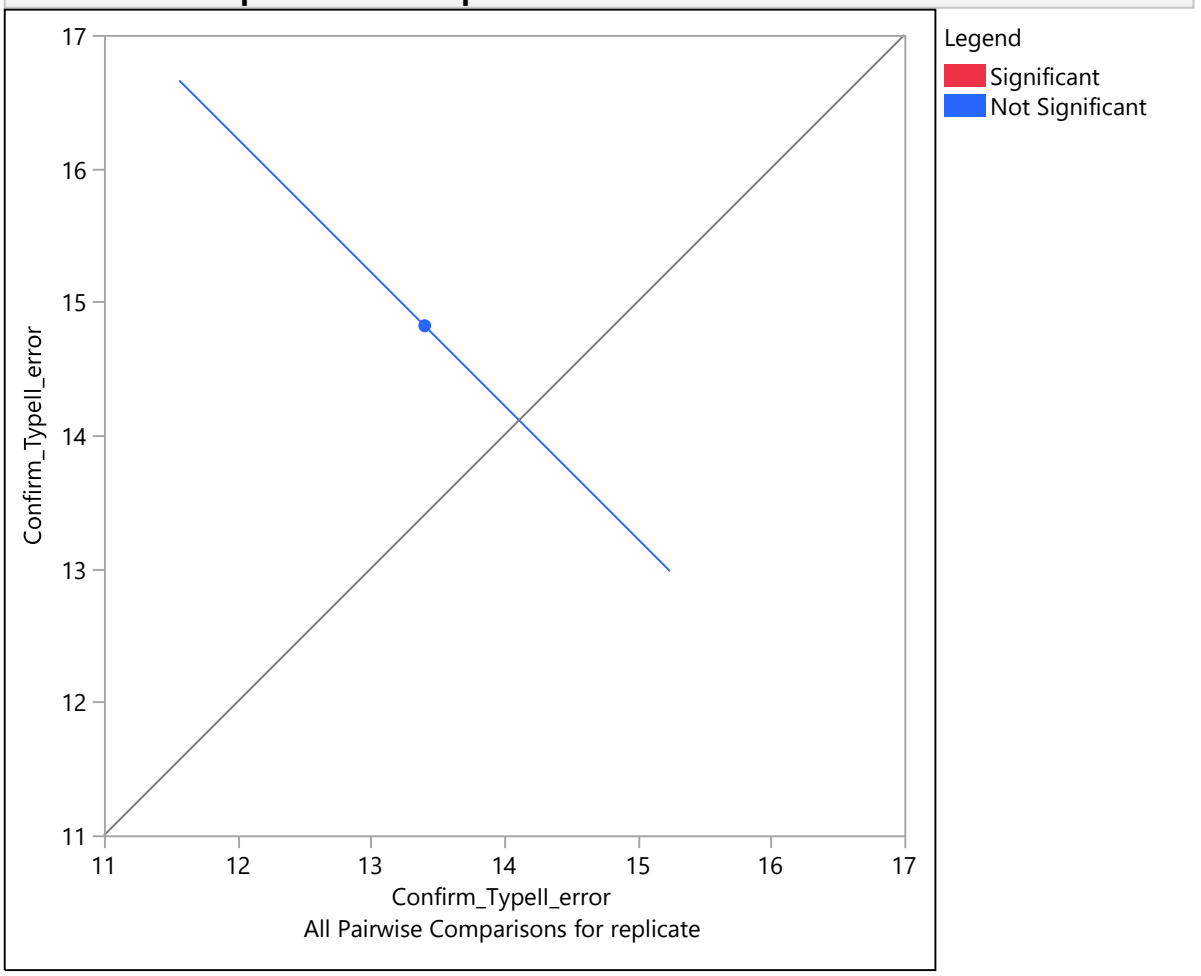
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-1.41442	1.871749	-0.76	0.4502	-5.09232	2.263471

Response Confirm_TypeII_error

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 0
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

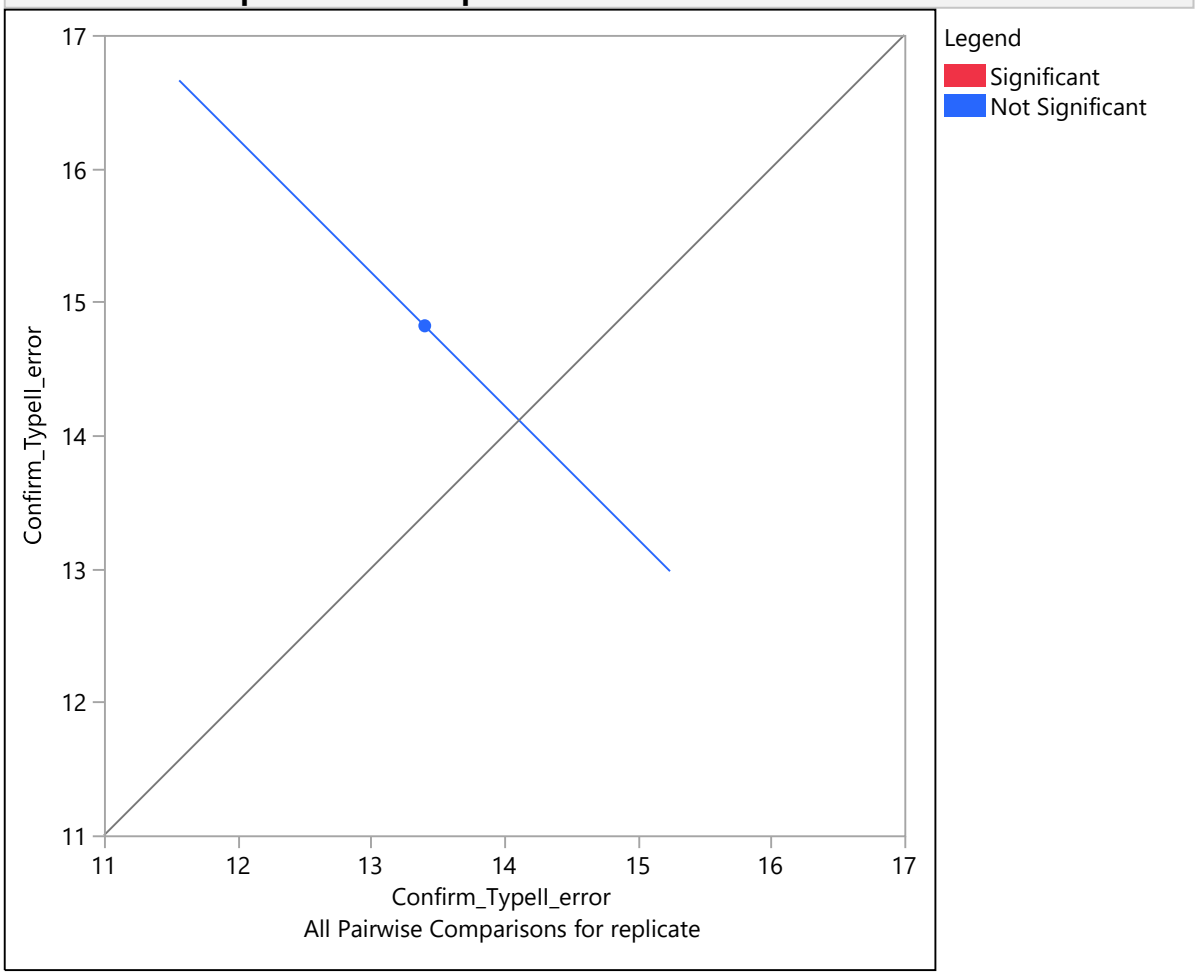
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-1.41442	1.871749	-0.76	0.4502	-5.09232	2.263470

Response Confirm_TypeII_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	15.766909	1.7508611	477	12.326555	19.207263
1	17.181332	1.7508611	477	13.740978	20.621686

Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey-Kramer

All Pairwise Differences

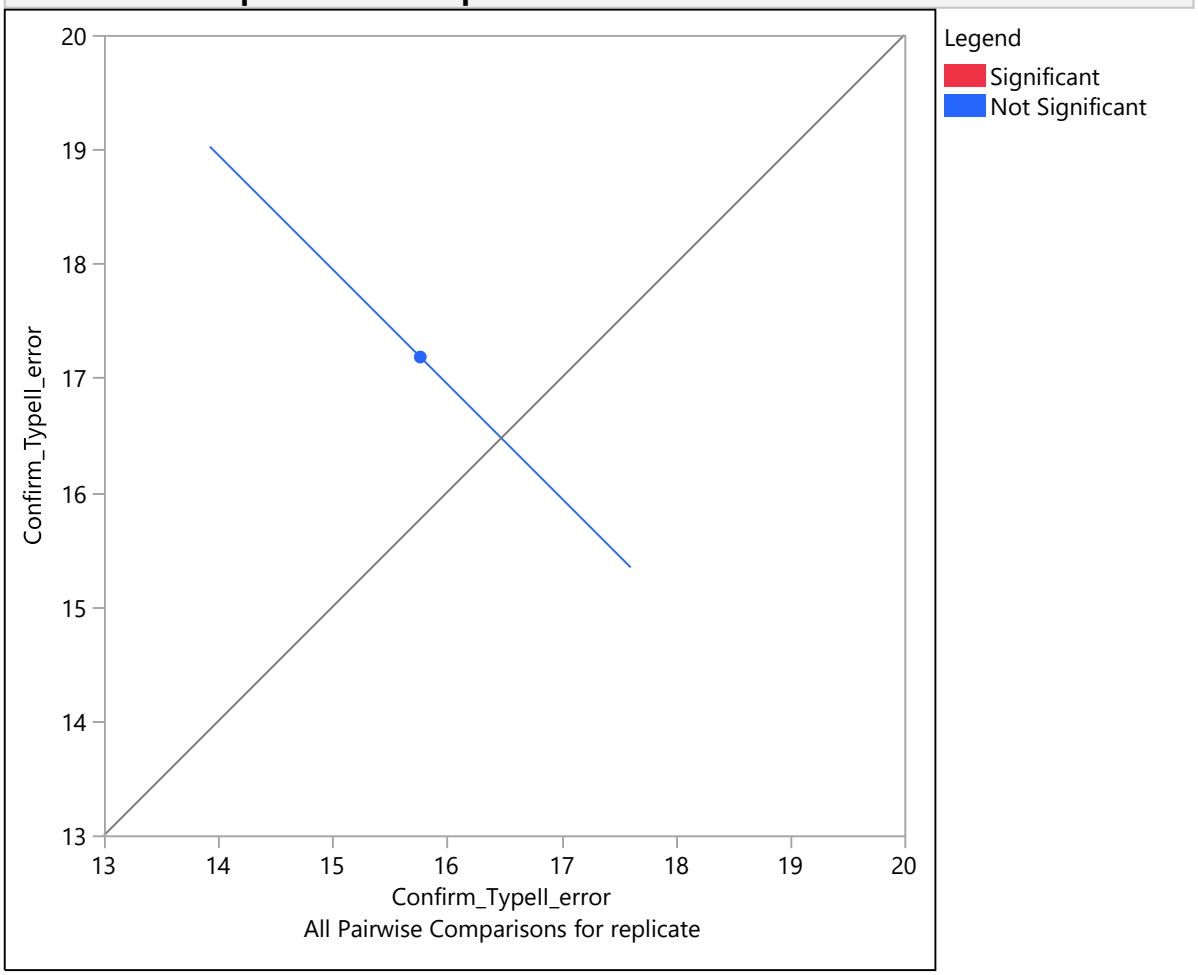
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-1.41442	1.871749	-0.76	0.4502	-5.09232	2.263471

Response Confirm_TypeII_error

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 1
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

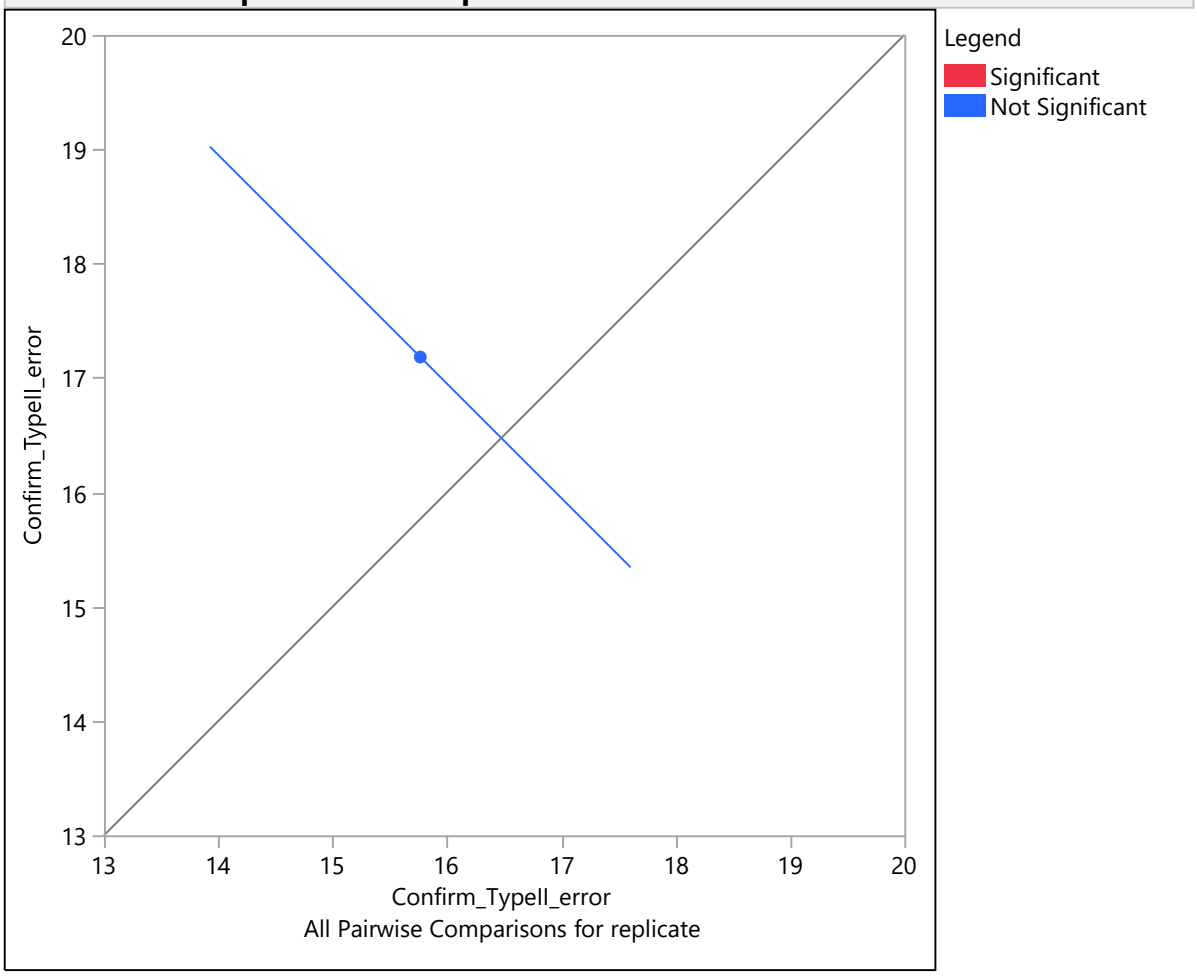
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-1.41442	1.871749	-0.76	0.4502	-5.09232	2.263470

Response Confirm_TypeII_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Detect_Typel_error

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	19.471654	1.4569024	477	16.608914	22.334394
1	17.742080	1.4569024	477	14.879340	20.604820

Tukey HSD All Pairwise Comparisons

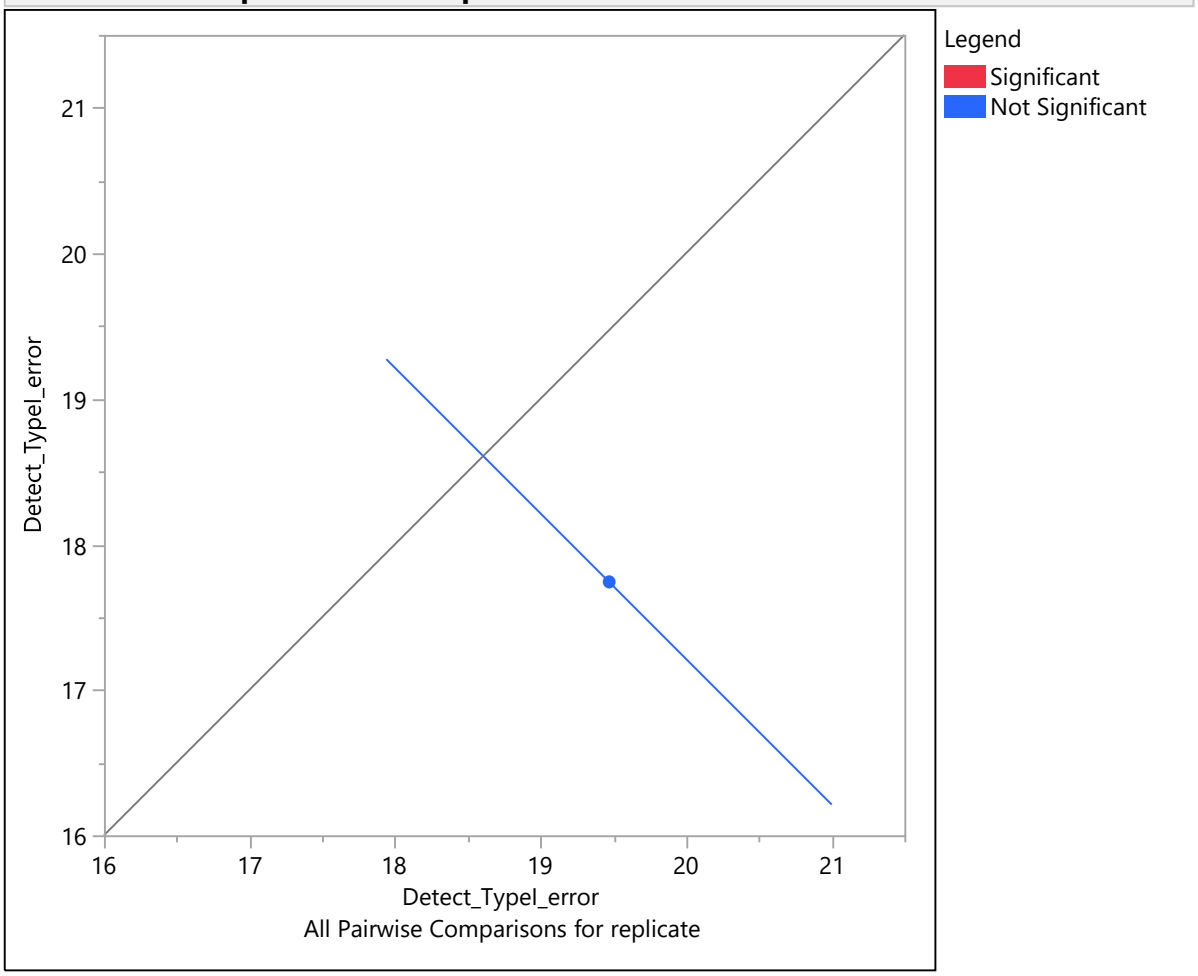
corp level = -1

Quantile = 1.96495 , Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.729574	1.557494	1.11	0.2673	-1.33083	4.789973

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

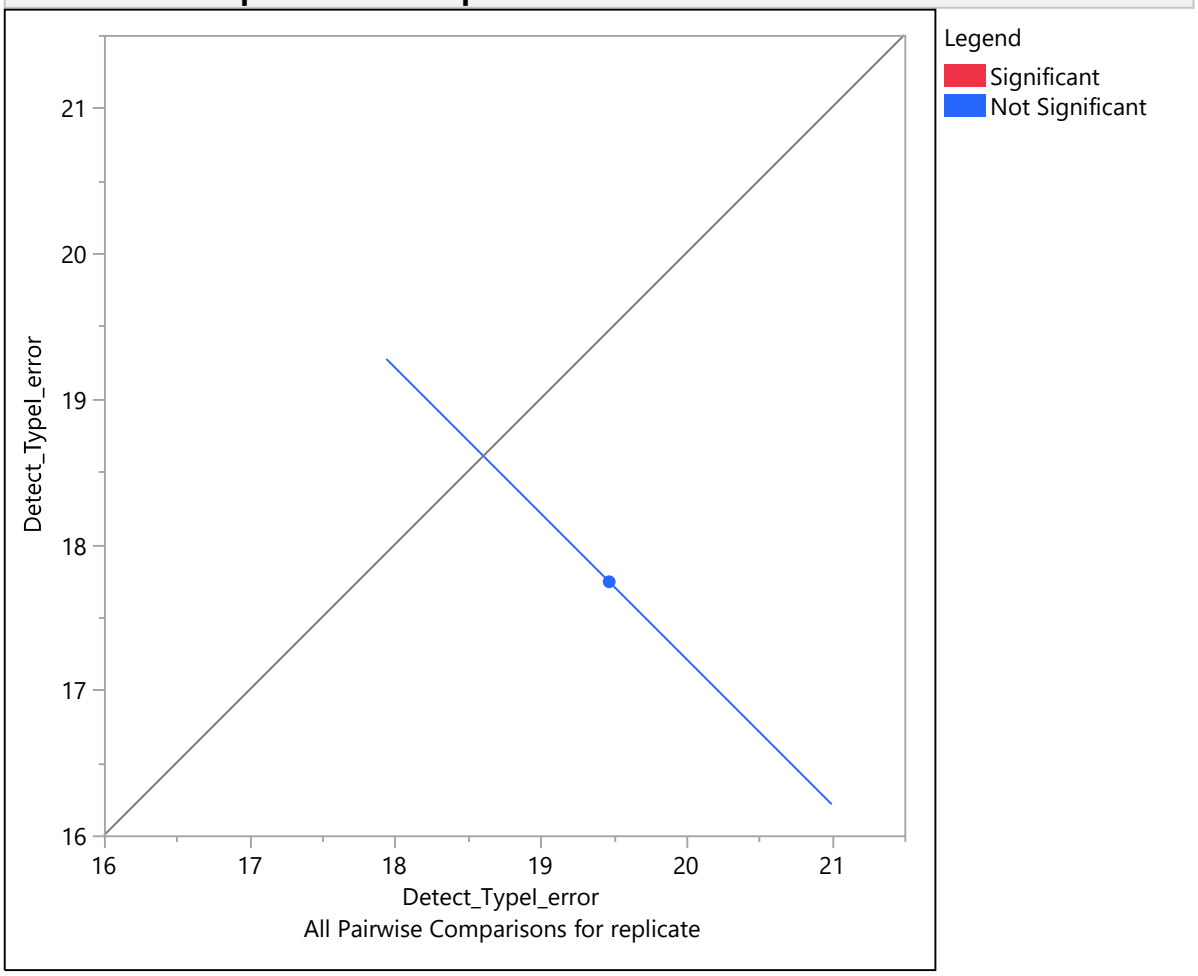
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.729574	1.557494	1.11	0.2673	-1.33082	4.789971

Response Detect_Typel_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	19.291254	1.1013147	477	17.127227	21.455282
1	17.561681	1.1013147	477	15.397653	19.725709

Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey

All Pairwise Differences

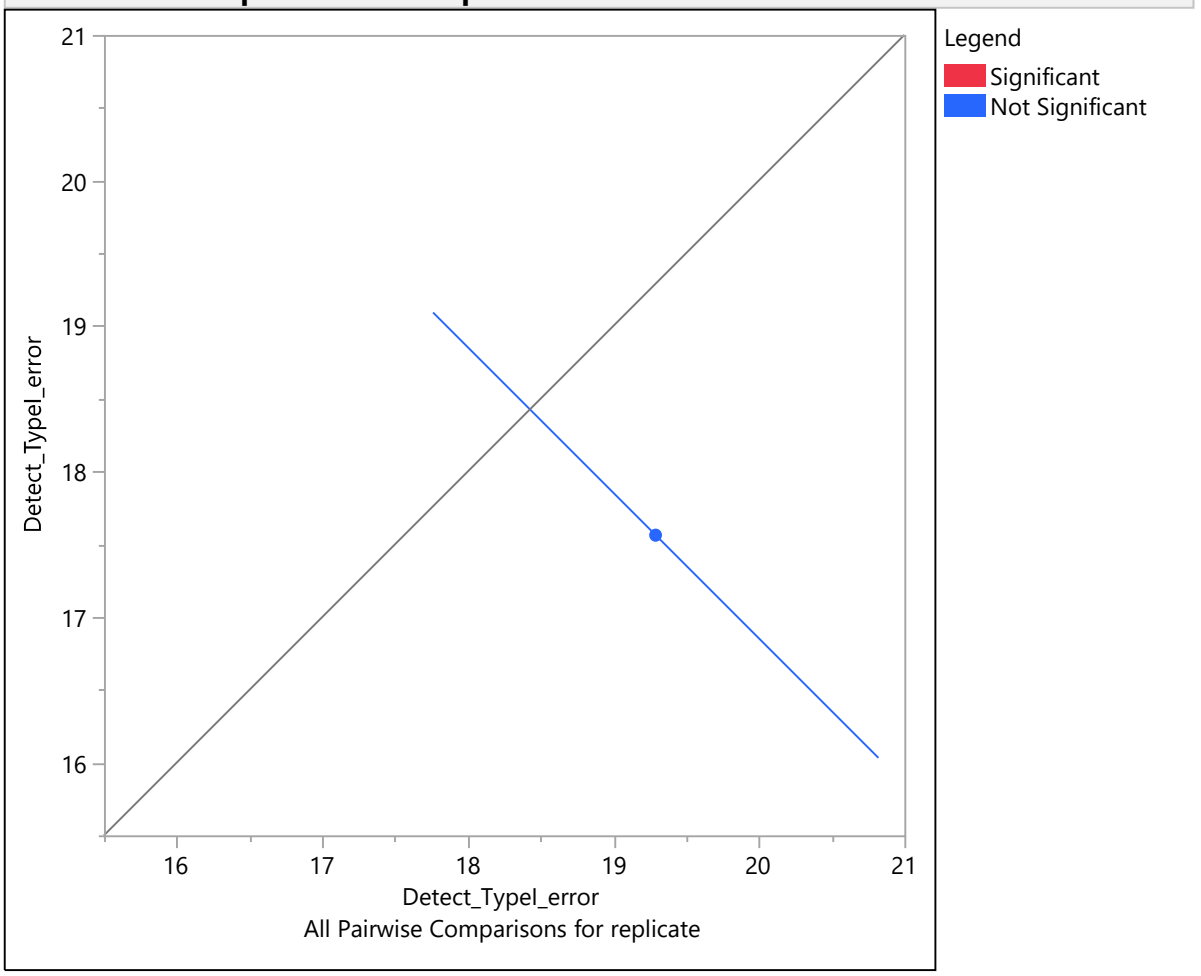
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.729574	1.557494	1.11	0.2673	-1.33083	4.789973

Response Detect_Typel_error

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 0
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

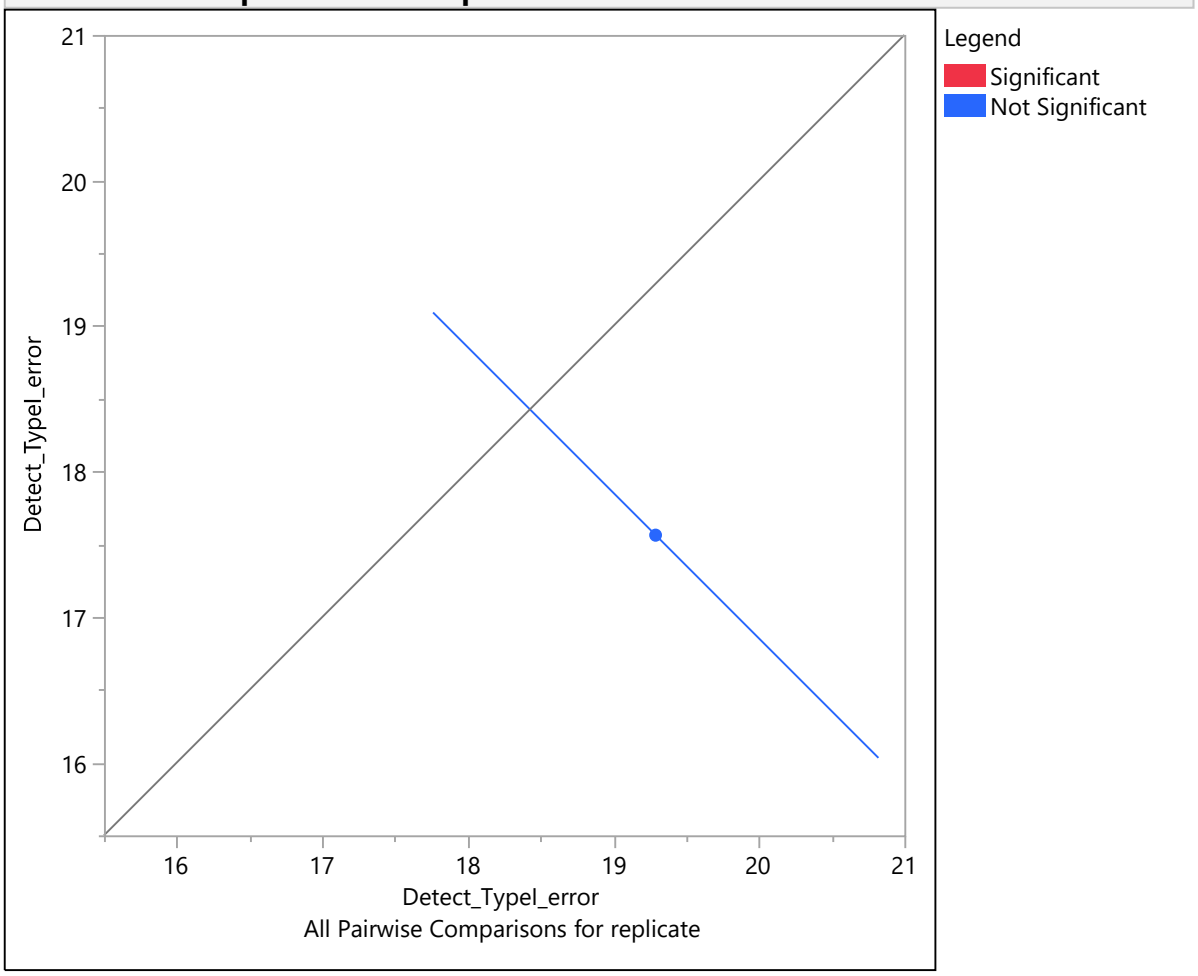
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.729574	1.557494	1.11	0.2673	-1.33082	4.789971

Response Detect_Typel_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	19.110855	1.4569024	477	16.248115	21.973595
1	17.381281	1.4569024	477	14.518541	20.244021

Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey-Kramer

All Pairwise Differences

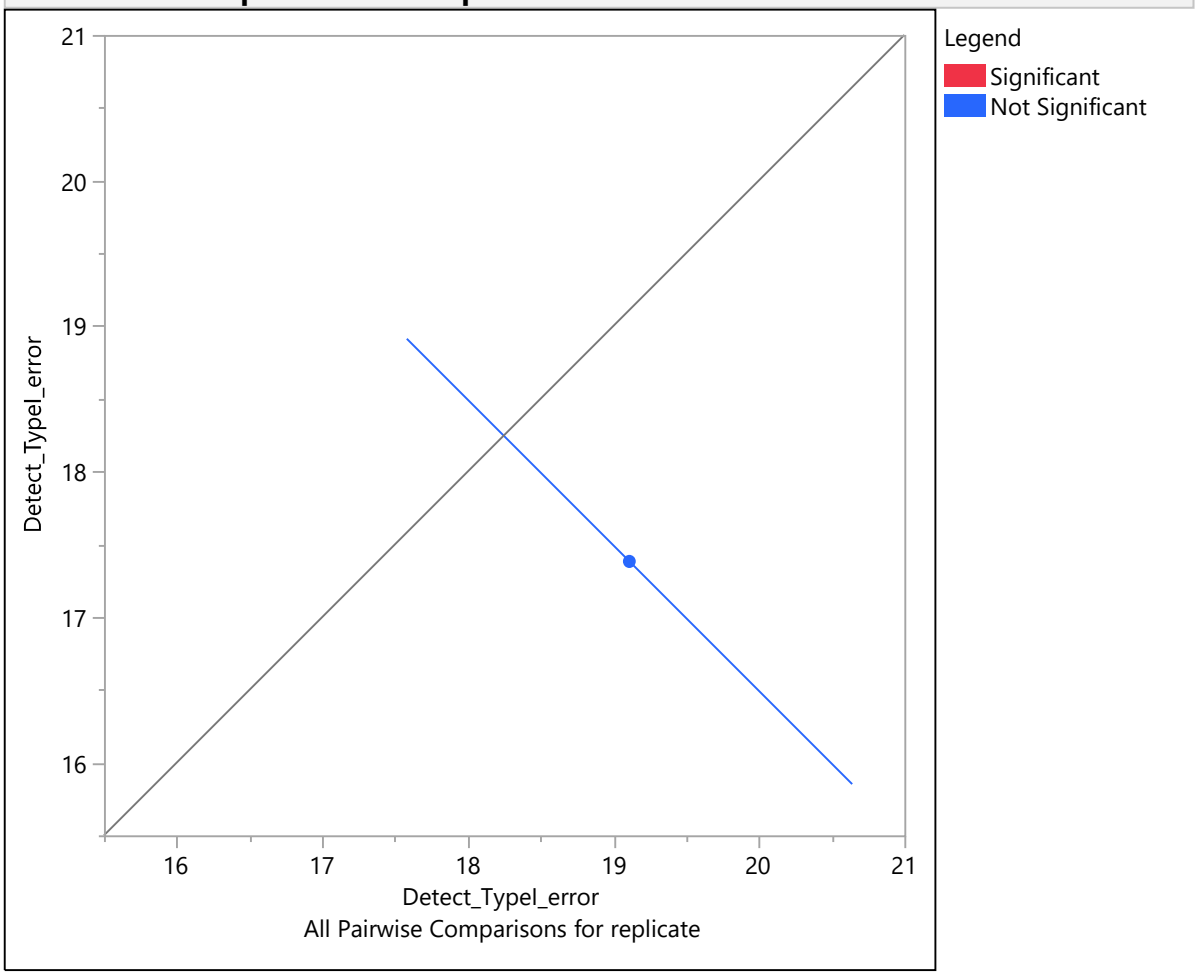
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.729574	1.557494	1.11	0.2673	-1.33083	4.789973

Response Detect_Typel_error

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 1
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

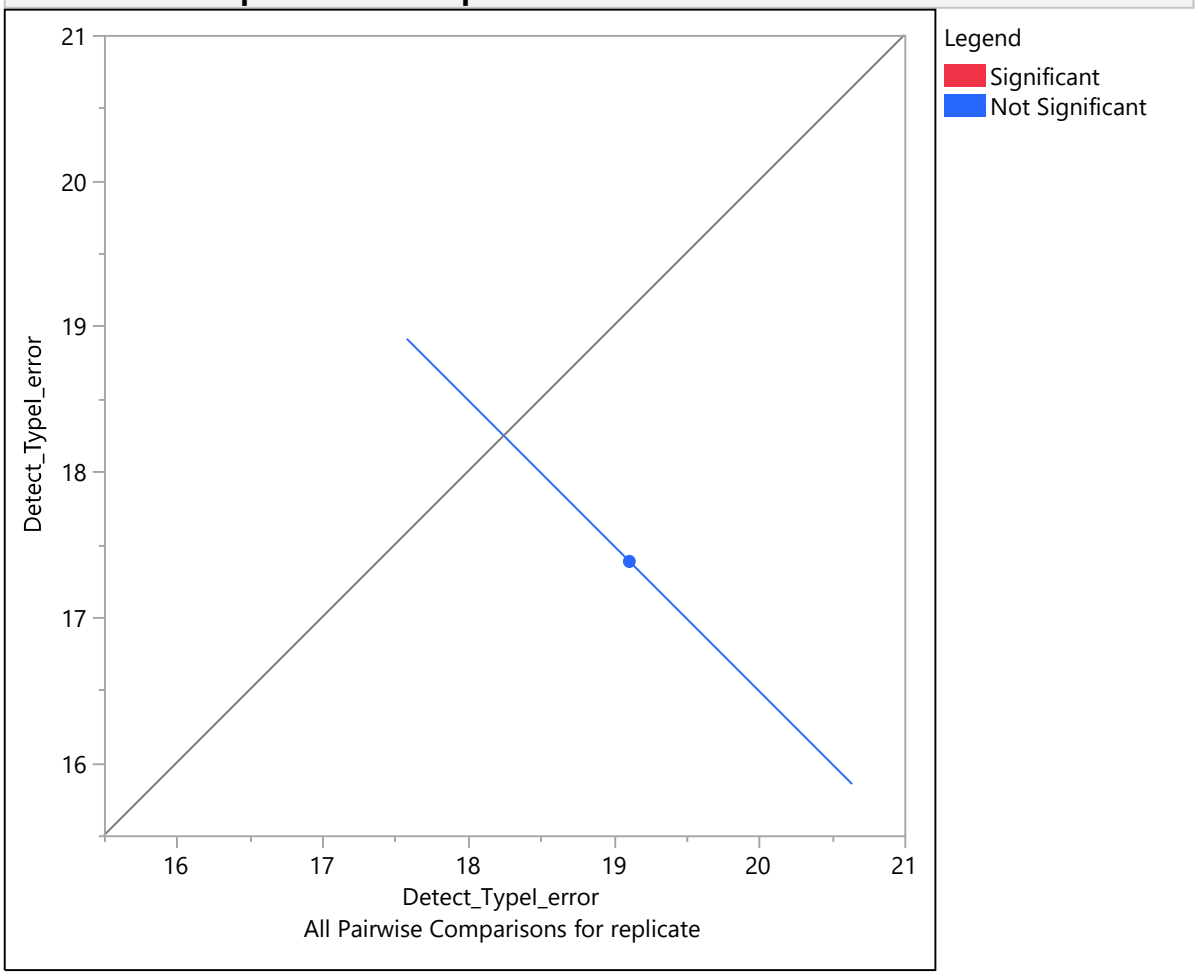
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.729574	1.557494	1.11	0.2673	-1.33082	4.789971

Response Detect_Typel_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Confirm_Typel_error

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	8.9605680	1.3066456	477	6.3930752	11.528061
1	7.6263514	1.3066456	477	5.0588586	10.193844

Tukey HSD All Pairwise Comparisons

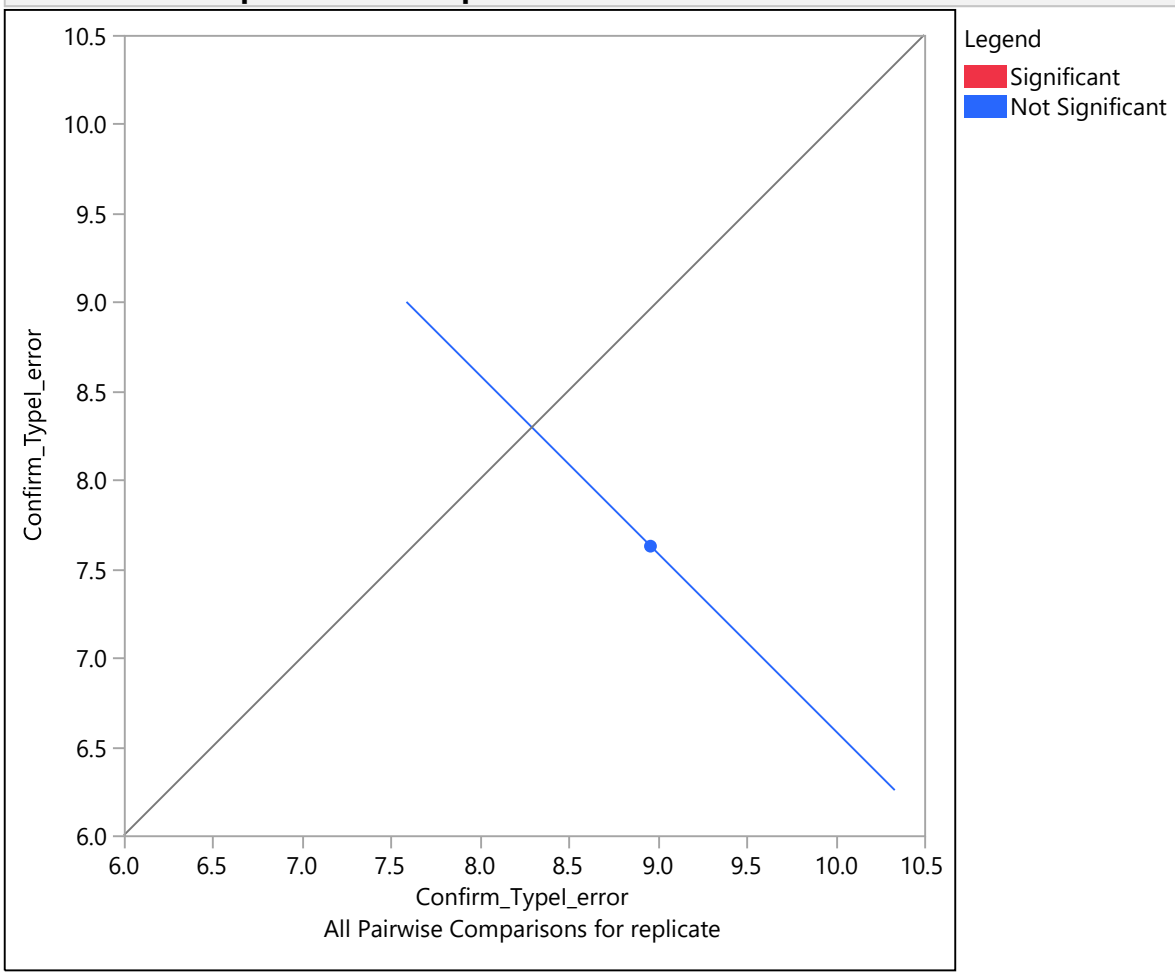
corp level = -1

Quantile = 1.96495 , Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.334217	1.396863	0.96	0.3400	-1.41055	4.078983

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

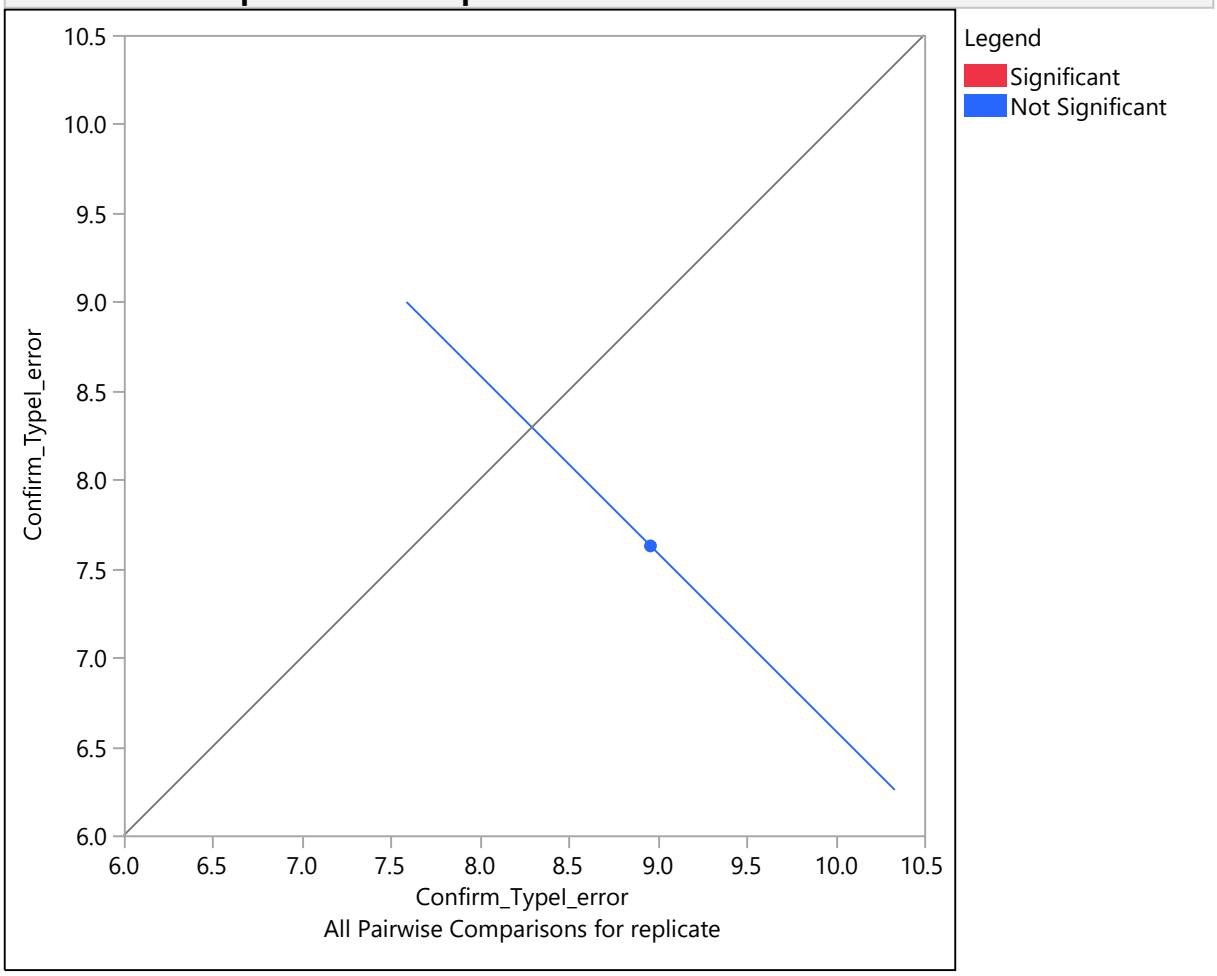
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.334217	1.396863	0.96	0.3400	-1.41055	4.078982

Response Confirm_Typel_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	9.6864100	0.98773121	477	7.7455678	11.627252
1	8.3521934	0.98773121	477	6.4113512	10.293036

Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey

All Pairwise Differences

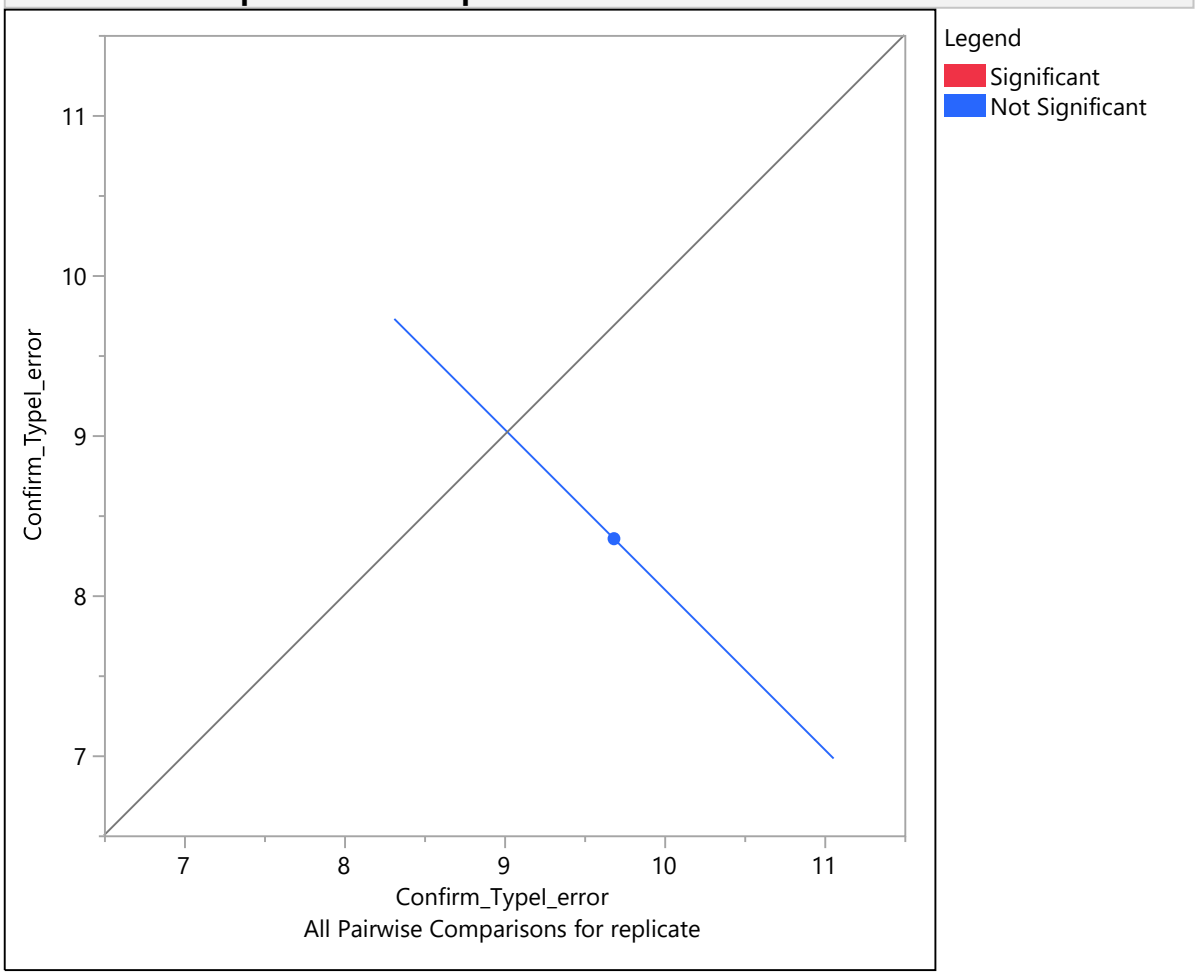
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.334217	1.396863	0.96	0.3400	-1.41055	4.078983

Response Confirm_Typel_error

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 0
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

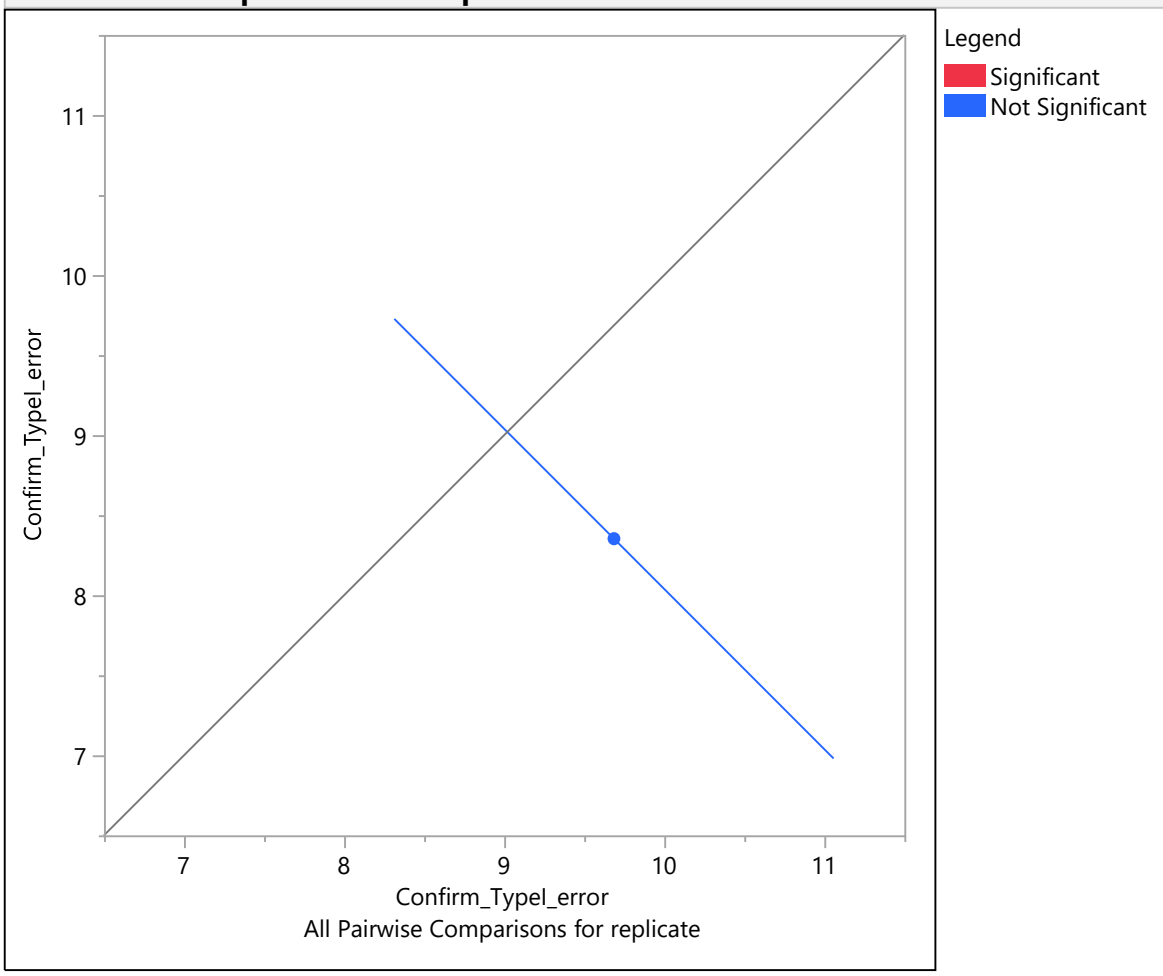
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.334217	1.396863	0.96	0.3400	-1.41055	4.078982

Response Confirm_TypeI_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	10.412252	1.3066456	477	7.8447591	12.979745
1	9.078035	1.3066456	477	6.5105425	11.645528

Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey-Kramer

All Pairwise Differences

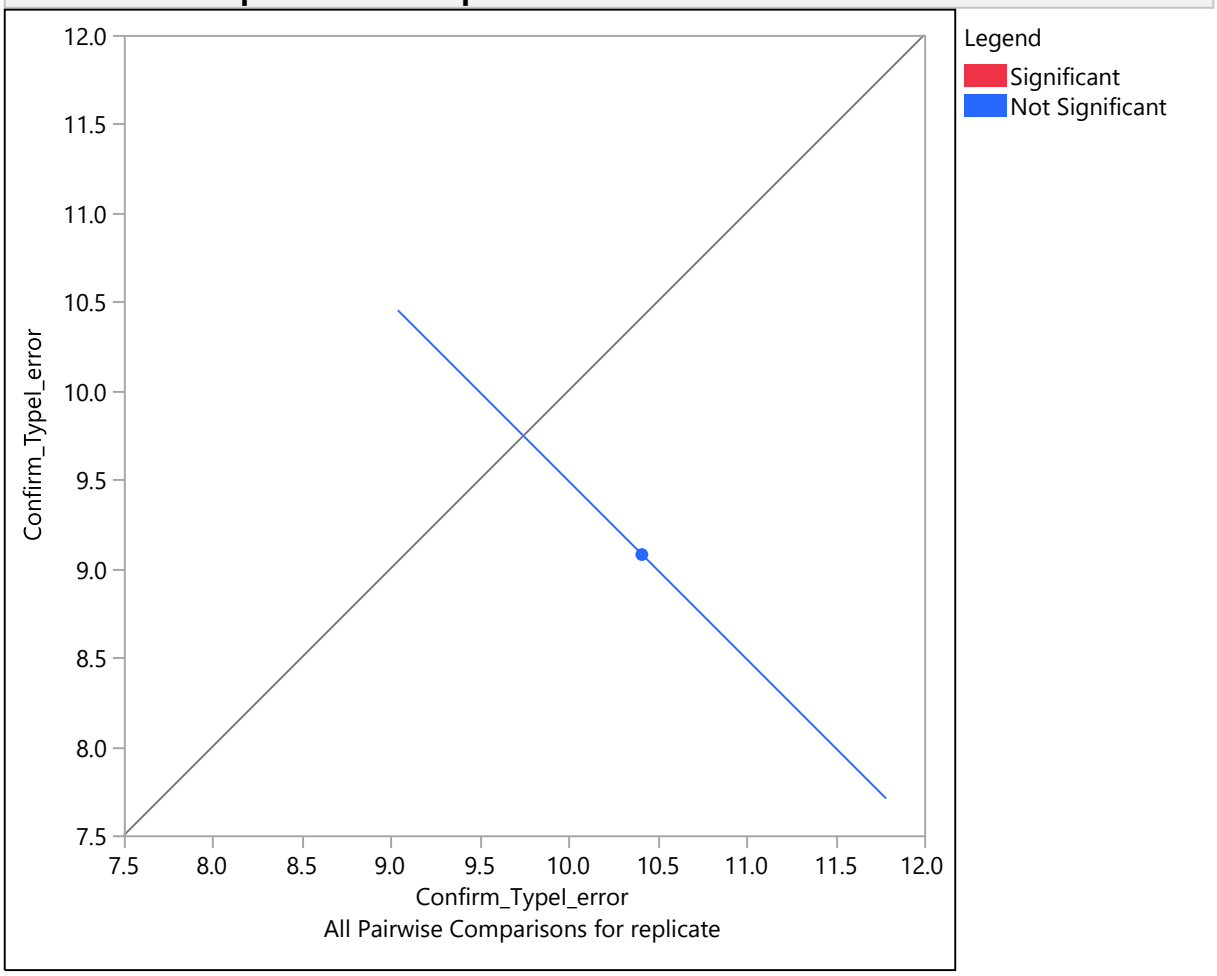
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.334217	1.396863	0.96	0.3400	-1.41055	4.078983

Response Confirm_Typel_error

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 1
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

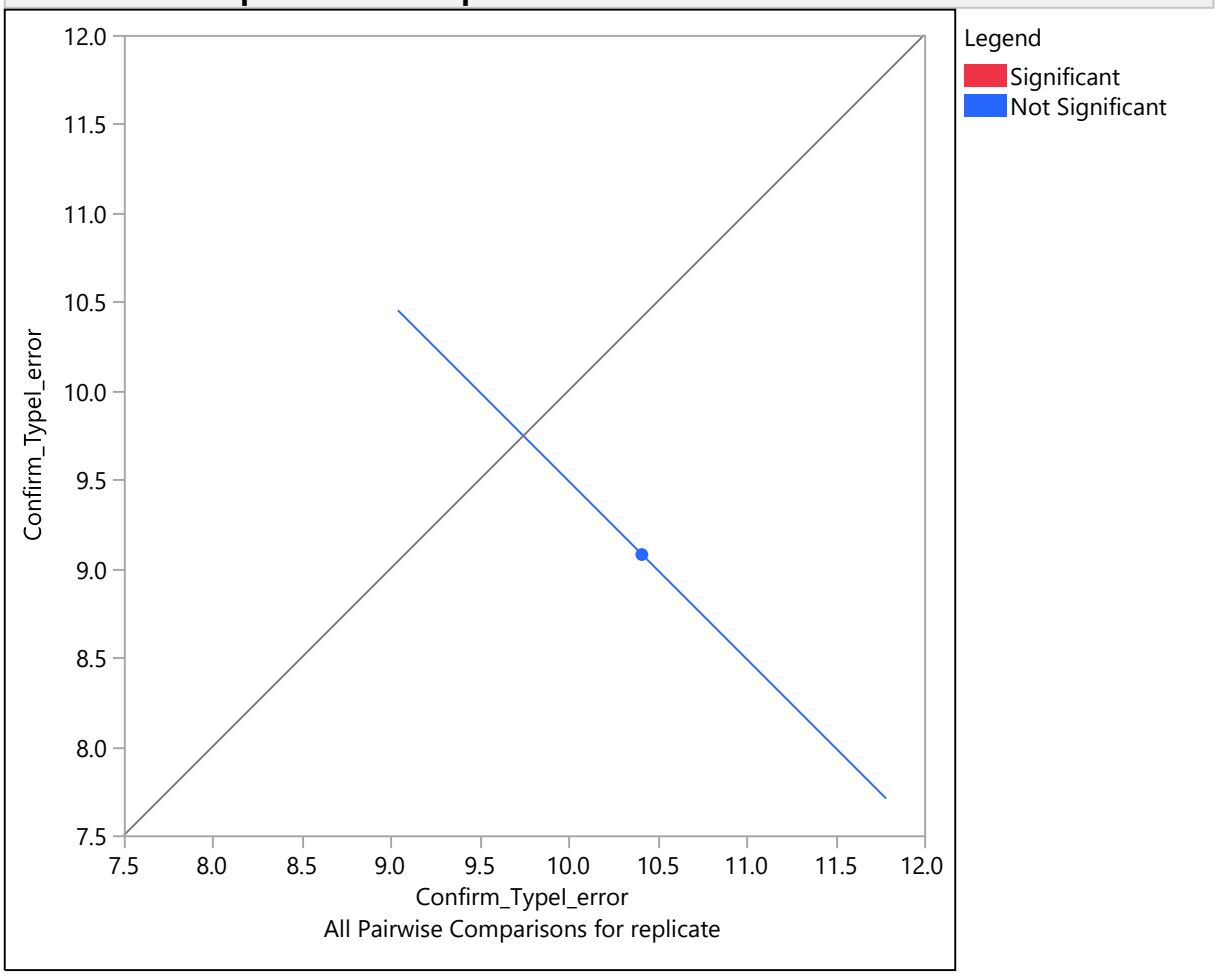
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	1.334217	1.396863	0.96	0.3400	-1.41055	4.078982

Response Confirm_TypeI_error

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Mission_Time

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	12.029091	0.36760985	477	11.306756	12.751426
1	12.014591	0.36760985	477	11.292256	12.736926

Tukey HSD All Pairwise Comparisons

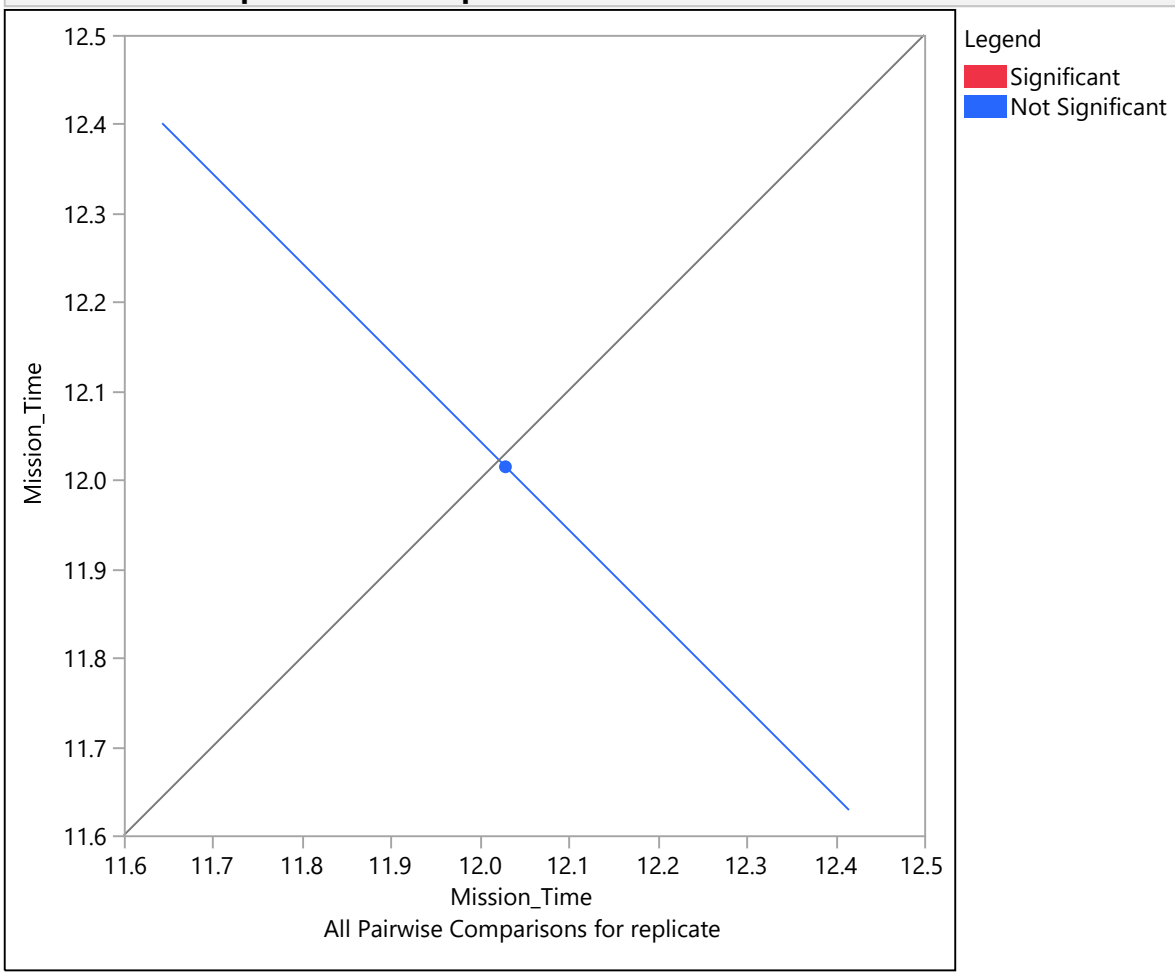
corp level = -1

Quantile = 1.96495 , Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.0145000	0.3929915	0.04	0.9706	-0.757709	0.7867088

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

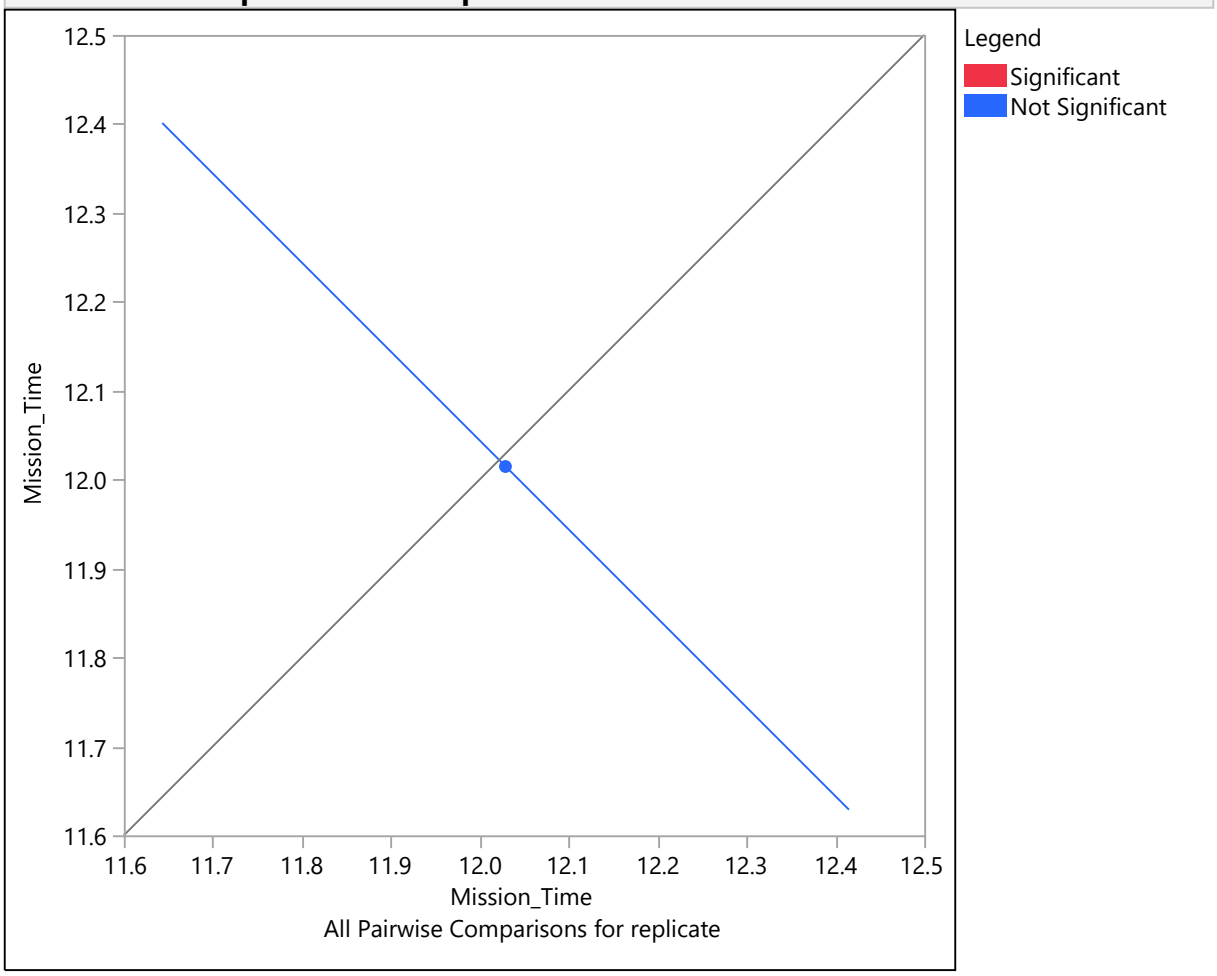
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.0145000	0.3929915	0.04	0.9706	-0.757708	0.7867085

Response Mission_Time

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	11.026708	0.27788693	477	10.480674	11.572742
1	11.012208	0.27788693	477	10.466174	11.558242

Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey

All Pairwise Differences

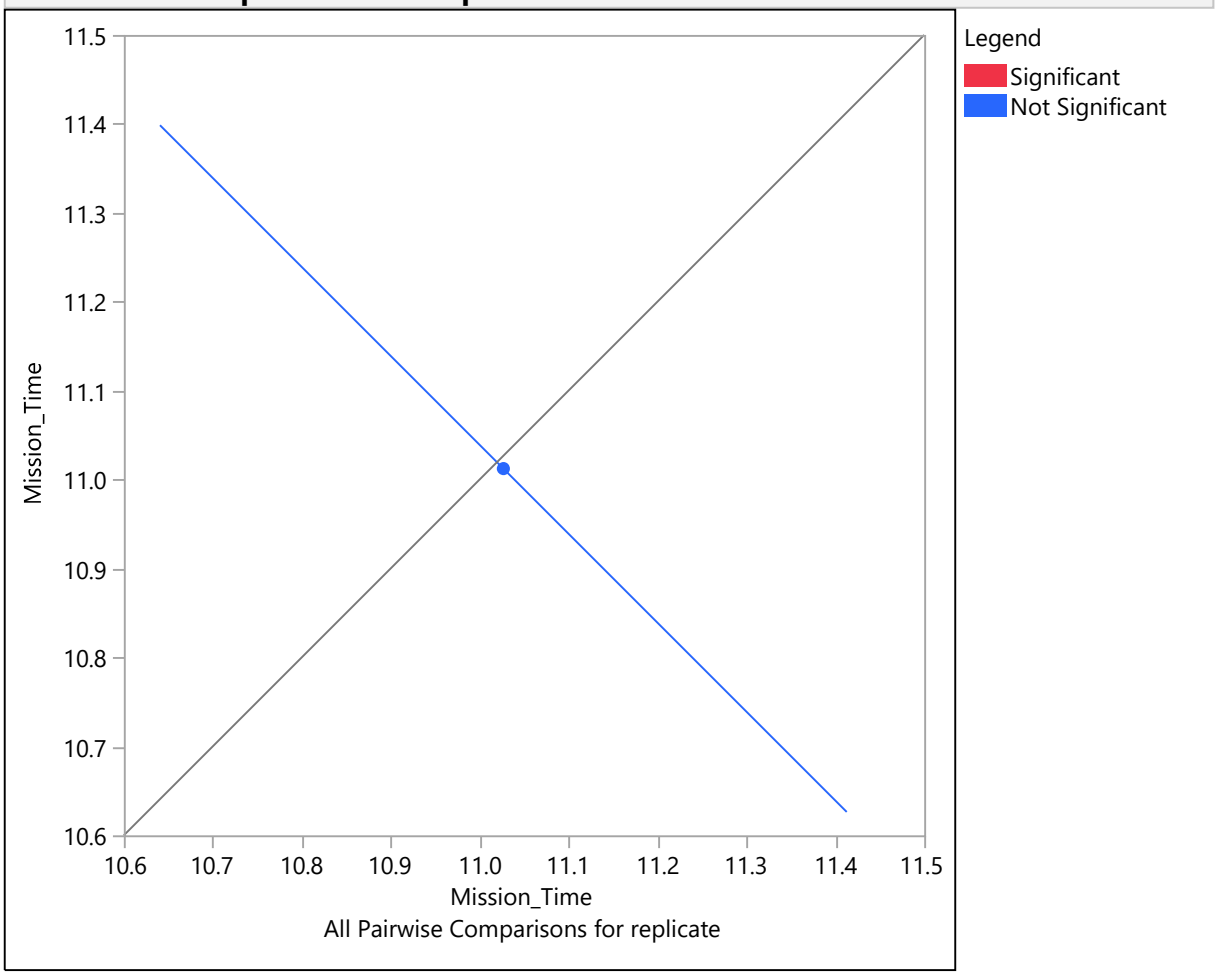
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.0145000	0.3929915	0.04	0.9706	-0.757709	0.7867088

Response Mission_Time

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 0
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

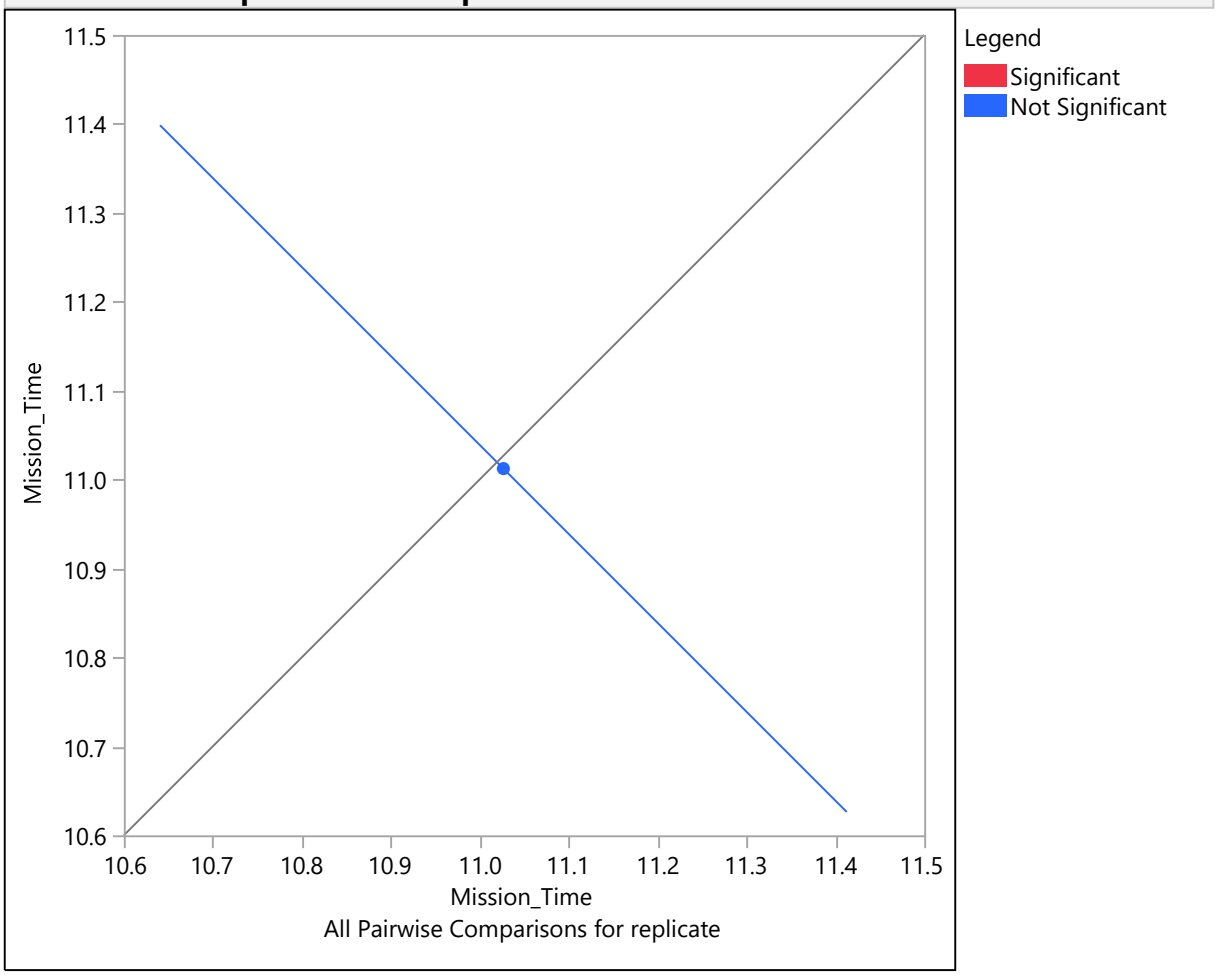
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.0145000	0.3929915	0.04	0.9706	-0.757708	0.7867085

Response Mission_Time

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	10.024326	0.36760985	477	9.3019906	10.746660
1	10.009826	0.36760985	477	9.2874906	10.732160

Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey-Kramer

All Pairwise Differences

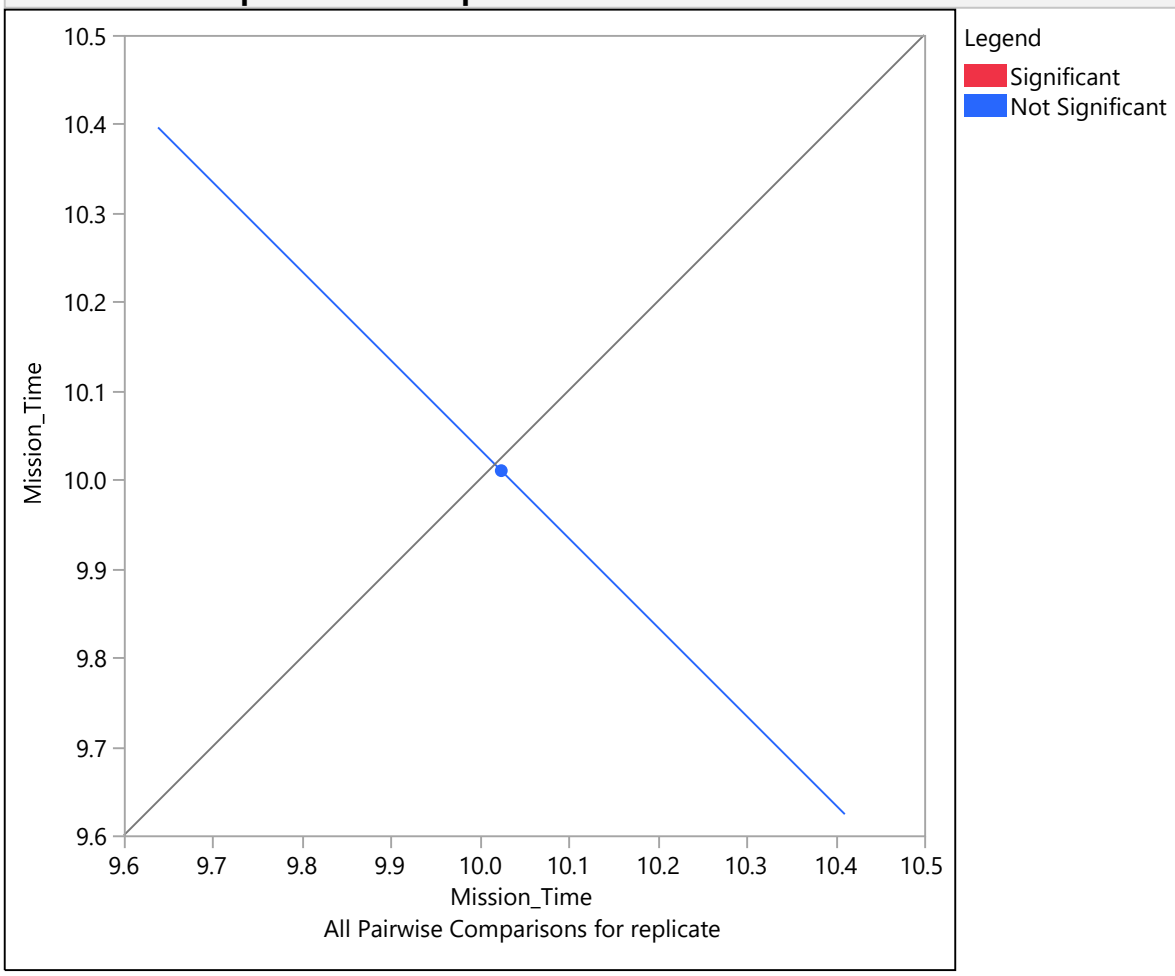
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.0145000	0.3929915	0.04	0.9706	-0.757709	0.7867088

Response Mission_Time

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 1
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

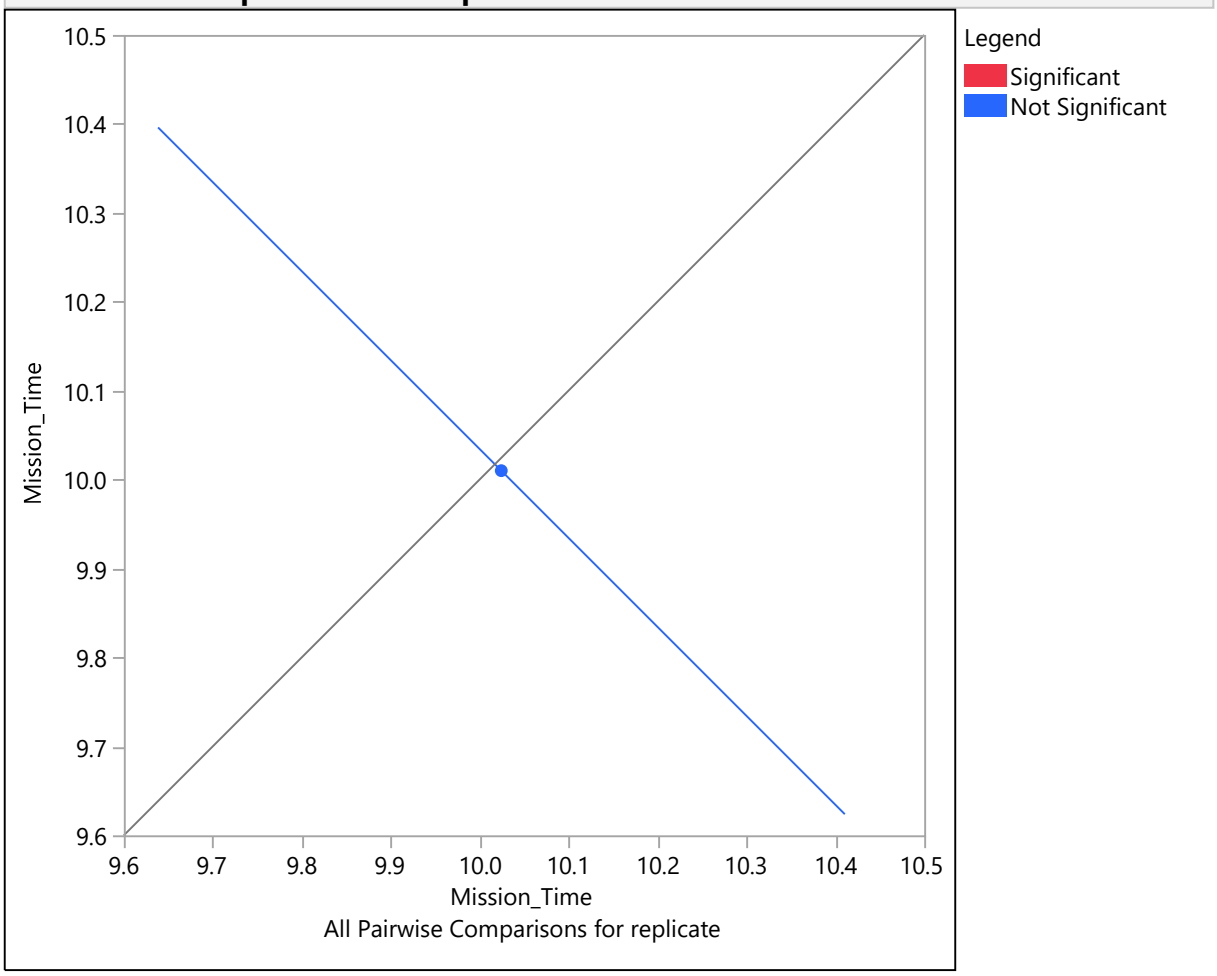
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.0145000	0.3929915	0.04	0.9706	-0.757708	0.7867085

Response Mission_Time

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Percent Covered

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	104.03622	0.46686824	477	103.11885	104.95359
1	104.07716	0.46686824	477	103.15979	104.99453

Tukey HSD All Pairwise Comparisons

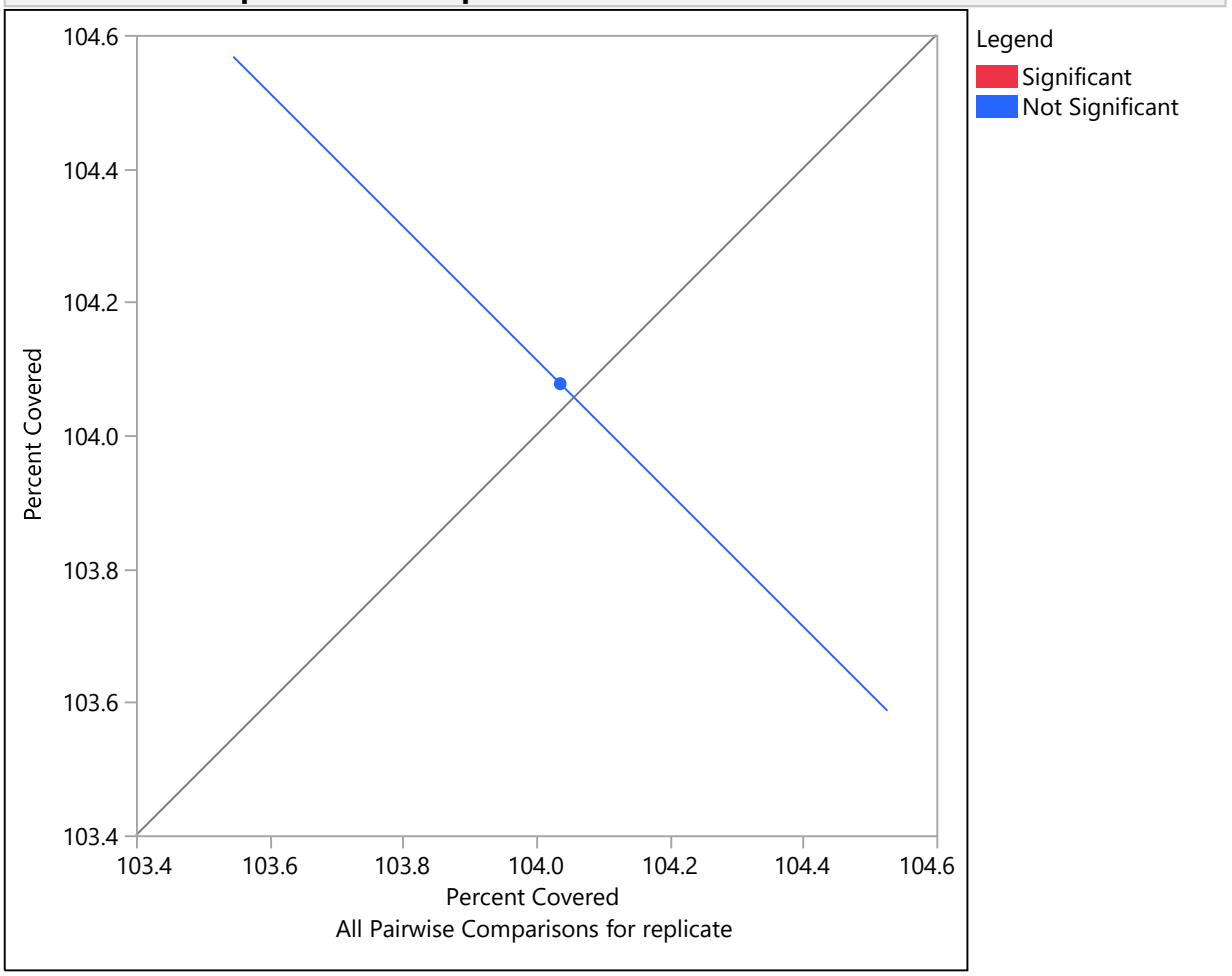
corp level = -1

Quantile = 1.96495 , Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.040939	0.4991031	-0.08	0.9347	-1.02165	0.9397740

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

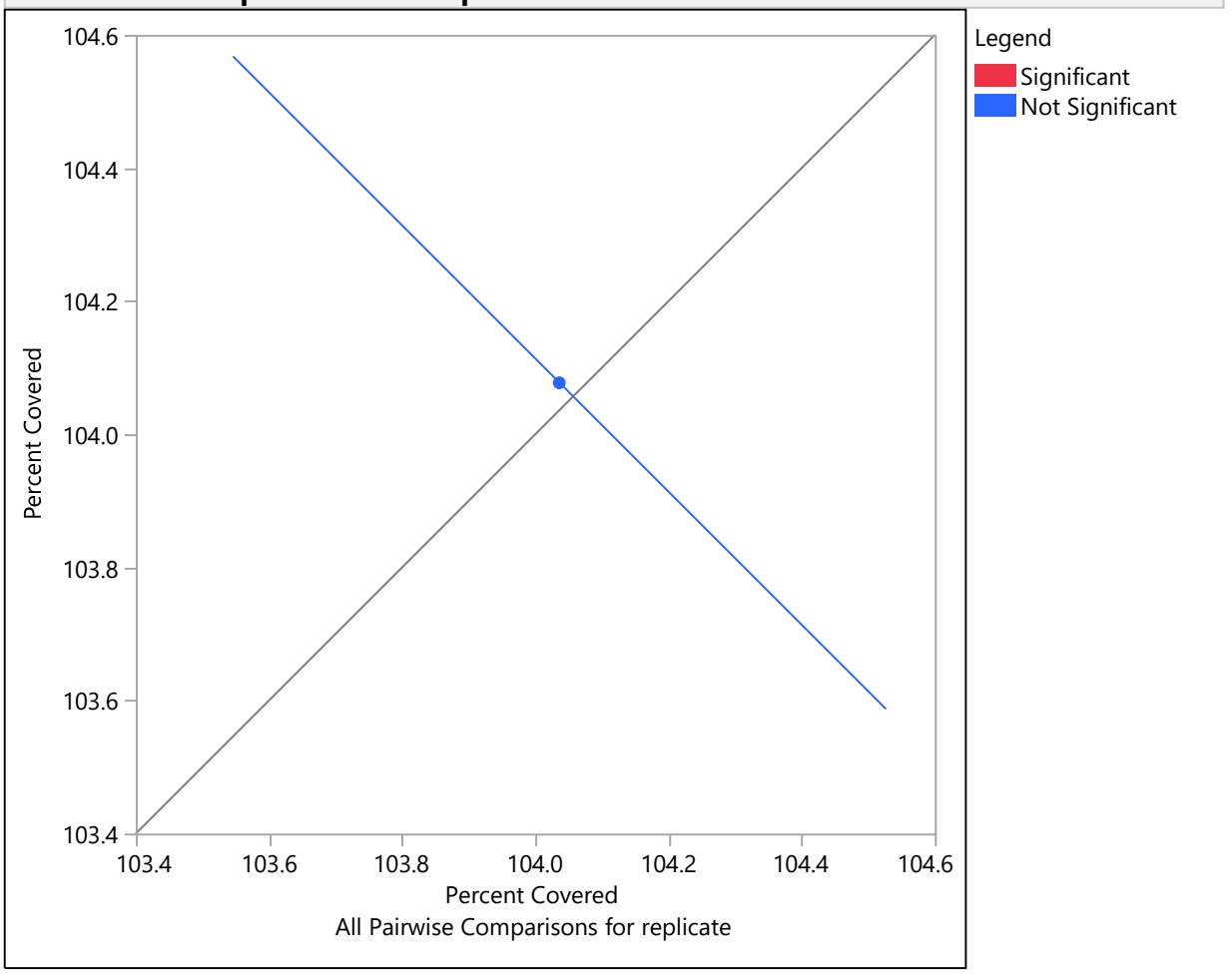
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.040939	0.4991031	-0.08	0.9347	-1.02165	0.9397735

Response Percent Covered

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	104.05928	0.35291922	477	103.36581	104.75275
1	104.10022	0.35291922	477	103.40675	104.79369

Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96495, Adjusted DF = 477.0, Adjustment = Tukey

All Pairwise Differences

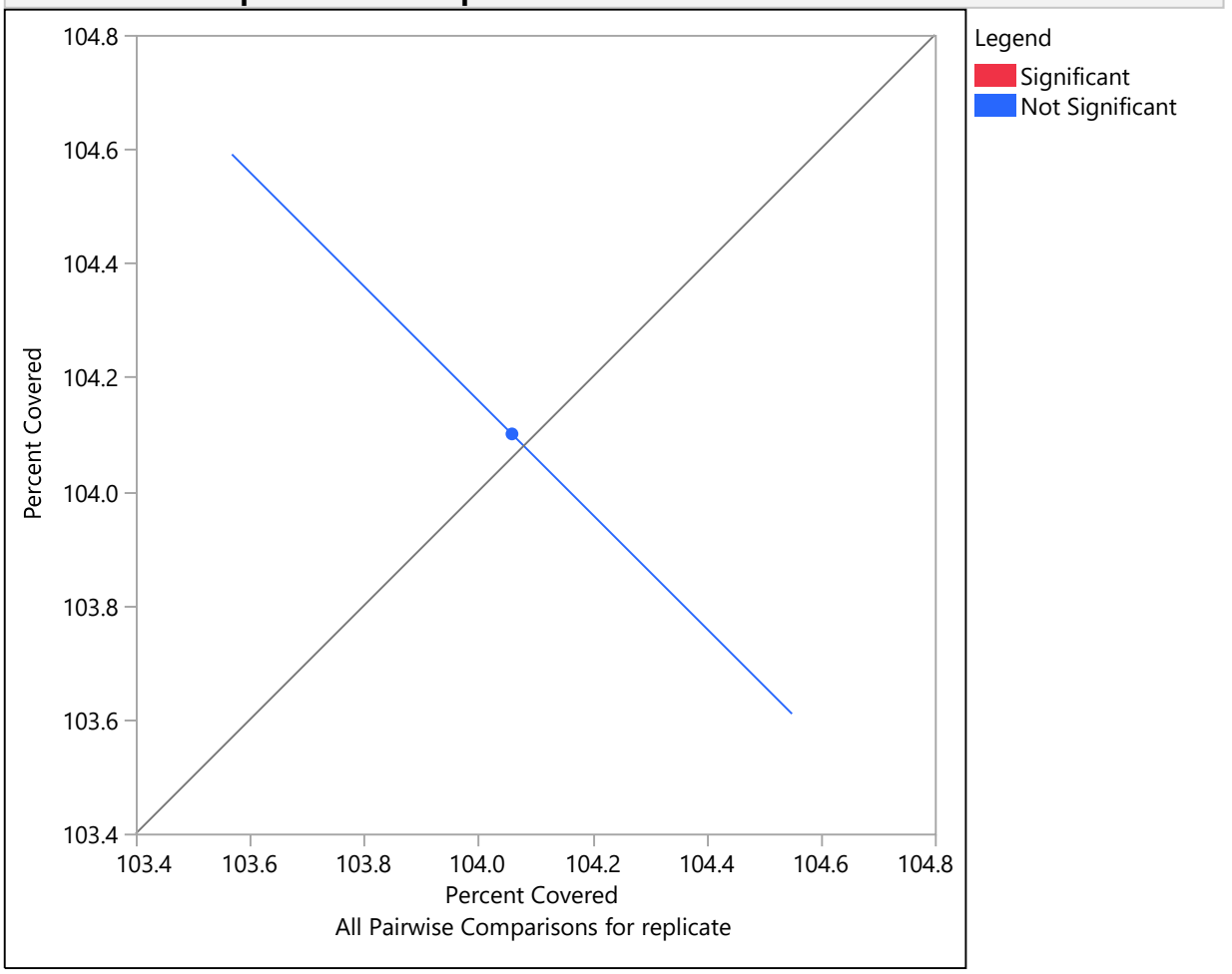
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.040939	0.4991031	-0.08	0.9347	-1.02165	0.9397740

Response Percent Covered

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 0
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

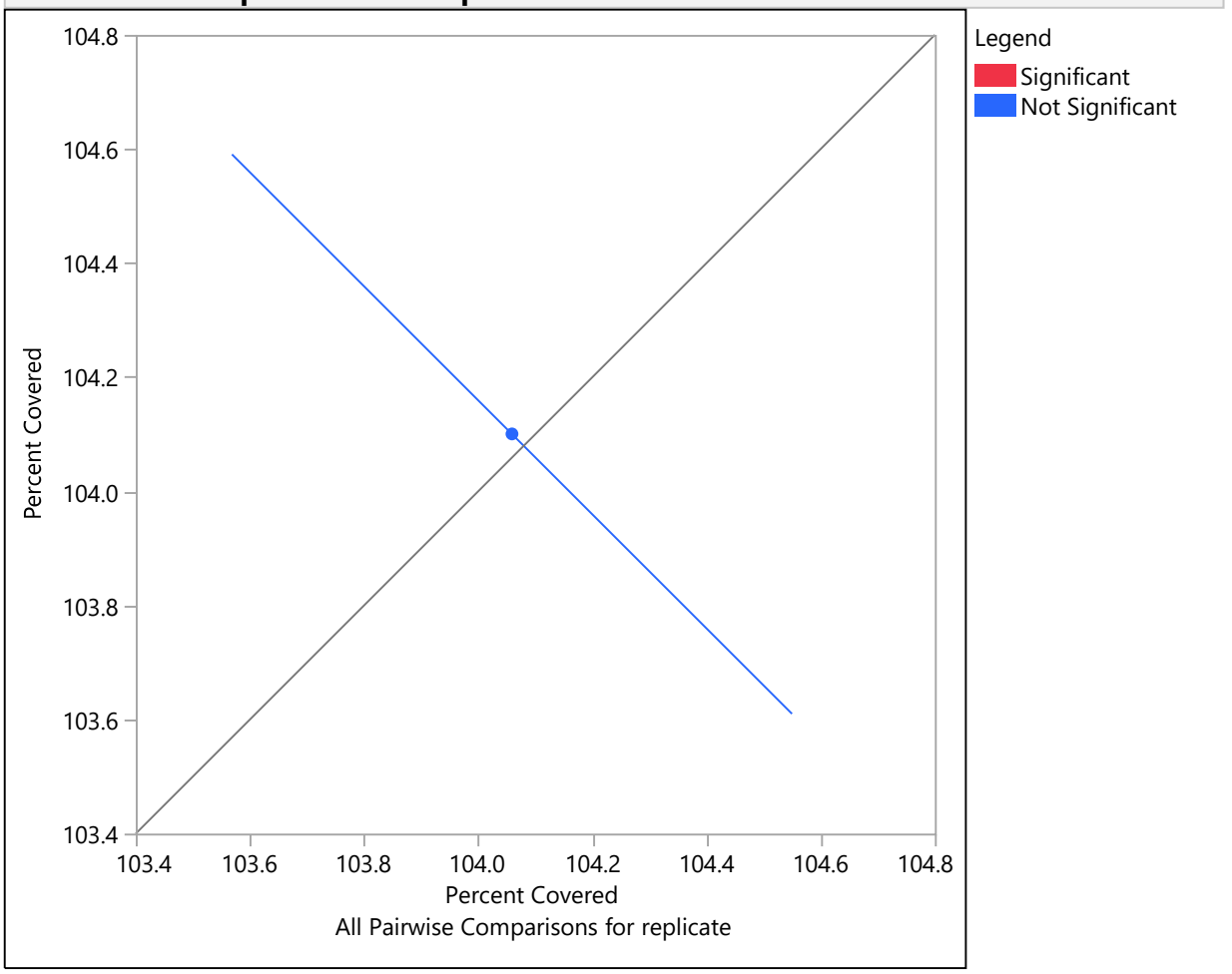
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.040939	0.4991031	-0.08	0.9347	-1.02165	0.9397735

Response Percent Covered

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	104.08234	0.46686824	477	103.16497	104.99971
1	104.12328	0.46686824	477	103.20591	105.04065

Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey-Kramer

All Pairwise Differences

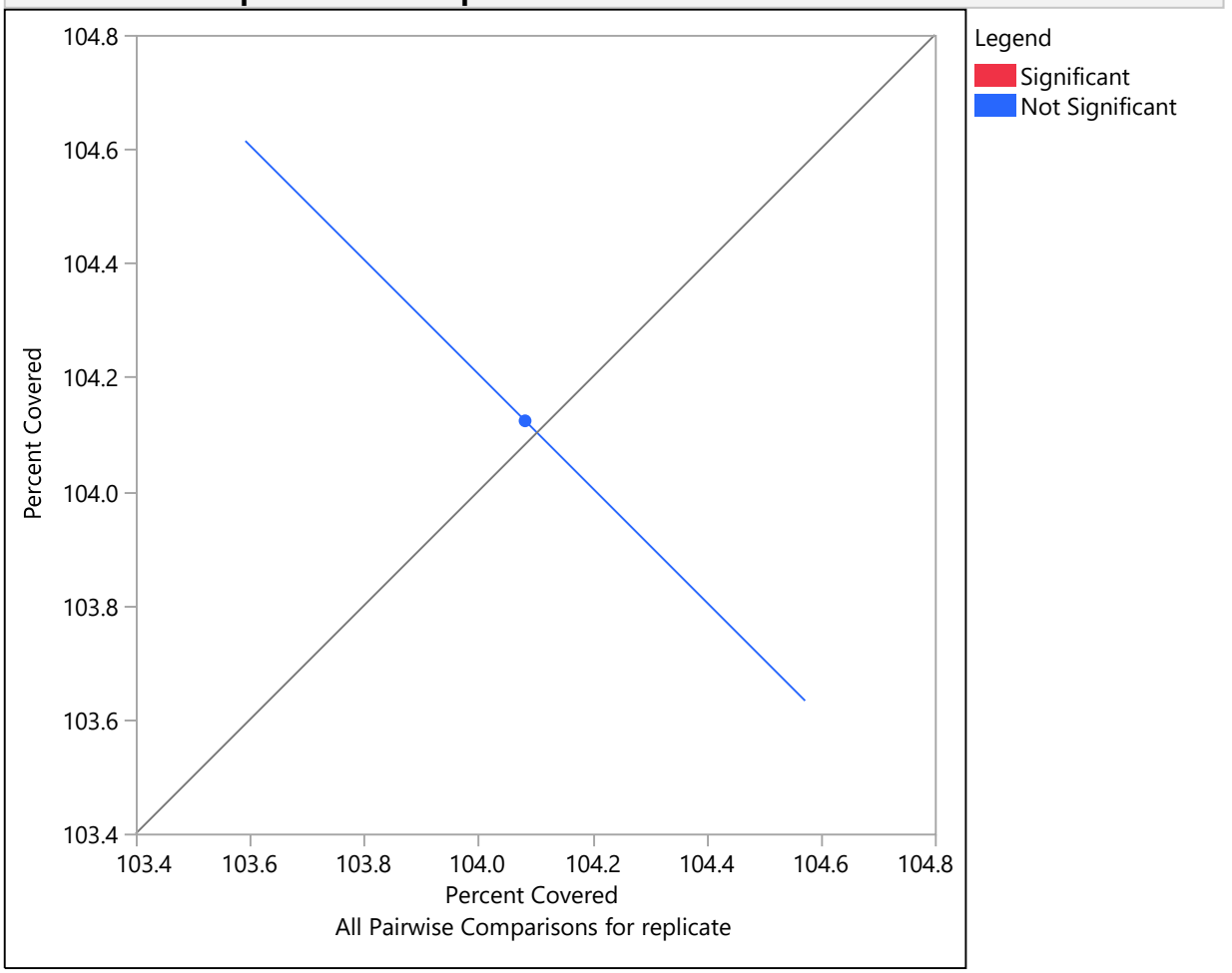
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.040939	0.4991031	-0.08	0.9347	-1.02165	0.9397740

Response Percent Covered

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 1
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

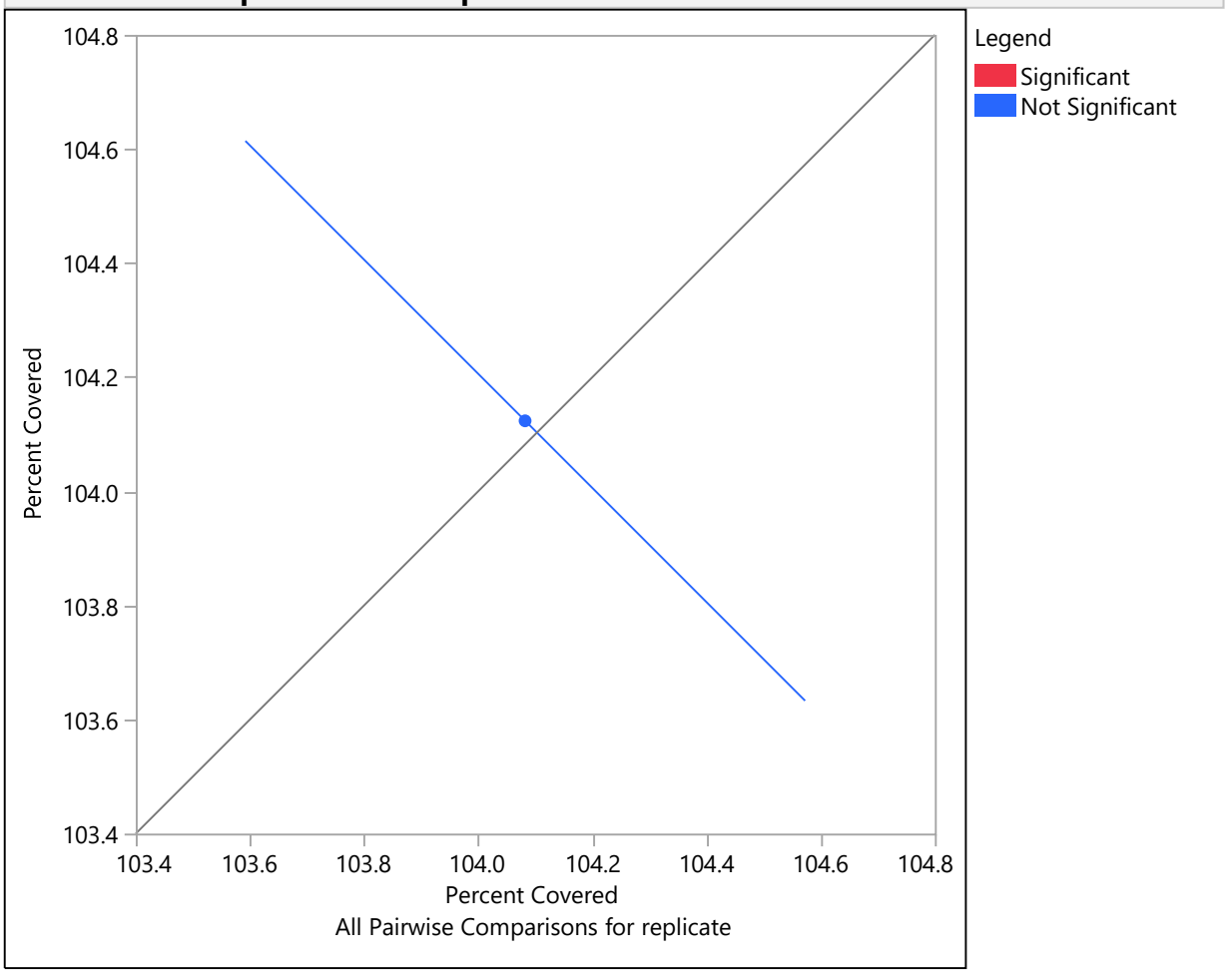
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-0.040939	0.4991031	-0.08	0.9347	-1.02165	0.9397735

Response Percent Covered

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Percent_Correct_Confirmations

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	43.582722	2.7641026	477	38.151399	49.014044
1	43.247257	2.7641026	477	37.815934	48.678579

Tukey HSD All Pairwise Comparisons

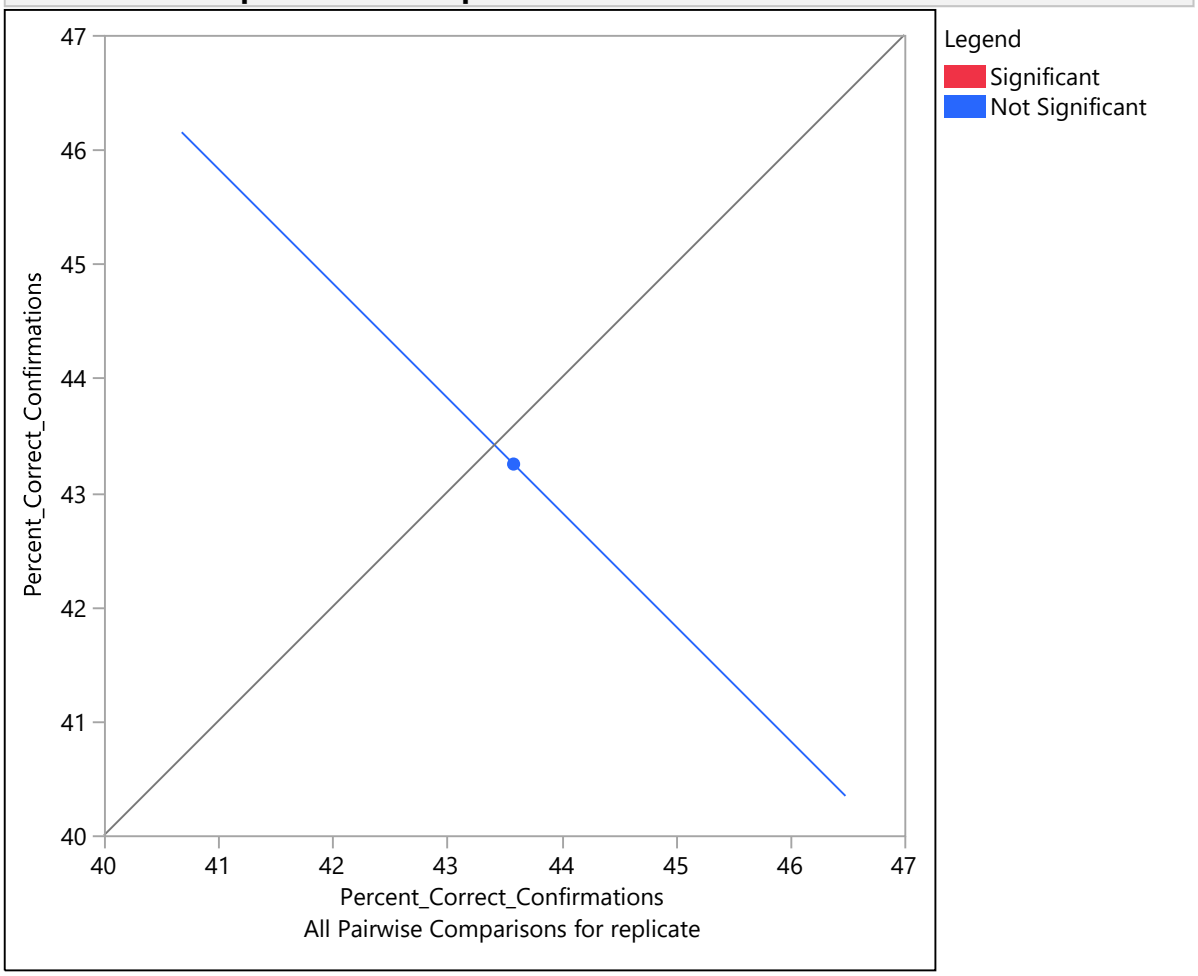
corp level = -1

Quantile = 1.96495 , Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.3354649	2.954950	0.11	0.9097	-5.47087	6.141796

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

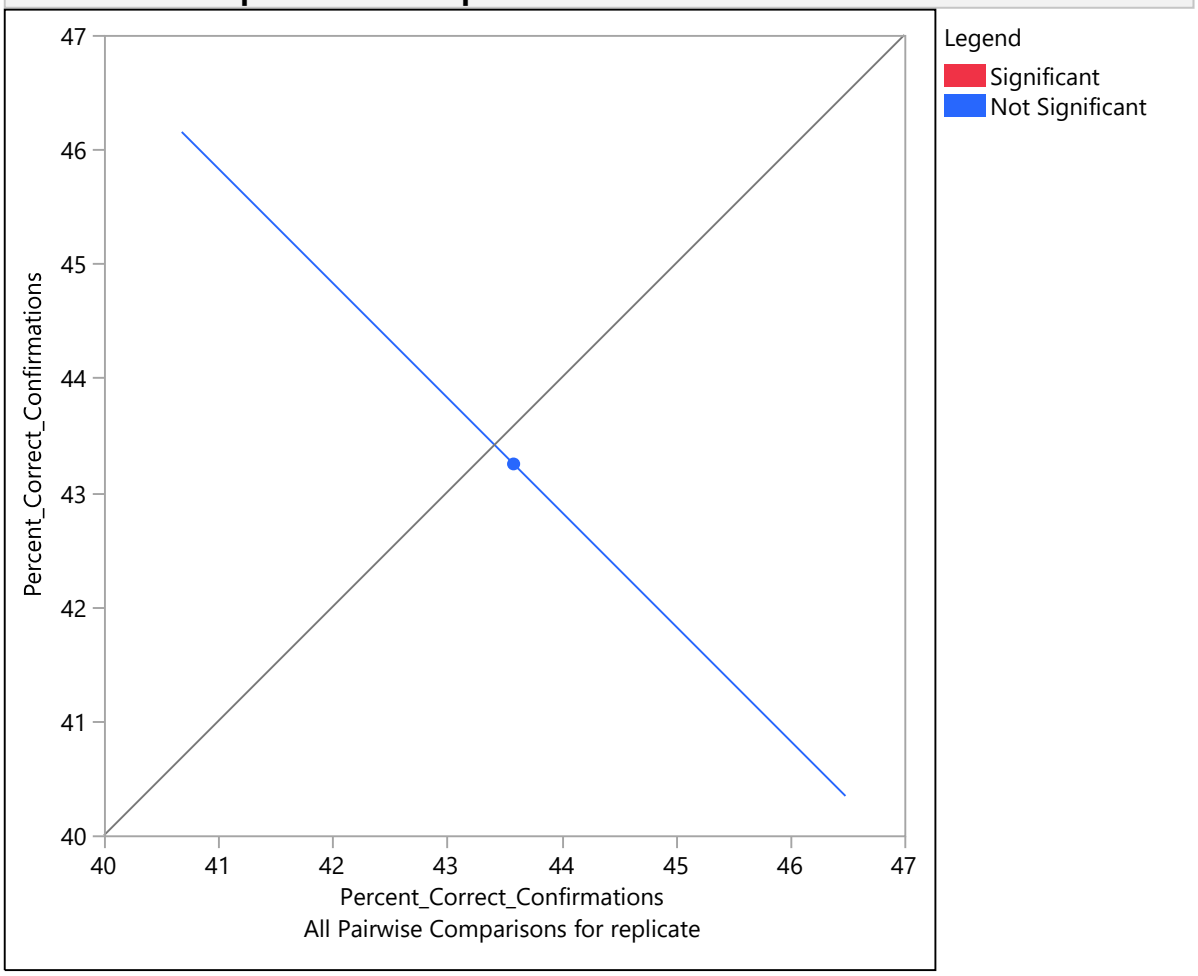
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.3354649	2.954950	0.11	0.9097	-5.47086	6.141793

Response Percent_Correct_Confirmations

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	43.230766	2.0894651	477	39.125072	47.336460
1	42.895301	2.0894651	477	38.789607	47.000995

Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey

All Pairwise Differences

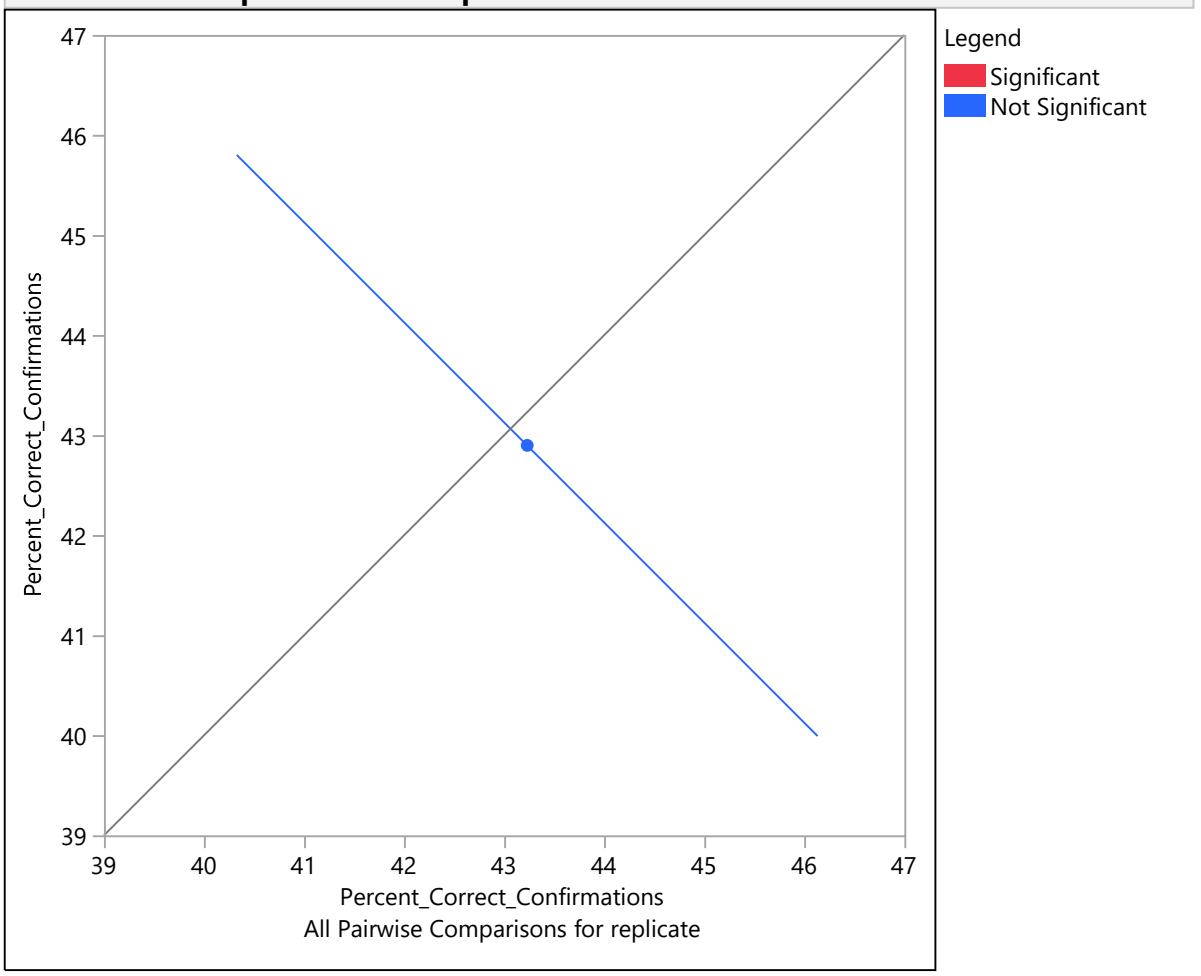
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.3354649	2.954950	0.11	0.9097	-5.47087	6.141796

Response Percent_Correct_Confirmations

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 0
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

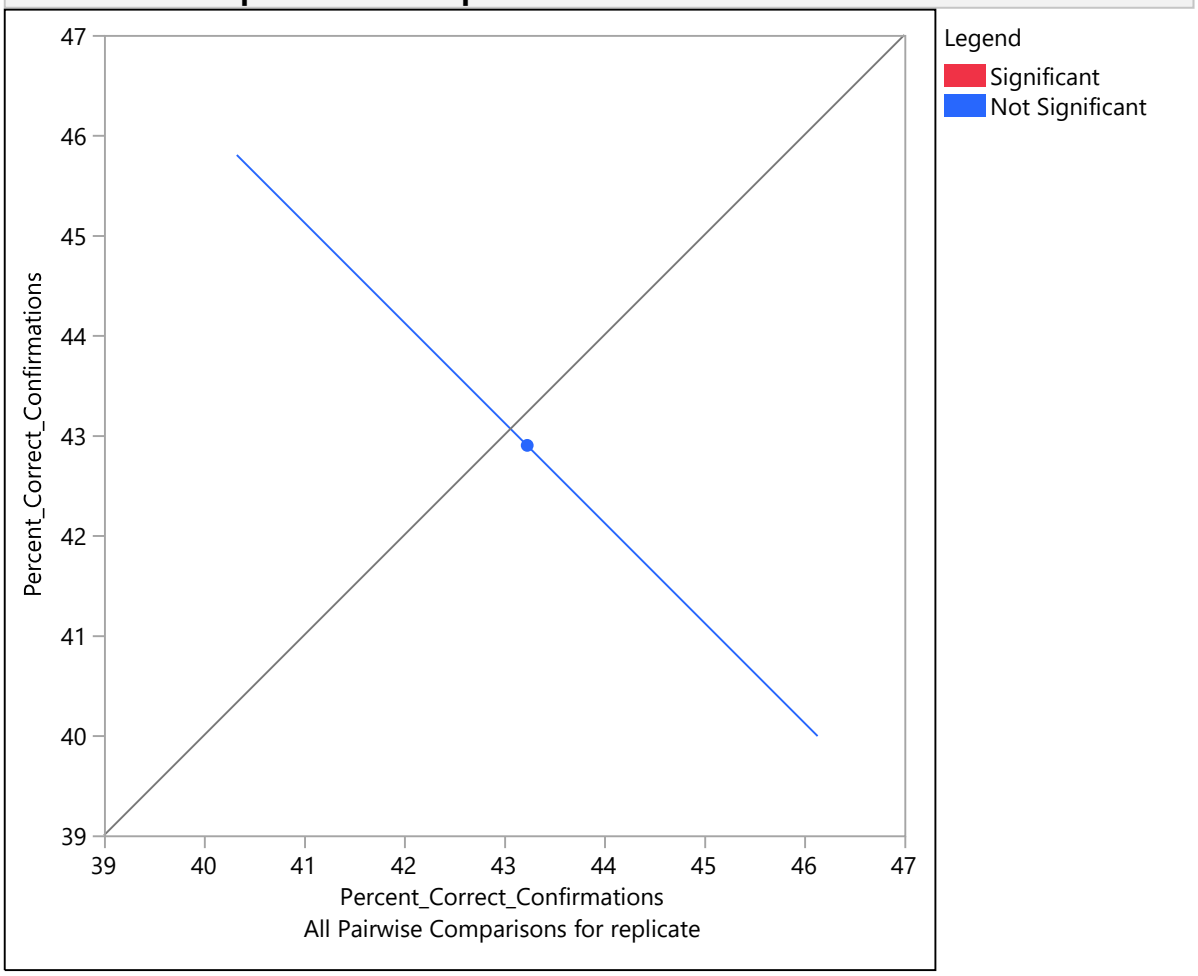
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.3354649	2.954950	0.11	0.9097	-5.47086	6.141793

Response Percent_Correct_Confirmations

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	42.878810	2.7641026	477	37.447488	48.310133
1	42.543345	2.7641026	477	37.112023	47.974668

Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey-Kramer

All Pairwise Differences

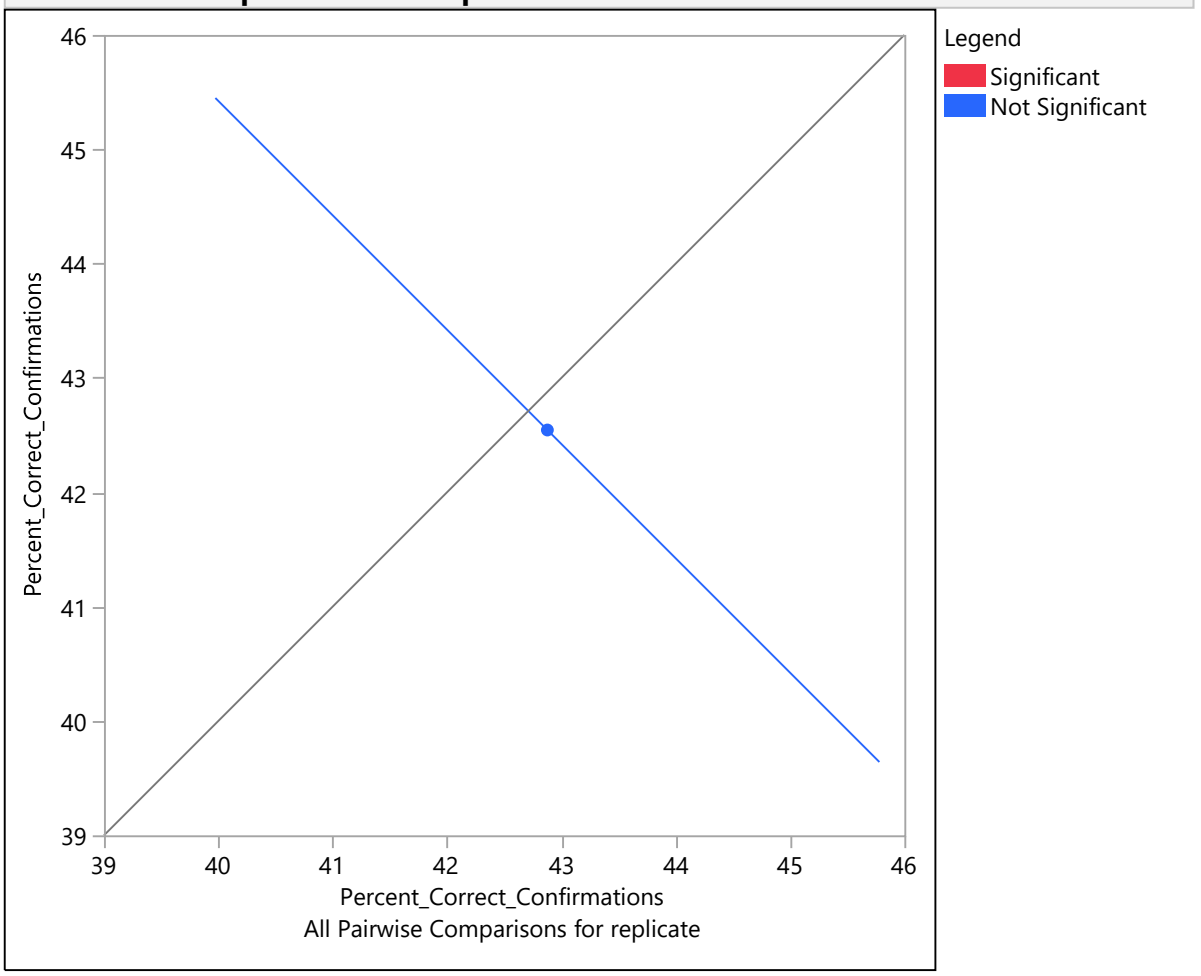
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.3354649	2.954950	0.11	0.9097	-5.47087	6.141796

Response Percent_Correct_Confirmations

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 1
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

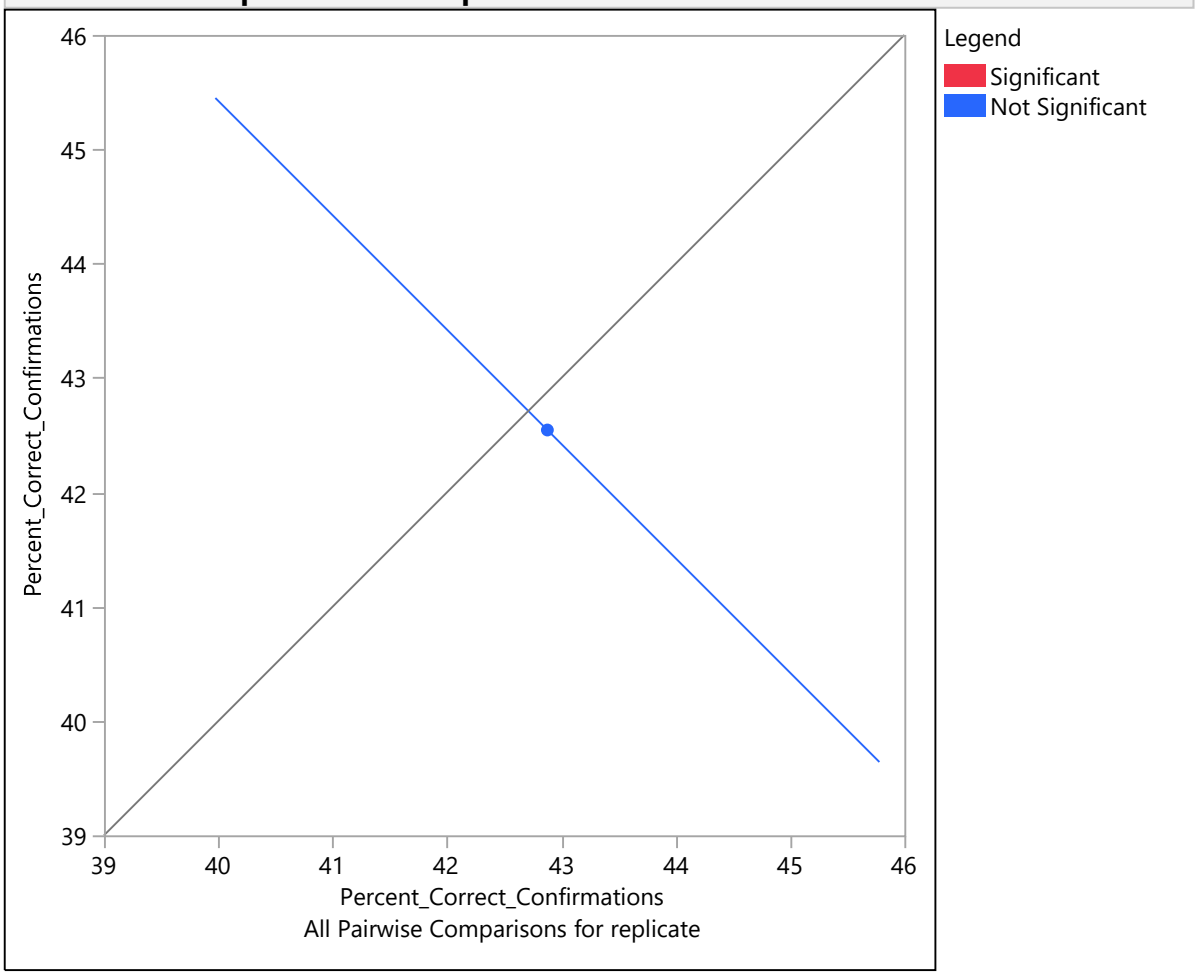
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.3354649	2.954950	0.11	0.9097	-5.47086	6.141793

Response Percent_Correct_Confirmations

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Percent_Confirmed

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	24.366177	1.7041291	477	21.017649	27.714704
1	23.813562	1.7041291	477	20.465034	27.162090

Tukey HSD All Pairwise Comparisons

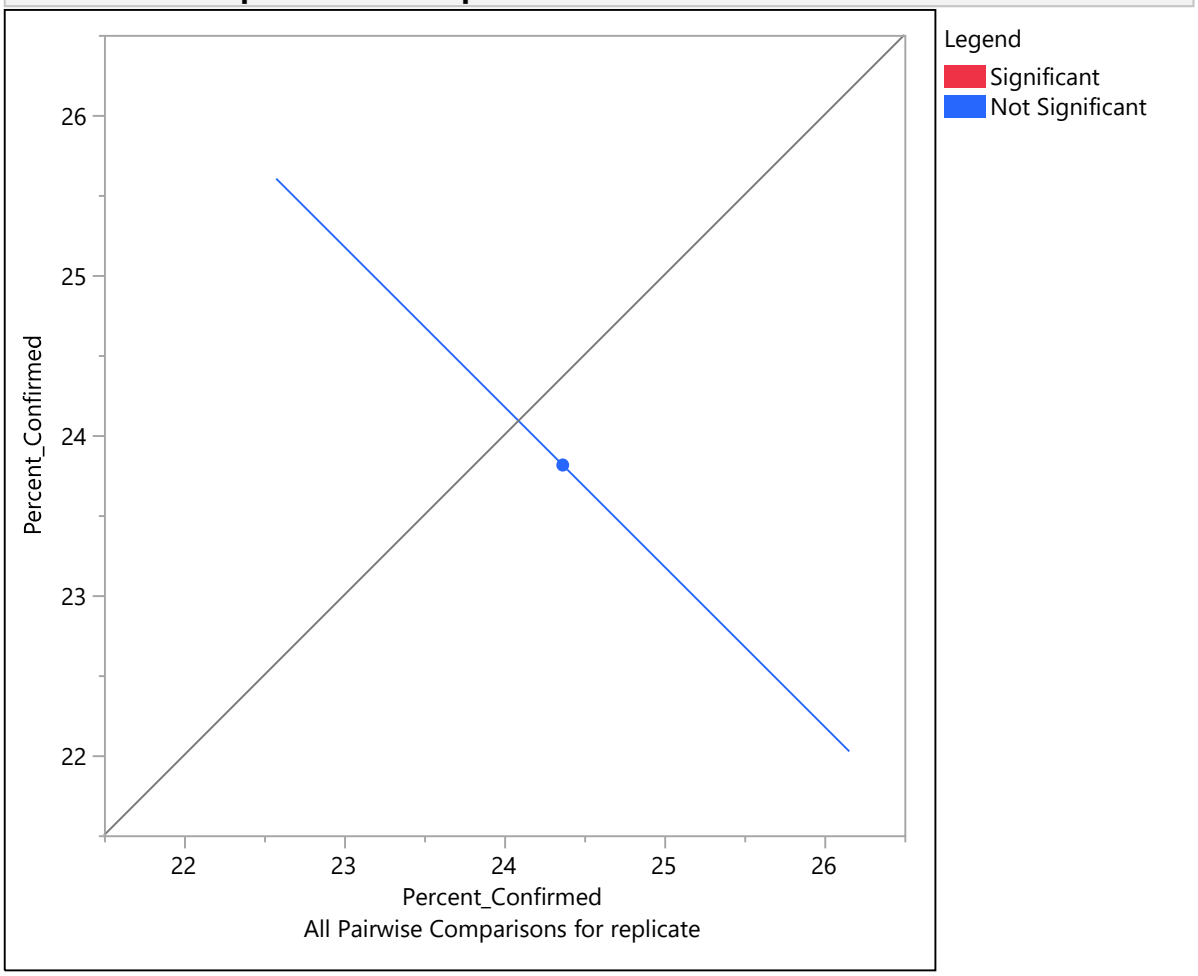
corp level = -1

Quantile = 1.96495 , Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.5526144	1.821791	0.30	0.7618	-3.02711	4.132343

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

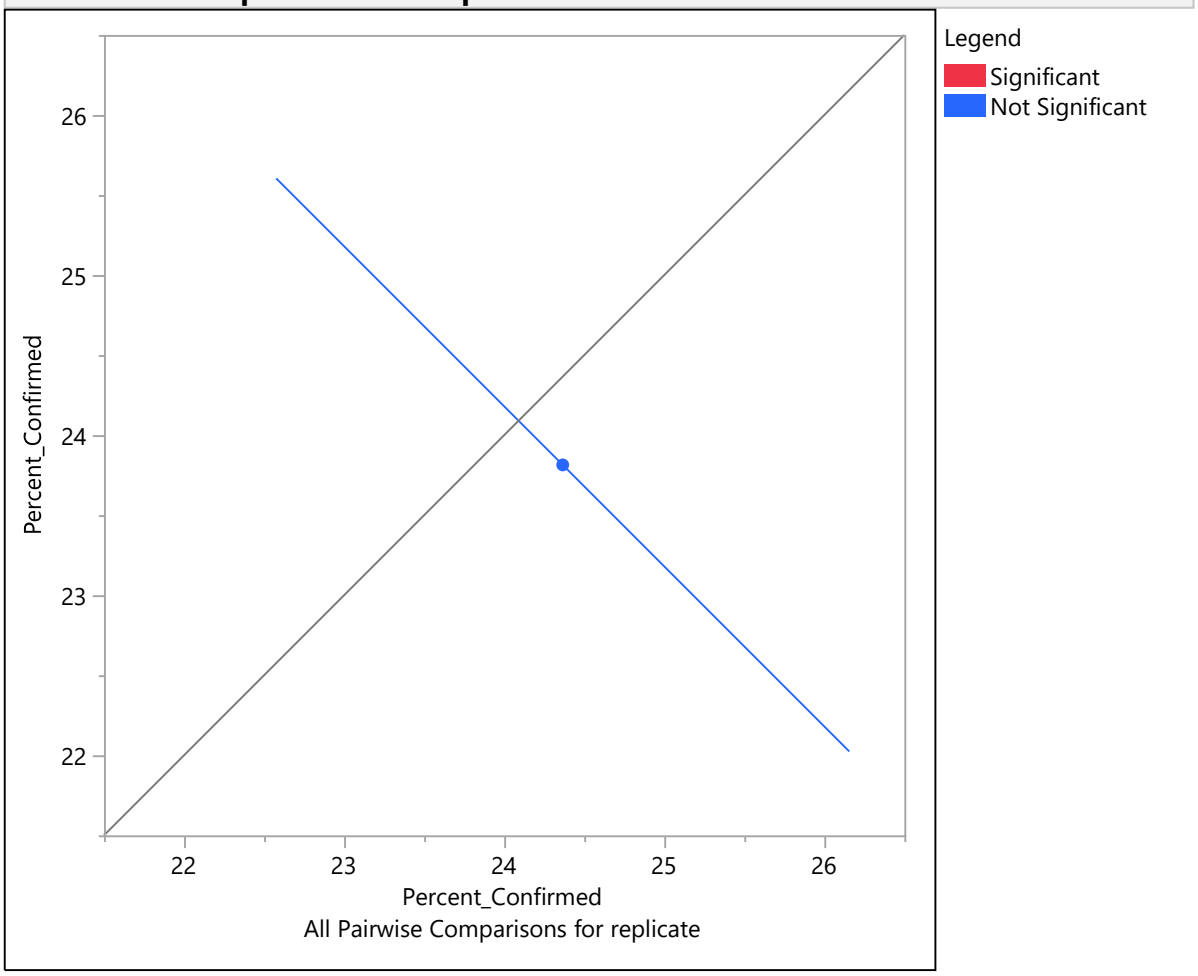
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.5526144	1.821791	0.30	0.7618	-3.02711	4.132341

Response Percent_Confirmed

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	23.349420	1.2882005	477	20.818171	25.880669
1	22.796805	1.2882005	477	20.265556	25.328055

Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey

All Pairwise Differences

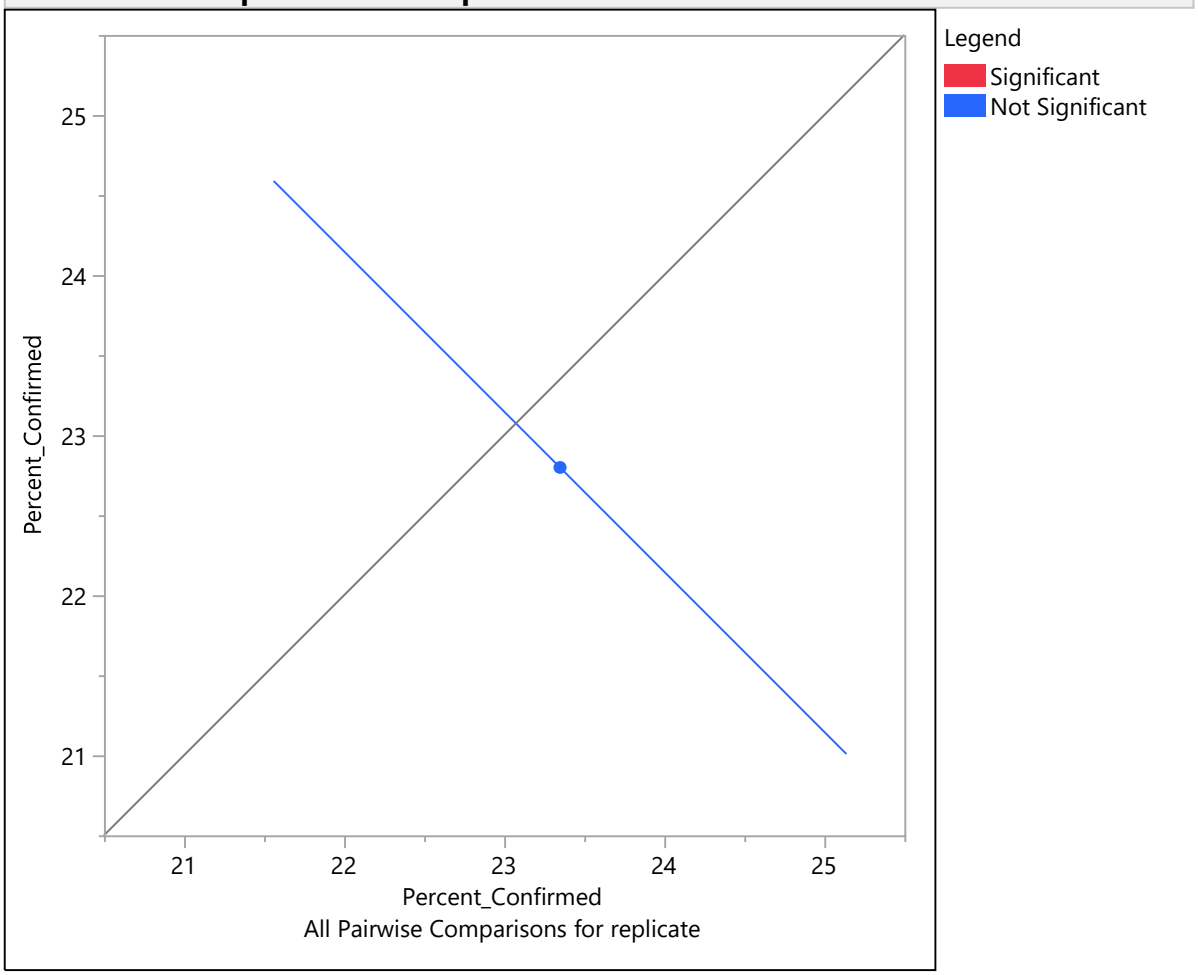
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.5526144	1.821791	0.30	0.7618	-3.02711	4.132343

Response Percent_Confirmed

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 0
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

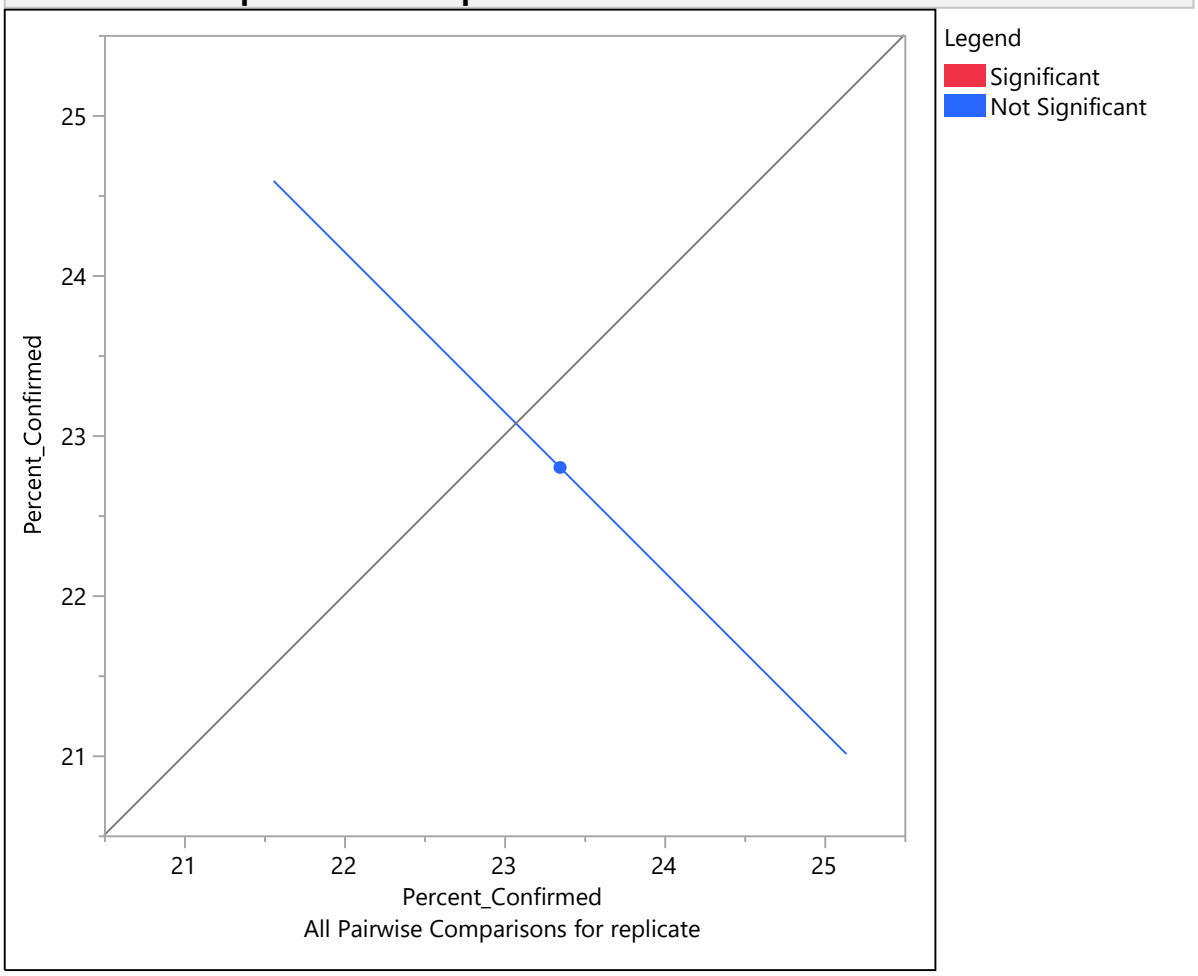
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.5526144	1.821791	0.30	0.7618	-3.02711	4.132341

Response Percent_Confirmed

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	22.332663	1.7041291	477	18.984135	25.681191
1	21.780049	1.7041291	477	18.431521	25.128577

Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96495 , Adjusted DF = 477.0 , Adjustment = Tukey-Kramer

All Pairwise Differences

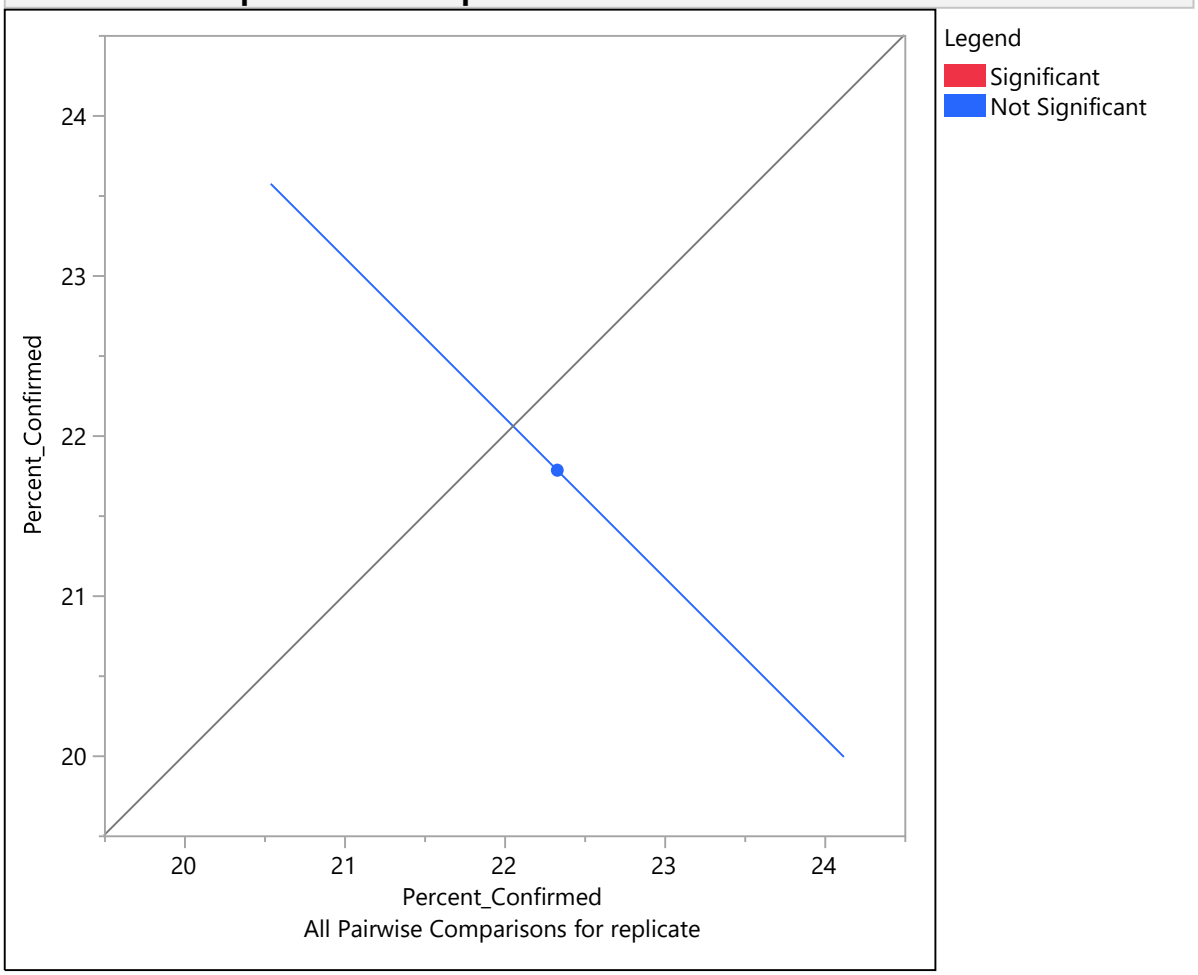
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.5526144	1.821791	0.30	0.7618	-3.02711	4.132343

Response Percent_Confirmed

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Student's t All Pairwise Comparisons

corp level = 1
Quantile = 1.96495 , DF = 477.0

All Pairwise Differences

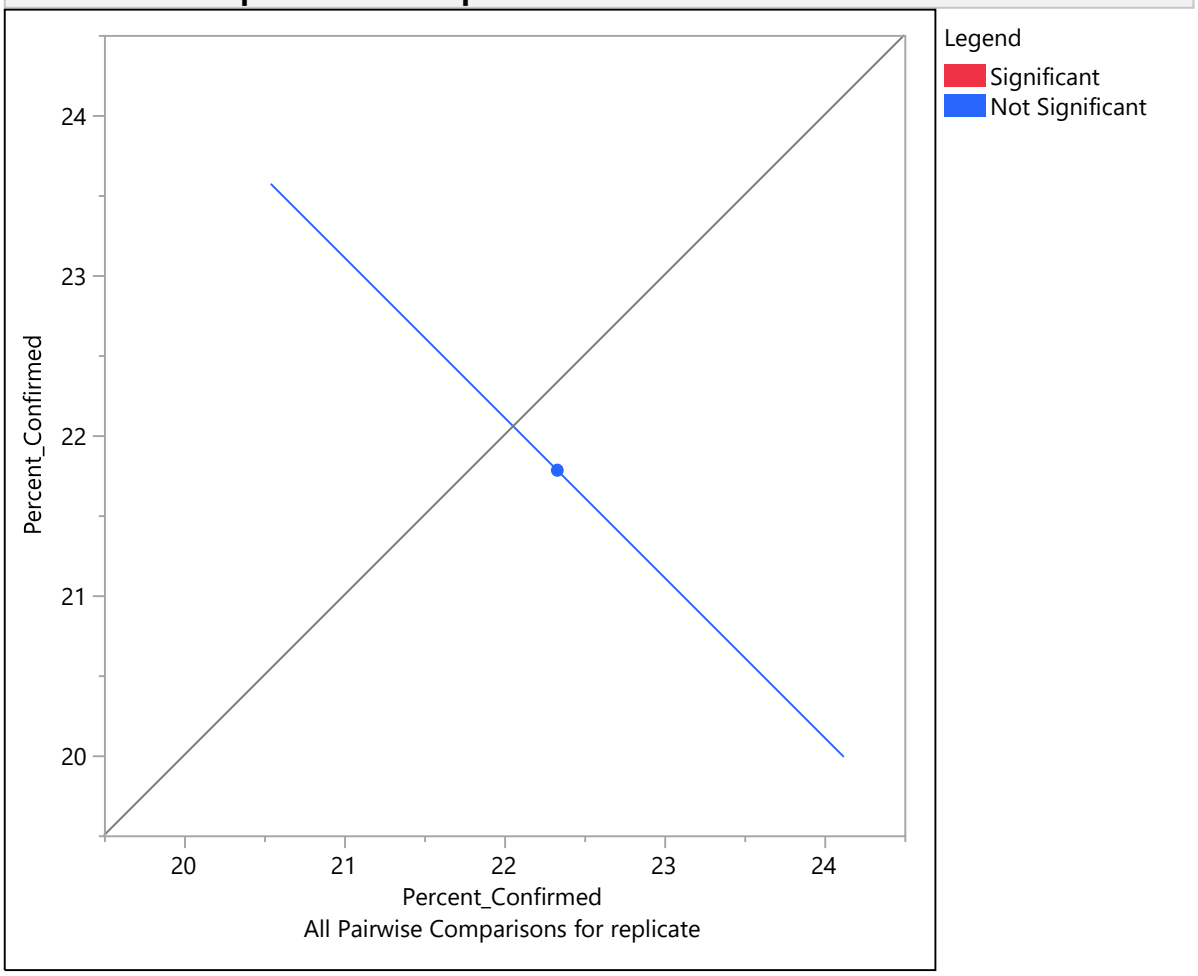
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0.5526144	1.821791	0.30	0.7618	-3.02711	4.132341

Response Percent_Confirmed

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Response Responsiveness

Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = -1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	3.23237e-6	1.49166e-6	462	3.01082e-7	6.16365e-6
1	3.45762e-6	1.51981e-6	462	4.71019e-7	6.44421e-6

Tukey HSD All Pairwise Comparisons

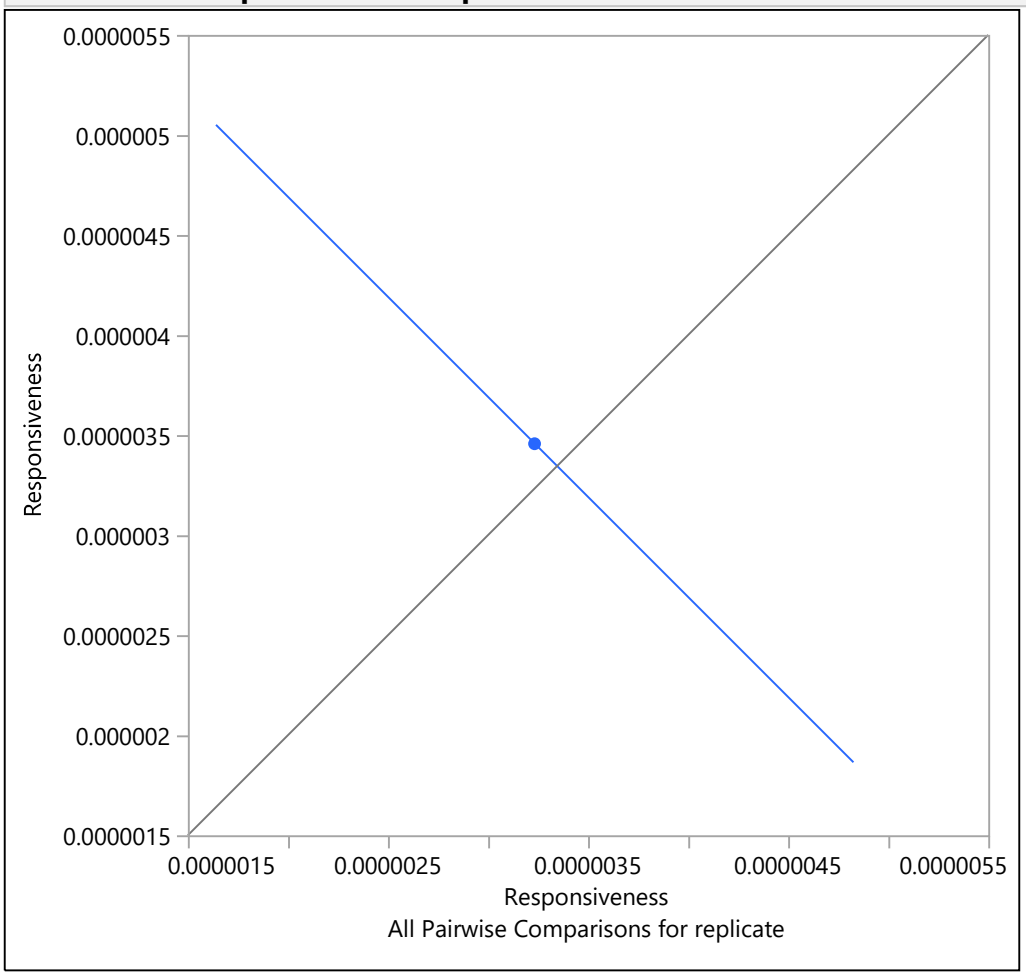
corp level = -1

Quantile = 1.96511 , Adjusted DF = 462.0, Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-2.252e-7	1.621e-6	-0.14	0.8895	-3.411e-6	2.9602e-6

All Pairwise Comparisons Scatterplot



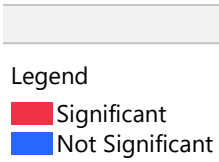
Student's t All Pairwise Comparisons

corp level = -1

Quantile = 1.96511 , DF = 462.0

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-2.252e-7	1.621e-6	-0.14	0.8895	-3.411e-6	2.9602e-6

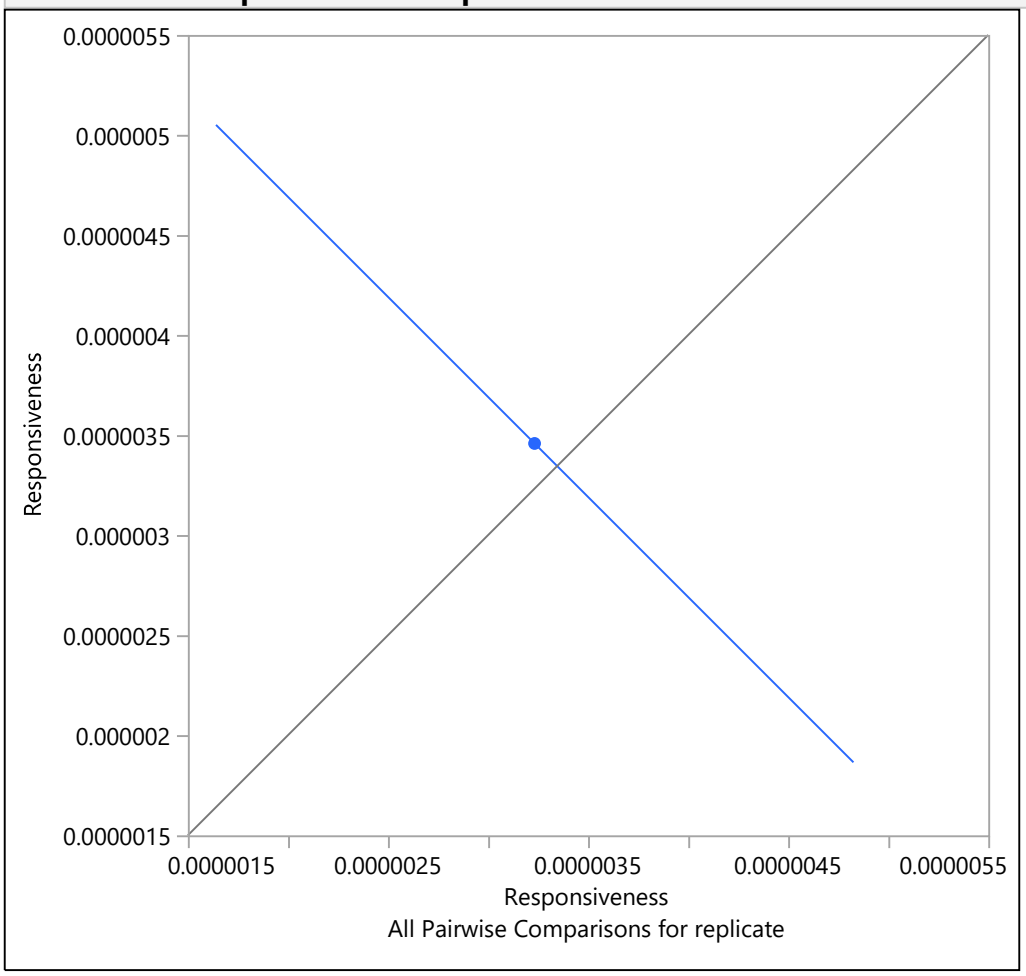

Legend
Significant
Not Significant

Response Responsiveness

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 0

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	3.41187e-6	1.12759e-6	462	1.19603e-6	5.62772e-6
1	3.63712e-6	1.16457e-6	462	1.34861e-6	5.92564e-6

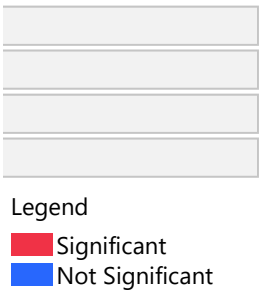
Tukey HSD All Pairwise Comparisons

corp level = 0

Quantile = 1.96511 , Adjusted DF = 462.0 , Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-2.252e-7	1.621e-6	-0.14	0.8895	-3.411e-6	2.9602e-6

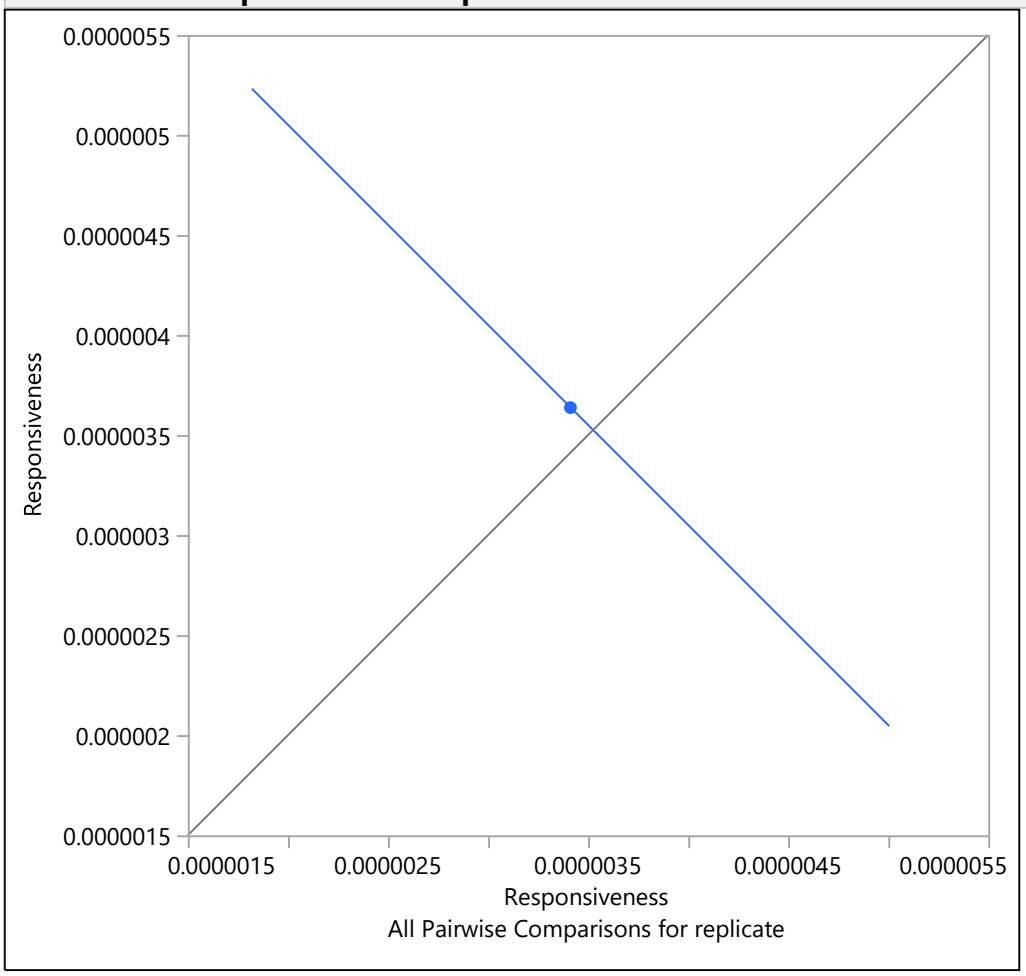

Legend
Significant
Not Significant

Response Responsiveness

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot

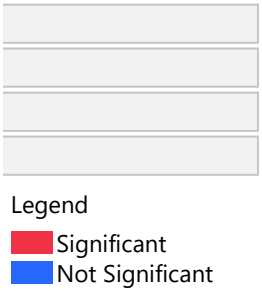


Student's t All Pairwise Comparisons

corp level = 0
Quantile = 1.96511 , DF = 462.0

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-2.252e-7	1.621e-6	-0.14	0.8895	-3.411e-6	2.9602e-6

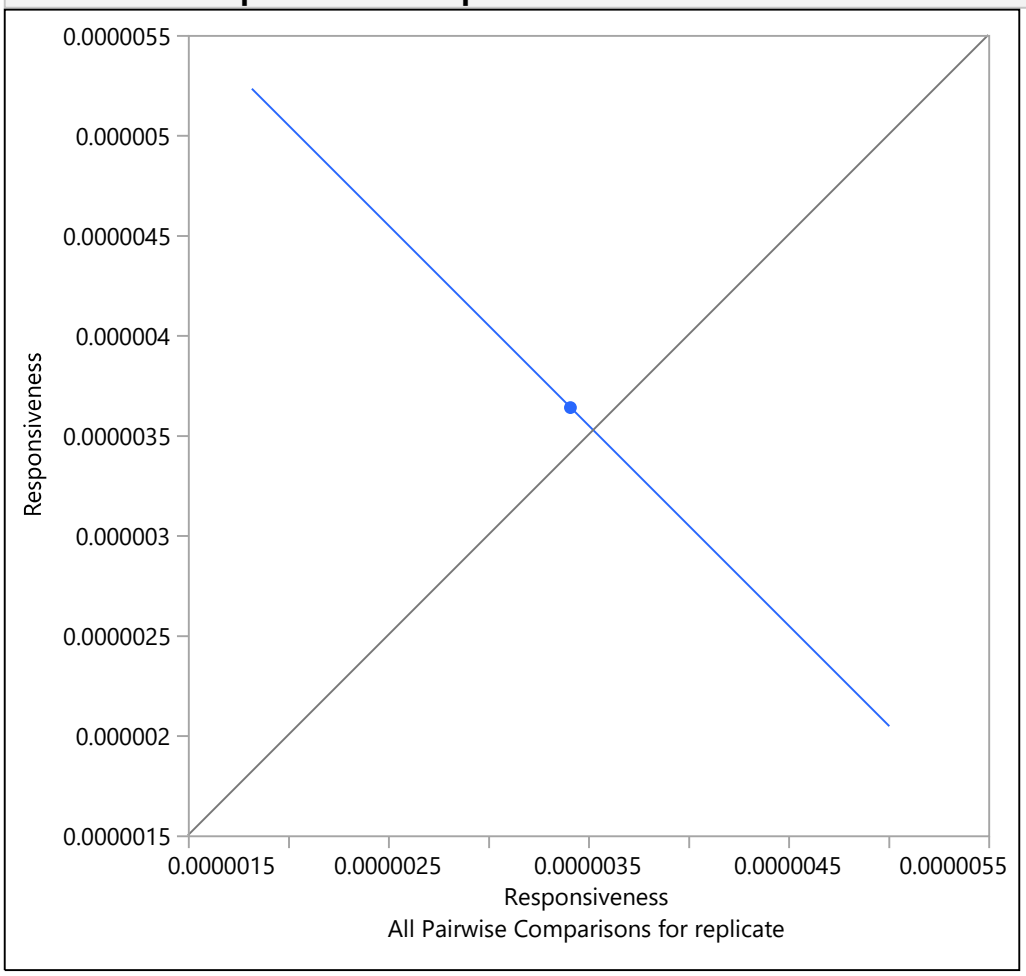

Legend
Significant
Not Significant

Response Responsiveness

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot



Multiple Comparisons for User-Defined Estimates

User-Defined Estimates

corp level = 1

replicate	Estimate	Std Error	DF	Lower 95%	Upper 95%
0	3.59138e-6	1.49166e-6	462	6.60101e-7	6.52267e-6
1	3.81663e-6	1.51981e-6	462	8.30038e-7	6.80323e-6

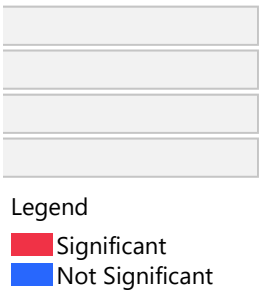
Tukey HSD All Pairwise Comparisons

corp level = 1

Quantile = 1.96511 , Adjusted DF = 462.0 , Adjustment = Tukey

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-2.252e-7	1.621e-6	-0.14	0.8895	-3.411e-6	2.9602e-6

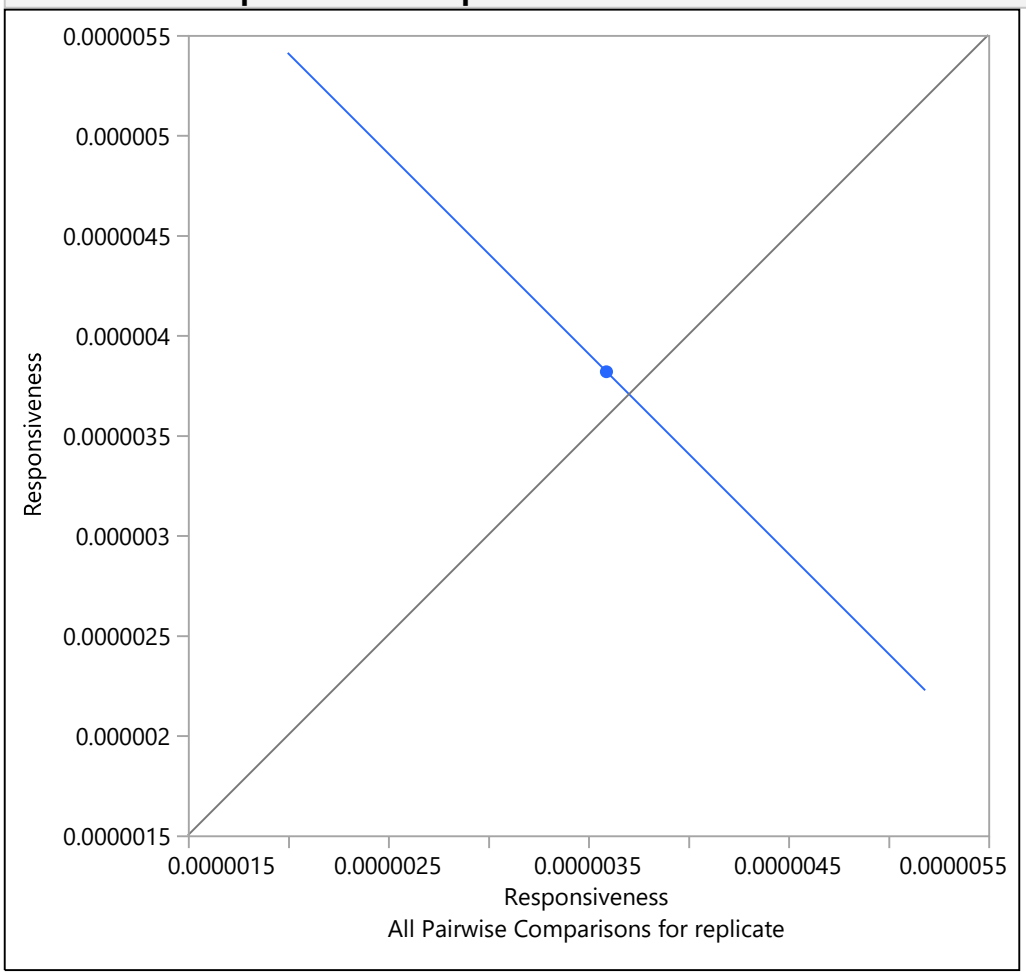

Legend
Significant
Not Significant

Response Responsiveness

Multiple Comparisons for User-Defined Estimates

Tukey HSD All Pairwise Comparisons

All Pairwise Comparisons Scatterplot

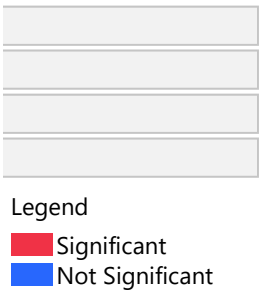


Student's t All Pairwise Comparisons

corp level = 1
Quantile = 1.96511 , DF = 462.0

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	-2.252e-7	1.621e-6	-0.14	0.8895	-3.411e-6	2.9602e-6

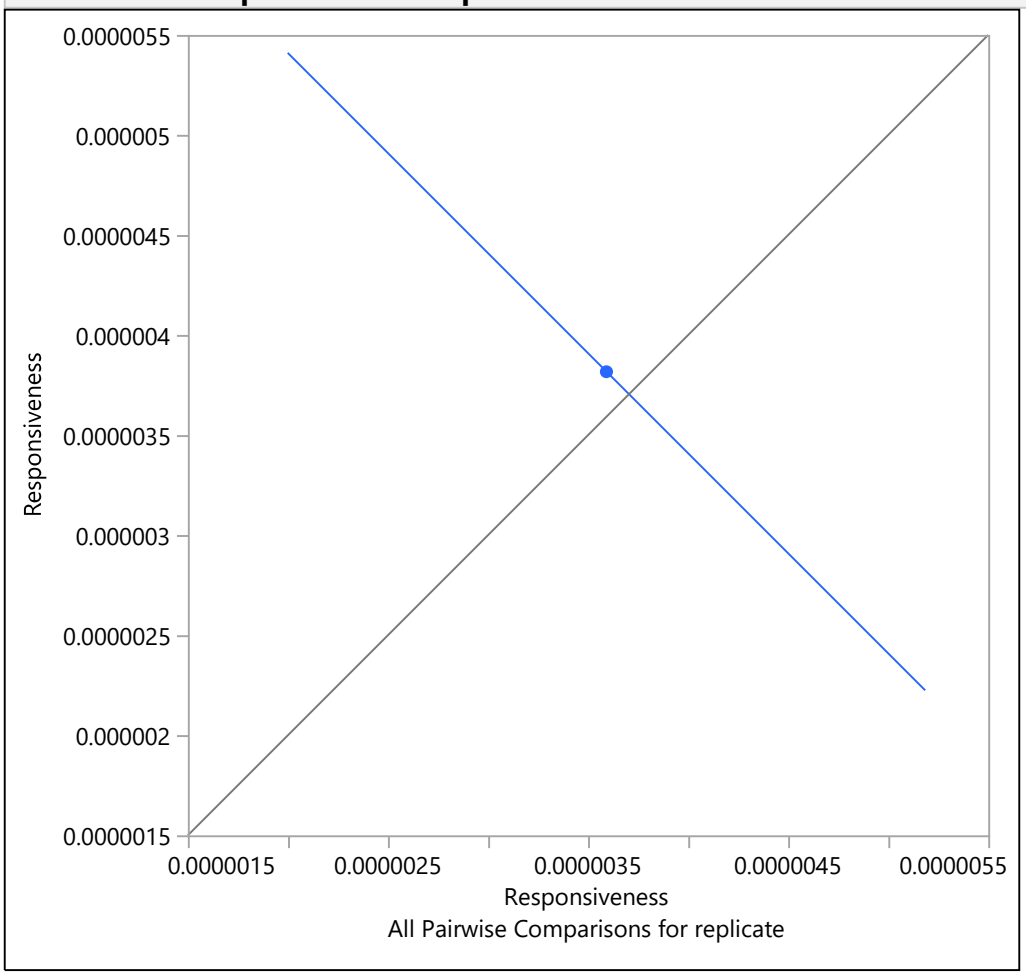

Legend
Significant
Not Significant

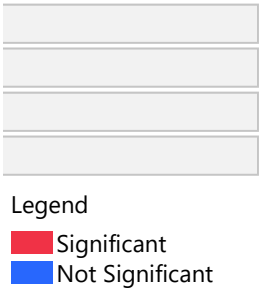
Response Responsiveness

Multiple Comparisons for User-Defined Estimates

Student's t All Pairwise Comparisons

All Pairwise Comparisons Scatterplot




Legend
Significant
Not Significant

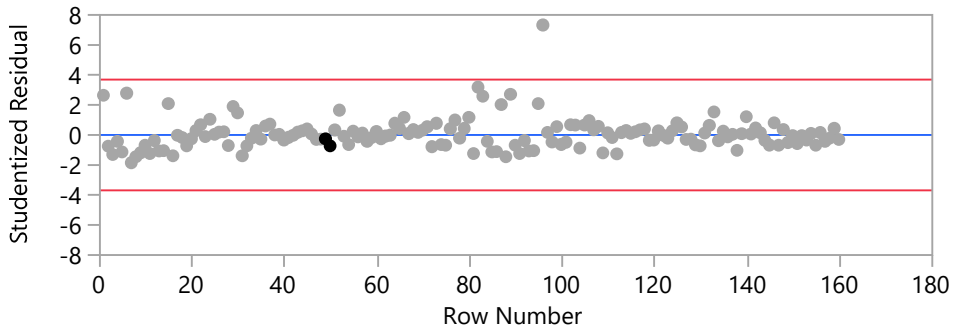
Appendix K. Response Surface JMP Outputs: One Vehicle Operation

Response Detect_TypeII_error

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	61	11115.262	182.217	0.9290
Pure Error	83	16280.610	196.152	**Prob > F**
Total Error	144	27395.872		0.6160
				Max RSq
				0.7097

Studentized Residuals

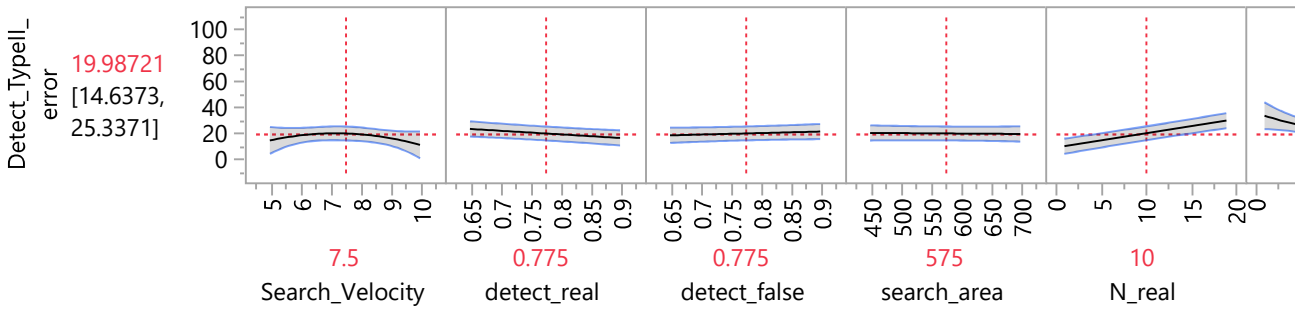


Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Effect Tests

Source	Nparm	DF	Sum of Squares	F Ratio	Prob > F
Search_Velocity	1	1	380.428	1.9996	0.1595
detect_real	1	1	1577.619	8.2924	0.0046*
detect_false	1	1	273.948	1.4399	0.2321
search_area	1	1	20.565	0.1081	0.7428
N_real	1	1	12741.613	66.9733	<.0001*
N_false	1	1	8739.290	45.9360	<.0001*
Search_Velocity*Search_Velocity	1	1	366.687	1.9274	0.1672
Search_Velocity*detect_real	1	1	185.681	0.9760	0.3248
Search_Velocity*detect_false	1	1	314.052	1.6507	0.2009
detect_real*detect_false	1	1	556.064	2.9228	0.0895
detect_real*N_real	1	1	2872.939	15.1009	0.0002*
Search_Velocity*N_false	1	1	181.826	0.9557	0.3299
detect_false*N_false	1	1	282.123	1.4829	0.2253
N_real*N_false	1	1	185.523	0.9752	0.3251
N_false*N_false	1	1	220.610	1.1596	0.2834

Prediction Profiler

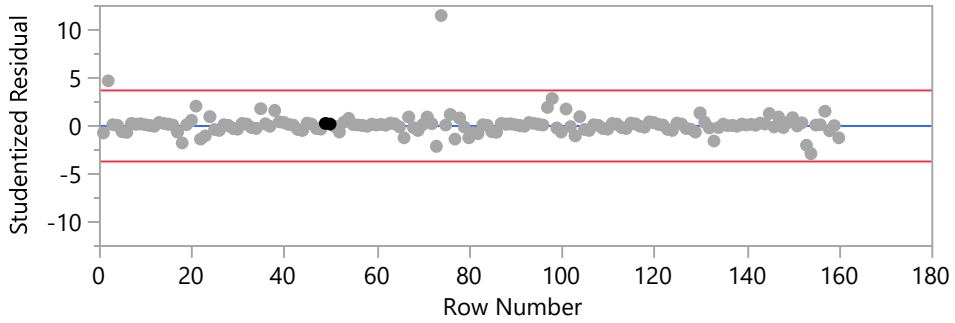


Response Confirm_TypeII_error

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	60	5253.396	87.557	0.7734
Pure Error	83	9396.469	113.210	**Prob > F**
Total Error	143	14649.864		0.8524
				Max RSq
				0.6370

Studentized Residuals

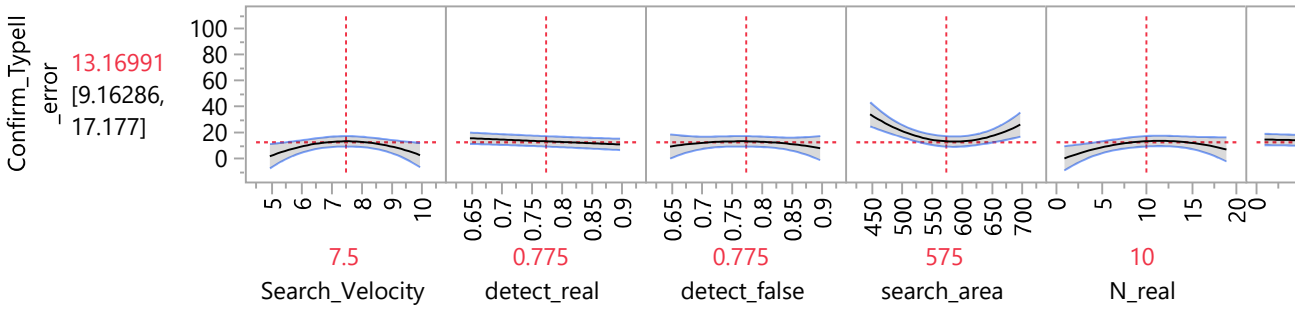


Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	46.301289	9.439308	4.91	<.0001*
Search_Velocity	0.172081	0.352389	0.49	0.6261
detect_real	-19.0181	7.047771	-2.70	0.0078*
detect_false	-4.87438	7.047771	-0.69	0.4903
search_area	-0.031553	0.007048	-4.48	<.0001*
N_real	0.381193	0.097886	3.89	0.0002*
N_false	-0.157433	0.097886	-1.61	0.1100
(Search_Velocity-7.5)*(Search_Velocity-7.5)	-1.769334	0.707035	-2.50	0.0135*
(detect_false-0.775)*(detect_false-0.775)	-293.1883	282.814	-1.04	0.3016
(detect_real-0.775)*(search_area-575)	0.1574549	0.057256	2.75	0.0067*
(search_area-575)*(search_area-575)	0.0010761	0.000283	3.80	0.0002*
(detect_real-0.775)*(N_real-10)	-1.45936	0.795227	-1.84	0.0686
(search_area-575)*(N_real-10)	-0.002979	0.000795	-3.75	0.0003*
(N_real-10)*(N_real-10)	-0.119153	0.054555	-2.18	0.0306*
(detect_real-0.775)*(N_false-10)	0.8239796	0.795227	1.04	0.3019
(search_area-575)*(N_false-10)	0.0012715	0.000795	1.60	0.1121
(N_real-10)*(N_false-10)	-0.013985	0.011045	-1.27	0.2075

Prediction Profiler

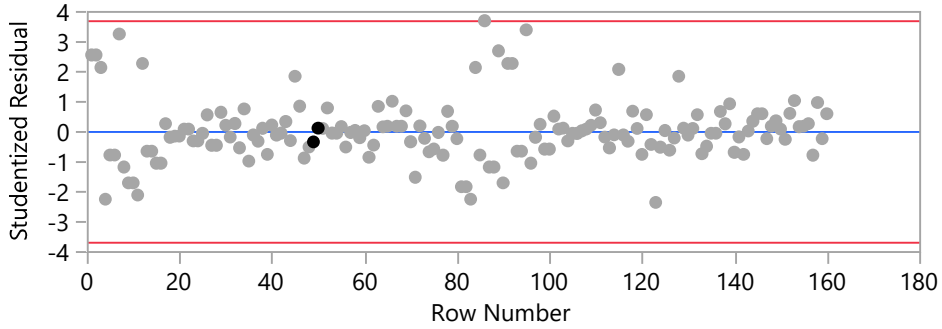


Response Detect_Typel_error

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	33	3301.803	100.055	0.6395
Pure Error	117	18306.761	156.468	**Prob > F**
Total Error	150	21608.564		0.9303
				Max RSq
				0.5732

Studentized Residuals



Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	40.686212	10.51619	3.87	0.0002*
detect_real	9.2738232	8.357384	1.11	0.2689
detect_false	-41.28353	8.357384	-4.94	<.0001*
search_area	0.0014062	0.008357	0.17	0.8666
N_real	-0.801424	0.116075	-6.90	<.0001*
N_false	0.9460895	0.116075	8.15	<.0001*
(detect_false-0.775)*(N_real-10)	1.9758132	0.942996	2.10	0.0378*
(detect_real-0.775)*(N_false-10)	-1.004174	0.942996	-1.06	0.2886
(detect_false-0.775)*(N_false-10)	-1.223314	0.942996	-1.30	0.1965
(search_area-575)*(N_false-10)	0.0008155	0.000943	0.86	0.3885

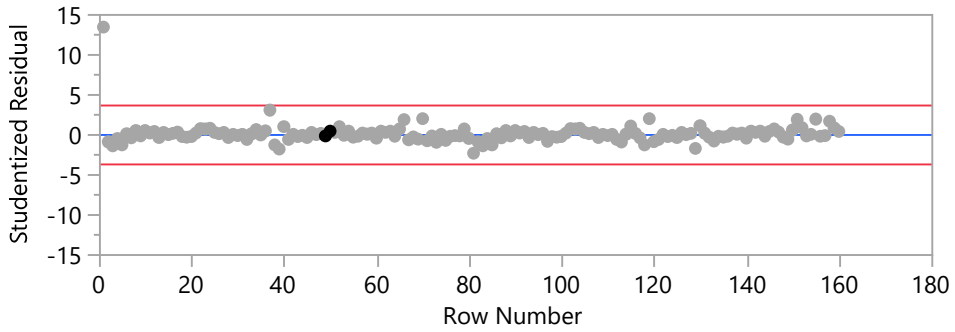
Prediction Profiler

Response Confirm_TypeI_error

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	60	5158.644	85.9774	0.9193
Pure Error	83	7762.211	93.5206	**Prob > F**
Total Error	143	12920.855		0.6316
				Max RSq
				0.6732

Studentized Residuals

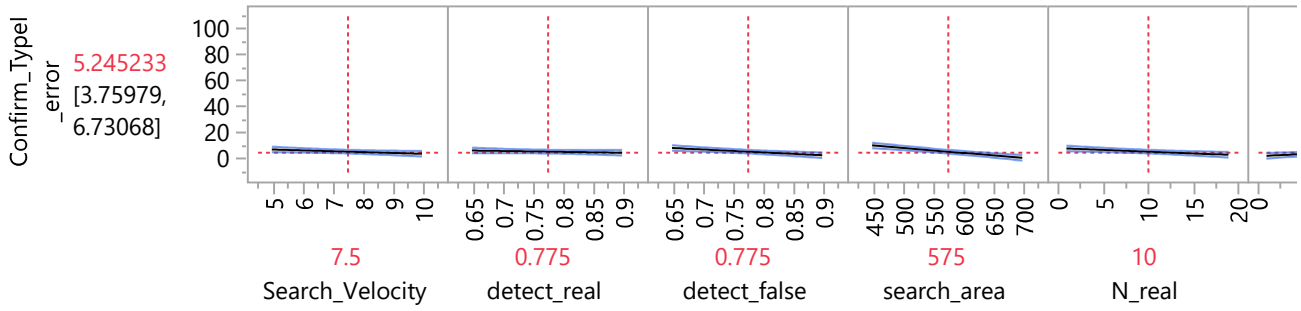


Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	53.630396	8.690516	6.17	<.0001*
Search_Velocity	-0.65598	0.330941	-1.98	0.0494*
detect_real	-6.912147	6.618821	-1.04	0.2981
detect_false	-22.26512	6.618821	-3.36	0.0010*
search_area	-0.037838	0.006619	-5.72	<.0001*
N_real	-0.26853	0.091928	-2.92	0.0041*
N_false	0.3589383	0.091928	3.90	0.0001*
(Search_Velocity-7.5)*(detect_real-0.775)	3.3725112	2.688578	1.25	0.2117
(Search_Velocity-7.5)*(search_area-575)	0.0051846	0.002689	1.93	0.0558
(detect_real-0.775)*(search_area-575)	0.0726818	0.053772	1.35	0.1786
(detect_false-0.775)*(search_area-575)	0.1761873	0.053772	3.28	0.0013*
(Search_Velocity-7.5)*(N_real-10)	0.0790678	0.037341	2.12	0.0360*
(detect_real-0.775)*(N_real-10)	0.7518675	0.746827	1.01	0.3158
(search_area-575)*(N_real-10)	0.0020418	0.000747	2.73	0.0070*
(detect_false-0.775)*(N_false-10)	-0.919268	0.746827	-1.23	0.2204
(search_area-575)*(N_false-10)	-0.002754	0.000747	-3.69	0.0003*
(N_real-10)*(N_false-10)	-0.010997	0.010373	-1.06	0.2909

Prediction Profiler

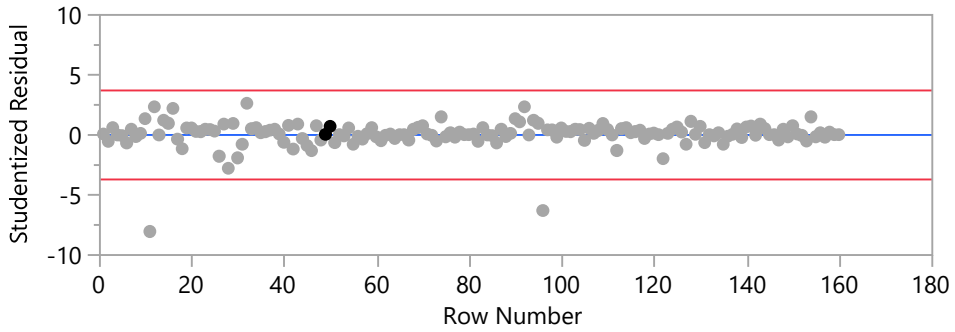


Response Percent_Detected

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	59	3016.7795	51.1319	1.1364
Pure Error	83	3734.4529	44.9934	**Prob > F**
Total Error	142	6751.2324		0.2930
				Max RSq
				0.7856

Studentized Residuals

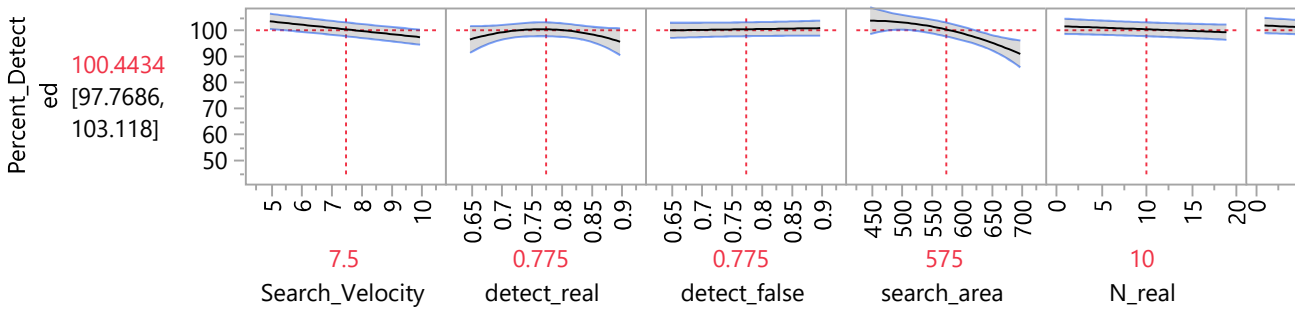


Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	141.949	6.424476	22.10	<.0001*
Search_Velocity	-1.216906	0.24006	-5.07	<.0001*
detect_real	-3.556619	4.801206	-0.74	0.4601
detect_false	3.3492823	4.801206	0.70	0.4866
search_area	-0.0511	0.004801	-10.64	<.0001*
N_real	-0.124934	0.066683	-1.87	0.0631
N_false	-0.158604	0.066683	-2.38	0.0187*
(detect_real-0.775)*(detect_real-0.775)	-278.6331	163.2477	-1.71	0.0900
(Search_Velocity-7.5)*(search_area-575)	-0.009408	0.00195	-4.82	<.0001*
(detect_real-0.775)*(search_area-575)	-0.044474	0.039005	-1.14	0.2561
(search_area-575)*(search_area-575)	-0.000199	0.000163	-1.22	0.2257
(Search_Velocity-7.5)*(N_real-10)	-0.042215	0.027087	-1.56	0.1213
(detect_real-0.775)*(N_real-10)	0.5299708	0.541739	0.98	0.3296
(search_area-575)*(N_real-10)	-0.000855	0.000542	-1.58	0.1166
(Search_Velocity-7.5)*(N_false-10)	-0.049159	0.027087	-1.81	0.0717
(detect_real-0.775)*(N_false-10)	1.2244152	0.541739	2.26	0.0253*
(search_area-575)*(N_false-10)	-0.001411	0.000542	-2.60	0.0102*
(N_real-10)*(N_false-10)	0.0165489	0.007524	2.20	0.0295*

Prediction Profiler

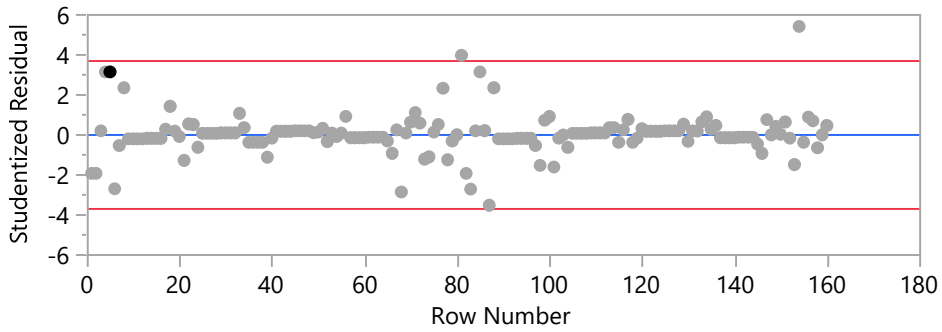


Response Percent_Correct_Confirmations

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	29	9936.189	342.627	1.0248
Pure Error	117	39118.719	334.348	**Prob > F**
Total Error	146	49054.908		0.4434
				Max RSq
				0.7805

Studentized Residuals



Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	113.56363	16.4051	6.92	<.0001*
detect_real	26.140778	12.76344	2.05	0.0423*
detect_false	24.989936	12.76344	1.96	0.0521
search_area	-0.178795	0.012763	-14.01	<.0001*
N_real	1.0346708	0.17727	5.84	<.0001*
N_false	-0.99826	0.17727	-5.63	<.0001*
(detect_real-0.775)*(detect_real-0.775)	-528.2151	486.6769	-1.09	0.2796
(detect_real-0.775)*(search_area-575)	-0.207884	0.103691	-2.00	0.0468*
(detect_false-0.775)*(search_area-575)	-0.211849	0.103691	-2.04	0.0428*
(search_area-575)*(search_area-575)	-0.000611	0.000487	-1.25	0.2117
(search_area-575)*(N_real-10)	-0.00809	0.00144	-5.62	<.0001*
(N_real-10)*(N_real-10)	-0.118907	0.093881	-1.27	0.2073
(search_area-575)*(N_false-10)	0.0073796	0.00144	5.12	<.0001*
(N_real-10)*(N_false-10)	0.032161	0.020002	1.61	0.1100

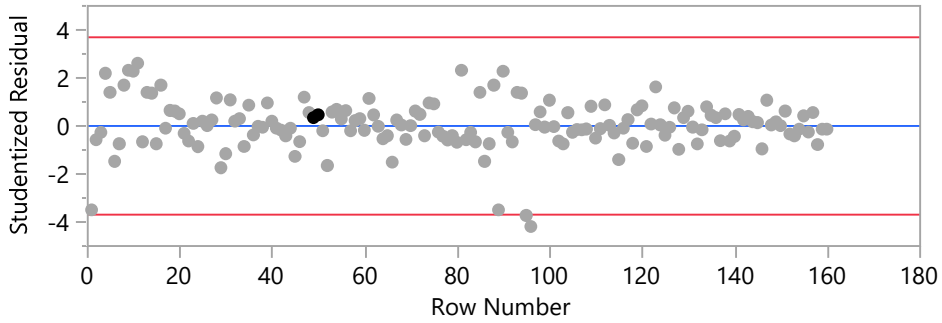
Prediction Profiler

Response Percent_Correct_Detected

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	16	3775.208	235.951	0.6806
Pure Error	135	46802.184	346.683	**Prob > F**
Total Error	151	50577.392		0.8091
				Max RSq
				0.2078

Studentized Residuals

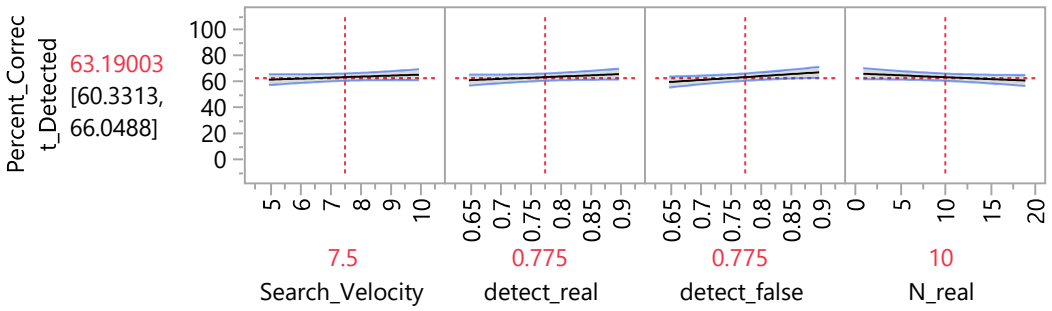


Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	23.059503	14.93809	1.54	0.1248
Search_Velocity	0.7630558	0.637181	1.20	0.2330
detect_real	18.383115	12.74362	1.44	0.1512
detect_false	29.758624	12.74362	2.34	0.0209*
N_real	-0.290224	0.176995	-1.64	0.1031
(Search_Velocity-7.5)*(detect_real-0.775)	5.1230634	5.176482	0.99	0.3239
(detect_real-0.775)*(detect_false-0.775)	-116.8649	103.5296	-1.13	0.2608
(detect_real-0.775)*(N_real-10)	4.410524	1.437912	3.07	0.0026*
(detect_false-0.775)*(N_real-10)	-2.067798	1.437912	-1.44	0.1525

Prediction Profiler

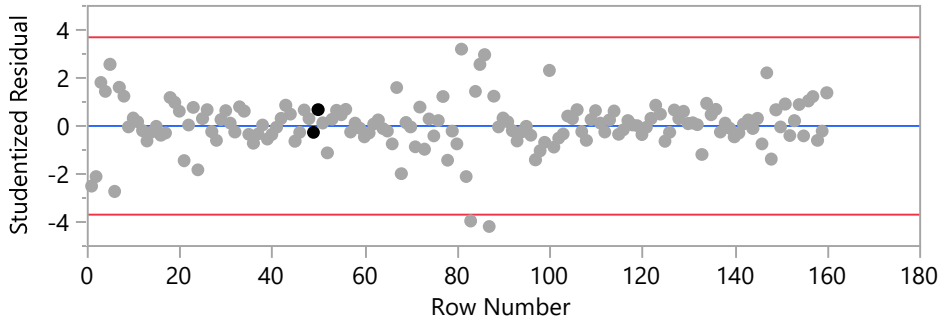


Response Percent_Confirmed

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	56	6239.515	111.420	1.3965
Pure Error	83	6622.381	79.788	**Prob > F**
Total Error	139	12861.895		0.0826
				Max RSq
				0.8857

Studentized Residuals



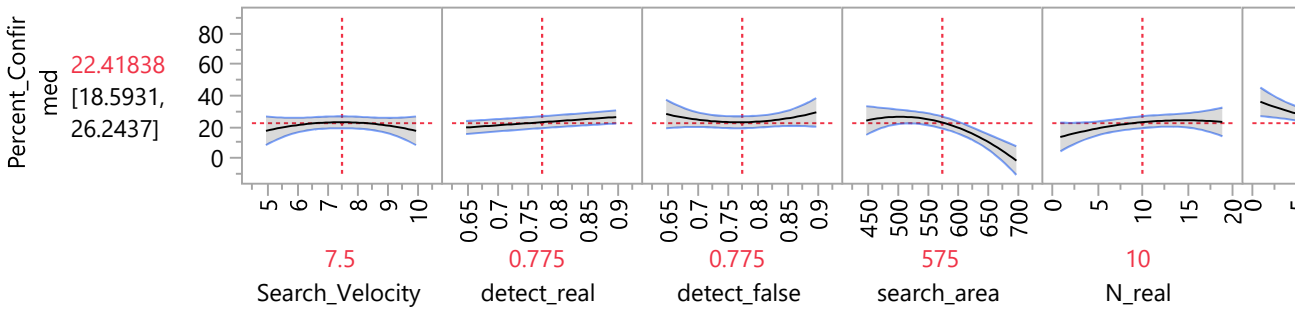
Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	59.217343	8.972664	6.60	<.0001*
Search_Velocity	-0.004785	0.334902	-0.01	0.9886
detect_real	26.842105	6.698046	4.01	<.0001*
detect_false	4.1307815	6.698046	0.62	0.5384
search_area	-0.1026	0.006698	-15.32	<.0001*
N_real	0.5344153	0.093028	5.74	<.0001*
N_false	-0.71164	0.093028	-7.65	<.0001*
(Search_Velocity-7.5)*(Search_Velocity-7.5)	-0.877147	0.691754	-1.27	0.2069
(Search_Velocity-7.5)*(detect_real-0.775)	5.3026316	2.720759	1.95	0.0533
(detect_real-0.775)*(detect_false-0.775)	-55.78947	54.41518	-1.03	0.3070
(detect_false-0.775)*(detect_false-0.775)	369.1412	276.7015	1.33	0.1844
(detect_real-0.775)*(search_area-575)	-0.218947	0.054415	-4.02	<.0001*
(search_area-575)*(search_area-575)	-0.000751	0.000277	-2.71	0.0075*
(detect_real-0.775)*(N_real-10)	1.5131579	0.755766	2.00	0.0472*
(detect_false-0.775)*(N_real-10)	-1.089181	0.755766	-1.44	0.1518
(search_area-575)*(N_real-10)	-0.004152	0.000756	-5.49	<.0001*
(N_real-10)*(N_real-10)	-0.057329	0.053376	-1.07	0.2847
(detect_real-0.775)*(N_false-10)	-1.403509	0.755766	-1.86	0.0654
(search_area-575)*(N_false-10)	0.0054313	0.000756	7.19	<.0001*
(N_real-10)*(N_false-10)	0.015229	0.010497	1.45	0.1491
(N_false-10)*(N_false-10)	0.0819957	0.053376	1.54	0.1268

Response Percent_Confirmed

Prediction Profiler

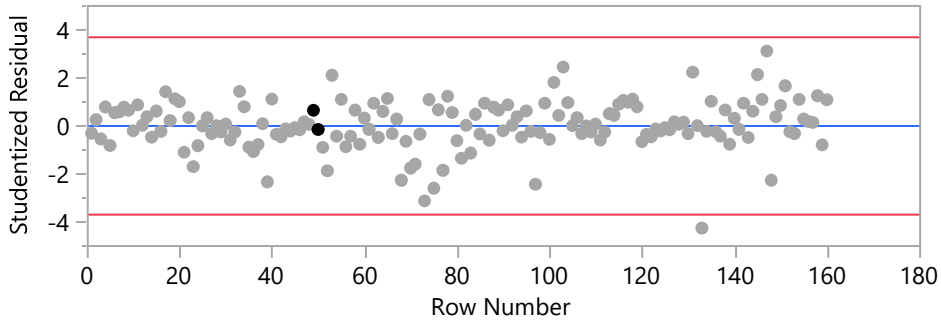


Response Mission Time

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	58	21.612942	0.372637	1.0293
Pure Error	83	30.049720	0.362045	**Prob > F**
Total Error	141	51.662662		0.4470
				Max RSq
				0.9922

Studentized Residuals

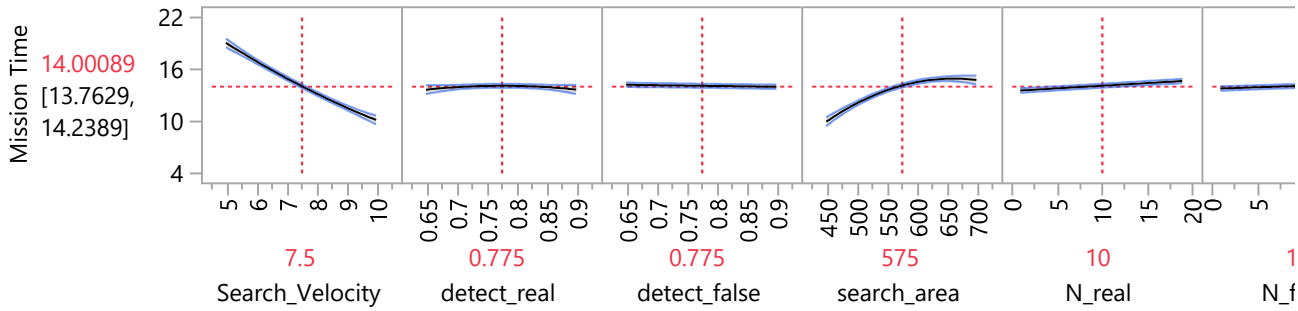


Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	15.951629	0.564329	28.27	<.0001*
Search_Velocity	-1.776593	0.021074	-84.30	<.0001*
detect_real	0.1158081	0.421485	0.27	0.7839
detect_false	-0.921364	0.421485	-2.19	0.0305*
search_area	0.0191813	0.000421	45.51	<.0001*
N_real	0.0613377	0.005854	10.48	<.0001*
N_false	0.0355387	0.005854	6.07	<.0001*
(Search_Velocity-7.5)*(Search_Velocity-7.5)	0.0800006	0.040179	1.99	0.0484*
(detect_real-0.775)*(detect_real-0.775)	-28.10643	16.07144	-1.75	0.0825
(Search_Velocity-7.5)*(search_area-575)	-0.004678	0.000171	-27.32	<.0001*
(detect_false-0.775)*(search_area-575)	0.0071704	0.003424	2.09	0.0380*
(search_area-575)*(search_area-575)	-0.000112	0.000016	-6.94	<.0001*
(Search_Velocity-7.5)*(N_real-10)	-0.007466	0.002378	-3.14	0.0021*
(detect_real-0.775)*(N_real-10)	0.0716956	0.047558	1.51	0.1339
(search_area-575)*(N_real-10)	-0.000478	4.756e-5	-10.04	<.0001*
(Search_Velocity-7.5)*(N_false-10)	-0.004754	0.002378	-2.00	0.0475*
(detect_false-0.775)*(N_false-10)	-0.069045	0.047558	-1.45	0.1488
(search_area-575)*(N_false-10)	-0.000267	4.756e-5	-5.62	<.0001*
(N_real-10)*(N_false-10)	-0.000991	0.000661	-1.50	0.1357

Prediction Profiler



Appendix L. Response Surface JMP Outputs: Two Vehicle Operation

Least Squares Fit

Effect Summary

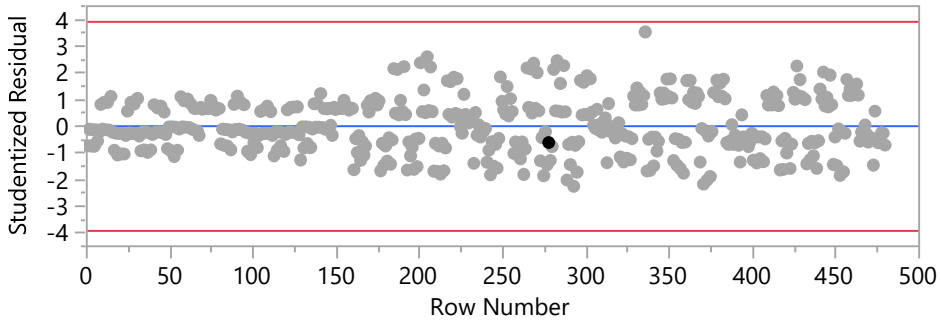
Source	LogWorth	PValue	
Search_Velocity	82.112	0.00000	
corp level*corp level	19.967	0.00000	
corp level	9.802	0.00000	^
Search_Velocity*corp level	5.355	0.00000	
N_real	5.001	0.00001	
N_real*corp level	4.345	0.00005	
Search_Velocity*N_real	2.265	0.00543	
N_false	2.189	0.00647	
N_false*corp level	1.253	0.05585	
Search_Velocity*N_false	1.118	0.07626	
N_real*N_false	0.984	0.10365	
detect_real*detect_false	0.357	0.43996	
detect_real*N_false	0.348	0.44857	
detect_false*N_real	0.346	0.45089	
N_real*N_real	0.331	0.46641	
detect_false	0.266	0.54230	^
detect_false*corp level	0.259	0.55063	
Search_Velocity*Search_Velocity	0.244	0.57034	
detect_real	0.229	0.58959	^
Search_Velocity*detect_real	0.198	0.63384	
Search_Velocity*detect_false	0.193	0.64143	
detect_real*corp level	0.178	0.66448	

Response Mission_Time

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	106	297.0866	2.80270	0.3142
Pure Error	351	3130.7513	8.91952	**Prob > F**
Total Error	457	3427.8379		1.0000
				Max RSq
				0.6583

Studentized Residuals



Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	18.63482	1.342004	13.89	<.0001*
Search_Velocity	-1.319778	0.055051	-23.97	<.0001*
detect_real	0.5943434	1.101018	0.54	0.5896
detect_false	-0.496818	1.101018	-0.45	0.6520
N_real	0.0683207	0.015292	4.47	<.0001*

Least Squares Fit

Response Mission_Time

Parameter Estimates

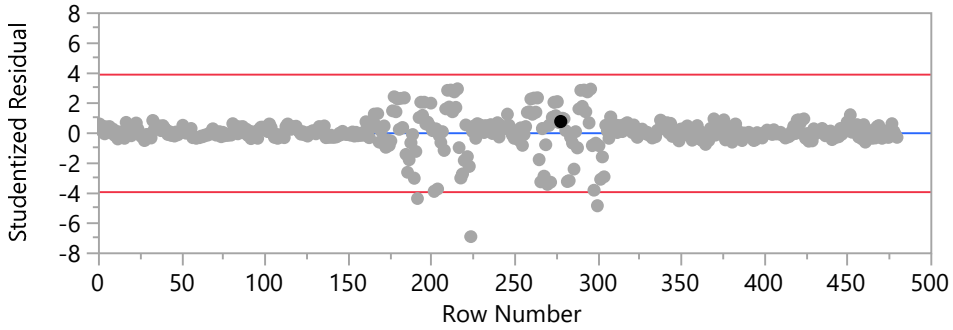
Term	Estimate	Std Error	t Ratio	Prob>|t|
N_false	0.0418287	0.015292	2.74	0.0065*
corp level	-1.002383	0.153101	-6.55	<.0001*
(Search_Velocity-7.5)*(Search_Velocity-7.5)	-0.053156	0.09359	-0.57	0.5703
(Search_Velocity-7.5)*(detect_real-0.775)	-0.111167	0.447236	-0.25	0.8038
(Search_Velocity-7.5)*(detect_false-0.775)	0.2084167	0.447236	0.47	0.6414
(detect_real-0.775)*(detect_false-0.775)	2.8291667	8.944713	0.32	0.7519
(Search_Velocity-7.5)*(N_real-10)	-0.017352	0.006212	-2.79	0.0054*
(detect_false-0.775)*(N_real-10)	-0.008669	0.124232	-0.07	0.9444
(N_real-10)*(N_real-10)	-0.005264	0.007221	-0.73	0.4664
(Search_Velocity-7.5)*(N_false-10)	-0.011037	0.006212	-1.78	0.0763
(detect_real-0.775)*(N_false-10)	-0.007812	0.124232	-0.06	0.9499
(N_real-10)*(N_false-10)	-0.002814	0.001725	-1.63	0.1036
(Search_Velocity-7.5)*corp level	0.0188371	0.067423	0.28	0.7801
(detect_real-0.775)*corp level	-0.129773	1.348466	-0.10	0.9234
(detect_false-0.775)*corp level	-0.510303	1.348466	-0.38	0.7053
(N_real-10)*corp level	0.0280871	0.018729	1.50	0.1344
(N_false-10)*corp level	0.0237069	0.018729	1.27	0.2062
corp level*corp level	2.5976328	0.265178	9.80	<.0001*

Response Percent Covered

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	106	1006.368	9.4940	0.2959
Pure Error	351	11261.965	32.0854	**Prob > F**
Total Error	457	12268.332		1.0000
				Max RSq
				0.2102

Studentized Residuals



Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	108.09409	2.538846	42.58	<.0001*
Search_Velocity	-0.282361	0.104147	-2.71	0.0070*
detect_real	-0.386792	2.082941	-0.19	0.8528
detect_false	1.2701834	2.082941	0.61	0.5423
N_real	-0.080376	0.02893	-2.78	0.0057*
N_false	-0.053136	0.02893	-1.84	0.0669
corp level	0.0230602	0.289641	0.08	0.9366
(Search_Velocity-7.5)*(Search_Velocity-7.5)	-0.015661	0.177057	-0.09	0.9296

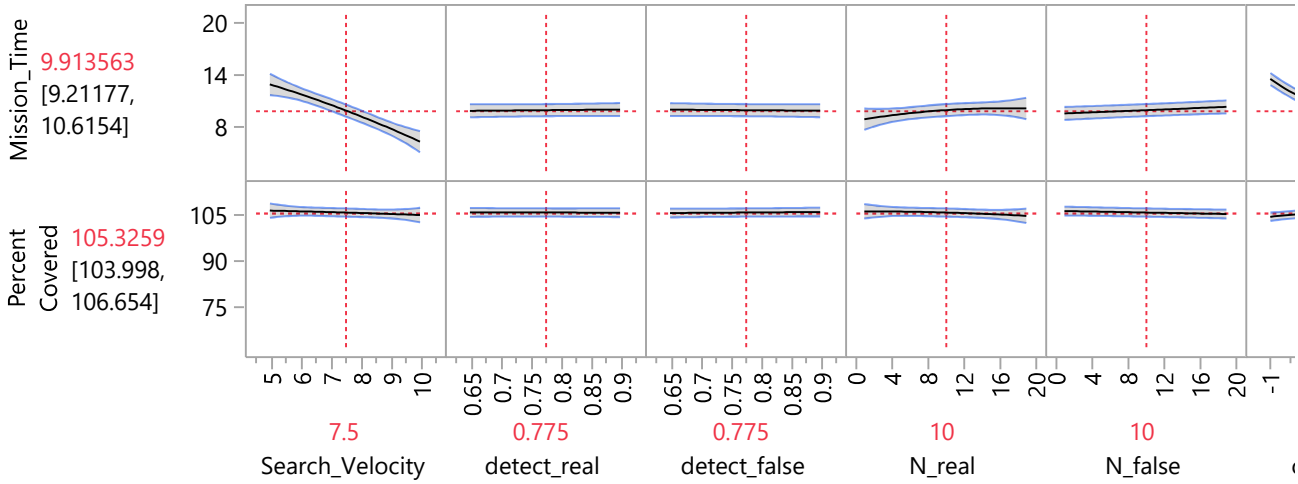
Least Squares Fit

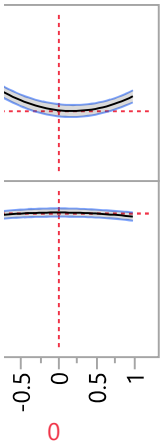
Response Percent Covered

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
(Search_Velocity-7.5)*(detect_real-0.775)	0.4032906	0.846095	0.48	0.6338
(Search_Velocity-7.5)*(detect_false-0.775)	-0.381189	0.846095	-0.45	0.6525
(detect_real-0.775)*(detect_false-0.775)	-13.07944	16.9219	-0.77	0.4400
(Search_Velocity-7.5)*(N_real-10)	-0.01107	0.011751	-0.94	0.3467
(detect_false-0.775)*(N_real-10)	-0.177346	0.235026	-0.75	0.4509
(N_real-10)*(N_real-10)	-0.004143	0.013662	-0.30	0.7619
(Search_Velocity-7.5)*(N_false-10)	-0.014542	0.011751	-1.24	0.2165
(detect_real-0.775)*(N_false-10)	-0.178255	0.235026	-0.76	0.4486
(N_real-10)*(N_false-10)	0.0021022	0.003264	0.64	0.5199
(Search_Velocity-7.5)*corp level	-0.592748	0.127554	-4.65	<.0001*
(detect_real-0.775)*corp level	-1.107204	2.551072	-0.43	0.6645
(detect_false-0.775)*corp level	1.5236683	2.551072	0.60	0.5506
(N_real-10)*corp level	-0.145938	0.035432	-4.12	<.0001*
(N_false-10)*corp level	-0.067925	0.035432	-1.92	0.0559
corp level*corp level	-1.332831	0.501673	-2.66	0.0082*

Prediction Profiler





corp level

Least Squares Fit

Effect Summary

Source	LogWorth	PValue
Search_Velocity	212.456	0.00000
N_false	193.339	0.00000
search_area	155.210	0.00000
corp level*corp level	82.585	0.00000
Search_Velocity*search_area	65.186	0.00000
N_real	53.814	0.00000
corp level	45.994	0.00000 ^
search_area*corp level	34.464	0.00000
N_real*N_false	21.617	0.00000
detect_false	20.730	0.00000
detect_real	17.277	0.00000
detect_false*N_false	12.697	0.00000
Search_Velocity*N_real	10.780	0.00000
Search_Velocity*corp level	10.219	0.00000
search_area*N_real	8.560	0.00000
N_real*corp level	8.252	0.00000
N_false*corp level	5.702	0.00000
Search_Velocity*N_false	4.862	0.00001
detect_real*N_false	3.433	0.00037
detect_real*N_real	2.927	0.00118
search_area*N_false	2.554	0.00279
search_area*search_area	2.097	0.00801
detect_false*N_real	1.799	0.01588
N_real*N_real	1.442	0.03615
N_false*N_false	0.862	0.13733
detect_false*detect_false	0.792	0.16151
detect_false*search_area	0.766	0.17141
detect_false*corp level	0.724	0.18895
detect_real*detect_false	0.694	0.20250
detect_real*detect_real	0.665	0.21612
Search_Velocity*detect_false	0.603	0.24944
Search_Velocity*Search_Velocity	0.484	0.32785
Search_Velocity*detect_real	0.408	0.39101
detect_real*corp level	0.275	0.53134
detect_real*search_area	0.200	0.63065

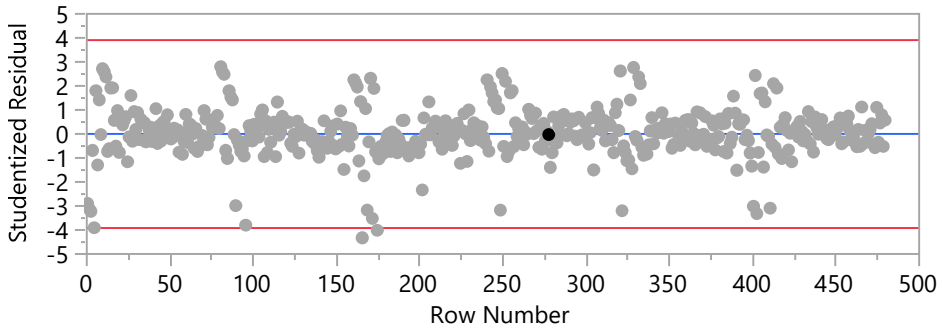
Least Squares Fit

Response Percent_Correct_Detected

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	195	56134.47	287.869	0.7520
Pure Error	249	95314.69	382.790	**Prob > F**
Total Error	444	151449.16		0.9814
				Max RSq
				0.4781

Studentized Residuals



Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	10.305325	10.01871	1.03	0.3042
Search_Velocity	0.1023603	0.37124	0.28	0.7829
detect_real	29.441522	7.424793	3.97	<.0001*
detect_false	43.475704	7.424793	5.86	<.0001*
search_area	-0.011194	0.007425	-1.51	0.1323
N_real	-0.055185	0.103122	-0.54	0.5928
N_false	-0.035686	0.103122	-0.35	0.7295
corp level	0.7449319	1.032445	0.72	0.4710
(Search_Velocity-7.5)*(Search_Velocity-7.5)	0.4287897	0.781341	0.55	0.5834
(Search_Velocity-7.5)*(detect_real-0.775)	1.4668492	3.015965	0.49	0.6270
(detect_real-0.775)*(detect_real-0.775)	91.5159	312.5363	0.29	0.7698
(Search_Velocity-7.5)*(detect_false-0.775)	-2.983167	3.015965	-0.99	0.3231
(detect_real-0.775)*(detect_false-0.775)	-10.26892	60.3193	-0.17	0.8649
(detect_false-0.775)*(detect_false-0.775)	164.49836	312.5363	0.53	0.5989
(Search_Velocity-7.5)*(search_area-575)	-0.001714	0.003016	-0.57	0.5700
(detect_real-0.775)*(search_area-575)	-0.023386	0.060319	-0.39	0.6984
(detect_false-0.775)*(search_area-575)	-0.076872	0.060319	-1.27	0.2032
(search_area-575)*(search_area-575)	-0.000202	0.000313	-0.65	0.5188
(Search_Velocity-7.5)*(N_real-10)	-4.965e-5	0.041888	-0.00	0.9991
(detect_real-0.775)*(N_real-10)	2.734648	0.837768	3.26	0.0012*
(detect_false-0.775)*(N_real-10)	-2.028278	0.837768	-2.42	0.0159*
(search_area-575)*(N_real-10)	0.0001708	0.000838	0.20	0.8386
(N_real-10)*(N_real-10)	0.0279692	0.060289	0.46	0.6429
(Search_Velocity-7.5)*(N_false-10)	-0.015747	0.041888	-0.38	0.7072
(detect_real-0.775)*(N_false-10)	-1.792136	0.837768	-2.14	0.0330*
(detect_false-0.775)*(N_false-10)	1.547349	0.837768	1.85	0.0654
(search_area-575)*(N_false-10)	0.0015931	0.000838	1.90	0.0579
(N_real-10)*(N_false-10)	0.0027363	0.011636	0.24	0.8142
(N_false-10)*(N_false-10)	-0.054202	0.060289	-0.90	0.3691
(Search_Velocity-7.5)*corp level	-0.146783	0.454674	-0.32	0.7470
(detect_real-0.775)*corp level	-0.431546	9.093477	-0.05	0.9622

Least Squares Fit

Response Percent_Correct_Detected

Parameter Estimates

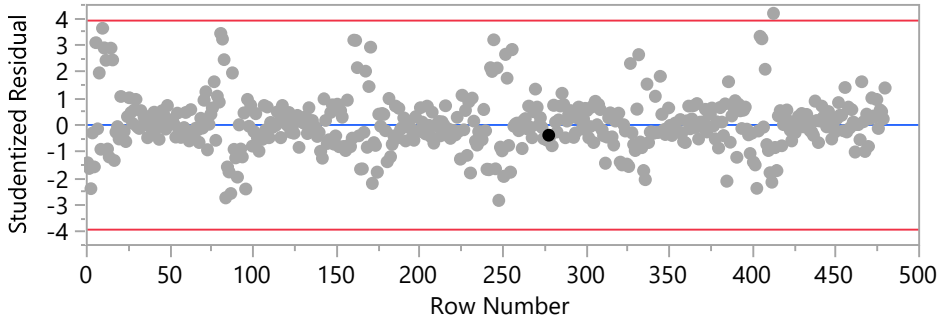
Term	Estimate	Std Error	t Ratio	Prob>|t|
(detect_false-0.775)*corp level	-2.444293	9.093477	-0.27	0.7882
(search_area-575)*corp level	-0.008918	0.009093	-0.98	0.3273
(N_real-10)*corp level	-0.141188	0.126298	-1.12	0.2642
(N_false-10)*corp level	-0.094075	0.126298	-0.74	0.4567
corp level*corp level	2.1280093	1.788247	1.19	0.2347

Response Percent_Correct_Confirmations

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	195	74866.09	383.929	0.7101
Pure Error	249	134626.39	540.668	**Prob > F**
Total Error	444	209492.48		0.9938
				Max RSq
				0.7307

Studentized Residuals



Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-30.69702	11.78318	-2.61	0.0095*
Search_Velocity	0.1672568	0.436622	0.38	0.7019
detect_real	78.851749	8.73243	9.03	<.0001*
detect_false	20.756791	8.73243	2.38	0.0179*
search_area	-0.015076	0.008732	-1.73	0.0850
N_real	2.1763319	0.121284	17.94	<.0001*
N_false	-1.500359	0.121284	-12.37	<.0001*
corp level	-0.351956	1.214277	-0.29	0.7721
(Search_Velocity-7.5)*(Search_Velocity-7.5)	-0.438266	0.918949	-0.48	0.6337
(Search_Velocity-7.5)*(detect_real-0.775)	2.1582879	3.54713	0.61	0.5432
(detect_real-0.775)*(detect_real-0.775)	394.68506	367.5795	1.07	0.2835
(Search_Velocity-7.5)*(detect_false-0.775)	-0.34846	3.54713	-0.10	0.9218
(detect_real-0.775)*(detect_false-0.775)	6.812989	70.9426	0.10	0.9235
(detect_false-0.775)*(detect_false-0.775)	167.97103	367.5795	0.46	0.6479
(Search_Velocity-7.5)*(search_area-575)	0.001923	0.003547	0.54	0.5880
(detect_real-0.775)*(search_area-575)	0.0247355	0.070943	0.35	0.7275
(detect_false-0.775)*(search_area-575)	-0.056116	0.070943	-0.79	0.4294
(search_area-575)*(search_area-575)	-0.000264	0.000368	-0.72	0.4724
(Search_Velocity-7.5)*(N_real-10)	-0.041014	0.049266	-0.83	0.4056
(detect_real-0.775)*(N_real-10)	2.1632823	0.985314	2.20	0.0286*
(detect_false-0.775)*(N_real-10)	0.9235647	0.985314	0.94	0.3491

Least Squares Fit

Response Percent_Correct_Confirmations

Parameter Estimates

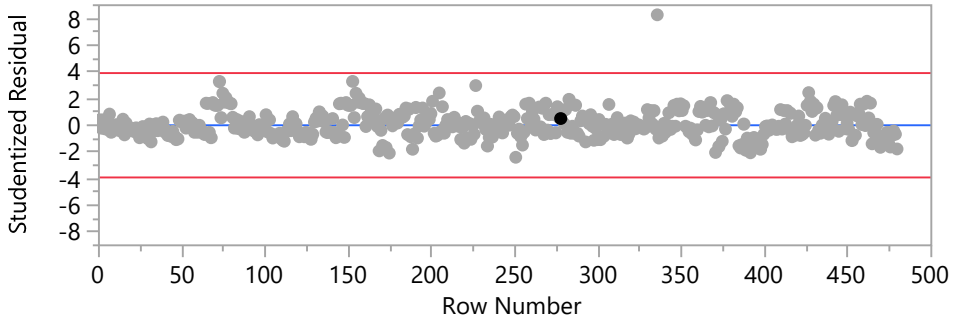
Term	Estimate	Std Error	t Ratio	Prob>|t|
(search_area-575)*(N_real-10)	0.001998	0.000985	2.03	0.0432*
(N_real-10)*(N_real-10)	-0.149019	0.070907	-2.10	0.0361*
(Search_Velocity-7.5)*(N_false-10)	-0.043457	0.049266	-0.88	0.3782
(detect_real-0.775)*(N_false-10)	-3.536018	0.985314	-3.59	0.0004*
(detect_false-0.775)*(N_false-10)	1.4816632	0.985314	1.50	0.1334
(search_area-575)*(N_false-10)	0.0023399	0.000985	2.37	0.0180*
(N_real-10)*(N_false-10)	0.0215819	0.013685	1.58	0.1155
(N_false-10)*(N_false-10)	0.0568206	0.070907	0.80	0.4234
(Search_Velocity-7.5)*corp level	-0.400772	0.53475	-0.75	0.4540
(detect_real-0.775)*corp level	2.1683772	10.695	0.20	0.8394
(detect_false-0.775)*corp level	-6.810244	10.695	-0.64	0.5246
(search_area-575)*corp level	-0.013367	0.010695	-1.25	0.2120
(N_real-10)*corp level	0.0283959	0.148542	0.19	0.8485
(N_false-10)*corp level	0.0125477	0.148542	0.08	0.9327
corp level*corp level	2.6914355	2.103189	1.28	0.2013

Response Mission_Time

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	195	444.96152	2.28185	5.7477
Pure Error	249	98.85410	0.39700	**Prob > F**
Total Error	444	543.81562		<.0001*
				Max RSq
				0.9892

Studentized Residuals



Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	7.9545497	0.60035	13.25	<.0001*
Search_Velocity	-1.319778	0.022246	-59.33	<.0001*
detect_real	0.5943434	0.444915	1.34	0.1823
detect_false	-0.496818	0.444915	-1.12	0.2647
search_area	0.0185881	0.000445	41.78	<.0001*
N_real	0.0683207	0.006179	11.06	<.0001*
N_false	0.0418287	0.006179	6.77	<.0001*
corp level	-1.002383	0.061867	-16.20	<.0001*
(Search_Velocity-7.5)*(Search_Velocity-7.5)	-0.045862	0.04682	-0.98	0.3278
(Search_Velocity-7.5)*(detect_real-0.775)	-0.111167	0.180725	-0.62	0.5388
(detect_real-0.775)*(detect_real-0.775)	6.3217988	18.72806	0.34	0.7359

Least Squares Fit

Response Mission_Time

Parameter Estimates

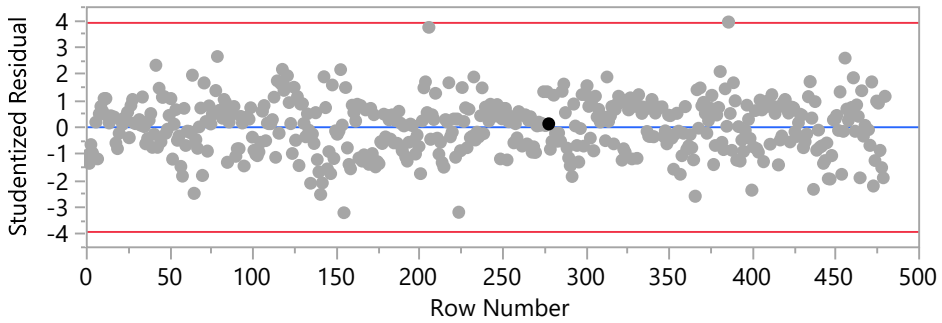
Term	Estimate	Std Error	t Ratio	Prob>|t|
(Search_Velocity-7.5)*(detect_false-0.775)	0.2084167	0.180725	1.15	0.2494
(detect_real-0.775)*(detect_false-0.775)	2.8291667	3.614504	0.78	0.4342
(detect_false-0.775)*(detect_false-0.775)	13.801799	18.72806	0.74	0.4615
(Search_Velocity-7.5)*(search_area-575)	-0.003693	0.000181	-20.44	<.0001*
(detect_real-0.775)*(search_area-575)	0.0011667	0.003615	0.32	0.7470
(detect_false-0.775)*(search_area-575)	0.0015683	0.003615	0.43	0.6646
(search_area-575)*(search_area-575)	-1.445e-5	1.873e-5	-0.77	0.4407
(Search_Velocity-7.5)*(N_real-10)	-0.017352	0.00251	-6.91	<.0001*
(detect_real-0.775)*(N_real-10)	-0.014294	0.050201	-0.28	0.7760
(detect_false-0.775)*(N_real-10)	-0.008669	0.050201	-0.17	0.8630
(search_area-575)*(N_real-10)	8.9514e-5	5.02e-5	1.78	0.0753
(N_real-10)*(N_real-10)	-0.004701	0.003613	-1.30	0.1938
(Search_Velocity-7.5)*(N_false-10)	-0.011037	0.00251	-4.40	<.0001*
(detect_real-0.775)*(N_false-10)	-0.007812	0.050201	-0.16	0.8764
(detect_false-0.775)*(N_false-10)	-0.045382	0.050201	-0.90	0.3665
(search_area-575)*(N_false-10)	0.0000801	5.02e-5	1.60	0.1113
(N_real-10)*(N_false-10)	-0.002814	0.000697	-4.04	<.0001*
(N_false-10)*(N_false-10)	-0.002338	0.003613	-0.65	0.5179
(Search_Velocity-7.5)*corp level	0.0188371	0.027245	0.69	0.4897
(detect_real-0.775)*corp level	-0.129773	0.544907	-0.24	0.8119
(detect_false-0.775)*corp level	-0.510303	0.544907	-0.94	0.3495
(search_area-575)*corp level	0.0073757	0.000545	13.54	<.0001*
(N_real-10)*corp level	0.0280871	0.007568	3.71	0.0002*
(N_false-10)*corp level	0.0237069	0.007568	3.13	0.0018*
corp level*corp level	2.5976328	0.107157	24.24	<.0001*

Response Perception_Accuracy

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	195	8129.944	41.6920	1.1819
Pure Error	249	8783.261	35.2741	**Prob > F**
Total Error	444	16913.205		0.1068
				Max RSq
				0.9389

Studentized Residuals



Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Least Squares Fit

Response Perception_Accuracy

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	75.968288	3.348046	22.69	<.0001*
Search_Velocity	-0.087734	0.124061	-0.71	0.4798
detect_real	-6.29421	2.481212	-2.54	0.0115*
detect_false	24.87312	2.481212	10.02	<.0001*
search_area	0.0013047	0.002481	0.53	0.5993
N_real	0.4250571	0.034461	12.33	<.0001*
N_false	-1.825802	0.034461	-52.98	<.0001*
corp level	2.1140586	0.345022	6.13	<.0001*
(Search_Velocity-7.5)*(Search_Velocity-7.5)	0.0766012	0.261108	0.29	0.7694
(Search_Velocity-7.5)*(detect_real-0.775)	-0.865391	1.007873	-0.86	0.3910
(detect_real-0.775)*(detect_real-0.775)	129.37136	104.4432	1.24	0.2161
(Search_Velocity-7.5)*(detect_false-0.775)	0.330026	1.007873	0.33	0.7435
(detect_real-0.775)*(detect_false-0.775)	25.727862	20.15746	1.28	0.2025
(detect_false-0.775)*(detect_false-0.775)	146.46561	104.4432	1.40	0.1615
(Search_Velocity-7.5)*(search_area-575)	-0.000577	0.001008	-0.57	0.5675
(detect_real-0.775)*(search_area-575)	0.0096985	0.020157	0.48	0.6307
(detect_false-0.775)*(search_area-575)	0.0276137	0.020157	1.37	0.1714
(search_area-575)*(search_area-575)	-0.000278	0.000104	-2.66	0.0080*
(Search_Velocity-7.5)*(N_real-10)	-0.014457	0.013998	-1.03	0.3023
(detect_real-0.775)*(N_real-10)	0.0572896	0.279965	0.20	0.8380
(detect_false-0.775)*(N_real-10)	0.0464394	0.279965	0.17	0.8683
(search_area-575)*(N_real-10)	-0.000066	0.00028	-0.24	0.8139
(N_real-10)*(N_real-10)	-0.005237	0.020147	-0.26	0.7950
(Search_Velocity-7.5)*(N_false-10)	-0.020456	0.013998	-1.46	0.1446
(detect_real-0.775)*(N_false-10)	-0.077543	0.279965	-0.28	0.7819
(detect_false-0.775)*(N_false-10)	2.1226407	0.279965	7.58	<.0001*
(search_area-575)*(N_false-10)	-3.284e-5	0.00028	-0.12	0.9067
(N_real-10)*(N_false-10)	0.0399338	0.003888	10.27	<.0001*
(N_false-10)*(N_false-10)	0.029989	0.020147	1.49	0.1373
(Search_Velocity-7.5)*corp level	0.0063703	0.151943	0.04	0.9666
(detect_real-0.775)*corp level	0.9982227	3.038852	0.33	0.7427
(detect_false-0.775)*corp level	-3.998226	3.038852	-1.32	0.1890
(search_area-575)*corp level	0.0021733	0.003039	0.72	0.4749
(N_real-10)*corp level	0.0101629	0.042206	0.24	0.8098
(N_false-10)*corp level	0.2033815	0.042206	4.82	<.0001*
corp level*corp level	2.0258798	0.597595	3.39	0.0008*

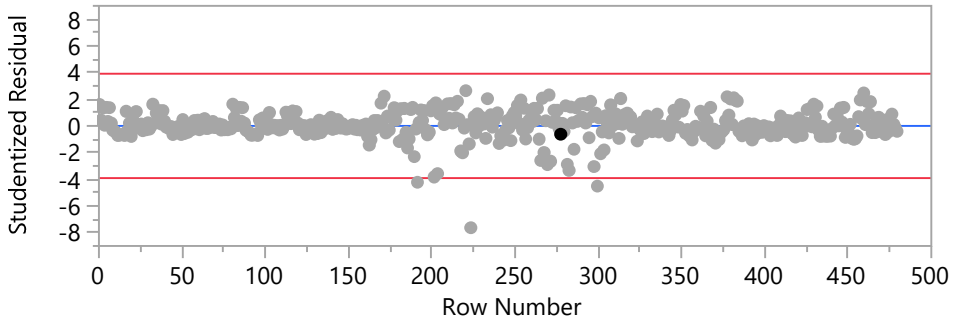
Least Squares Fit

Response Percent Covered

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	195	4526.8927	23.2148	4.8396
Pure Error	249	1194.4103	4.7968	**Prob > F**
Total Error	444	5721.3029		<.0001*
				Max RSq
				0.9162

Studentized Residuals



Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	121.65461	1.947269	62.47	<.0001*
Search_Velocity	-0.282361	0.072155	-3.91	0.0001*
detect_real	-0.386792	1.443106	-0.27	0.7888
detect_false	1.2701834	1.443106	0.88	0.3792
search_area	-0.023578	0.001443	-16.34	<.0001*
N_real	-0.080376	0.020043	-4.01	<.0001*
N_false	-0.053136	0.020043	-2.65	0.0083*
corp level	0.0230602	0.200669	0.11	0.9086
(Search_Velocity-7.5)*(Search_Velocity-7.5)	-0.012582	0.151864	-0.08	0.9340
(Search_Velocity-7.5)*(detect_real-0.775)	0.4032906	0.586193	0.69	0.4918
(detect_real-0.775)*(detect_real-0.775)	18.544354	60.74556	0.31	0.7603
(Search_Velocity-7.5)*(detect_false-0.775)	-0.381189	0.586193	-0.65	0.5158
(detect_real-0.775)*(detect_false-0.775)	-13.07944	11.72385	-1.12	0.2652
(detect_false-0.775)*(detect_false-0.775)	-23.8997	60.74556	-0.39	0.6942
(Search_Velocity-7.5)*(search_area-575)	-0.003042	0.000586	-5.19	<.0001*
(detect_real-0.775)*(search_area-575)	-0.003833	0.011724	-0.33	0.7438
(detect_false-0.775)*(search_area-575)	0.0080205	0.011724	0.68	0.4943
(search_area-575)*(search_area-575)	-8.304e-6	6.075e-5	-0.14	0.8913
(Search_Velocity-7.5)*(N_real-10)	-0.01107	0.008142	-1.36	0.1746
(detect_real-0.775)*(N_real-10)	-0.011898	0.162831	-0.07	0.9418
(detect_false-0.775)*(N_real-10)	-0.177346	0.162831	-1.09	0.2767
(search_area-575)*(N_real-10)	-0.000988	0.000163	-6.07	<.0001*
(N_real-10)*(N_real-10)	-0.003905	0.011718	-0.33	0.7391
(Search_Velocity-7.5)*(N_false-10)	-0.014542	0.008142	-1.79	0.0748
(detect_real-0.775)*(N_false-10)	-0.178255	0.162831	-1.09	0.2742
(detect_false-0.775)*(N_false-10)	0.0026738	0.162831	0.02	0.9869
(search_area-575)*(N_false-10)	-0.00049	0.000163	-3.01	0.0028*
(N_real-10)*(N_false-10)	0.0021022	0.002262	0.93	0.3531
(N_false-10)*(N_false-10)	0.0021101	0.011718	0.18	0.8572
(Search_Velocity-7.5)*corp level	-0.592748	0.088372	-6.71	<.0001*
(detect_real-0.775)*corp level	-1.107204	1.767437	-0.63	0.5313

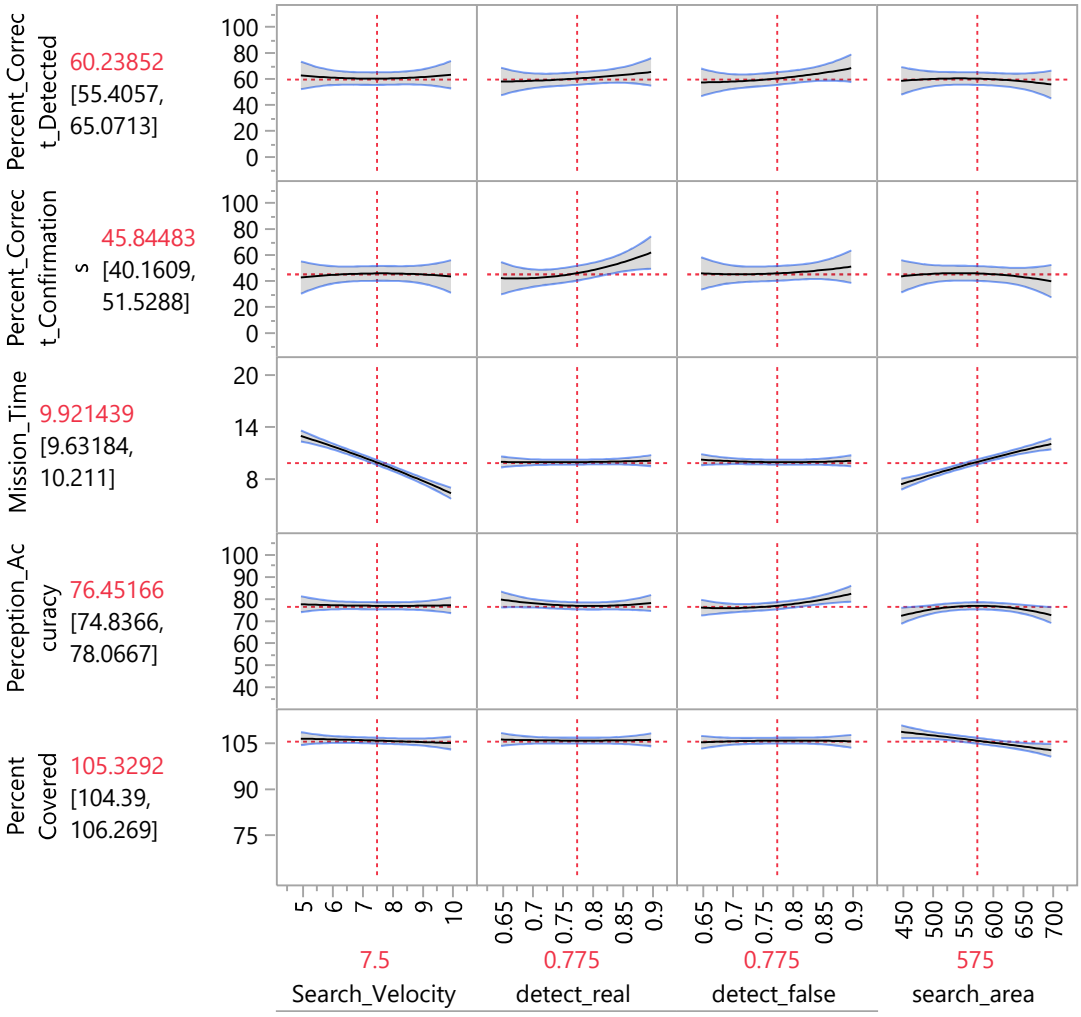
Least Squares Fit

Response Percent Covered

Parameter Estimates

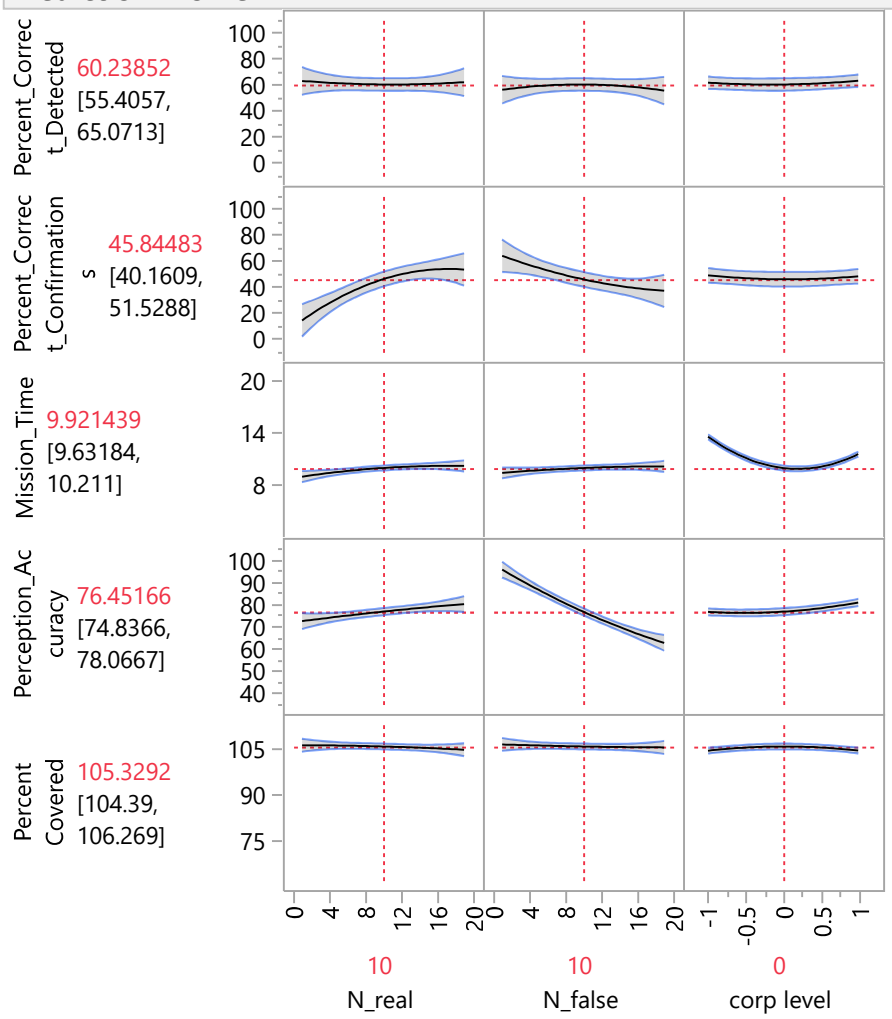
Term	Estimate	Std Error	t Ratio	Prob>|t|
(detect_false-0.775)*corp level	1.5236683	1.767437	0.86	0.3891
(search_area-575)*corp level	-0.022875	0.001767	-12.94	<.0001*
(N_real-10)*corp level	-0.145938	0.024548	-5.95	<.0001*
(N_false-10)*corp level	-0.067925	0.024548	-2.77	0.0059*
corp level*corp level	-1.332831	0.347569	-3.83	0.0001*

Prediction Profiler



Least Squares Fit

Prediction Profiler



Appendix M. Responsiveness: One Vehicle Operation

Response Max

Whole Model

Actual by Predicted Plot

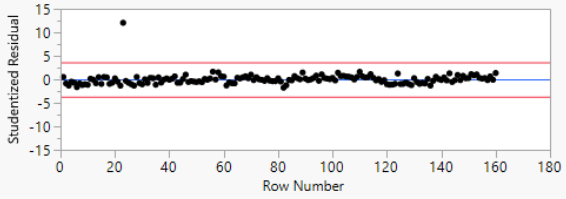
Effect Summary

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	1	3.8141e-15	3.814e-15	3.4092
Pure Error	157	1.7564e-13	1.119e-15	Prob > F
Total Error	158	1.7946e-13		0.0667
				Max RSq
				0.0675

Residual by Predicted Plot

Studentized Residuals



Externally Studentized Residuals with 95% Simultaneous Limits (Bonferroni)

Summary of Fit

RSquare	0.04726
RSquare Adj	0.04123
Root Mean Square Error	3.37e-8
Mean of Response	4.07e-7
Observations (or Sum Wgts)	160

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	1	8.9019e-15	8.902e-15	7.8375
Error	158	1.7946e-13	1.136e-15	Prob > F
C. Total	159	1.8836e-13		0.0058*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	4.4478e-7	1.375e-8	32.34	<.0001*
search_area	-6.57e-11	2.35e-11	-2.80	0.0058*

Figure 40. Maximum responsiveness model for single vehicle operation

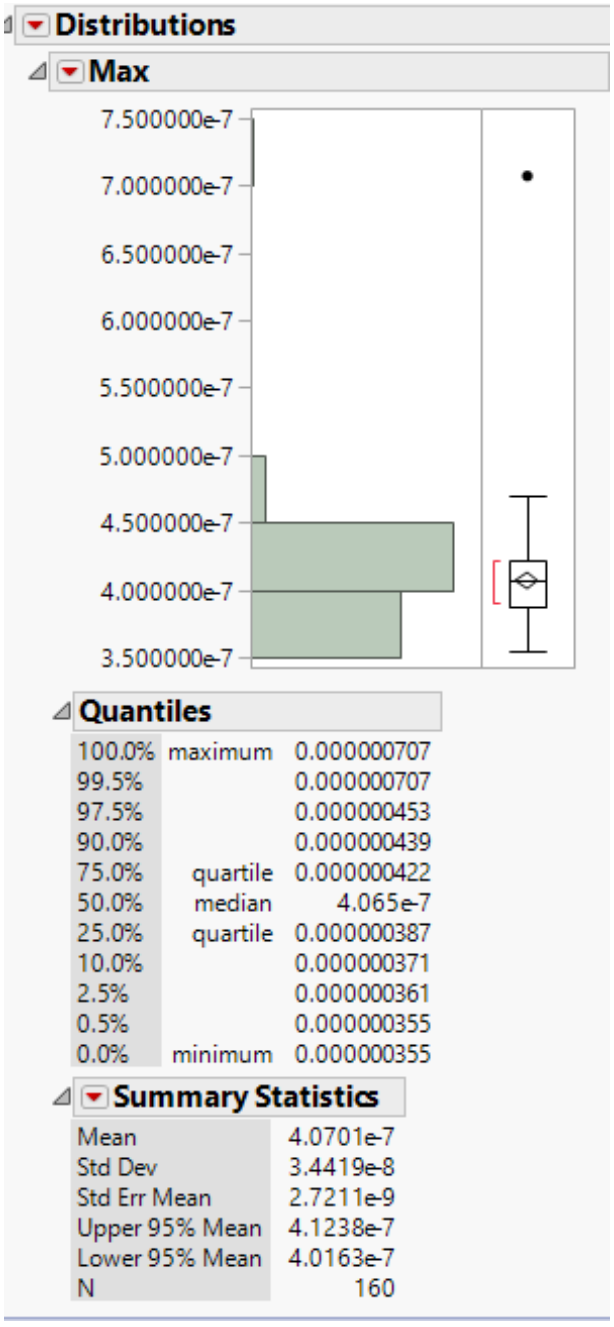


Figure 41. Distribution and descriptive statistics for maximum responsiveness for one vehicle operation

Appendix N. Responsiveness: Two Vehicle Operation

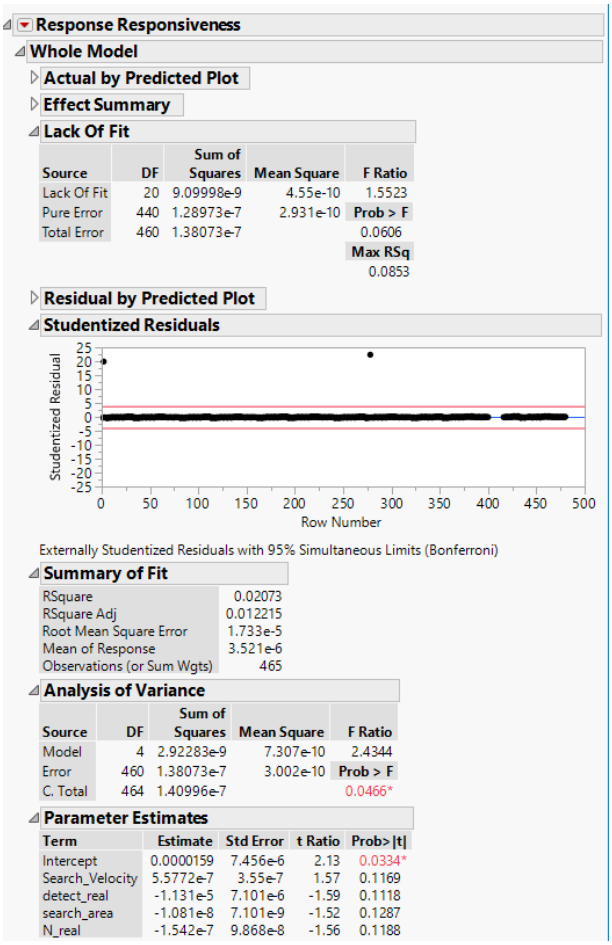


Figure 42. Maximum responsiveness model for two vehicle operation

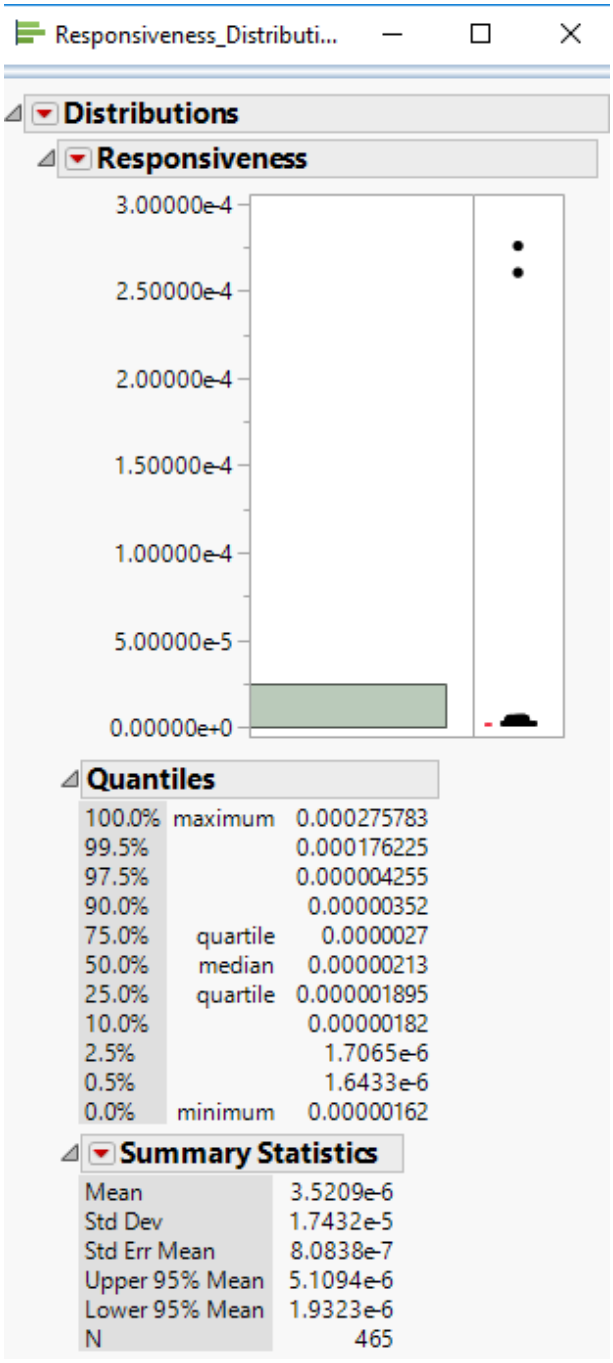


Figure 43. Distribution and descriptive statistics for maximum responsiveness for two vehicle operation

Appendix O. Response Surface Optimization Code

File - RunOptimization.py

```
from Aframe_Test.Optimization.ResponseSurfaces import *
from Aframe_Test.Optimization.DesirabilityConfig import DesirabilityConfig
from Aframe_Test.Optimization.Desirability import *

#all
# cooperation_space = np.array([-1, 0, 1])
# NReal_space = np.arange(1,19,1)
# NFalse_space = np.arange(1,19,1)
# Boundary_space = np.arange(450,700,1)
# DetectFalse_space = np.arange(0.65,0.9,.01)
# DetectReal_space = np.arange(0.65,0.9,.01)
# SearchVelocity_space = np.arange(5,10,1)
# desirability_max = 0

#Low density:
# cooperation_space = np.array([-1, 0, 1])
# NReal_space = [1] #np.arange(1,19,1)
# NFalse_space = [1] #np.arange(1,19,1)
# Boundary_space = [450] #np.arange(450,700,1)
# DetectFalse_space = np.arange(0.65,0.9,.01)
# DetectReal_space = np.arange(0.65,0.9,.01)
# SearchVelocity_space = np.arange(5,10,1)
# desirability_max = 0


#medium density:
# cooperation_space = np.array([-1, 0, 1])
# NReal_space = [10] #np.arange(1,19,1)
# NFalse_space = [10] #np.arange(1,19,1)
# Boundary_space = [700] #np.arange(450,700,1)
# DetectFalse_space = np.arange(0.65,0.9,.01)
# DetectReal_space = np.arange(0.65,0.9,.01)
# SearchVelocity_space = np.arange(5,10,1)
# desirability_max = 0
#
# #high density:
cooperation_space = np.array([-1, 0, 1])
NReal_space = [19] #np.arange(1,19,1)
NFalse_space = [19] #np.arange(1,19,1)
Boundary_space = [700] #np.arange(450,700,1)
DetectFalse_space = np.arange(0.65,0.9,.01)
DetectReal_space = np.arange(0.65,0.9,.01)
SearchVelocity_space = np.arange(5,10,1)
desirability_max = 0

desirability_data_init = pd.DataFrame(data={'cooperation':[], 'NReal': [], 'NFalse':[], 'Boundary
':[], 'SearchVelocity':[], 'DetectFalse':[], 'DetectReal':[], 'desirability':[]})

for n in cooperation_space:
    for NReal in NReal_space:
        for NFalse in NFalse_space:
            for Boundary in Boundary_space:
                for Velocity in SearchVelocity_space:
                    for DetFalse in DetectFalse_space:
                        for DetReal in DetectReal_space:
                            surface = ResponseSurfaces(SearchVelocity=Velocity, DetectReal=
DetReal, DetectFalse=DetFalse,
                                                       SearchBoundary=Boundary, NReal=NReal,
NFalse=NFalse, CooperationLevel=n)
                            surface.get_missionTime()
                            surface.get_detect()  # this model is working well
                            surface.get_confirm()
                            surface.get_perceptionAccuracy()
                            surface.get_covered()
                            predictMissionTime = surface.missionTime
                            predictDetect = surface.detect
                            predictConfirm = surface.confirm
                            predictPreception = surface.perceptionAccuracy
                            predictCovered = surface.covered
                            desirability = Desirability()
                            desirability.get_desirability(ResponseTime=predictMissionTime,
ResponseArea=predictCovered, ResponsePercep=predictPreception, ResponseDetect=predictDetect,
ResponseConfirm=predictConfirm)
                            desirability_N = desirability.desirability
                            if desirability_N > 0.35:
```

File - RunOptimization.py

```
                                desirability_dict = {'cooperation':[n],'NReal': [NReal], 'NFalse
':[NFalse], 'Boundary':[Boundary], 'SearchVelocity':[Velocity], 'DetectFalse':[DetFalse], '
DetectReal':[DetReal], 'desirability':[desirability_N]}
                                desirability_data = desirability_data_init.append(pd.DataFrame(
data=desirability_dict))
                                desirability_data_init = desirability_data

                            if desirability_N > desirability_max:
                                desirability_max = desirability_N
                                print(desirability_max)
                            #if desireability calculated is > previous, save settings


print(desirability_max)
desirability_data_init.to_csv('desirability_output', index=False)


```

File - ResponseSurface.py

```
import numpy as np
import pandas as pd


class ResponseSurfaces:
    def __init__(self, SearchVelocity=10, DetectReal=0.9, DetectFalse=0.9, NReal=19, NFalse=1, CooperationLevel=0, SearchBoundary = None):
        self.SearchVelocity = SearchVelocity
        self.DetectReal = DetectReal
        self.DetectFalse = DetectFalse
        self.NReal = NReal
        self.Nfalse = NFalse
        self.CooperationLevel = CooperationLevel
        self.SearchBoundary = SearchBoundary
        self.detect = None

    def get_detect(self):
        self.detect = None

        Search_Velocity = self.SearchVelocity
        detect_real = self.DetectReal
        detect_false = self.DetectFalse
        search_area = self.SearchBoundary
        N_real = self.NReal
        N_false = self.Nfalse
        corp_level = self.CooperationLevel


        self.detect = 10.5042256245727 + 0.102659851825734 * Search_Velocity + 29.44240254977 * detect_real + 43.480692831673 * detect_false + -0.0111759668600259 * search_area + -0.0548972680538209 * N_real + -0.0354941682435795 * N_false + 0.682433756842006 * corp_level + (Search_Velocity - 7.5) * ((Search_Velocity - 7.5) * 0.424399082578523) + (Search_Velocity - 7.5) * ((detect_real - 0.774999999999995) * 1.46684917075) + (detect_real - 0.774999999999995) * ((detect_real - 0.774999999999995) * 89.7596330313833) + (Search_Velocity - 7.5) * ((detect_false - 0.774999999999995) * -2.98316742941667) + (detect_real - 0.774999999999995) * ((detect_false - 0.774999999999995) * -10.268921325) + (detect_false - 0.774999999999995) * ((detect_false - 0.774999999999995) * 162.742089191384) + (Search_Velocity - 7.5) * ((search_area - 575) * -0.00171435612775) + (detect_real - 0.774999999999995) * ((search_area - 575) * -0.023386355265) + (detect_false - 0.774999999999995) * ((search_area - 575) * -0.0768717738616667) + (search_area - 575) * ((search_area - 575) * -0.000203573700301898) + (Search_Velocity - 7.5) * ((N_real - 10) * -0.0000496457743055491) + (detect_real - 0.774999999999995) * ((N_real - 10) * 2.73464797891204) + (detect_false - 0.774999999999995) * ((N_real - 10) * -2.02827832409722) + (search_area - 575) * ((N_real - 10) * 0.000170769088773148) + (N_real - 10) * ((N_real - 10) * 0.0276304535837291) + (Search_Velocity - 7.5) * ((N_false - 10) * -0.0157465503159722) + (detect_real - 0.774999999999995) * ((N_false - 10) * -1.79213603775463) + (detect_false - 0.774999999999995) * ((N_false - 10) * 1.54734901405093) + (search_area - 575) * ((N_false - 10) * 0.00159307747821759) + (N_real - 10) * ((N_false - 10) * 0.00273632533789864) + (N_false - 10) * ((N_false - 10) * -0.0545404619100985) + (Search_Velocity - 7.5) * ((corp_level - (-0.00204081632653061)) * -0.146782607030304) + (detect_real - 0.774999999999995) * ((corp_level - (-0.00204081632653061)) * -0.431545510909137) + (detect_false - 0.774999999999995) * ((corp_level - (-0.00204081632653061)) * -2.4442925184849) + (search_area - 575) * ((corp_level - (-0.00204081632653061)) * -0.00891785011636377) + (N_real - 10) * ((corp_level - (-0.00204081632653061)) * -0.141187967041245) + (N_false - 10) * ((corp_level - (-0.00204081632653061)) * -0.0940747782659928) + (corp_level - (-0.00204081632653061)) * ((corp_level - (-0.00204081632653061)) * 2.0372912980427)


    def get_confirm(self):
        self.confirm = None

        Search_Velocity = self.SearchVelocity
        detect_real = self.DetectReal
        detect_false = self.DetectFalse
        search_area = self.SearchBoundary
        N_real = self.NReal
        N_false = self.Nfalse
        corp_level = self.CooperationLevel

        self.confirm = (-30.6970200318902) + 0.167256755899992 * Search_Velocity + 78.8517485859965 * detect_real + 20.7567914657156 * detect_false + -0.0150763932676564 * search_area + 2.17633192147053 * N_real + -1.50035911936054 * N_false + -0.351955739471856 * corp_level + (Search_Velocity - 7.5) * ((Search_Velocity - 7.5) * -0.438266084712616) + (Search_Velocity - 7.5) * ((detect_real - 0.774999999999996) * 2.158287853525) + (detect_real - 0.774999999999996) * ((detect_real - 0.774999999999996) * 394.685056194954) + (Search_Velocity - 7.5) * ((detect_false - 0.774999999999996) * -0.348459840275001) + (detect_real - 0.774999999999996) * ((detect_false -
```

File - ResponseSurface.py

```
0.774999999999996) * 6.81298895583336) + (detect_false - 0.774999999999996) * ((detect_false - 0.774999999999996) * 167.971029474958) + (Search_Velocity - 7.5) * ((search_area - 575) * 0.001923020331025) + (detect_real - 0.774999999999996) * ((search_area - 575) * 0.0247354553504999) + (detect_false - 0.774999999999996) * ((search_area - 575) * -0.0561157953788333) + (search_area - 575) * ((search_area - 575) * -0.000264387723005038) + (Search_Velocity - 7.5) * ((N_real - 10) * -0.0410140213045139) + (detect_real - 0.774999999999996) * ((N_real - 10) * 2.16328230651157) + (detect_false - 0.774999999999996) * ((N_real - 10) * 0.923564667817129) + (search_area - 575) * ((N_real - 10) * 0.00199800527816898) + (N_real - 10) * ((N_real - 10) * -0.149018698131118) + (Search_Velocity - 7.5) * ((N_false - 10) * -0.0434570194200231) + (detect_real - 0.774999999999996) * ((N_false - 10) * -3.53601790155787) + (detect_false - 0.774999999999996) * ((N_false - 10) * 1.48166318232176) + (search_area - 575) * ((N_false - 10) * 0.00233987999002546) + (N_real - 10) * ((N_false - 10) * 0.0215818878228202) + (N_false - 10) * ((N_false - 10) * 0.0568205838236144) + (Search_Velocity - 7.5) * (corp_level * -0.400772358483333) + (detect_real - 0.774999999999996) * (corp_level * 2.16837723766664) + (detect_false - 0.774999999999996) * (corp_level * -6.8102441180909) + (search_area - 575) * (corp_level * -0.0133665598823333) + (N_real - 10) * (corp_level * 0.0283959077066491) + (N_false - 10) * (corp_level * 0.0125476924869517) + corp_level * (corp_level * 2.69143552192813)


    def get_perceptionAccuracy(self):
        self.perceptionAccuracy = None
        Search_Velocity = self.SearchVelocity
        detect_real = self.DetectReal
        detect_false = self.DetectFalse
        search_area = self.SearchBoundary
        N_real = self.NReal
        N_false = self.Nfalse
        corp_level = self.CooperationLevel


        self.perceptionAccuracy = 75.9682879441665 + -0.0877340315454522 * Search_Velocity + -6.29420962646591 * detect_real + 24.873119933736 * detect_false + 0.00130465794040388 * search_area + 0.425057115563973 * N_real + -1.82580242979238 * N_false + 2.11405859406252 * corp_level + (Search_Velocity - 7.5) * ((Search_Velocity - 7.5) * 0.0766011851646056) + ((Search_Velocity - 7.5) * ((detect_real - 0.774999999999996) * -0.86539111375) + (detect_real - 0.774999999999996) * ((detect_real - 0.774999999999996) * 129.37136355916) + (Search_Velocity - 7.5) * ((detect_false - 0.774999999999996) * 0.330026045916666) + (detect_real - 0.774999999999996) * ((detect_false - 0.774999999999996) * 25.727862415) + (detect_false - 0.774999999999996) * ((detect_false - 0.774999999999996) * 146.46560755916) + (Search_Velocity - 7.5) * ((search_area - 575) * -0.00057668705675) + (detect_real - 0.774999999999996) * ((search_area - 575) * 0.00969853512166666) + (detect_false - 0.774999999999996) * ((search_area - 575) * 0.0276137190283333) + (search_area - 575) * ((search_area - 575) * -0.000278218148014167) + (Search_Velocity - 7.5) * ((N_real - 10) * -0.0144573017280093) + (detect_real - 0.774999999999996) * ((N_real - 10) * 0.0572895615046297) + (detect_false - 0.774999999999996) * ((N_real - 10) * 0.0464394227083333) + (search_area - 575) * ((N_real - 10) * -0.00006595900942129 64) + (N_real - 10) * ((N_real - 10) * -0.00523680900478913) + (Search_Velocity - 7.5) * ((N_false - 10) * -0.0204563086354167) + (detect_real - 0.774999999999996) * ((N_false - 10) * -0.0775433788657423) + (detect_false - 0.774999999999996) * ((N_false - 10) * 2.12264069965278) + (search_area - 575) * ((N_false - 10) * -0.0000328419951620368) + (N_real - 10) * ((N_false - 10) * 0.0399338295746528) + (N_false - 10) * ((N_false - 10) * 0.0299890010466514) + (Search_Velocity - 7.5) * (corp_level * 0.00637031319696977) + (detect_real - 0.774999999999996) * (corp_level * 0.998222696060605) + (detect_false - 0.774999999999996) * (corp_level * -3.99822592151516) + (search_area - 575) * (corp_level * 0.00217331220545455) + (N_real - 10) * (corp_level * 0.0101629378914142) + (N_false - 10) * (corp_level * 0.203381472710439) + corp_level * (corp_level * 2.02587984275)


    def get_missionTime(self):
        self.missionTime = None
        Search_Velocity = self.SearchVelocity
        detect_real = self.DetectReal
        detect_false = self.DetectFalse
        search_area = self.SearchBoundary
        N_real = self.NReal
        N_false = self.Nfalse
        corp_level = self.CooperationLevel

        self.missionTime = 18.6348198770386 + -1.31977777777778 * Search_Velocity + 0.594343434343422 * detect_real + -0.496818181818194 * detect_false + 0.0683207070707072 * N_real + 0.0418287037037037 * N_false + -1.0023828125 * corp_level + (Search_Velocity - 7.5) * ((Search_Velocity - 7.5) * -0.0531560821484924) + (Search_Velocity - 7.5) * ((detect_real - 0.774999999999996) * -0.111166666666667) + (Search_Velocity - 7.5) * ((detect_false - 0.774999999999996) * 0.208416666666667) + (detect_real - 0.774999999999996) * ((detect_false - 0.
```

File - ResponseSurface.py

```
774999999999996) * 2.82916666666666) + (Search_Velocity - 7.5) * ((N_real - 10) * -0.0173518518518519) + (detect_false - 0.774999999999996) * ((N_real - 10) * -0.00866898148148139) + (N_real - 10) * ((N_real - 10) * -0.00526410098882493) + (Search_Velocity - 7.5) * ((N_false - 10) * -0.011037037037037) + (detect_real - 0.774999999999996) * ((N_false - 10) * -0.00781249999999998) + (N_real - 10) * ((N_false - 10) * -0.00281362525720165) + (Search_Velocity - 7.5) * (corp_level * 0.018837121212121) + (detect_real - 0.774999999999996) * (corp_level * -0.129772727272727) + (detect_false - 0.774999999999996) * (corp_level * -0.51030303030303) + (N_real - 10) * (corp_level * 0.0280871212121212) + (N_false - 10) * (corp_level * 0.0237068602693603) + corp_level * (corp_level * 2.5976328125)



    def get_covered(self):
        self.covered = None
        Search_Velocity = self.SearchVelocity
        detect_real = self.DetectReal
        detect_false = self.DetectFalse
        search_area = self.SearchBoundary
        N_real = self.NReal
        N_false = self.Nfalse
        corp_level = self.CooperationLevel

        self.covered = 108.094086419738 + -0.28236101469697 * Search_Velocity + -0.386792267676709 * detect_real + 1.27018343252531 * detect_false + -0.0803759091386075 * N_real + -0.0531356397615031 * N_false + 0.0230601513437483 * corp_level + (Search_Velocity - 7.5) * ((Search_Velocity - 7.5) * -0.0156606301955765) + (Search_Velocity - 7.5) * ((detect_real - 0.774999999999996) * 0.403290585749999) + (Search_Velocity - 7.5) * ((detect_false - 0.774999999999996) * -0.381189392416667) + (detect_real - 0.774999999999996) * ((detect_false - 0.774999999999996) * -13.0794448483333) + (Search_Velocity - 7.5) * ((N_real - 10) * -0.0110700020497685) + (detect_false - 0.774999999999996) * ((N_real - 10) * -0.177345574282407) + (N_real - 10) * ((N_real - 10) * -0.00414271745336239) + (Search_Velocity - 7.5) * ((N_false - 10) * -0.0145422918993055) + (detect_real - 0.774999999999996) * ((N_false - 10) * -0.178255406319444) + (N_real - 10) * ((N_false - 10) * 0.00210217519257973) + (Search_Velocity - 7.5) * (corp_level * -0.59274806780303) + (detect_real - 0.774999999999996) * (corp_level * -1.10720432272727) + (detect_false - 0.774999999999996) * (corp_level * 1.52366832757575) + (N_real - 10) * (corp_level * -0.145938457984007) + (N_false - 10) * (corp_level * -0.0679253659974747) + corp_level * (corp_level * -1.33283141053125)



        # x = np.array([[self.SearchVelocity],[self.DetectReal],[self.DetectFalse], [self.NReal], [self.Nfalse ], [self.CooperationLevel]])
        # areaData = pd.read_csv('area.csv')
        # b = areaData.Linear.array.to_numpy()
        # intercept = areaData.Intercept.array.to_numpy()
        # b0 = intercept[0]
        # B = areaData.loc[:,['velocity', 'detect_real', 'detect_false', 'Nreal', 'Nfalse', 'Coop']]
        # B = B.to_numpy()
        # mult = np.array([[1, .5, .5, .5, .5, .5],
        #                  [.5, 1, .5, .5, .5, .5],
        #                  [.5, .5, 1, .5, .5, .5],
        #                  [.5, .5, .5, 1, .5, .5],
        #                  [.5, .5, .5, .5, 1, .5],
        #                  [.5, .5, .5, .5, .5, 1]
        #                  ])
        #
        # B = np.multiply(mult,B)
        #
        # yPredict = b0 + np.matmul(x.transpose(), b) + np.matmul(np.matmul(x.transpose(), B), x)
        # self.covered = yPredict[0]



```

File - DesirabilityConfig.py

```
class DesirabilityConfig:
    def __init__(self, time = None):

        self.WeightTime = .15
        self.WeightArea = .15
        self.WeightDetect = .225
        self.WeightConfirm = .225
        self.WeightPerception = .25

        self.valueLowTime = .45
        self.valueMedTime = .4
        self.valueHighTime = .15

        self.LowTime = 5
        self.MedTime = 12
        self.HighTime = 21

        self.valueLowArea = 0.05
        self.valueMedArea = .15
        self.valueHighArea = .8

        self.LowArea = 60
        self.MedArea = 85
        self.HighArea = 115


        self.valueLowDetect = 0.02
        self.valueMedDetect = 0.38
        self.valueHighDetect = 0.6


        self.LowDetect = 60
        self.MedDetect = 85
        self.HighDetect = 110

        self.valueLowConfirm = .05
        self.valueMedConfirm = .4
        self.valueHighConfirm = .55


        self.LowConfirm = 60
        self.MedConfirm = 80
        self.HighConfirm = 110



        self.valueLowPercep = .05
        self.valueMedPercep = .45
        self.valueHighPercep = .6



        self.LowPercep = 35
        self.MedPercep = 70
        self.HighPercep = 105
```

File - Desirability.py

```
from Aframe_Test.Optimization.DesirabilityConfig import DesirabilityConfig

Config = DesirabilityConfig()

class Desirability():
    def __init__(self):

        self.MissionTimeRange = [Config.LowTime, Config.MedTime, Config.HighTime]
        self.AreaRange = [Config.LowArea, Config.MedArea, Config.HighArea]
        self.PercepRange = [Config.LowPercep, Config.MedPercep, Config.HighPercep]
        self.DetectRange = [Config.LowDetect, Config.MedDetect, Config.HighDetect]
        self.ConfirmRange = [Config.LowConfirm, Config.MedConfirm, Config.HighConfirm]


    def get_desirability(self, ResponseTime, ResponseArea, ResponsePercep, ResponseDetect, ResponseConfirm):
        self.desirability = None

        if ResponseTime >= self.MissionTimeRange[0] and ResponseTime < self.MissionTimeRange[1]:
            desirabilityTime = Config.valueLowTime*ResponseTime/Config.LowTime
        elif ResponseTime >= self.MissionTimeRange[1] and ResponseTime < self.MissionTimeRange[2]:
            desirabilityTime = Config.valueMedTime*ResponseTime/Config.MedTime
        elif ResponseTime > self.MissionTimeRange[1] and ResponseTime <= self.MissionTimeRange[2]:
            desirabilityTime = Config.valueHighTime*ResponseTime/Config.HighTime
        else:
            desirabilityTime = 0

        if ResponseArea < self.AreaRange[1] and ResponseArea >= self.AreaRange[0]:
            desirabilityArea = Config.WeightArea*Config.valueLowArea*ResponseArea*self.AreaRange[0]/Config.LowArea
        elif ResponseArea < self.AreaRange[2] and ResponseArea >= self.AreaRange[1]:
            desirabilityArea = Config.WeightArea*Config.valueMedArea*ResponseArea*self.AreaRange[1]/Config.MedArea
        elif ResponseArea > self.AreaRange[1] and ResponseArea <= self.AreaRange[2]:
            desirabilityArea = Config.WeightArea*Config.valueHighArea*ResponseArea*self.AreaRange[2]/Config.HighArea
        else:
            desirabilityArea = 0

        if ResponsePercep >= self.PercepRange[0] and ResponsePercep < self.PercepRange[1]:
            desirabilityPercep = Config.valueLowPercep*ResponsePercep/Config.LowPercep
        elif ResponsePercep >= self.PercepRange[1] and ResponsePercep < self.PercepRange[2]:
            desirabilityPercep = Config.valueMedPercep*ResponsePercep/Config.MedPercep
        elif ResponsePercep <= self.PercepRange[2] and ResponsePercep > self.PercepRange[1]:
            desirabilityPercep = Config.valueHighPercep*ResponsePercep/Config.HighPercep
        else:
            desirabilityPercep = 0

        if ResponseDetect < self.DetectRange[1] and ResponseDetect >= self.DetectRange[0]:
            desirabilityDetect = Config.valueLowDetect*ResponseDetect/Config.LowDetect
        elif ResponseDetect < self.DetectRange[2] and ResponseDetect >= self.PercepRange[1]:
            desirabilityDetect = Config.valueMedDetect*ResponseDetect/Config.MedDetect
        elif ResponseDetect > self.DetectRange[1] and ResponseDetect <= self.DetectRange[2]:
            desirabilityDetect = Config.valueHighDetect*ResponseDetect/Config.HighDetect
        else:
            desirabilityDetect = 0

        if ResponseConfirm < self.ConfirmRange[1] and ResponseConfirm >= self.ConfirmRange[0]:
            desirabilityConfirm = Config.valueLowConfirm*ResponseConfirm/Config.LowConfirm
        elif ResponseConfirm < self.ConfirmRange[2] and ResponseConfirm >= self.ConfirmRange[1]:
            desirabilityConfirm = Config.valueMedConfirm*ResponseConfirm/Config.MedConfirm
        elif ResponseConfirm > self.ConfirmRange[1] and ResponseConfirm <= self.ConfirmRange[2]:
            desirabilityConfirm = Config.valueHighConfirm*ResponseConfirm/Config.HighConfirm
        else:
            desirabilityConfirm = 0

        self.desirability = Config.WeightArea*desirabilityArea + Config.WeightConfirm*desirabilityConfirm + \
                            Config.WeightDetect*desirabilityDetect + Config.WeightPerception*desirabilityPercep + \
                            Config.WeightTime*desirabilityTime
```

File - OptimalConfiguration.py

```
import pandas as pd

high = pd.read_csv('desirability_output_high')
med = pd.read_csv('desirability_output_medium')
low = pd.read_csv('desirability_output_low')

high_des = high.desirability
high_des = high_des.to_frame()
med_des = med.desirability
med_des = med_des.to_frame()
low_des = low.desirability
low_des = low_des.to_frame()


max_high_des = high_des.max()
configuration_index_high = high_des.set_index('desirability').index.get_loc(float(max_high_des))
configuation_high = high.iloc[configuration_index_high]


max_med_des = med_des.max()
configuration_index_med = med_des.set_index('desirability').index.get_loc(float(max_med_des))
configuation_med = med.iloc[configuration_index_med]

max_low_des = low_des.max()
configuration_index_low = low_des.set_index('desirability').index.get_loc(float(max_low_des))
configuation_low = low.iloc[configuration_index_low]

print('stop')
```

Appendix P. JMP Use Guide

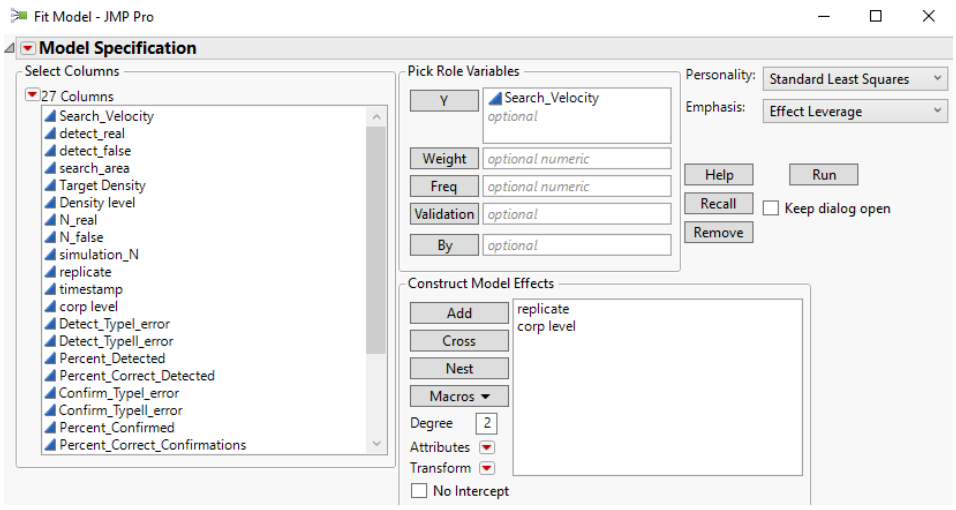
JMP Use Guide

Data entry:

In this research, data was entered from the analysis conducted in Python. The columns from the design of experiment test matrix and those from analysis were combined into one CSV file. Data columns required include: independent variables, replicate number, cooperation level (if any), and response variables.

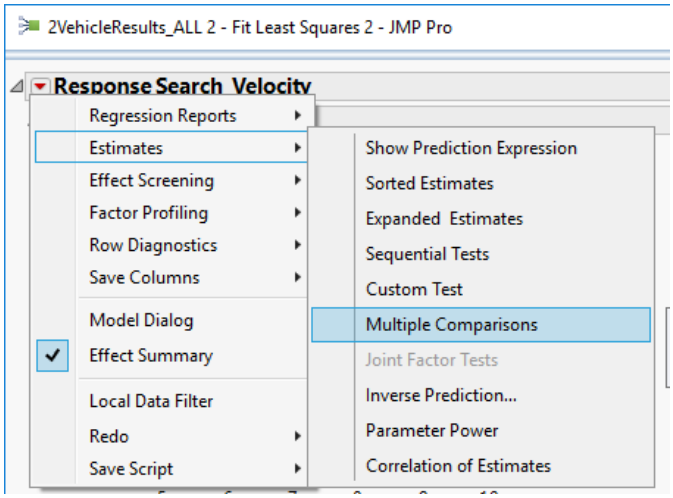
Multiple Comparison Test:

Click on **analyze** and then **fit model** to get the following dialogue window.

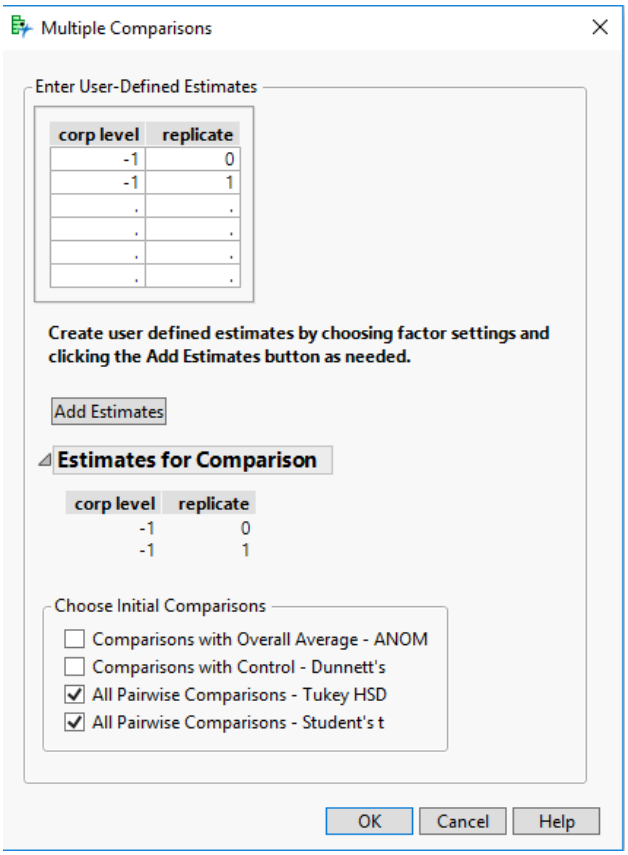


Select one **response** variable, click **Y** to add. Select **replicate** and click **add** to input as an independent variable. For multi-vehicle calculations, all levels were done simultaneously so **cooperation level** was also added as an independent variable. Click **Run.**

In the produced model, click the **red downward facing arrow** next to Response (shown below). Select **estimates** and then **multiple comparison.**



In this new dialogue box, enter replicate 0 and 1, and then -1, -1 cooperation level. Click **add estimates**. **Select all pairwise comparisons, Tukey and Student's t**. Click **ok**. Repeat this with 0, 0 cooperation level and 1, 1 cooperation level with replicate the same.



Investigate print outs. Look for blue lines to show that there is no significant difference between groups.

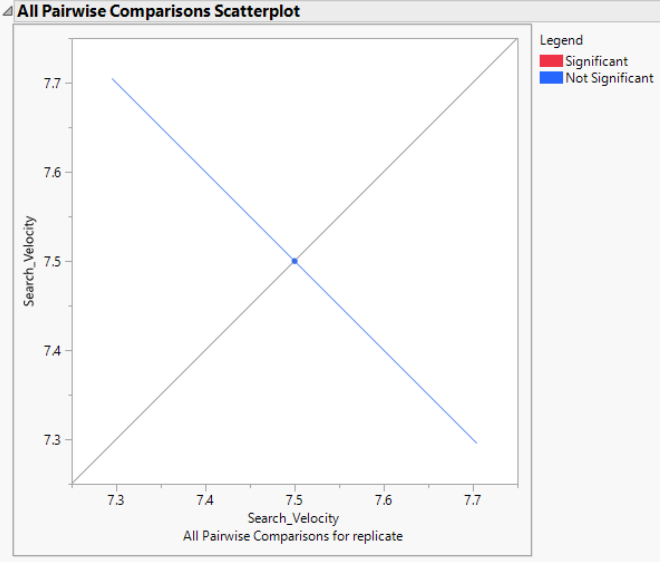
Student's t All Pairwise Comparisons

corp level = -1
Quantile = 1.96495, DF = 477.0

All Pairwise Differences

replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0	0.2079399	0.00	1.0000	-0.408591	0.4085914

All Pairwise Comparisons Scatterplot



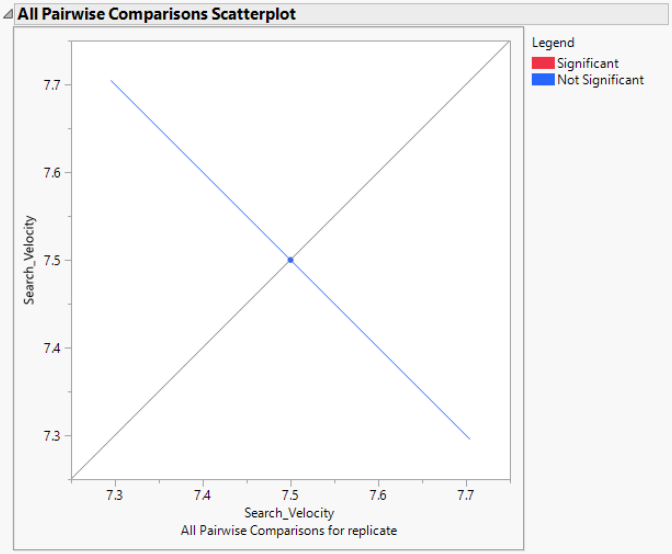
Tukey HSD All Pairwise Comparisons

corp level = -1
Quantile = 1.96495, Adjusted DF = 477.0, Adjustment = Tukey-Kramer

All Pairwise Differences

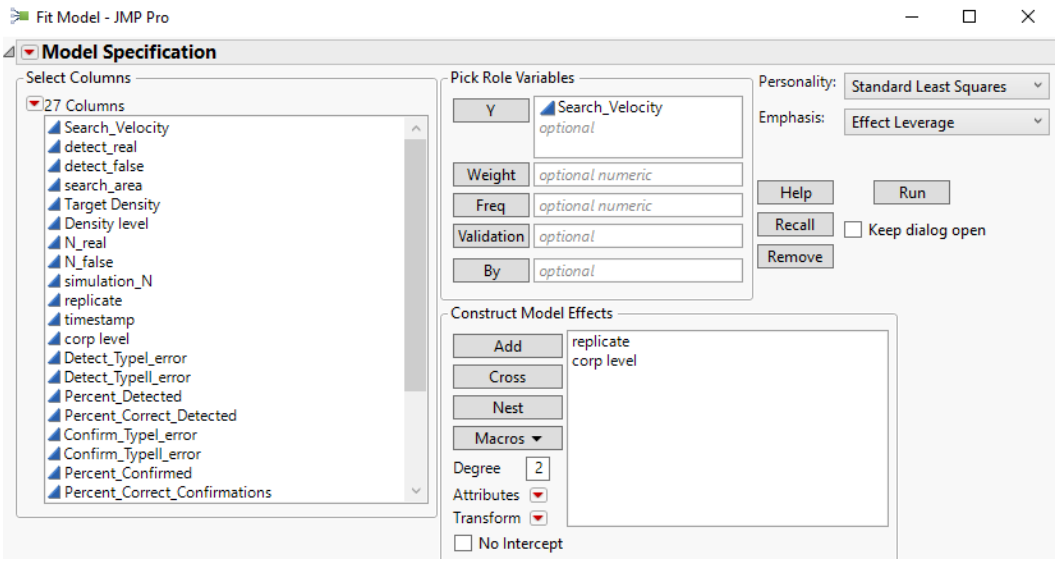
replicate	-replicate	Difference	Std Error	t Ratio	Prob>|t|	Lower 95%	Upper 95%
0	1	0	0.2079399	0.00	1.0000	-0.408592	0.4085916

All Pairwise Comparisons Scatterplot



Creating a statistical model:

To create a statistical model, click **analyze** and then **fit model**. Select **independent variables**, select **macros** and then **response surface**. Select **dependent variable(s)**, click **Y** and then click **Run**.



Analyzing a statistical model:

If desired, remove factors that have large p values. Check lack of fit and studentized residuals. If studentized residuals do not show, click red arrow, row diagnostics, and plot studentized residuals. If lack of fit does not show click red arrow, regression report, and then lack of fit. The parameter estimates are used to create the prediction equation.

Easy steps to input prediction equation into code:

One the model is validated, click the red arrow, save columns, and prediction formula. This step will save a column of predicted values in your JMP table. Go to that column, right click and select formula. This will open a dialogue box shown below with the prediction formula. Open a word document. Click in the white space and hold. Drag the equation into a word document. Repeat for all response variables. Save the word document as a text file. It is helpful to do a search and replace on variable names. In this research, the colons were replaced with nothing and variables with two words were replaced with underscores instead of spaces for easy coding. To paste into python, copy the equation and paste without formatting (right click, paste without formatting in Pycharm). This should paste in one line.


Pred Formula Detect_Type1_error - JMP Pro
Filter
29 Columns
Row
Numeric
Transcend
Trigonome
Character
Comparisc
Conditiona
Probability
Discrete Pr
Statistical
Random
Date Time
Row State
Assignmer
Parametric
Finance
Sear...locity
detect_real
detect_false
search_area
Targ...nsity
Dens... level
N_real
N_false
simu...on_N
replicate
timestamp
corp level
Dete...error
Dete...error
Perc...ected
Perc...ected
Conf...error
Conf...error
Perc...irmed
Perc...tions
Area...vered
Missi...Time
Resp...ness
fail_r...onse
Robustness
Perc...uracy
Perc...vered
Pred...error
Pred...error
Constant
0
1
2
-1
pi
42.319753066
-0.123567052 • Search_Velocity
8.6131654247 • detect_real
-35.95429697 • detect_false
-0.004791305 • search_area
-0.891108919 • N_real
0.9747305028 • N_false
-0.180399325 • corp level
(Search_Velocity - 7.5) • ((Search_Velocity - 7.5) • -0.193052541)
(Search_Velocity - 7.5) • ((detect_real - 0.775) • -0.579718937)
(detect_real - 0.775) • ((detect_real - 0.775) • -77.22101643)
(Search_Velocity - 7.5) • ((detect_false - 0.775) • -0.826209397)
(detect_real - 0.775) • ((detect_false - 0.775) • -18.49213261)
(detect_false - 0.775) • ((detect_false - 0.775) • -126.3438235)
(Search_Velocity - 7.5) • ((search_area - 575) • 0.001280016)
(detect_real - 0.775) • ((search_area - 575) • -0.007984639)
(detect_false - 0.775) • ((search_area - 575) • 0.0011012216)
(search_area - 575) • ((search_area - 575) • 0.000242779)
(Search_Velocity - 7.5) • ((N_real - 10) • 0.02953744)
(detect_real - 0.775) • ((N_real - 10) • -1.223873932)
(detect_false - 0.775) • ((N_real - 10) • 1.6255332439)
OK
Cancel
Apply
Help

Appendix Q. Test Plan

Autonomous Wide Area Search Test Plan

Contents

The approach of this document is based upon *Best practices for highly effective test design; Part 2 – Beginner's guide to design of experiments in T&E* by the STAT T&E COE

Assumptions

- No learning
- Cyber secure network
- Not flying in rain, human agent execute RTL if weather changes

Scope:

Iteration 1

- Wide Area Search
- Autonomous target detection report
- Static targets of approximately 8.43x2.46m
- Simulated environment
- One sensor
- One agent
- Rotary vehicle type
- Optimal search interest: Find as many targets as possible given accessible resources
- Priority in finding all targets in search area
- Second priority its to confirm locations
- On a given search pattern (we are not researching the pattern)

Iteration 2

- Wide Area Search
- Autonomous target detection report
- Static targets of approximately 8.43x2.46m
- Simulated environment
- One sensor
- 2 agents
- High, moderate, and low cooperation levels
- Rotary vehicle type
- Optimal search interest: Find as many targets as possible given accessible resources
- Priority in finding all targets in search area
- Second priority its to confirm locations
- On a given search pattern (we are not researching the pattern)

Future Iterations and areas of future research:

- Moving Targets
- Multiple Sensors
- Obstacles
- Alternate search patterns

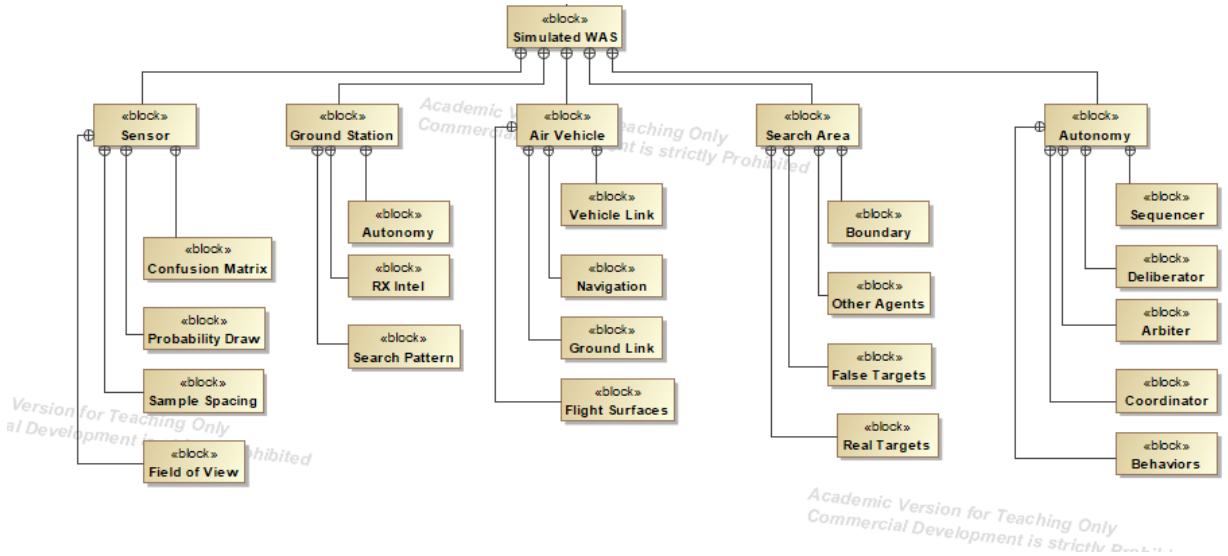
- Contour Map output
- Capture world model baseline to extrapolate to real world

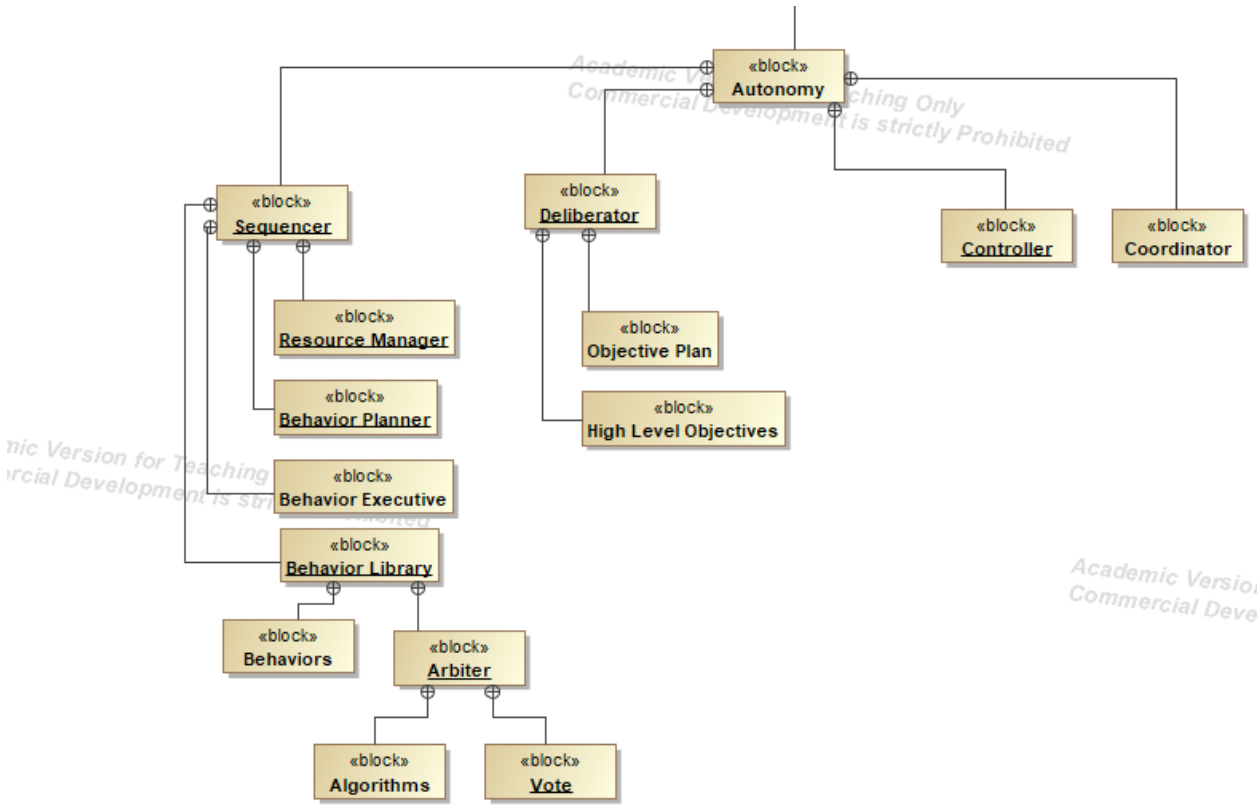
System Boundary

- Air Vehicle
- Search area
- Information being passed
- Recipient of intel

We are not concerned about the ground station in this test or project, just the AV, the area being searched, the intelligence gathered, and if it is useful to the recipient

System Decomposition (WBS)





Mission: Search mission area for targets and report detected and confirmed target locations

Definition of mission success:

- Search area complete
- List of target locations
- List of confirmed target locations

User Questions

- What percentage of targets in a given area can I expect the system to find?
- How accurate is the target location list reported?
- How accurate is the system reasoning?
- Does the system have the ability to adjust reasoning following failure?
- How many AVs are required to search X area in no more than Y time with Z accuracy?

Problem Type: Optimal configuration to achieve the mission

Optimization implies objectives that are to be maximized or minimized using system constraints.

Problem Statement

What configuration of design parameters will maximize area searched and percentage of real targets found?

User Requirements: What the system needs to do

only writing requirements that the team has to implement, not things that have already been done, or incorporating a ground station, this is being done in simulation

- The system shall search a given area for targets
    - The system shall detect targets
    - The system shall make declarations on targets (what it thinks it is)
    - The system shall log locations of targets
    - The system shall send
        - Telemetry
        - Target locations
        - Target declarations
- The system shall search in an efficient manner
    - The system shall monitor health (fuel, sensor health, etc)
    - The system shall adjust flight path
        - According to health (to include sensor health and failures)
        - According to obstacles
- The system shall confirm target location

Qualification Requirements / Test Objectives for Autonomy

Reusable optimization test, no requirement measures. Instead, compare TPMs of design to chosen figures of merit (sections to follow)

- The system shall be responsive enough to maintain a real time internal state
- The system shall be able to detect sensor failure (on/off)
- The system shall alter reasoning to account for a failed sensor

TPM

Measure	WBS Element
Responsiveness	Autonomy, Agent level, distribution
Robustness	Autonomy, Multi-Agent
Perception accuracy (wrongly impacted decisions)	Autonomy, Multi-Agent
Type I error (detect & confirm)	Sensor
Type II error (detect & confirm)	Sensor
Target identification confidence	Sensor
Percent targets identified (detect & confirm)	Sensor
Area covered	Air Vehicle
Area coverage rate	Air Vehicle

Notes on TPMs:

- Responsiveness: assess ability to react to perception, is the agent getting lost in planning
- Perception accuracy: Impact of perception error on decisions made by the autonomous system

Figures of Merit

Are specified by the user and compared to the TPMs to make mission level decisions for fielding. The FOMs are determined by the threats of a given environment that decision makers intend to implement an autonomous WAS agent. The simulation in the framework allows decision makers to input their configuration and determine if they are willing to accept the reported risk.

Test to aid Design Decisions

- Constrain problem and use simulation to find:
    - Minimum number of agents required (area searched in amount of time)
    - Minimum sensor quality (payload selection) (to achieve desired intel accuracy)

Notes: Idea to optimize a configuration given all other constraints, have the simulation step with minimum and continue to simulate until all other figures of merit are met

Optimization Objectives

- Maximize area searched
- Maximized area detected
- Maximize confirmed targets
- Minimize mission time required

System Constraints

- Fuel available
- Logic, what is enough?

  The user can constrain certain parameters based upon their mission environment to configure remaining parameters (see ideas for testing to aid design decisions).

Testing Constraints

- Validity of assumptions made
- Time
- Computing power
- Mission Planner
- SITL

Nuisance Factors

Source of Variability	Simulate	Cannot Simulate
Placement of Targets	X	
Probability Draw Sensor	X	
Environmental		X

Measures of Effectiveness (MOEs)

- Responsiveness: the system can respond to its environment in a timely manner
  - Internal state can be assumed to be real time
- Robust: the system can operate given sensor failure
- Search effectiveness: effective use of resources, replace C2 of human agents in this case
- Type I error, confirmed and detected

- Type II error, confirmed and detected
- Percent area searched

Measures of Performance (MOPs)

- Mission Time
- Area covered
- Worse case responsiveness
- Percent false positive (Type I error)
- Percent detected
- Percent fault injection detection
- Target report accuracy
- Confirmation accuracy
- Perception accuracy

Notes: Perception accuracy would be how accurate is the plan. How different is the solution to that given truth? Idea would be to iterate each plan given truth and another given sensed. How many plans are the same? How many are different?

Response Variables

Responsiveness:

$$t_{response} - t_{detect}$$

The amount of time for a handed down objective plan to be put into action. The timestamp of the objective plan is compared to the timestamp of the task status to begin as "in progress."

Robustness: Percent fault injection detection

If one of the agents goes down, the system should be able to respond accordingly.

$$\frac{N_{fualt\ response}}{N_{fault\ injections}}$$

A score of 1 detects all faults, meaning that the system is fully able to recover after a vehicle failure. A fault response is one where a logic is given to respond. Have a way to flag no solution and log what the problem was to create one.

Percent Detected: The percent of real targets detected

This metric reflects the percentage of real targets detected while searching the area.

$$\frac{N_{real\ targets\ found}}{N_{total\ real\ targets\ in\ area}} * 100\%$$

False Positive Detected (Type I error): The percent of real target declarations that were in truth false targets. The same for type II with the numerator equal to number of false negatives.

$$\frac{N_{false\ positive}}{N_{total\ real\ declarations}} * 100\%$$

Percent Confirmed: The percent of real targets confirmed (percent confirmed truths in code)

This metric reflects the percentage of real targets confirmed while searching the area.

$$\frac{N_{real\ targets\ confirmed}}{N_{total\ real\ targets\ in\ area}} * 100\%$$

Percent Confirmed Confirmations: The percent of confirmed targets confirmed correctly (percent confirmed confirmations in code)

This metric reflects the ability of the agent(s) to confirm targets. The number of correctly confirmed targets is divided by the number of targets the agent could confirm (revisit)

$$\frac{N_{real\ targets\ confirmed}}{N_{confirmations}} * 100\%$$

False Positive Confirmed (Type I Error): The percent of real target declarations that were in truth false targets. The same for type II with the numerator equal to number of false negatives.

$$\frac{N_{false\ positive\ confirmations}}{N_{total\ real\ declarations}} * 100\%$$

Search Effectiveness Score: Metric to represent effective search

In WAS, optimization problem of: min time, max target accuracy, min vehicles used, min AV loss, max area searched. Use weights of importance and do a vector sum as a score. The results of this can be used to compare design choices and gives a metric for the objective of searching efficiently.

Value Based Thinking (VBT) approach:

$$Value = \Sigma\, w * V(x_i)$$

W is the weight of an attribute. The summation of all must equal 1. $V(x_i)$ is the value function which ranges from 0 to 1. X_i is the performance metric for a given attribute. A value function has an associated curve. It can any shape. Some common ones are piecewise, linear, or square root. This is dependent on user needs and should be implemented in a changeable way. Table 1 shows fields that will need to be determined by the user.

Attribute	Weight	$0\ Value$	$1\ Value$	Function of utility curve
Target Accuracy				
Time				
Vehicles Used				
Agents Lost				
Area Searched				

An example:

$$SES = 0.3V(x_i)_{test\ accuracy} + 0.2V(x_i)_{time} + 0.05V(x_i)_{N_{agents}} + 0.15V(x_i)_{N_{lost\ agents}} + 0.3V(x_i)_{area\ searched}$$

Attribute	Weight	$\mathbf{0\ Value}$	$\mathbf{1\ Value}$	Shape
Target Accuracy	0.3	$x_i \leq 0.56$	$x_i \geq 0.72$	linear
Time	0.2	$x_i \geq$ 3 hours	$x_i \leq$ 0.5 hours	Square root
Vehicles Used	0.05	$x_i \geq$ 5 vehicles	$x_i <$1 vehicle	linear
Agents Lost	0.15	$x_i \geq 2$	$x_i \leq 0$	linear
Area Searched	0.03	$x_i \leq 0$	$x_i \geq 1.0$	linear

Attribute Metrics:

$$Target\ Accuracy = \left(\frac{N_{correct} - N_{wrong}}{N_{real}}_{detect} * \frac{N_{correct} - N_{wrong}}{N_{real}}_{confirmed} \right)$$

$$Time = \frac{t_{seconds}}{3600}$$

$$N_{lost\ agents} = N$$

$$A_{searched} = Percent\ of\ given\ search\ area\ covered$$

Design Parameters

What can be varied in the design?

- Altitude
- Max Velocity
- Weights in arbiter
- Confusion Matrix
- Number of targets
- Search area boundary
- Sensor field of view
- Sensor performance decay rate WRT increate in altitude
- Detect real probability
- Detect false probability

Test Considerations

- Test boundaries of performance
- Test as many rules at the same time
- Rules often specify boundaries
- Test as many common points at the same time
- Transient profiles exist—test enough profiles (at different modes) to give mission assurance
- What is the minimum rule set? How does it recover?
- Use replication to understand noise, account for different sources of error
- Are mode changes robust?
- Maybe we don't test or concern ourselves with validating out of the box capabilities
- Circular Error Probability (CEP), is there a way to utilize dynamics to minimize error
    - Want spatial diversity to close in error
- Bounds of design parameters are a trade space, if I can very them how does the system perform, each vertex is design metric
- Aid developmental test

Factors and Levels

Single Vehicle Test

Factor	High	Low
FOV	**39°**	
Search Velocity	10 m/s	5 m/s
Detect Real	0.9	0.65
Detect False	0.9	0.65
Search Area	700 m	200 m
N Real Targets	19	1
N False Targets	1	19
Altitude	**150**	

2 Vehicle Test

Factor	High	Low
FOV	**39°**	
Search Velocity	10 m/s	5 m/s
Detect Real	0.9	0.65
Detect False	0.9	0.65
Search Area	700 m	200 m
N Real Targets	19	1
N False Targets	1	19
Altitude	**150**	
Cooperation	**High, Moderate, Low**	

Experimental Design

Full factorial with three replicates

Required Data

Each run will need to be saved with the response data. This will allow one to assess the input parameters from the test matrix. Other option is to have a message with all input parameters. The following are data that will need to be logged by LCM:

- Perception Error
  - N Plan given truth ≠Plan given sensed
  - N Plan given truth = Plan given sensed
- Responsiveness
  - Timestamp objective time handed
  - Timestamp first action
- Robustness
  - N fault injections
  - N no response (thinking no response can be triggered by a max time to come up with a new plan)
  - Some sort of state to inform developers the failure mode and what will need to be considered in their code
- Percent detected
  - N real targets identified
  - N real targets in search area
- Type I error detect
  - N type I error declarations
  - N total real targets in search area (truth)
- Type II error detect
  - N type II error declarations
  - N total real targets in search area (truth)
- Type I error confirm
  - N type I error confirm declarations
  - N total real targets in search area (truth)
- Type II error confirm
  - N type II error confirm declarations
  - N total real targets in search area (truth)
- Percent confirmed
  - N real targets confirm declaration
  - N real targets in search area
- Search Effectiveness Score
  - Time to mission completion
  - N vehicles

- N vehicle failure (fuel empty, collision)
- Area searched

Testing Autonomy Background:

Testing autonomy is complex for many reasons. This test attempts to test the system and its objectives as well as the underlying logic of the agent. The Figure 1 depicts the following breakdown of autonomy.

Autonomy exists inside the system. It consists of a prebuilt autopilot, which is reasonably trusted by users. Autopilots have been used in commercial flight with human supervision and reliably implements actuation for flight control. Configuration parameters and constraints are the bounds and parameters used in the autonomous agent to constrain the performance. These are user inputs that drive the logic of the agent. The portion of autonomy requires more assurance is the agent's ability to perceive the world around it and to make the appropriate decisions regarding it.

Assuming the system is able to reach the overall mission objectives, decision makers what to know if they can trust the decide portion of autonomy. To do this, simulation is helpful in assessing the robustness of the logic and rules. What will the system do in a wide variety of situations and is there adequate logic to handle them? To asses this, there are a number of items of concern according to research and problem reports: responsiveness, robustness, trustworthiness, reasoning, and robot intent. In this project, there is an attempt to capture the responsiveness, robustness, and perception accuracy of the agent.

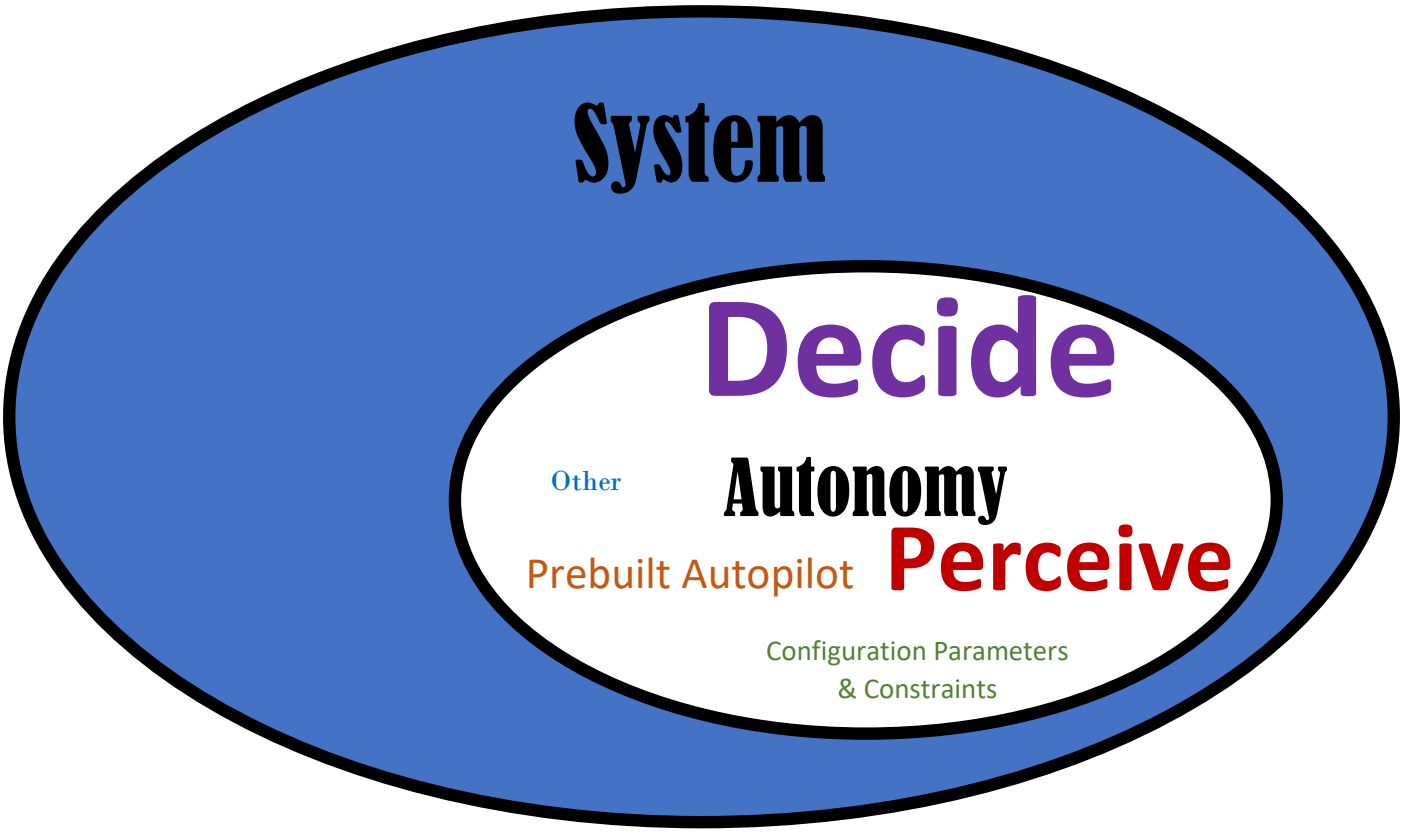


Figure 1: Autonomy Graphic

Appendix R. Multi-Domain Glossary

Glossary

Term	Domain	Description	Citation
Abstract Goal	AI	The specific tasks behaviors accomplish.	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Activation Path	AI	An abstract and standardized definition of a behavior. Consists of an initial condition, post condition, required data, abstract goal, control setting, and a vote.	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Activity Diagram	SysML	Represents behavior in terms of the order in which actions execute based on the availability of their inputs, outputs, and control, and how the actions transform the inputs to outputs	A Practical Guide to SysML: The Systems Modeling Language, 2nd Ed
Allocated Architecture	SE	Complete description of the systems design, including the functional architecture allocated to the physical architecture, derived input/output, technology and system-wide, trade-off, and qualification requirements for each component, an interface architecture that has been integrated as one of the components and complete documentation of the design and major design decisions.	Buede and Miller, 2016
Arbiter	AI	Algorithms that give a behavior action vote value to scale the affect or selection of a behavior from a set of available behaviors. An example would be a vector summation arbiter.	Unified Behavior Framework for Reactive Robot Control, Woolley, Brian G & Peterson, Gilbert L
Autonomy	AI	The ability to make decisions using sensory information without human interaction.	Adapted from: Autonomy and Unmanned Vehicles, Mahmoudzadeh, Somaiyeh, Powers, David, Bairam Zadeh, Reza
Behavior	AI and SE	A primitive transfer function from sensor inputs to motor outputs. Includes all types of behavior such as leaf and composite that generate actuation. In SysML, behaviors are activities that have been allocated to a block in a diagram.	Buede and Miller, 2016, A Practical Guide to SysML: The Systems Modeling Language, 2nd Ed
Behavior Diagrams	SysML	Can be an activity diagram, sequence diagram, state machine diagram, or use case diagram. Composed of activities and the flow and information between them. These diagrams specify how the components interact within the system and how the system interacts with external systems.	A Practical Guide to SysML: The Systems Modeling Language, 2nd Ed
Behavior Library	AI	All possible behaviors the controller can choose from. Only behavior library requires knowledge of the addition or removal of behaviors, minimizing changes throughout the system when adapting to a new environment.	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Behavior Model	SE	Defines the control, activation, and termination of system functions that is needed to meet the performance requirements of the system.	(Buede and Miller, 2016)
Behavior Planner	AI	Behavior planner generates set of behaviors that satisfy objective plan (OP). It uses the behavior representation to generate plans that are composed of a set of behaviors and ordering constraints necessary to accomplish OP.	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Block	SysML	A general modeling concept in SysML that is used to model entities that have structure, such as subsystems, hardware, software, physical objects, and abstract entities. A block can represent any real or abstract entity that can be conceptualized as a structural unit with one ore more distinguishing features. Blocks often capture hierarchy.	A Practical Guide to SysML: The Systems Modeling Language, 2nd Ed
Block Definition Diagram	SysML	Represents structural elements called blocks, and their composition and classification (UML class diagram)	A Practical Guide to SysML: The Systems Modeling Language, 2nd Ed
Component	SE	Subset of physical realization (and the physical architecture) of the system to which a subset of the system's functions have been (will be) allocated.	Buede and Miller, 2016
Composite Behavior	AI	The combination of two ore more behaviors with an arbiter to produce a single action output.	Unified Behavior Framework for Reactive Robot Control, Woolley, Brian G & Peterson, Gilbert L
Configuration Items	SE	lowest-level components in the physical architecture	Buede and Miller, 2016

Control Setting	AI	The controls, often motor controls, that a behavior is programmed to affect. An attribute of a behavior's activation path.	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Controller	AI	Executor Manages behaviors and their implementation. Implements one or more feedback control loops, often selected from a library of transfer functions or Behaviors.	Erann Gat. On Three-Layer Architectures. Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems, pages 195–210, 1998.
Coordinator	AI	Expands on capabilities, allowing prioritization of tasks and managing negotiations for multi-agent applications. Multi-agent task allocation to perform multiple tasks simultaneously.	HAMR : A Hybrid Multi-Robot Control Architecture, Hooper, Daylond & Peterson, Gilbert
Deliberator	AI	Performs high-level reasoning tasks that include task decomposition, task allocation, and planning. Generates new tasks from sensor data and processing that occur during task performance.	HAMR : A Hybrid Multi-Robot Control Architecture, Hooper, Daylond & Peterson, Gilbert
Design	SE	Preliminary activity that has the purpose of satisfying the needs of the stakeholders, begins in the mind of the lead engineer but has to be transformed into model employing visual formats in a highly skilled manner for success to be achieved.	Buede and Miller, 2016
Engineering of a System	SE	Engineering discipline that develops, matches, and trades off requirements, function, and alternate system resources to achieve a cost effective, lifecycle balanced product based upon the needs of stakeholders.	Buede and Miller, 2016
Fault	SE	Defect in the system that can cause an error. Faults can be permanent or temporary depending on an internal malfunction or external transient.	Buede and Miller, 2016
Figure of Merit	SE	Describes a specific system property or attribute for a given environment and context; a FOM is measured within the system.	Buede and Miller, 2016
Functional Architecture	SE	A logical architecture that defines what the system must do, a decomposition of the system's top-level functions.	Buede and Miller, 2016
Functional Requirements	SE	The 2 - 7 functions that are the first-level decomposition of the system's functions.	Buede and Miller, 2016
Fundamental Objective	SE	Aggregation of the essential set of objectives that summarizes the current decision context and is yet relevant to the evaluation of the options under consideration.	Buede and Miller, 2016
Fundamental Objectives Hierarchy	SE	Subdivision of the fundamental objective into value objects that more meaningfully define the fundamental objective, thereby forming a value structure.	Buede and Miller, 2016
Initial Condition	AI	Represent environment variables that when true generate an action recommendation and vote from behavior. There are two types: active and passive. An activation path attribute	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Integration	SE	Process of assembling the system from its components, which must be assembled from their configuration items.	(Buede and Miller, 2016)
Interface	SE	Connection for hooking to another system (external interface) or for hooking one system component to another (internal interface). The interface of a system contains both a logical element and a physical element (or link) that are responsible for carrying items (electrochemical energy or information) from one component or system to another.	(Buede and Miller, 2016)
Internal Block Diagram	SysML	Represents interconnection and interfaces between the parts of a block (modified UML composite structure diagram)	A Practical Guide to SysML: The Systems Modeling Language, 2nd Ed
Items	SE	Inputs that are received by the system, the outputs that are sent by the system to other systems, and the inputs that are generated internally to the system and sent to other parts of the system to assist in the transformation process for which the system is responsible.	Buede and Miller, 2016
Leaf Behavior	AI	An atomic behavior that cannot be broken down any further that requires no arbitration.	Unified Behavior Framework for Reactive Robot Control, Woolley, Brian G & Peterson, Gilbert L
Lifecycle	SE	Begins with the gleam in the eyes of the users or stakeholders, is followed by the definition of the stakeholders' needs by the systems engineers, includes developmental design and integration, goes through production and operational use, usually involved refinement, and finishes with the retirement and disposal of the system.	Buede and Miller, 2016
Measure of Effectiveness (MOE)	SE	Variable that describes how well a system carries out a task or set of tasks within a specific context; a MOE is measured outside the system for a defined environment and state of context variables.	Buede and Miller, 2016
Measure of Performance (MOP)	SE	Variable that describes a specific system property or attribute for a given environment and context. A MOP is measured within the system.	Buede and Miller, 2016

Mission Requirements	SE	Requirements that relate to objectives of the stakeholders that are defined in the context of the subsystem, not the system itself.	Buede and Miller, 2016
Model	SE	Any incomplete representation of reality, an abstraction. The essence of a model is the question or set of question that the model can reliably answer.	Buede and Miller, 2016
Objectives Hierarchy	SE	Hierarchy of objectives that are important to the system's stakeholders in a value sense; that is, the stakeholders would (should) be willing to pay to obtain increased performance (or decreased cost) in any one of these objectives. The definition of the natural subsets of the fundamental objectives into a collection of performance requirements.	Buede and Miller, 2016
Operational Concept	SE	Vision for what the system is (in general terms), a statement of mission requirements, and a description of how the system will be used. The shared vision is based on the perspective of the system's stakeholders of how the system will be developed, produced, deployed, trained, operated, and maintained, refined, and retired to overcome some operational problem and achieve the stakeholders operational needs and objectives. The vision requirements are stated in terms of measures of effectiveness. The operational concept includes a collection of scenarios; one ore more for each group of stakeholders in each relevant phase of the system's life-cycle.	Buede and Miller, 2016
Package Diagram	SysML	Represents the organization of a model in terms of the packages that contain model elements	A Practical Guide to SysML: The Systems Modeling Language, 2nd Ed
Physical Architecture	SE	Resources for every function identified in the functional architecture. The general physical architecture is a description of the partitioned elements of the physical architecture without any specification of the performance characteristics of the physical resources that comprise each element.	Buede and Miller, 2016
Physical Model	SE	Representation of an entity in 3-D space and can be divided into full-scale mock-up, subscale mock up, breadboard, and electronic mock-up.	Buede and Miller, 2016
Post Condition	AI	Set of environment effects that the behavior intends to achieve. This intent is based upon action recommendations for the behavior given an initial condition. Post condition may invalidate other goals. An activation path attribute	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Prototype	SE	Physical model of the system that ignores certain aspects of the system, glosses over other aspects, and is fairly representative of a third segment of aspects of the system. The prototype can range from a subscale model of the system to a paper display (storyboard) of the user interface of the system.	Buede and Miller, 2016
Qualification	SE	Process of verifying and validating the system design and then obtaining the stakeholders' acceptance of the design.	Buede and Miller, 2016
Qualification Methods	SE	Inspection, analysis and simulation, instrumented test, and demonstration.	Buede and Miller, 2016
Qualification Requirements	SE	Requirements that address the needs to qualify the system as being designed right, the right system, and an acceptable system.	Buede and Miller, 2016
Qualitative Model	SE	Model that provides symbolic, textual, or graphical answers. Symbolic models are based on logic or set theory. Textual models are based in verbal descriptions. Graphical models use either elements of mathematical graph theory or simply artistic graphics to represent a hierarchical structure, the flow of items or data through the system's function, or the dynamic-interaction of the system's components.	Buede and Miller, 2016
Quantitative Model	SE	Model that provides answers that are numerical; these models can be either analytic, simulation, or judgmental models.	Buede and Miller, 2016
Required Data	AI	Represents set of sensors (or data) required for behavior to function properly. This data includes computed data that is not directly from a sensor.	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Resource Manager	AI	Resource manager monitors system resources (hardware and data) and optimizes based on planned objectives and power management in relation to concurrent tasks. RM also answers queries about the prospects for a behavior's activation based on resource availability, allowing the system to dynamically respond to low battery life, failure of sensors, etc.	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Sequence Diagram	SysML	Represents behavior in terms of a sequence of messages exchanged between systems, or between parts of the system.	A Practical Guide to SysML: The Systems Modeling Language, 2nd Ed
Sequencer	AI	Enables and disables behaviors to achieve an objective plan, maintains internal state to alert deliberator if new plan is needed. Can consist of behavior executive, behavior library, resource manager, and behavior planner. The sequencer receives the objective plan from the deliberator and delivers arbitrated hierarchy of behaviors to the controller.	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Stakeholder	SE	Owner and/or bill payer, developer, producer or manufacture, tester, deployer, trainer, operator, user, victim, maintainer, sustainer, product improved, and decommissioned. Each stakeholder has significantly different perspective of the system and the system's requirements.	Buede and Miller, 2016

State machine diagram	SysML	Represents behavior of an entity in terms of its transitions between states triggered by events.	A Practical Guide to SysML: The Systems Modeling Language, 2nd Ed
Structure Diagrams	SysML	Can be a block definition diagram or internal block diagram. Each is of type block that shows structure within the design. Examples include physical architecture and IBDs.	A Practical Guide to SysML: The Systems Modeling Language, 2nd Ed
System	SE	Set of components (subsystems, segments) acting together to achieve a set of common objectives via the accomplishment of a set of tasks.	Buede and Miller, 2016
System Requirements	SE	Translation (or derivation) of the original requirements into engineering terminology.	Buede and Miller, 2016
System Task or Function	SE	Set of functions that must be performed to achieve a specific objective.	Buede and Miller, 2016
Systems Thinking	SE	Holistic look at the whole of systems, interrelationships with outside forces (systems, external systems, and context), and the properties of systems, especially emergence and stability.	Buede and Miller, 2016
Task	AI	A behavior hierarchy selected by the Sequencer to accomplish an objective plan goal.	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Task Plan	AI	A solved solution to an objective plan consisting of one or more sequenced tasks that together achieve the overall objective plan goal.	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J
Traceable	SE	Pertaining to requirements, each derived requirement must be traceable to an originating requirement via some unique name or number.	Buede and Miller, 2016
Validation	SE	Process of determining that the systems engineering process has produced the right system, based upon the needs expressed by the stakeholder.	Buede and Miller, 2016
Verification	SE	Matching of configuration items, components, subsystems, and the right system to their corresponding requirements to ensure that each has been built right.	Buede and Miller, 2016
Verification Plan	SE	How the qualification data will be used to determine that the real system conforms to the design that was developed.	Buede and Miller, 2016
Vote	AI	A vote is a value for a given behavior when in a state in which it acts. A vote of 0 is given when in a state that the behavior would not be useful and some other value when in a different state . Used by the composite behavior's arbiter to determine the composite behavior's action output.	Dynamic Behavior Sequencing for Hybrid Robot Architectures, Peterson, Gilbert L; Duffy, Jeffrey P; Hooper, Daylond J

Bibliography

Air Force Research Lab AFRL. Air Force Research Laboratory Test and Evaluation, Verification and Validation of Autonomous Systems: Challenge Exploration Final Report. Technical report, 2014.

Darryl K Ahner and Carl R Parson. Workshop Report: Test and Evaluation of Autonomous Systems. Technical report, STAT T&E Center of Excellence, 2016. URL `https://www.afit.edu/stat/doclib.cfm?dl=105`.

Michael Baudin, Maria Christopoulou, Michael Baudin, Yann Collette, and Jean-Marc Martinez. pyDOE, 2009. URL `https://github.com/tisimst/pyDOE`.

Brian Berthold, Trevor Bihl, Chadwick Cox, Todd Jenkins, and Logan Leland. Probabilistic reasoning for real-time UAV decision and control. In *Sensors and Systems for Space Applications XII*, volume 11017, pages 110170A–1 – 110170A–7. International Society for Optics and Photonics, SPIE, 2019. doi: 10.1117/12.2519451. URL `https://doi.org/10.1117/12.2519451`.

Trevor Bihl. *Using JMP: A Practical Guide*. SAS Press, 2017.

Trevor Bihl, Chadwick Cox, and Todd Jenkins. Finding common ground by unifying autonomy indices to understand needed capabilities. In Khanh D. Pham and Genshe Chen, editors, *Sensors and Systems for Space Applications XI*, volume 10641, pages 132 – 149. International Society for Optics and Photonics, SPIE, 2018. doi: 10.1117/12.2303670. URL `https://doi.org/10.1117/12.2303670`.

Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, 2 1986. ISBN 9780262521123.

Rodney Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, pages 14–23, 1986. ISSN 2374-8710. doi: 10.1109/JRA.1986.1087032.

Luis A Cortes. Best practices for highly effective test design ; Part 2 Beginners guide to design of experiments in T & E. Technical report, STAT T&E COE, 2014.

Ruth A David and Paul Nielsen. Defense Science Board Summer Study on Autonomy. Technical report, DSB, 6 2016.

Dough Decker and David Jacques. A Theory of Wide Area Search and Engagement. *Military Operations Research*, pages 37–57, 2007.

Defense Science Board. The Role of Autonomy in DoD Systems. Technical report, DoD, 7 2012.

Taylor B. Dunkel. Investigation of Cooperative Behavior in Autonomous Wide Area Search Munitions. Master's thesis, Air Force Institute of Technology, 3 2002.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Mass. : Addison-Wesley, 1995. ISBN 9780201633610.

Erann Gat. On Three-Layer Architectures. *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, pages 195–210, 1998.

Daniel P. Gillen. Cooperative Behavior Schemes for Improving the Effectiveness of Autonomous Wide Area Search Munitions. Master's thesis, Air Force Institute of Technology, 3 2003.

Jeremy Gray and David Jacques. Reference Architecture for Development of Autonomous Systems. 2019.

Sabine Hauert, Severin Leven, Maja Varga, Fabio Ruini, Angelo Cangelosi, Jean Christophe Zufferey, and Dario Floreano. Reynolds flocking in reality with fixed-wing robots: Communication range vs. maximum turning rate. *IEEE International Conference on Intelligent Robots and Systems*, pages 5015–5020, 2011. ISSN 2153-0858. doi: 10.1109/IROS.2011.6048729.

Daylond Hooper and Gilbert Peterson. HAMR: A Hybrid Multi-Robot Control Architecture. 2009.

David Jacques. Interview, 2019.

David Jacques and Meir Pachter. A Theoretical Foundation for Cooperative Search, Classification, and Target Attack. 2004.

Nidhi Karla and Susan Paddock. Driving to Safety: How Many Miles of Driving Would it Take to Demonstrate Autonomous Vehicle Reliability? Technical report, RAND Corporation, 2016. URL `rand.org/content/dam/rand/pubs/research_reports/RR1400/RR1478/RAND_RR1478.pdf`.

Somaiyeh MahmoudZadeh, David M. W. Powers, and Reza Bairam Zadeh. *Autonomy and Unmanned Vehicles: Augmented Reactive Mission and Motion Planning Architecture*. Springer Nature, 2019. ISBN 9789811322440.

Raj P. Malhotra, Michael J. Pribilski, Patrick A. Toole, and Craig Agate. Decentralized asset management for collaborative sensing. In Thomas George, Achyut K. Dutta, and M. Saif Islam, editors, *Micro- and Nanotechnology Sensors, Systems, and Applications IX*, volume 10194, pages 403 – 414. International Society for Optics and Photonics, SPIE, 2017. doi: 10.1117/12.2263728. URL `https://doi.org/10.1117/12.2263728`.

Sonia Martinez. *UAV Cooperative Decision and Control: Challenges and Practical Approaches*. Society for Industrial and Applied Mathematics, 2008. ISBN 978-0898716641.

James Mattis. Summary of the 2018 National Defense Strategy of The United States of America. Technical report, Department of Defense, 2018.

Susan J Milton. *Introduction to Probability and Statistics: Principles and Application for Engineering and the Computer Sciences*. McGraw Hill Education, 4 edition, 2003. ISBN 978-0-07-063694-1.

Douglas C Montgomery. *Introduction to Linear Regression Analysis*. Wiley, 5 edition, 2012. ISBN 978-0-470-54281.

Douglas C Montgomery. *Design and Analysis of Experiments*. Wiley, 9 edition, 2017. ISBN 978-1-119-11347-8.

Robin R Murphy. *Introduction to AI Robotics*. MIT Press, 2000. ISBN 9780262133838.

Raymond H Myers. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley, 4 edition, 2016. ISBN 978-1-118-91601-8.

Office Aerospace Studies OAS. Analysis of Alternatives (AoA) Handbook: A Practical Guide for Analyses of Alternatives. Technical report, Office Aerospace Studies, 07 2010.

Sang M. Park. Analysis for Cooperative Behavior Effectiveness of Autonomous Wide Area Search Munitions. Master's thesis, Air Force Institute of Technology, 8 2002.

Gilbert Peterson, Jeffrey Duffy, and Daylond Hooper. Dynamic Behavior Sequencing for Hybrid Robot Architectures. *Journal of Intelligent Robotic Systems*, 64(2): 179–196, 2011. doi: 10.1007/s10846-010-9535-3.

William Roberts, Patrick Meyer, Samuel Seifert, Ethan Evans, Michael Steffens, and Dimitri Marvis. A Flexible Problem Definition for Event-Based Missions for Evaluation of Autonomous System Behavior. pages 1–7, 2018. doi: 10.1109/OCEANS.2018.8604771.

Timothy D. Ross, Jeffrey P. Duffy, Richard J. Thomas, Wesley A. Jones, Jordan Garcia, Hyatt B. Baker, and Andrew C. Rice. Modeling the performance of modern sensor systems. In Riad I. Hammoud and Timothy L. Overman, editors, *Automatic Target Recognition XXIX*, volume 10988, pages 224 – 238. International Society for Optics and Photonics, SPIE, 2019. doi: 10.1117/12.2518887. URL `https://doi.org/10.1117/12.2518887`.

Brian Woolley and Gilbert Peterson. Unified Behavior Framework for Reactive Robot Control. *IEEE Journal of Intelligent and Robotic Systems*, pages 155–176, 2009. doi: 10.1007/s10846-008-9299-1.

Greg Zacharias. Autonomous Horizons: The Way Forward. Technical report, Air University, 3 2019. URL `https://www.airuniversity.af.edu/Portals/10/AUPress/Books/b_0155_zacharias_autonomous_horizons.pdf`.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–03–2020	Master's Thesis	Sept 2019 — Mar 2020

4. TITLE AND SUBTITLE

Development, Test, and Evaluation of Autonomous Unmanned Aerial Systems in a Simulated Wide Area Search Scenario: An Implementation of the Autonomous Systems Reference Architecture

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Cheney, Katherine E, 2nd Lt
King, David D, 2nd Lt

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENV-MS-20-M-220

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Lab Sensors Directorate AFRL/RY
WPAFB OH 45433
DSN 312-785-3636, COMM 937-255-3636
Email: trevor.bihl.2@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFIT/ENV

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The implementation and testing of autonomous systems is challenging due to the inherent design complexity, infinite test spaces, and lack of autonomy specific measures. This research instantiates the Autonomous System Reference Architecture (ASRA) on a wide area search (WAS) scenario as a test bed to study the reusability of ASRA and evaluate the system's ability to achieve efficient WAS and make decisions. To achieve this, metrics of autonomy were derived from literature requirements for autonomous systems implementing reactive architectures and control: responsiveness, robustness, and perception accuracy. The test implemented a face centered cubic design of experiments over 4 scenarios including a single vehicle, and 3 levels of cooperation between two vehicles. Following a rigorous test plan, the tests were conducted in simulation implementing automated testing and expedited analysis. The test results were used to create a response surface model to characterize the system and conduct multiple response optimization to determine an optimal configuration that maximizes area searched, percent detected, and perception accuracy in a given target density.

15. SUBJECT TERMS

Autonomy, UAS, Optimization, Response Surface Methodology, Framework, Reference Architecture, Test, Evaluation, Simulation, Cooperation, Coordination, RSM

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. David R. Jacques, AFIT/ENV
U	U	U	U	344	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x3329; david.jacques@afit.edu