3-22-2019

Imitating Human Responses via a Dual-Process Model Approach

Matthew A. Grimm

IMITATING HUMAN RESPONSES
VIA A DUAL-PROCESS MODEL APPROACH

THESIS

Matthew A. Grimm, 2d Lt, USAF

AFIT-ENG-MS-19-M-030

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-19-M-030

IMITATING HUMAN RESPONSES VIA A DUAL-PROCESS MODEL APPROACH

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

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 Computer Engineering

Matthew A. Grimm, BS

2d Lt Grimm, USAF

March 2019

AFIT-ENG-MS-19-M-030

IMITATING HUMAN RESPONSES VIA A DUAL-PROCESS MODEL APPROACH

Matthew A. Grimm, BS

2d Lt, USAF

Committee Membership:

Dr. Gilbert L. Peterson, PhD
Chair

Dr. Michael E. Miller, PhD
Member

Dr. Robert F. Mills, PhD
Member

AFIT-ENG-MS-19-M-030

Abstract

Human-autonomous system teaming is becoming more prevalent in the Air Force and in society. Often, the concept of a shared mental model is discussed as a means to enhance collaborative work arrangements between a human and an autonomous system. The idea being that when the models are aligned, the team is more productive due to an increase in trust, predictability, and apparent understanding. This research presents the Dual-Process Model using multivariate normal probability density functions (DPM-MN), which is a cognitive architecture algorithm based on the psychological dual-process theory. The dual-process theory proposes a bipartite decision-making process in people. It labels the intuitive mode as "System 1" and the reflective mode as "System 2". The current research suggests by leveraging an agent which forms decisions based on a dual-process model, an agent in a human-machine team can maintain a better shared mental model with the user. Evaluation of DPM-MN in a game called *Space Navigator* shows that DPM-MN presents a successful dual-process theory motivated model.

To Father and Mother

Acknowledgments

I would like to express my sincere appreciation to my research advisor, Dr. Gilbert Peterson, for his guidance and support throughout the course of this thesis effort. The insight and experience were certainly appreciated.

I would also like to thank all of my teachers and the staff at AFIT who taught me invaluable lessons and prepared me for my lifelong search for knowledge.

Matthew A. Grimm

Table of Contents

Page

Abstract iv

Acknowledgments vi

List of Figures x

List of Tables xiii

Chapter 1. Introduction 1

1.1 Hypothesis 3

1.2 Research Goals 4

1.2.1 DoD Goals. 5

1.3 DPM-MN High Level Overview 7

1.3.1 Input-Output. 9

1.3.2 DPM-MN Characterized as a Dual-Process Model 9

1.4 Evaluation 10

1.5 Thesis Outline 11

Chapter 2. Literature Review 12

2.1 Human-Machine Teaming 12

2.2 Concept Drift 18

2.3 Novelty Detection 20

2.4 Cognitive Architectures 21

2.4.1 Social Cognition Models. 24

2.4.2 Dual-Process Theory Implementations. 26

2.5 DPM-MN Building Blocks 27

2.5.1 CLARION. 27

2.5.2 Qualia Modeling Framework. 30

2.5.3 Bindewald Clustering-Based Online Player Modeling. 33

2.6 Summary 35

Chapter 3. Dual-Process Model using Multivariate Normal Probability Density Functions 36

3.1 DPM-MN Overview 36

3.2 DPM-MN Learning Characteristics 37

3.3 DPM-MN Behavior Examples 45
3.4 Summary 47
Chapter 4. Methodology 48
4.1 Notional Dataset Experiment 48
4.1.1 Notional Environment/Data. 49
4.1.2 Notional Test Strategy. 52
4.1.3 Notional Measures. 54
4.2 Space Navigator Dataset 54
4.2.1 Space Navigator Data. 55
4.2.1.1 Irreducible Error in Space Navigator. 56
4.2.2 Hyperparameter and Validation Process. 59
4.2.3 Individual Player Test. 64
4.2.4 Generic Player Test. 64
4.2.5 Space Navigator Measures. 66
4.3 Summary 68
Chapter 5. Results 69
5.1 Notional Dataset Results 69
5.1.1 Additional DPM-MN notional dataset analysis 73
5.1.2 Notional Dataset Hypothesis Closure. 77
5.2 Space Navigator Dataset Results 78
5.2.1 Hyperparameter Search Results. 78
5.2.1.1 Individual Player Hyperparameter Results. 78
5.2.1.2 Averaged Player Hyperparameter Results. 79
5.2.1.3 Generic Player Hyperparameter Results. 80
5.2.2 Individual Player Test Results. 80
5.2.3 Generic Player Test. 87
5.2.4 Space Navigator Results Initial Discussion. 89
5.2.5 Space Navigator Dataset Hypothesis Closure. 90
5.3 Summary 91
Chapter 6. Conclusions 92
6.1 Discussion 92

6.1.1 Overall Space Navigator Results Discussion. 92
6.1.2 DPM-MN Notional Dataset Results Discussion. 94
6.2 Future Work 96
6.2.1 Algorithm Optimizations. 96
6.2.2 Narrow Application. 98
6.2.3 Section Feature Value Collisions. 99
6.2.4 Problems with State-Response True Correlation. 100
6.2.5 System Two Multicollinearity. 101
6.2.6 DPM-MN Priority Improvements. 103
6.3 Future Application 105
6.4 Summary 107
References 108
Appendix A: Research Community Goals 117
Appendix B: Dual-Process Theory Influence on DPM-MN 120
Appendix C: DPM-MN Dual-Process Theory Functions 123
Concept Drift. 123
Overwriting Previous Knowledge. 123
Retaining Past Experience Memory. 124
Aggregation of Experiences to Form a General Concept. 125
Outlier Sensitivity. 126
Online Learning. 127
Balance Between Implicit and Deliberate Systems. 128
Appendix D: Bias-Variance Tradeoff in Experimental Design 129
Appendix E: Data Preprocessing 131
Appendix F: Other Ship Gaussian Weight Function Multiplier 142
Appendix G: Interpolation Method 144
Appendix H: Trajectory Euclidean Average Algorithm Computed 148
Appendix I: Parameters Assumed 149
Appendix J: Trajectory Representations 151
Appendix K: Data Distribution Non-Normality 155
Appendix L: Grimm LSTM Machine Learning Project 157

List of Figures

Page

Figure 1: Abstraction of System 1 and System 2.....10
Figure 2: High-level CLARION representation (Helie, et al., 2011). 28
Figure 3: High-level representation of QMF (Vaughan, et al., 2016). 31
Figure 4: CBOP paradigm (Bindewald, et al., 2017).....34
Figure 5: DPM-MN overview. 37
Figure 6:DPM-MN flowchart. 38
Figure 7: Addition of new points to System 1. 39
Figure 8: Creation of a new System 2 concept. 41
Figure 9: Windowing. 42
Figure 10: Refreshing of System 2 concepts. 42
Figure 11: Overwriting of another concept. 43
Figure 12: Revoking. 44
Figure 13: Concept drift. 45
Figure 14: Novelty detection example. 45
Figure 15: Abstract answer example. 46
Figure 16: Inference. 47
Figure 17: Notional dataset step one. 49
Figure 18: Notional dataset step two. 49
Figure 19: Notional dataset step three. 50
Figure 20: Notional dataset step four. 50
Figure 21: Notional dataset step five. 51
Figure 22: Notional dataset step two and three. 51
Figure 23: Notional dataset all data. 52
Figure 24: Test points zoomed out. 53
Figure 25: Test points zoomed in. 53
Figure 26: Example of irreducible error curve. 57
Figure 27: Median irreducible error curve of all players. 58
Figure 28: Close up of median irreducible error curve. 59

Figure 29: K-fold cross validation.s 63
Figure 30: SVM and DPM-MN correct points. 71
Figure 31: SVM and DPM-MN orange class correct. 72
Figure 32: SVM and DPM-MN non-orange class correct. 72
Figure 33: Immediate class groupings. 73
Figure 34: Blue class concept drift. 74
Figure 35: Orange class predominance and blue class return. 74
Figure 36: Windowing of orange concept. 75
Figure 37: Refreshing of blue concept 76
Figure 38: Final DPM-MN notional dataset trained model. 76
Figure 39: Correlation heatmap. 83
Figure 40: High performance predictions. 84
Figure 41: Low performance predictions. 85
Figure 42: Unique examples. 86
Figure 43: Trajectories unaware of important information. 87
Figure 44: Predicted trajectories in the generic experiment. 89
Figure 45: Notional dataset final model revisited. 94
Figure 46: Raw data input example. 131
Figure 47: A Space Navigator screen capture highlighting important game objects. 131
Figure 48: Important parts of Space Navigator. 133
Figure 49: Initial data capture and graphical representation. 134
Figure 50: Translation of objects. 135
Figure 51: Rotation of objects. 135
Figure 52: Scaling of objects and addition of zone lines. 136
Figure 53: Example of non-normalized feature values for a normalized state space. 137
Figure 54: Scatter matrix of features. 138
Figure 55: Data Euclidean distance. 139
Figure 56: Data trajectory X value residual error averages. 140
Figure 57: Data trajectory Y value residual error averages. 141
Figure 58: Relevance factor calculation. 142
Figure 59: Interpolation example. 145

Figure 60: t-SNE trajectory class visualization. 151
Figure 61: First example of characteristic medoid trajectory. 152
Figure 62: Second example of characteristic medoid trajectory. 153
Figure 63: Third example of characteristic medoid trajectory. 154
Figure 64: Log-normal distribution of test. 155
Figure 65: Cumulative distribution function. 155
Figure 66: Boxplot of trajectory difference. 156

List of Tables

Page

Table 1: DPM-MN parameter optimization. 53

Table 2: SVM optimized hyperparameters. 70

Table 3: DPM-MN optimized hyperparameters. 70

Table 4: Notional dataset accuracy results. 71

Table 5: Hyperparameters selected for each player and individual results. 78

Table 6: Average of variables across players. 79

Table 7: Hyperparameters selected for the generic test. 80

Table 8: Individual players results (lower is better). 80

Table 9: Individual player DPM-MN (unique) additional data. 82

Table 10: Average of variables across players additional data. 82

Table 11: Generic player results (lower is better) 87

Table 12: DPM-MN (generic) test additional data. 88

Table 13: Generic straight-line results (lower is better). 146

IMITATING HUMAN RESPONSES

VIA A DUAL-PROCESS MODEL APPROACH

Chapter 1. Introduction

Any corporation or group with common goals includes teams in some capacity. The effectiveness of these teams depends greatly on their teamwork abilities. Teamwork between agents of any kind, humans or computers, requires fundamental elements to act effectively, this includes both know-how related to the task at hand, as well as the knowledge to cooperate (Lemoine, et al., 2002). Fundamental elements of knowing how to cooperate include communication, organization, ability to build trust, and a shared mental model (SMM) to facilitate understanding. An SMM is the concept of "knowledge structures held by members of a team that enable them to form accurate explanations and expectations for the task, and, in turn, coordinate their actions and adapt their behavior to demands of the task and other team members" (Jonker, et al., 2010).

SMMs are a paramount factor for the success of cooperative work (Hanna, et al., 2018). An abundant amount of existing research explores SMMs, but the research mainly focused on teams only comprised of humans (Grim, et al., 2016; Tarola, et al., 2018; Wang, et al., 2009; Yen, et al., 2003).With the increase in the number of and complexity of computer agents interacting with human teams, an increased focus has risen in having the computer agent build a shared mental model to strengthen teamwork. The computer agent must somehow algorithmically formalize the SMM. Some of the benefits of a shared

mental model include an increase in trust, the creation of a shared goal, significant improvement on team effectiveness (Hanna, et al., 2018), a shared knowledge perspective (Abdulrahman, et al., 2018), a shared understanding of the task and team roles (Jonker, et al., 2010), and enhanced team communication (Wang, et al., 2009). Without an SMM, the computerized agent becomes situationally unaware of its teammates' decision-making process. As a result, the agent will make ineffective decisions solely based on its own cognitive model with no consideration of possible teammate desires.

One way to formalize an SMM in a computer is through developing a cognitive architecture to simulate the mental model of the human teammate. Jonker, et al. (Jonker, et al., 2010) investigates the notion of an SMM and designs an ontological model using UML. The explained rules and concepts of an SMM then motivate an example implementation in the Blocks World for Teams (BW4T) problem domain. Hodhod and Magerko (Hodhod, et al., 2016) bolster an SMM representation through the Co-creative Cognitive Architecture (CoCoA). CoCoA is built on the principles of a minimalistic design, a confidence factor, the use of fuzzy logic, and the construction of knowledge rules. It ultimately creates an SMM between two improvisational agents to co-create stories. Fan and Yen (X. Fan, et al., 2011) created an architecture called Shared Mental Models for all (SMMall). It implements a hidden Markov model to predict a person's cognitive workload. SMMall is not only able to maintain an SMM of the whole team, but it can also divide the team members into sub teams each with their own subgroup SMM.

The Dual-Process Model using multivariate normal probability density functions (DPM-MN) algorithm instantiates the dual-process theory as an SMM. The presented DPM-MN behaves similar to one notion of how psychologists believe people think – dual-

process theory. To limit the scope of the work, DPM-MN only includes dual-process learning characteristics (Sun, 2015). Future work could extend DPM-MN to include dual-process accounts of reasoning (J. S. B. T. Evans, 2003). The dual-process accounts of reasoning use a belief-logic framework to describe the task of each system. System 1 represents an associative, belief-based, and instinctual problem-solving method. System 2 corresponds to a general intelligence, abstract, and higher-level thinking approach, which enables it override System 1 when System 1 reaches a conclusion with low confidence.

Some of the key dual-process learning characteristics DPM-MN implements include memory preservation, balance between implicit and deliberate systems, aggregation of experiences to form a general concept, online learning, overwriting previous knowledge, retaining past experience through memory, concept drift, and outlier sensitivity.

DPM-MN characteristics are not a full solution, but it is a starting point for the knowledge representation and search concept. By selecting a limited number of important cognitive elements to functionalize in the DPM-MN algorithm, DPM-MN leaves some cognitive elements out. Conversely, the interaction between the chosen cognitive elements will implicitly emerge into new functions. It is possible to ally DPM-MN with other cognitive architectures to come up with an artificial general intelligence (AGI) solution. Also, because of the modularity of the learning concepts, DPM-MN may adjust in the future to gain a closer representation of the most critical thinking and learning components of a human.

1.1 Hypothesis

The need for autonomous systems to align with a human teammate's mental model motivates the DPM-MN architecture. DPM-MN and all of its functions orient towards the

simulation of a target human's learning process. After enough experiences with the human teammate, the autonomous agent should possess an approximation of the human's mental model. This research hypothesizes that a human-machine teaming agent motivated by a dual-process learning theory model maintains a more accurate mental model of the user.

1.2 Research Goals

The evaluation of the hypothesis is broken into two research goals: the development of DPM-MN and the evaluation of DPM-MN with human participants.

The development of DPM-MN is made of two separate systems that each utilize multivariate normal probability density functions to make predictions. System 1 makes associative predictions while System 2 makes reflective predictions. DPM-MN also implements many different cognitive learning functions such as the aggregation of experiences to form general concepts and the overwriting of previous knowledge.

Evaluation of DPM-MN uses the *Space Navigator* dataset. The measure for a mental model match is that the DPM-MN model outputs a trajectory that is most similar to those output by the human.

The main goal is to build a cognitive architecture that functions similarly to a human mind by learning the associated mental model. Succeeding in this goal would greatly benefit human-machine teams and AGI. Many future AI solutions will most likely involve correct management of human-robot teams. As a result, it is important that the relationship between human and robot teammates improves. One way to do this is by capacitating robots with the ability to anticipate and imitate human behavior by building a replica of their shared mental model. This gained ability, in turn, improves anthropomorphism and predictability which are key components for trust in teams

(Bindewald, et al., 2018). Human-robot teams have the potential to be more enjoyable and perform better if trust is improved through increased perceived reliability.

An example of the game chess illustrates the main behavioral research goal. Given a single state instance of a game of chess, the desired AI's next move should not necessarily be the most optimal or rewarding move. It should be the move a specific human player would make. DPM-MN should allow a person to feel like they are playing against the individual that participated in training.

Similarly, in a team setting, assume two people named Chris and Joe have been pilot and co-pilot partners for a year. Joe has to move away, so a DPM-MN utilizing AI that was trained on Joe replaces Chris's co-pilot. The co-pilot AI will imitate Joe's operational behaviors so that Chris feels like he is still flying with Joe. Instead of aiming for the most optimal functionality, DPM-MN aims to capture the behavioral idiosyncrasies of unique people.

1.2.1 *DoD Goals.*

The next push in AI research will involve algorithms dealing with explainable AI and unexpected queries (Launchbury, 2017). DPM-MN's structure favors these two concepts. DPM-MN can be visualized through dimensionality reduction to allow a real-time learning process demonstration. DPM-MN's gathering and learning of knowledge are intuitively understood with a visualization. Independent observations in System 1 get transferred to System 2 as a generalized concept once there exists a large enough grouping of instances in System 1. A separation of concepts is shown with the DPM-MN visuals in Chapter 3.

DPM-MN also covers unexpected queries. In this case, an unexpected query is when the system has never seen an observation before. System 2 responds to the new observation first, but it is also added as an individual instance in System 1. When DPM-MN encounters the same observation again, there is more information to rely on. This process is similar to how a human learns.

An algorithm with cognitive flexibility is also important for autonomy. Volatile mapping of observations to the search space achieves cognitive flexibility in DPM-MN. Once a concept drift of a class occurs, DPM-MN quickly relearns the meaning of the class concept. This aims to achieve a constant evolution of meaning as DPM-MN gathers new information. When a person is learning a new task, they may react differently to the same situation depending on their level of proficiency or experience.

One of DPM-MN's goals is peer flexibility. If the robot in a human-machine team can accurately predict the action the human will take, then it can rapidly adjust its level of involvement. If a specific person reacts tragically to a given situation, DPM-MN may foresee the outcome and alert the robot to start taking a supervisor role.

Each of these goals not only further the Air Force's vision on the future of AI, but they enable better human-machine teaming. From a practicality standpoint, human-machine teaming will be the primary application all the previous goals aid in ultimately achieving. An AI assistant in a fighter aircraft can potentially revolutionize the way pilots fly (Schutte, 2015). Machines' constant vigilance will better defend against cyber-attacks when the AI can understand context. Drones may become capable of starting and completing missions autonomously. These examples are the future of warfare. Many different countries envision AI to become the "third revolution in warfare, after gunpowder

and nuclear arms" (Russell, et al., 2015). News sources reported Russian leader Putin saying "whoever leads in artificial intelligence will rule the world" (Meyer, 2017). They want to potentially have an advantage over the US in military power. China, Russia, and the US are all key players in the new arms race of AI. The use cases for AI in war involve complex and difficult problems. The vision for AI warfare heavily depends on research in the concepts previously mentioned such as cognitive flexibility, peer flexibility, human-machine teaming, explainable AI, and unexpected queries

1.3 DPM-MN High Level Overview

Psychologist William James (Colman, 2008), in 1890, proposed the idea of dual-process theory and linked it to social information processing. The dual-process theory involves two different systems that countervail each other. Automation characterizes System 1 while the use of working memory characterizes System 2 (J. Evans, et al., 2013). Additionally, System 2 is a domain for more abstract, explicit knowledge that is adjustable. System 1, on the other hand, harbors implicit individual knowledge pieces that are harder to alter.

DPM-MN represents the demonstrated knowledge of the observed human through state-response pairings. It is difficult to determine the exact thought process of a person when making decisions, but the mapping of patterned responses to a scenario encapsulate any possible rational, or lack thereof, that lead the human to their decision. Klein's Natural Decision Making (Klein, 2008) paradigm says people do not generate and compare options. Rather, they rely on prior experience. Klein's research validates the choice to build DPM-MN through a stream of experiences. Additionally, Klein described a recognition-primed decision model that asserts an intuitive component and an analytical component. The

intuitive component attempts to rapidly compare the situation to a previously similar situation. The analytical component finds a satisfactory option among different prospective solutions. DPM-MN's structure aligns with this model through the automatic response given if a state is close enough to an individual experience in System 1. If no similar previous experience exists in System 1, then System 2 determines the decision by considering multiple generalized concepts.

A dual-process model approach exploits the bias/variance tradeoff in machine learning for balancing benefits. System 1 implements the high variance component via importance of singular experiences, and System 2 implements the high bias component via a mapping of the generalized concepts. With both systems, the dual-process model can be sensitive to outliers while still accounting for the general solution when needed. By transitioning datapoints between these systems, a dual-process model recognizes and properly deals with concept drift. A change in the average feature values of a class over time defines concept drift.

DPM-MN uses Gaussian kernels to represent the area of influence each individual point in System 1, or each concept in System 2, possesses. Within the model of System 1 individual points' Gaussian kernels resemble a radial basis function. The probability density function inference solely depends on the distance from the center. On the other hand, System 2 Gaussian kernels are made utilizing the individual points from System 1 that compose the underlying concept distribution. When DPM-MN predicts a new input datapoint, each Gaussian kernel adds its own influence on the prediction relative to the new point's placement in the kernel's probability density function. Chapter 3 presents more information on the DPM-MN algorithm.

1.3.1 *Input-Output.*

DPM-MN functions by accepting a state input and outputting a response. This is considered a state-response pairing. The input maps to a position in the state space. After that, nearby observations are found using distance measurements. A class labels each observation, and each class represents a fixed response. The selected response class ultimately provides a response pertaining to the specific scenario.

As an example, imagine a situation where a person decides what mode of transportation they should use to get to work for a given day. Weather, the availability of transportation methods, the day of the week, and the number of traveling companions are factors that might shape the state space. These are examples of features which determine the state-space position. Some response examples may include riding a bike, taking the bus, carpooling, driving a personal car, and walking.

Although the experimental data comes from a specific state-response scenario, any situation with a state-response structure can theoretically use DPM-MN. Anywhere a decision is made based on the current circumstances is a high-level example. Because DPM-MN fits many generic problem spaces, there is a lot of responsibility to appropriately represent both the circumstantial knowledge and the response actions.

1.3.2 *DPM-MN Characterized as a Dual-Process Model*

DPM-MN is different from previous dual-process theory approaches because it heavily focuses on dynamic information such as concept drift and non-redundant information between System 1 and System 2. Furthermore, DPM-MN allows one-off instances to determine future decisions if the circumstance is nearly similar. This lets concept drift occur quickly and appropriately accommodates anomaly situations.

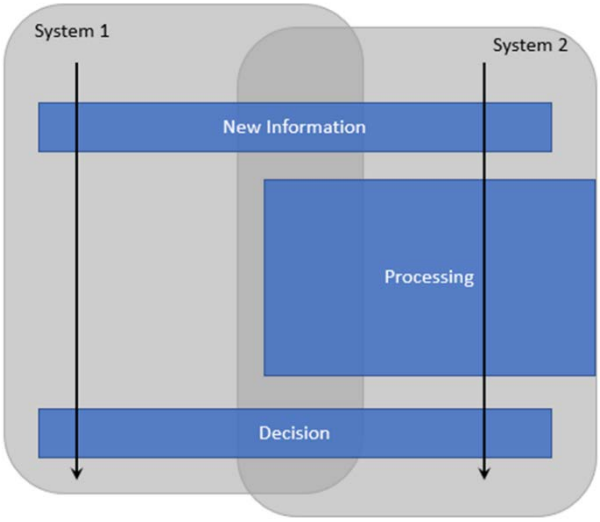

Figure 1: Abstraction of System 1 and System 2.

Figure 1 illustrates the basic concept of System 1 and System 2. System 1 is reactionary while System 2 depends on some processing. In DPM-MN, the processing needed to form a generalized representation in System 2 using the conglomeration of individual observations from System 1 satisfies the working-memory characteristic of a System 2 described in dual-process theory.

1.4 Evaluation

The evaluation of an SMM comparison is based on mimicking the human response. Better human predictions indicate a closer SMM between the algorithm and the human. It is possible for the SMM to develop based on a human that performs poorly on the specified task. DPM-MN creates an SMM which will lead to higher performance in the prediction of trajectories in the *Space Navigator* environment. This SMM will give statistically significant and better results in *Space Navigator* than baseline tests. In the future, a test should be conducted with humans to see if the DPM-MN algorithm creates a correct SMM.

1.5 Thesis Outline

Chapter 1 introduces the motivation to imitate human responses in a human-machine team. Imitation of a human teammate relies on the creation of an SMM. The dual-process theory provides a cognitive framework to help align an autonomous system with a human teammate's mental model. The dual-process theory inspires the DPM-MN implementation (Appendix B). The DPM-MN evaluation in the *Space Navigator* experiment shows that DPM-MN improves imitation of human player trajectory generation.

The following chapter presents related work on how other researchers have tackled a similar goal. The next chapter is an overview of the DPM-MN algorithm. Since there are many intricately related functions in DPM-MN, Chapter 3 individually illustrates each functionality. Chapter 4 discusses the methodology of the experiment. The setup of the experiment, the Bayesian optimization parameter search method, and the DPM-MN parameters are discussed. In Chapter 5, we analyze the results. Chapter 6 concludes the findings and guides potential future research.

Chapter 2. Literature Review

Dual-Process Model using multivariate normal probability density functions (DPM-MN) is a cognitive architecture inspired by the dual-process theory. Its purpose is to functionalize the human cognitive process during decision making and the storing of memory. DPM-MN should map the strategic learning process of a person to subsequently enable computer agents to interact with others in a human-like manner.

This chapter presents several topics related to different aspects of DPM-MN. The topics include machine learning focused topics of human-machine teaming, concept drift, novelty detection, and more psychological focused topics that include cognitive architectures, social cognition models, and dual-process theory model implementations. Afterwards, three specific papers that motivated much of the DPM-MN research are individually explored.

2.1 Human-Machine Teaming

A reliable cognitive architecture would bring about many practical effects in human-machine teams. Theoretically, DPM-MN or another cognitive architecture is trained on a specific person to learn their behavior. After it learns their behavior, the DPM-MN wielding robot can either better predict how the individual will react to a given situation, or the robot can take on and act with that unique personality. In order to guess the benefits that come from DPM-MN, it is assumed that personality, human-imitation, and teammate predictiveness all follow from the implementation of the proposed algorithm.

The first example of a socially cognitive robot being useful comes from Bindewald, et al. (Bindewald, et al., 2018). Trust between members of a team greatly affects the performance. In a human-robot team, a couple influencers of trust include the predictability

and the anthropomorphism amount of the robot. A personality provided through DPM-MN enhances both of these components. The robot will behave in a way more easily noticed by the human, and the robot would seem more human-like since it is imitating a human it used for training.

Personality can also affect the behavior of the human partner. Salam, et al. (Salam, et al., 2017) conducted an experiment to determine group engagement based on the human's personality and the robot's personality. Personality in this experiment was defined as either extroverted or introverted in order to make a clear distinction. They found that the most group engagement arises when both the human and robot are extroverted, and the worst results occurred when the human and robot are both introverted. Previous studies have suggested a different effect; humans enjoy working with robots with a similar personality (Park, et al., 2012). The extroversion results would align with this study, but the introverted results would not. Salam, et al. figured that different experimental settings caused the disagreement in findings between the two studies. Regardless of the ground truth, it becomes apparent that the human-robot teams differ depending on the personality type of the robot. This indicates the importance of including a robot personality to achieve a higher performance potential.

Choi, et al. (A. Choi, et al., 2015) had similar findings on the importance of robotic personality. This study focused on positive emotions from the robot versus negative emotions from the robot. They used physiological measures such as electrodermal activity and heart rate to objectively measure both arousal and psychological valence. The results showed the robot's capability to affect the inferential processes and the affective processes of a human participant. The inferential processes are where "emotion expressions provide

information about other's mental states" and affective processes are where "emotion expressions elicit emotion in the receiver which, in turn, impacts his or her decisions" (A. Choi, et al., 2015). This research found that more emotional humans cooperated more with and formed positive opinions of computers which exhibited positive emotion. Also, people who experienced less emotion were more likely to exploit the computers that showed submissive expressions such as regret. Other behavioral patterns were also noticed across different human personality and robot personality matchings (A. Choi, et al., 2015). Overall, it is noticeable how human-robot interactions can change and be manipulated depending on the personality of the human and the personality of the robot.

Other research experiments have found almost contradictory results. Lee, et al. (Lee, et al., 2006) discovered a complementarity attraction effect in their research where the human-robot team with complementary personalities performed best. The participants favored and enjoyed interacting with a robot with a complementary robot. Extroverts liked the social company of an introverted robot while introverts liked the social company of an extroverted robot. Lee, et al. (Lee, et al., 2006) makes note of different social rules emerging from the multiple AI social tests. The authors claim a similarity attraction rule occurs when the robot is a disembodied social actor and the opposite occurs when the robot is an embodied social actor. With embodied actors, people tend to treat the robot with the same social rules as a normal human. Their mind does not perfectly differentiate the fact that they are interacting with a robotic social agent. Lee, et al. in the end asks for more research to be completed to better uncover the human-robot social rules. Even though many of the social AI research seemingly conclude different results, there is a common

agreement among all of them: robots with personality significantly affect human behavior and attitudes.

AI will not be accepted into the general society unless the AI can be trusted. Sarkar, et al. (Sarkar, et al., 2017) show that the human perception of the robot more heavily depends on the personality of the robot rather than the performance of the robot. The researchers tested human-likeliness, likeability, trustworthiness, and perceived competence. The experiment was executed with an industrial setting in mind. The robotic co-worker was meant to aid humans in manufacturing tasks. The result showing the importance of the robot's personality opens a pathway of understanding for the acceptance of commercial AI products. Sooner or later, human-robot teams will need to implement a robot with a personality.

It is important to test human-robot teams against human-human teams to better understand what benefits a robot may bring to a team. Harriott, et al. (Harriott, et al., 2015) conducted a few experiments to observe the performance and mental workload of human-robot teams versus human-human teams. Their results showed a lowering of mental workload when the human-robot team performed the task, but performance between the two teams did not vary by too much. Though, the authors concluded some observations about the human-robot team. It is important for the human and robot to understand the other's perspective, goals, and decision-making process. Being able to predict what the other team agent is going to do can greatly improve a team's dynamic. Lastly, human-robot teams need to leverage the advantages humans have and the advantages robots possess in order to efficiently allocate tasks. Experimental limitations hindered these potential performance improvements. If these areas were improved, performance most likely would

have improved along with workload. DPM-MN attempts to target some of these performance areas.

Along with lowering the mental workload of a human-robot team, robots with a personality can also affect stress coping abilities (Lohani, et al., 2016). The idea is that a human-robot team where the human trusts and relies more on the robot can impact the human's perceived ability to cope with stress in a positive manner. Socially intelligent robots can provide their human teammate with socioemotional support through successful social interaction.

Robots can display their intelligence through proper mathematical calculations and choices, but some AI enthusiasts argue that "primate intelligence primarily evolved in adaptation to social complexity" (Dautenhahn, 2007). This is the social intelligence hypothesis (Gavrilets, et al., 2006). In essence, robots need to start with social intelligence to gain intellectual attributes such as interpretation, prediction, and manipulation of information. The acceptance of social intelligence as a crucial element for AI further validates the need for cognitive architectures such as DPM-MN. One way to capture social intelligence is through copying humans, like DPM-MN, through imitation learning. There is a lot to learn from the "richness and depth of human experiences" (Dautenhahn, 2007). Specifically, contextual adaptation depends on human experiences and the ability to restructure the meaning of a situation to fit the context of recent events. Many optimization and analytical problems are solved through traditional AI, but human intelligence will require a solid model of social intelligence.

The benefits from equipping robots with a personality will spread throughout human-robot teams found in normal society. The healthcare sector can use AI to fight

against labor shortages, the educational sector can use AI to reduce costs and improve quality, and the marketing sector can use AI to persuade shoppers in buying certain brands (Gonzalez-Jimenez, 2018). However, with improvement comes an expansion of human factors issues. Before socially intelligent AI can utilize their full potential, a few other problems will need to be better studied. These problems include deciding what role AI should play in the allocation of tasks, avoiding negative emergent behavior when teaching robots, correctly combining robot models with human models, and fighting against irrational fears (Sheridan, 2016)

With the spread of socially intelligent robots, it is important to distinguish the roles a human should have versus the roles a robot should have. Fitts list proposes that humans appear to surpass computers at tasks requiring judgement and inductive reasoning while computers appear to surpass humans at tasks requiring routine repetition, highly complex operations, and deductive reasoning (De Winter, et al., 2015). However, Sheridan brings up the following question: “If a job can be more efficiently done by a robot, should that job always be automated?” (Sheridan, 2016). In a future setting, the robot may possibly be able to perform better than the human in every sector. This requires the team designer to more deeply define the purpose of a human in a human-robot team (Schutte, 2015). One suggestion is to utilize AI to allow the human to perform optimally within the classic Yerkes-Dodson inverted U theory of performance (Diamond, et al., 2007).

Human-robot teams would gain a barrage of advantages if the robot could reconstruct the functions of a human teammate’s mind. DPM-MN tries to simulate the commonly adopted dual-process theory to create a well-performing cognitive architecture. A dual-process theory inspired algorithm may not be a standalone end solution, but it may

become a tool used with other algorithms to construct an AGI. Either way, it is an attempt to get closer to reaching the goal of AGI.

2.2 Concept Drift

Concept drift "primarily refers to an online supervised learning scenario when the relation between the input data and the target variable changes over time" (Gama, et al., 2014). It is sometimes referred to as adaptive learning. Another perspective is to think of concept drift as a change in the underlying distribution of data given a context. Problems involving a data stream over a long period of time are sometimes concerned with concept drift. A data stream describes when the input is sequential and temporal.

There are various types of concept drift including sudden, gradual, and reoccurring (Bifet, et al., 2011). Webb, et al. (Webb, et al., 2016) attempt to provide a framework for functionalizing the categorization of different types of concept drift. They use data descriptions such as drift magnitude, drift frequency, drift duration, drift recurrence, and drift predictability to formulate the equations for determining the concept drift type.

Gama, et al. (Gama, et al., 2014) created a taxonomy to help understand the differences between concept drift algorithm implementations. The taxonomy split the algorithms along the individual lines of memory, change detection, learning process, and loss estimation. Each of the axis provide in-depth explanations about the sub-categories and research examples for each.

Many solutions for the problem of concept drift take a monitoring approach. An example is ADWIN which oversees the raw sensor data or streaming error and provides a concept drift warning whenever it detects a large enough change (Gama, et al., 2014). These approaches performed well for their purpose, but they fail when a high input data

rate exists. Today, data can stream so quickly that it is improbable to label all of it (Woźniak, et al., 2016).

A concept drift algorithm that takes advantage of a dual-store structure is the Self-Adjusting Memory (SAM) algorithm (Losing, et al., 2017). Dual-store models involve a balance between short-term memory (STM) and long-term memory (LTM). SAM provides a general structure that deals with various types of concept drift without needing any hyperparameterization. The STM tracks the most recent concepts while the LTM tracks concepts previously revoked from the STM as time advances. Predictions are balanced between the memory types dependent on the recent accuracy rate. The prediction provided by each memory type uses a kNN classifier with a distance weighting. SAM is primarily different from DPM-MN because SAM focuses on building memory based on different types of sliding windows while DPM-MN focuses on building each system based on human-like cognitive learning characteristics.

Many recent concept drift algorithms focus on dealing with concept drift in the face of the increasing velocity of data. Some other concept drift solutions include using an ensemble of simple classifiers (Woźniak, et al., 2016), an algorithm that only requires 15% of the data to be labeled (Lindstrom, et al., 2010), and an algorithm taking advantage of probabilistic graphical models to capture context through a latent variable(Borchani, et al., 2015). Other real-world applications of concept drift modeling are provided by *CDCStream* (Ienco, et al., 2014) and two overviews covering the topic (Bifet, et al., 2011; Gama, et al., 2014). Some of the real-world applications of concept drift examined in the overviews are a movie recommender system, food sales prediction, and real-time mass flow prediction.

2.3 Novelty Detection

The goal of outlier/novelty/anomaly detection is to determine which data observations do not belong to the "normal" data distribution. The presence of normal data constructs the normal data distribution. After the distribution is created, new points are usually given something similar to a novelty score. The novelty score is checked against a subjective threshold to determine if the new point is an outlier or not (Pimentel, et al., 2014).

A predominant issue for contemporary novelty detection is the "curse of dimensionality". High-dimensionality can become the source of many different problems including the need to search a large space, distances between points becoming less informative, and the existence of a difficult relationship between "hubness" and a true outlier degree (Zimek, et al., 2012). Many novelty detection algorithms focus on overcoming the curse of dimensionality (Erfani, et al., 2016; Radovanović, et al., 2015).

A myriad number of outlier detection algorithms exist. Pimentel, et al. (Pimentel, et al., 2014) divide the algorithm types into the categories of probabilistic detection, distance-based detection, reconstruction-based detection, domain-based detection, and information-theoretic detection. Ahmed, et al. (Ahmed, et al., 2016) instead classify novelty detection algorithms into classification-based detection, statistical detection, information-theory detection, and clustering-based detection. Additionally, Agrawal organizes novelty detection algorithms into classification-based detection, clustering-based detection, and hybrid detection (Agrawal, et al., 2015). Each of these three survey papers provide several examples and comprehensive explanations for their categorizations. Although the survey papers are sourced from different academic backgrounds, there is obvious overlap in their grouping of novelty detection algorithms. Even though there are

many different algorithms, the fundamental goal is to differentiate between the “normal” and “abnormal” distribution locations.

The definition of “normal” is somewhat subjective. Thus, Lavin (Lavin, et al., 2015) came up with a benchmark for anomaly detection algorithms(Lavin, et al., 2015). The goal is to evaluate real world anomaly detectors in an objective manner. Real-world situations that caused an anomaly instance to occur motivated the hand-labeled dataset true values. The benchmark also takes the real-world performance measure of detection timing into consideration. It is important for an algorithm to quickly reveal anomaly observations in time critical settings such as during intrusion detection.

A real-world application of outlier detection can be seen in Djenouri’s research (Djenouri, et al., 2018). Djenouri detects traffic outliers using three different methods: statistical models, distance-based models, and pattern analysis. The experiment highlighted critical problems with Djenouri’s applied outlier detection. First, computation time can get very expensive, especially when during pattern analysis. Second, a temporal dataset constrains the scope of outlier detection. Djenouri could detect single-point extreme outliers, but had difficulty detecting a larger outlier window of time. Third, researchers should utilize speed improvement architecture through high-performance computing, database systems, and computational intelligence. Finally, it may be useful to repurpose an existing, more complex outlier detection method to fit Djenouri’s research problem.

2.4 Cognitive Architectures

Cognitive architectures in artificial intelligence are meant to model human cognition. A concrete example of a cognitive architecture is CogPrime (Goertzel, et al., 2013). CogPrime tries to serve as an Artificial General Intelligence (AGI) solution. It

acknowledges the complicated and complex nature of the AGI problem. CogPrime focuses on learning through pattern finding and the evolution of a network of memory-based hierarchies and heterarchies. The key to this model is the proclaimed cognitive synergy between the vast network. The network modules are interlocked enough to provide efficiency in the search for a solution but specialized enough to engender new functionalities. The authors believe beginning with possibly complicated and complex, but sound, models and applying a sustained effort towards building it will greatly improve AGI.

CogPrime and other cognitive architectures receive their engineering advantage through functionalizing the human thought, decision-making, and learning process. Historically, researchers have created cognitive architectures for three purposes: "to capture…the functions of reasoning, control, learning, memory, adaptivity, perception, and action", to design the basic building blocks necessary for the evolution of capabilities over time, and to reach human level intelligence (Lieto, Bhatt, et al., 2018). A variety of psychological and biological theories inspire cognitive architectures. Distributed Adaptive Control theory of mind and brain is the basis for DAC-h3 (Moulin-Frier, et al., 2017), the Functional Systems Theory is the basis for Vityaev's cognitive architecture (Vityaev, et al., 2018), and a physical symbol system hypothesis and the heuristic search hypothesis propel the Icarus cognitive architecture (D. Choi, et al., 2018).

Two progenitor cognitive architectures, Adaptive control of thought-rational (ACT-R) (Anderson, et al., 2004) and state, operator, and result (Soar) (Lehman, et al., 2006) inspire many modern cognitive architectures. Collaborating many different submodules to produce a single, functional module is the main idea of ACT-R. When the model needs a new component, it creates the component as a specialized module and fits

it into the grand scheme. Some examples of modules are the perceptual-motor module, the goal-module, and the declarative memory module. Soar is more structural. It works through designated memory segments. The memory representations are long-term memory (divided into procedural, semantic, and episodic memory), and short-term working memory. The working memory holds the current state. Similar to most cognitive architectures, both ACT-R and Soar contain learning mechanisms.

Cognitive architectures are not only meant for studying humans or creating a human-like robot. Recently, researchers proposed to implement cognitive architectures into self-driving cars and transportation systems (Chen, et al., 2018; Deng, et al., 2017; Jämsä, et al., 2013; Saucer, et al., 2018). Even though architectures mimic human cognition, the functions gained through the architecture are also beneficial to cars. Cars act similar to human-like agents that communicate with the driver and each other. The theory of cognitive psychology describes "sense and perception, memory and learning, reasoning, judgement and problem solving" (Deng, et al., 2017). The cognitive architecture provides the higher-level functioning that allows the car to perform tasks such as determining the human driver's mental or emotional state, pay attention to important information in the surrounding environment, and weigh decisions against each other.

Despite major advancements in cognitive architectures, researchers still need to solve certain critical issues. Lieto, et al. (Lieto, Lebiere, et al., 2018) outline the two primary problems of knowledge size and knowledge homogeneity. In relation to knowledge size, current cognitive architectures lack a solid knowledge base. Humans are able to learn and memorize an enormous amount of generalized information for everyday tasks while artificial intelligence typically remains limited in its problem space. Knowledge

homogeneity is an issue because of the theoretically high number of problem domains an AI agent needs to understand. It is difficult to encode every object and idea down to a common form for processing purposes.

For a plethora of examples of cognitive architectures, refer to Ye's survey of cognitive architectures in the past twenty years (Ye, et al., 2018).

2.4.1 *Social Cognition Models.*

Social cognition models are a subset of cognitive architectures. Social cognition models are unique because they are cognitive architectures where the environment is primarily a social setting. The endowed agent is meant to increase their social and emotional intelligence. A socially aware agent would especially be useful in human-robot teams (Baxter, et al., 2016; Infantino, et al., 2018).

Affective computing is "computing that relates to, arises from, or influences emotions" (Picard, 1995), and it heavily influences the social capability of AI. Schuller offers three sectors of emotion that are necessary for improving the affective computing ability of socially intelligent AI. These three areas are emotion recognition, emotion generation, and emotion augmentation (Schuller, et al., 2018). Emotion recognition allows a robot to determine what emotions a human is displaying while emotion generation allows a robot to display those same emotions so a human can relate. Researchers scarcely discuss emotion augmentation, but it means to use emotion as a factor in the cognitive process of planning, reasoning, and learning.

One of the more important social values relating to affective computing is trust. In human-robot teams, it is worthwhile to figure out how the robot can influence the trust of their teammate. Some factors that promote trust include reliability, validity, utility,

robustness, and false-alarm rate of the agent (Siau, 2018). An enhancement in trust of a robot also enhances the effectiveness of a human-robot team (Siau, 2018; Weiss, et al., 2017).

In a team, the robot must be able to socially read humans to act appropriately. Social intelligence includes the ability to understand the social setting. Social cues such as vocal laughs, visual smiles, and facial expressions can permit the robot to comprehend emotional states (Krakovsky, 2018; Weber, et al., 2018). Once the physical features are determined, an autonomous agent should deliberate the internal state of the human (Görür, et al., 2017). It could be possible for the robot to correctly understand the human's current ostensible emotional state but misunderstand their true hidden desires. After an emotional state prediction is made, a robot can emotionally influence the human (Bera, et al., 2018; Pereira, et al., 2015).

There already exists various complex social cognitive architectures (Azarnov, et al., 2018; J. Fan, et al., 2017; Lazzeri, et al., 2018; Lemaignan, et al., 2017; Rodríguez-Lera, et al., 2018; Sinclair, et al., 2017; Wiltshire, et al., 2017). The FACE humanoid robot (Lazzeri, et al., 2018) has a "believable facial display system based on biomimetic engineering". HiMoP creates a social cognitive architecture to structure a hierarchy of needs, and it also executes behaviors by using an assortment of finite-state machines (Rodríguez-Lera, et al., 2018). A final notable implementation feature is Wiltshire's (Wiltshire, et al., 2017) recommendation to embed a dual-processing theory module in an agent to account for type 1 and type 2 processes during social interactions.

2.4.2 *Dual-Process Theory Implementations.*

Cognitive architectures inspired by the dual-process theory all follow the same tenet of using some form of System 1 and System 2, or type 1 and type 2, modules. System 1 represents the implicit process while System 2 represents the explicit process. The implicit process is intuitive and defined by autonomy while the explicit process is reflective and requires working memory (J. Evans, et al., 2013). Each of the systems should interact in some manner to achieve positive cognitive functions. However, each implementation will surely contain a somewhat unique execution of the dual-process theory.

Augello (Augello, et al., 2016) uses System 1 and System 2 to obtain a multimodal quadrant of processing. Implicit and explicit processing and convergent or divergent processing split the quadrants. These give the states of “exploratory”, “reflective”, “tacit”, and “analytic”.

The MECA cognitive architecture has System 1 and System 2 as primary subsystems (Gudwin, et al., 2017). Within System 1 and System 2 exists many smaller components. In MECA, the dual-process theory harmonizes with Dynamic Subsumption, Conceptual Spaces, and Grounded Cognition.

As the dual-process theory architectures are iteratively analyzed (Augello, et al., 2017; Blythe, 2012; Dennis, et al., 2018; Lieto, et al., 2017; Potamianos, 2014; Strannegård, et al., 2013), some common themes become more apparent. System 1 and System 2 frequently represent low-level and high-level goal subsystems. The more abstracted calculations and intensive resource allocation occur in System 2 while the almost instant and near-obvious predictions occur in System 1. The researchers often exploit the nonlinear behavior that comes with dual system processing when interacting between System 1 and

System 2. Some authors activate System 2 only when System 1 cannot provide an answer while other authors infer in parallel and then compete the two answers. Sometimes this competition of answers serves as a check to see if System 2's reasoned answer agrees with System 1's intuitive answer.

2.5 DPM-MN Building Blocks

The dual-process theory also inspires a few more cognitive architectures such as CLARION (Helie, et al., 2011) and QMF (Vaughan, et al., 2016). These models each attempt to simulate human cognition in their own unique algorithm. There are differences between which cognitive functions are most prominent. They also notably differ on the knowledge representation, the data processing pipeline, and the interaction between their own System 1 and System 2 implementations. Dual-process theory does not inspire Bindewald's Clustering-Based Online Player Modeling (Bindewald, et al., 2017), but Bindewald, et al. deals with the same *Space Navigator* dataset that DPM-MN also attempts. CLARION, QMF, and Bindewald's algorithm inspired the creation and structure of DPM-MN. Ideas from the three inspirational models pervade the DPM-MN model.

2.5.1 *CLARION.*

CLARION (Helie, et al., 2011) breaks apart System 1 and System 2 into implicit and explicit knowledge. The aggregation of bottom-level implicit knowledge forms the top-level explicit knowledge. The model consists of two different subsystems, each with their own System 1 and System 2. The first subsystem is the Non-Action-Centered Subsystem (NACS) that builds the declarative long-term memory. The other subsystem is the Action-Centered Subsystem (ACS) which deals with procedural memory and executive function. These two subsystems represent the short-term versus long-term processing

duality. The NACS and ACS, along with their respective System 1 and System 2 architectures, interact with each other through activation nodes.

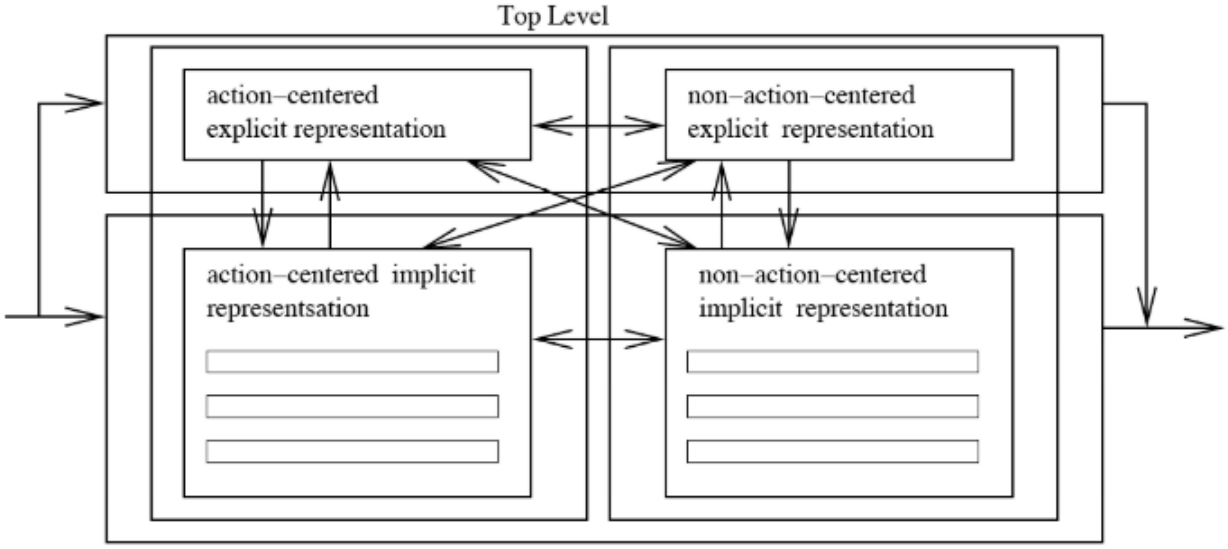

Figure 2: High-level CLARION representation (Helie, et al., 2011).

CLARION exhibits its own special topology for a cognitive architecture. It seems as though the cognition theory came first, and then the model reflects the desired functions. Instead of gaining an engineering advantage through mathematical theory, CLARION gains an engineering advantage through cognitive theory. Many prominent characteristics exist. The System 1 and System 2 sections are split via a distinction between implicit and explicit knowledge. The model learns implicit knowledge "through gradual trial-and-error learning" (Helie, et al., 2011). Bottom-up learning from System 1 builds the explicit knowledge in System 2. There also is a respect for an interaction and balance between both levels instead of treating them as completely independent functions.

Decision Field Theory (DFT) is the basis for the primary psychological idea. DFT places great importance on understanding the evolution of the model rather than only caring

about the end-state. Two main features encompass DFT: valence and preference. Valence is the "momentary advantage/disadvantage of an option in relation to the other options being considered" while preference of an option "refers to the accumulation of all the valences that this option has received in the past". In essence, DFT examines the path to the end-state at distinct points in time while many other models only use the process as a means to an end. For example, in neural networks, nobody cares about the parameter values of the network half-way through training.

CLARION elucidates other pivotal progression points for cognitive architectures. It advocates for a minimalism structure. The cognitive architecture should start out initially bare and "internal structures and representations should also be kept to a minimum" (Helie, et al., 2011). Rule-based reasoning and similarity-based reasoning are utilized to make decisions based on previous experiences. People make decisions based on simplified mental modeled concepts built up over time, and they also can make decisions based on how similar a current experience is to a previous experience they encountered. CLARION also realizes the need to decide which cognitive functions should be encoded in the cognitive architecture. There are so many psychological effects and people are different in general. It is crucial to include enough cognitive functions to simulate a human's mind, but not too many where the functions impede the effectiveness of each other because of complexity. For example, CLARION attempts to simulate "similarity effects, the attraction effect, the compromise effect, and the complex interaction between these phenomena" (Helie, et al., 2011).

Another important cognitive attribute is a decision-making confidence level. CLARION uses confidence levels to determine if a possible decision is sufficient. The

parameter search for the best performing confidence level gives information about the problem. The researchers exemplified how the training process of a cognitive architecture can give explanatory information. CLARION was able to model the cognitive theories of the unpacking principle and ascertainment bias. The unpacking principle occurs in a medical setting when a doctor gives a patient their diagnosis before all pertinent information is revealed. CLARION can achieve the unpacking principle through setting the decision-making confidence threshold to a low value. Ascertainment bias occurs when the doctor's diagnosis is based on prior beliefs. It is a form of stereotyping. This form of subjective decision making is important to capture when dealing with cognitive architectures. Even though it can lead to a poor diagnosis, the goal is to capture the human-decision making process itself which is inherently flawed.

CLARION is a primary influencer of DPM-MN. DPM-MN implements similar ideas and functions as CLARION such as bottom-up learning through trial-and-error experiences, the temporal aspect of valence and preference, a minimalist structure, rule-based reasoning, the expectation for lower level functions to enable higher level emergent behaviors, and the use of a confidence level in support of decision making. Many of these ideas and functions also come straight from the dual-process theory. CLARION gives DPM-MN the benefit of a clear example and explanation of which dual-process theory parts to focus on.

2.5.2 *Qualia Modeling Framework.*

The dual-process theory directly influences the Qualia Modeling Framework (QMF) (Vaughan, et al., 2016). It separates System 1 as an imitation of the autonomous mind while System 2 is an imitation of the reflective mind. The model utilizes two separate

ACT-R models to build the System 1 and System 2 implementations. A new input first encounters System 1. If the model provides a suboptimal answer, the input is transformed into a new state space using hypernetwork mathematics. After the state space transformation, the System 2 ACT-R model passes through the input. The final prediction accepts the resulting answer and uses it to update the System 1 autonomous ACT-R model.

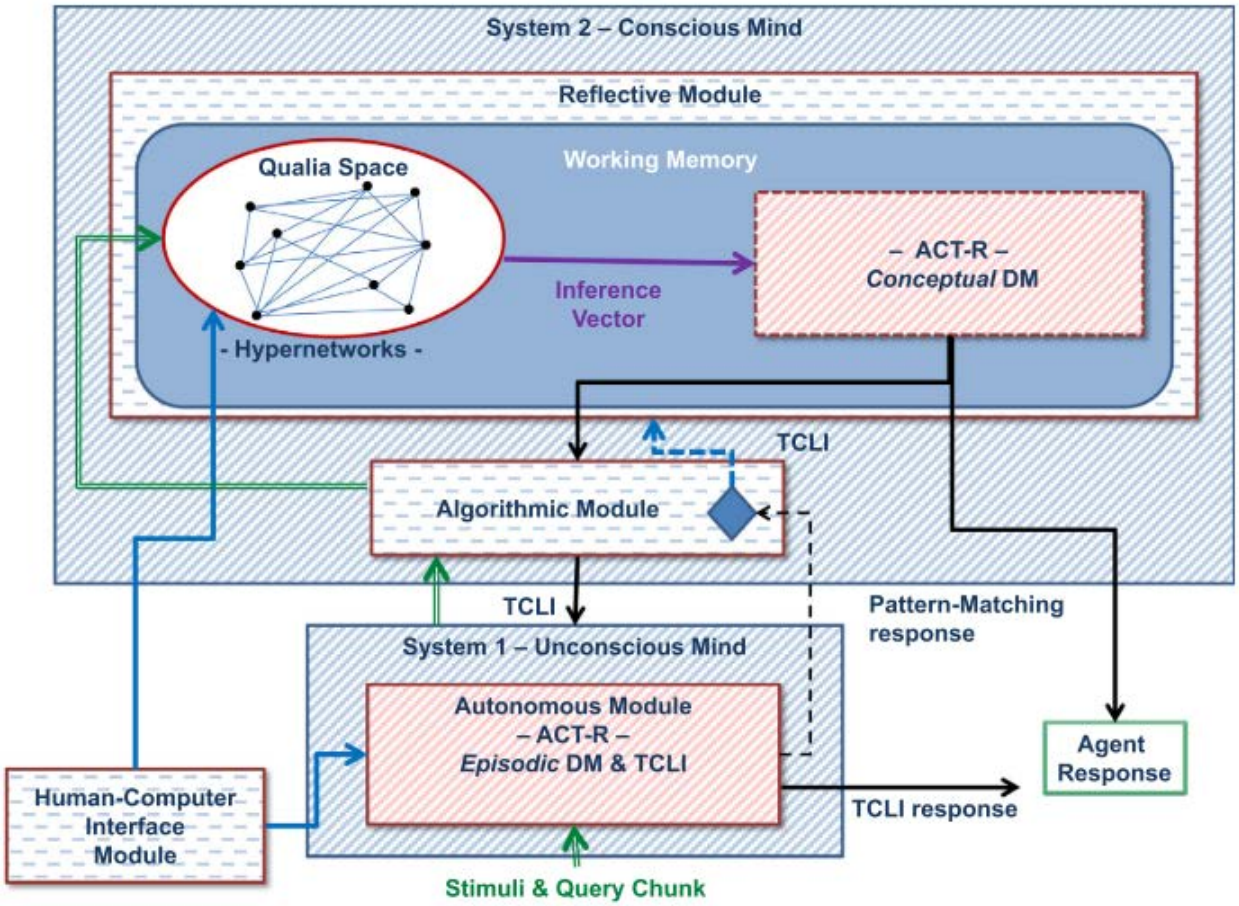

Figure 3: High-level representation of QMF (Vaughan, et al., 2016).

QMF possesses multiple unique properties that may be useful for other cognitive architectures. Vaughan, et al. believe experiences can model consciousness and qualia. In other words, data points representing distinct experiential points in time are sufficient enough to simulate the consciousness used in human decision making. Other cognitive

architectures often use a similar approach, but do not explicitly state this point. Dual-Process theory as the inspiration for the cognitive architecture serves as another explicit statement in the research paper. Psychological research inspires many other cognitive architectures, such as CLARION, that have a framework similar to the dual-process theory; however, researchers do not directly label it as a dual-process theory topology.

QMF also tries to serve as a generic structure for many different problems. As a result, QMF is flexible with its input and output. The target attribute and number of dimensions in the data can change in real time. Also, the model can still process data with incomplete feature information. QMF can look for spatial and temporal relationships. Transfer learning is possible for quick implementation. Humans can make use of previous knowledge for new tasks or new domains. This idea influenced QMF's attempt at model generalization.

Two other crucial components of a dual-process theory inspired cognitive architecture model are found in QMF: the algorithmic mind and real-time training. System 1 and System 2 are already known as the subsystems of the dual-process theory, but QMF adds one more with the algorithmic mind. The intended purpose of the algorithmic subsystem is to create a process for interaction between System 1 and System 2. The algorithmic module in QMF primarily acts as the gate-keeper of information flow. It decides whether the prediction consults System 2 or not. This can save computational time and increase speed since the information does not always reach System 2. The real-time training aspect of QMF imitates the continuous development of consciousness in a human. There is no point where a person stops learning. A cognitive architecture should be able to

take in experiences one at a time to constantly update rather than only functionalizing past data for current inference.

Even though QMF succeeds in creating an engineering advantage, confirmed during a malware classification experiment, it poses one major flaw in shaping itself after a dual-process theory. QMF can be flattened out and function exactly the same way. System 2 acts as a backup inferencing model in case System 1 fails rather than having a dependent interaction between the two. QMF is contrasted by CLARION which creates explicit concepts in System 2 through the build-up of singular instances in System 1. CLARION also has direct activation ties between the two sub systems.

QMF provides DPM-MN with the notion that human consciousness can be modeled through experiences. This is an important component since DPM-MN begins and builds its knowledge base solely on datapoints that represent experiences. Additionally, DPM-MN utilizes an algorithmic mind similar to QMF. System 1 and System 2 complete independent tasks within their own system, and then the algorithmic mind determines the effect each system has on the other. DPM-MN also imitates QMF by intending to operate in a real-time manner for the purpose of continuous learning. DPM-MN assembles the cognitive functions with the expectation of a constant flow of input.

2.5.3 *Bindewald Clustering-Based Online Player Modeling.*

Bindewald, et al. created the Clustering-Based Online Player Modeling (CBOP) (Bindewald, et al., 2017) approach. This model type initially develops a state-trajectory mapping through clustering. A weighting algorithm dependent on certain criteria such as existing cluster population and cluster variance then updates the model.

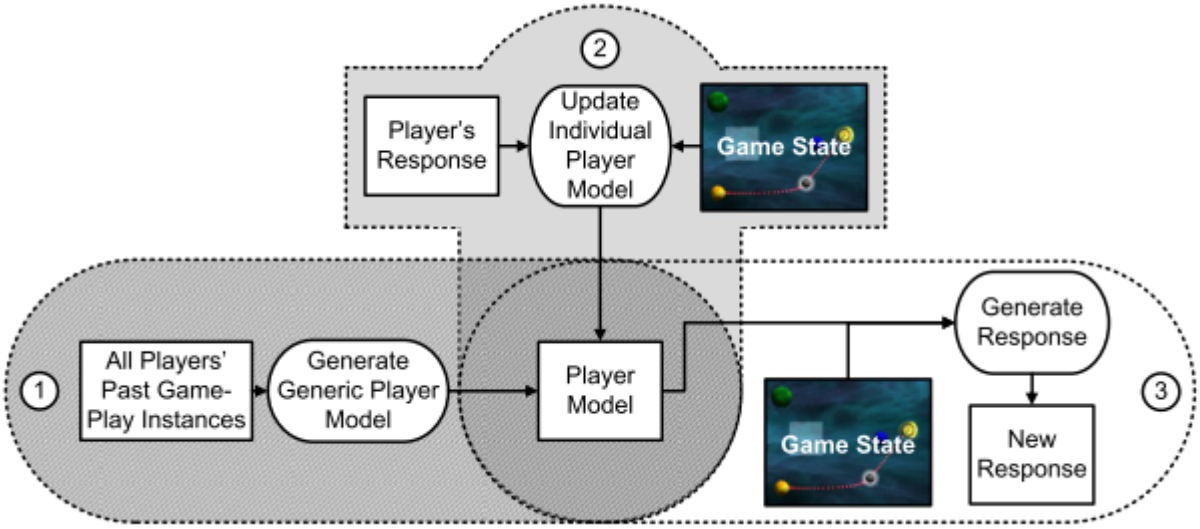

Figure 4: CBOP paradigm (Bindewald, et al., 2017).

CBOP is a predominantly relevant algorithm since it attempts to solve the same problem as DPM-MN. The data gathering, and data processing steps are very similar to the DPM-MN steps. Additionally, CBOP provides trajectories as whole items instead of providing the response point-by-point (Bindewald, et al., 2015). CBOP predicted the *Space Navigator* trajectories with a mean Average Coordinate Distance (ACD) of 0.2036 using specific player modeling.

Through their research, Bindewald, et al. discovered a few issues with the *Space Navigator* problem. First, the state-space representation could be more detailed. Some probably important features are left out such as the trajectory of other ships. Second, the experimental participants seemed to loosely keep a strategy. Given the same scenario, they would not draw the same trajectory. Finally, the CBOP model's learning mechanism did not necessarily change the underlying state-response mapping. It acted more as an update to the foundational model rather than a transition due to concept drift. As a result, any learning was minimally captured (Bindewald, et al., 2015).

CBOP's clustering methods motivated DPM-MN to take advantage of k-means clustering to find similar datapoints to make up the underlying distribution of a new System 2 concept. Also, CBOP found the importance in viewing the *Space Navigator* trajectories as a whole. This influenced the idea in DPM-MN to treat the responses as classes with attached prediction representations.

2.6 Summary

The lessons learned and discoveries found during these related research undertakings motivated DPM-MN. Overall, the dual-process theory cognitive architecture approaches share similarities with regard to the structure and the goal. All the models attempt to simulate human cognition through the distinction of System 1 and System 2 processes. There may be disagreements on the exact algorithm features or what is important, but each method has its own special way of trying to provide cognitive capabilities.

The paramount research related to DPM-MN includes the functionalization of cognitive processes, previously created models to actualize the functions, and research that directly motivates the DPM-MN architecture. Concept drift and novelty detection are critical cognitive functions specifically targeted during the inception of DPM-MN. Cognitive architectures in general, and those focused on social cognition or the dual-process theory, serve as completed examples of human decision-making models. Furthermore, cognitive architectures with a dual-process theory core such as CLARION, QMF, and CBOP are recent models with goals comparable to DPM-MN.

Chapter 3. Dual-Process Model using Multivariate Normal Probability Density Functions

The Dual-Process Model using multivariate normal probability density functions (DPM-MN) is a multifaceted algorithm. It has several interwoven functionalities to produce an outcome similar to the cognitive processes of a person (Appendix C). DPM-MN constructs a shared mental model (SMM), evaluated by prediction similarity, through explicitly stated cognitive learning functionality. The DPM-MN algorithm enables memory preservation, balance between implicit and deliberate systems, aggregation of experiences to form a general concept, online learning, overwriting previous knowledge, retaining past experience memory, concept drift, and outlier sensitivity.

This chapter first provides an overview of the DPM-MN algorithm. Then, the chapter explains each individual component of DPM-MN. These components include the refreshing of System 2 concepts, the addition of new points to System 1, the creation of a new System 2 concept, the windowing function, the overwriting of another concept in System 2, the revocation of points from System 2 to System 1, and concept drift. Finally, Chapter 3 illustrates examples of novelty detection, prediction using System 2 abstraction, and inference.

3.1 DPM-MN Overview

Figure 5 shows an overview of the DPM-MN model. The dotted line represents the learning path. The input can consist of multiple datapoints to reduce the computational cost. Once System 2 determines insufficient predictions of the input, System 1 updates with those insufficient predictions. System 1 discovers new generalized concepts to place into System 2. If System 2 needs to remove previous concepts to accommodate the newly

discovered concepts, a fraction of the underlying points that made up the newly deposed concepts return to System 1 to promote long term memory. The interrelation unit handles the specific interaction between System 1 and System 2.

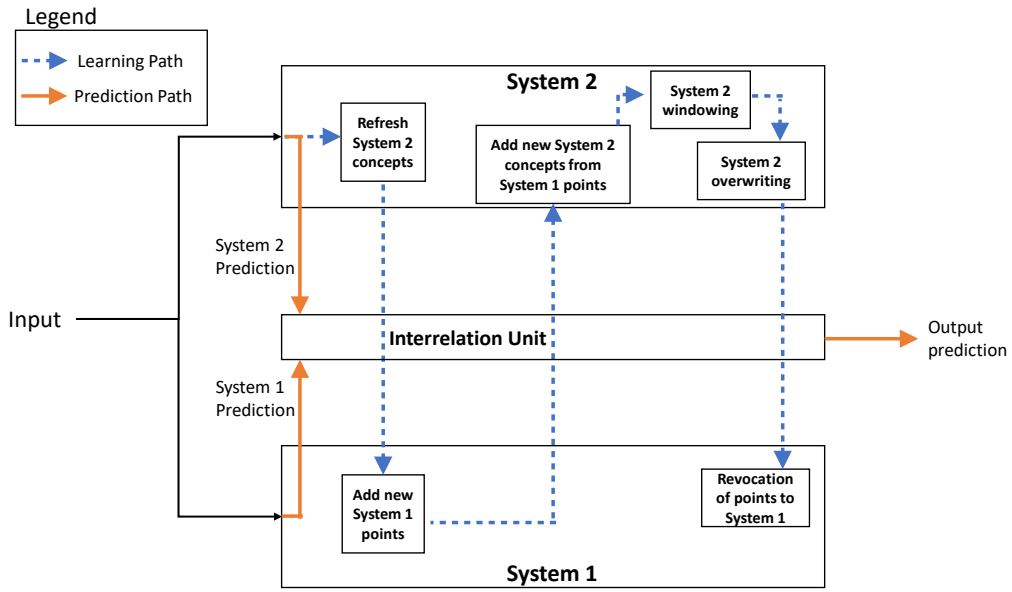

Figure 5: DPM-MN overview.

The solid arrow path shows the prediction process used for validation and testing. The input is simultaneously inserted into System 1 and System 2. Each system provides their prediction along with a confidence value. Within each system, the chosen prediction is the class with the highest confidence value. After each system predicts the input, the most confident of the two systems determines the final prediction that is ultimately output from DPM-MN.

3.2 DPM-MN Learning Characteristics

The DPM-MN algorithm builds an SMM via dual-process theory learning characteristics. These learning characteristics include memory preservation, balance between implicit and deliberate systems, aggregation of experiences to form a general

concept, online learning, overwriting previous knowledge, retaining past experience memory, concept drift, and outlier sensitivity.

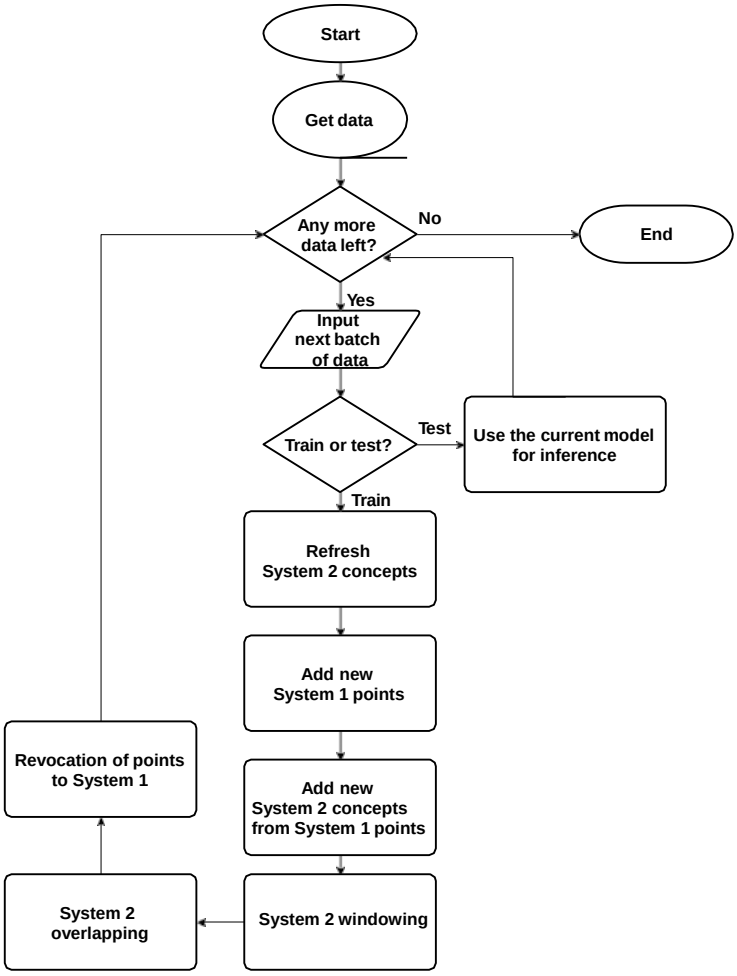

Figure 6:DPM-MN flowchart.

Figure 6 is the flowchart for DPM-MN at a high level. This chapter explains all of the processes. The training path in the flowchart is made of the various cognitive-like functions. DPM-MN allows additions or subtractions from the training path functions. It has a modular design so DPM-MN can easily accommodate any dual-process theory implementation change.

Algorithm Dual-process model using multivariate normal probability density functions (DPM-MN)

input: C_t = confidence threshold; E_t = entropy threshold; N_c = number of clusters during k-mean clustering; S_t = size threshold; S_{one} = system 1 influence; S_{two} = system 2 influence; W_s = system 2 window size
initialize: *system1Points* = empty; *system2Concepts* = empty; *data* = *Space Navigator* data
for *point* in *data* **do**
  if Training **then**
    $refreshThreshold = \frac{C_t}{W_s}$ >(Refresh System 2 Rules)
    if ((*concept* ∈ *system2Concepts*) > $refreshThreshold$) ∧ (*concept* is max prediction confidence for class group) **then**
      refresh *concept*
    if (*point* prediction < C_t) ∨ (*point* prediction is wrong) **then** >(Add new System 1 points)
      add the point to *system1Points*

    kMeans ← the *system1Points* groupings after k-means clustering where $k = N_c$ >(add new System 2 concepts)
    newConcepts = empty
    for *group* ∈ *kMeans* **do**
      groupSize = *size*(*group*)
      $groupEntropy = \frac{mostFrequentClass}{groupSize}$
      if (*groupSize* > S_t) ∧ (*groupEntropy* > E_t) **then**
        mostFreqPoints = points ∈ *group* that belong to the most frequent class
        add *mostFreqPoints* to *system2Concepts*
        remove *mostFreqPoints* from *system1Points*
        newConcepts ← *mostFreqPoints*

    rebukedPoints = empty
    numRevoke = integer value from $(S_t \times (\frac{0.5}{W_s}))$
    for *conceptClass* ∈ *sys2Concepts* classes **do** >(System 2 Windowing)
      if *size*(*conceptClass*) > W_s **then**
        window the older concepts in *conceptClass*
        rebukedPoints ← *numRevoke* of the points in the windowed concepts that are closest to the distribution mean

    for *newConcept* ∈ *newConcepts* **do** >(System 2 Overlapping)
      if overlapping of other concept happens (determined by C_t) **then**
        remove the overlapping points of the other concept from *system2Concepts*
        rebukedPoints ← *numRevoke* of the overlapped points of the other concept

    system1Points ← *rebukedPoints* >(Revocation of points to System 1)

  if Inference **then**
    system1PredictionClass, *system1PredictionConfidence* = *point* prediction from *system1Points* using S_{one}
    system2PredictionClass, *system2PredictionConfidence* = *point* prediction from *system2Concepts* using S_{two}
    if *system2PredictionConfidence* > *system1PredictionConfidence* **then**
      finalPrediction = *system2PredictionClass*
    else
      finalPrediction = *system1PredictionClass*

Algorithm 1:DPM-MN algorithm.

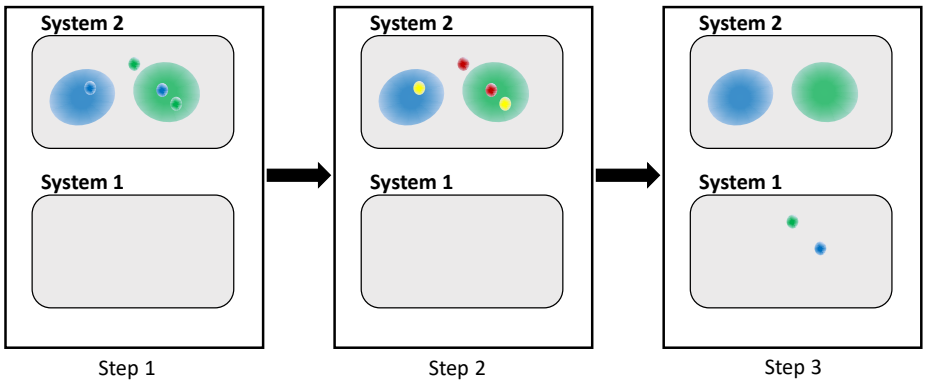

Figure 7: Addition of new points to System 1.

DPM-MN begins in a *tabula rasa,* or “blank slate”, state. This initial state conforms to the dual-state process model traits described by Sun (Sun, 2004). Specifically, DPM-MN maintains a minimalistic approach that grows through bottom-up learning. Beginning in a *tabula rasa* state empowers DPM-MN to learn new concepts through new experiences rather than beginning with a presupposed knowledge base.

The addition of new points to System 1 maintains a balance between the implicit and deliberate systems. Figure 7 shows how DPM-MN learns from new datapoints. The individually illustrated datapoints represent a new input and their color reveals their true classification. Step 1 shows the position and class of each of the points from the new input. Step 2 highlights the confidently correct points yellow and highlights the new System 1 points red. The confidence level and the correctness are the basis for the new System 1 points. The new System 1 points include a point that is correctly classified but has a low confidence (the green point turned red), and a point with high confidence but incorrectly predicted (the blue point turned red). Another way to think about the process in Figure 7 is to identify which new points are not highly confidently correct. The points that are not correct with high confidence become new System 1 points. Through this training process, DPM-MN maintains the correct concepts learned in System 2, while building the potentiality of new concepts in System 1.

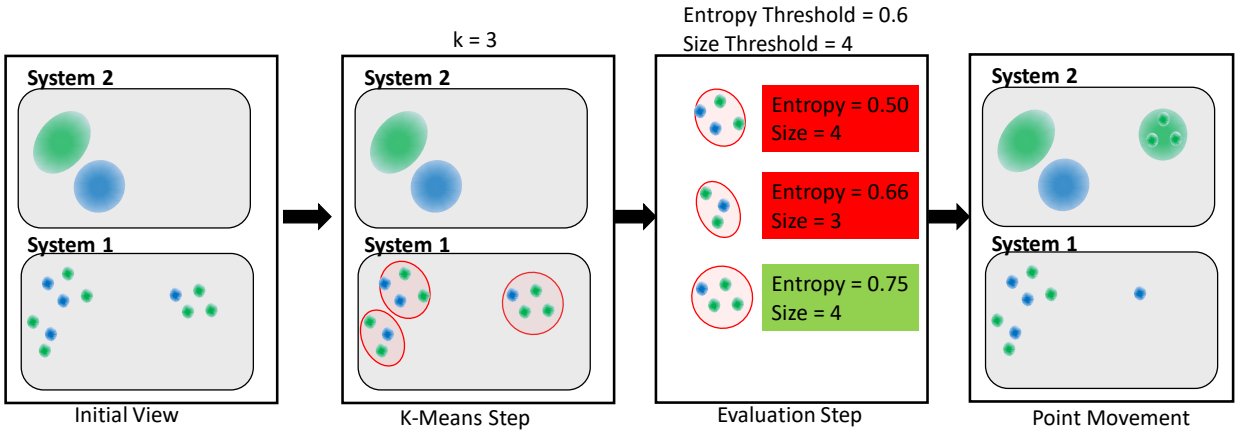

Figure 8: Creation of a new System 2 concept.

Bottom-up learning, or the aggregation of experiences to form a general concept, inspires the method of adding a new concept to System 2. Once enough outlier experiences occur, the new state-response pairings individualized in System 1 can become a generalized concept in System 2.

Figure 8 presents the formation of a new System 2 concept. The initial view is first shown. After that, System 1 groups individual points together using k-means clustering where k is equal to the number of clusters (N_c) hyperparameter. In this case, it is three. Once the clustered groups are assigned, System 1 tests each group for their size and their consistency. The size must at least be the size threshold (S_t) and the consistency must be at least the entropy threshold (E_t). The number of individual points in the group determines the size. The percentage makeup of the most frequently occurring class determines the consistency. If both the thresholds are met, the points belonging to the most frequently occurring class compose the underlying distribution for the new System 2 concept's Gaussian probability density function.

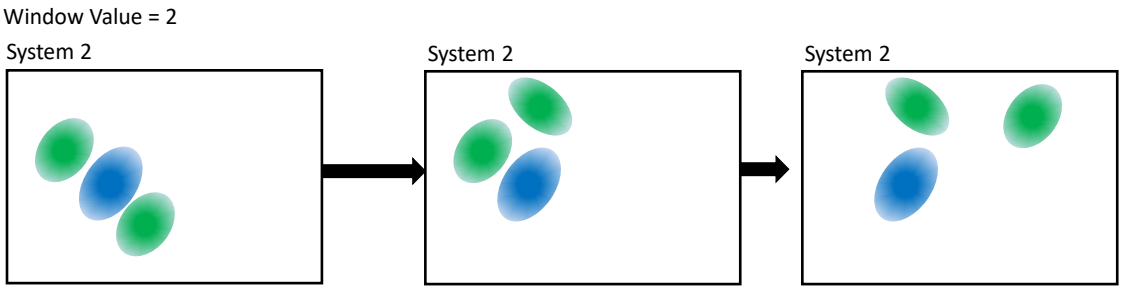

Figure 9: Windowing.

The windowing functionality in DPM-MN caters to an online learning environment. It also helps capture the balance between old concepts and new concepts in memory. The number of concepts per class in System 2 depends on each unique SMM's optimal learning style.

Figure 9 displays the idea of windowing for System 2 concepts. This is a memory controlling process. If the windowing hyperparameter is a large value, System 2 exhibits long-term memory. Windowing gives DPM-MN an ability to favor recent concepts over older concepts. The process acts as a conventional sliding window. When it comes time to remove a concept, the oldest concept is forgotten.

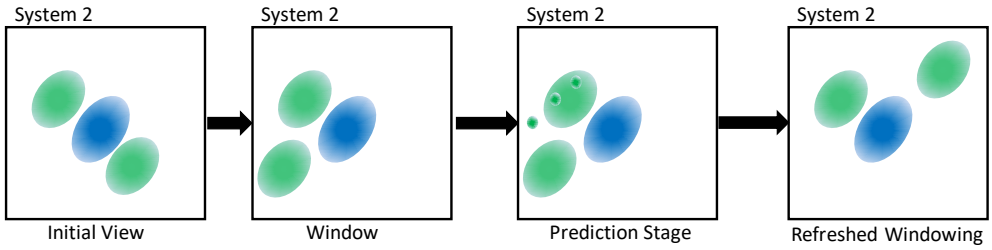

Figure 10: Refreshing of System 2 concepts.

Figure 10 exhibits an additional memory preservation function. System 2 remembers concepts that still hold relevance. A concept's frequency of being the correct prediction source determines the amount of relevancy. The prediction stage in Figure 10

shows the green concept on top as the provider for the correct prediction. Each training stage, System 2 refreshes a new concept from each class if they are sufficiently confident. In Figure 10, the top green concept meets the requirements to be refreshed. As a result, System 2 pushes the top-left green concept to the front of the window for the green class. Subsequently, the other green concept moves twice in a row.

Windowing and refreshing maintain the important memory in System 2. System 2 uses new samples to determine the relevancy of the System 2 concepts. With the addition of new System 2 concepts through aggregation of unique experiences in System 1, System 2 'forgets' older unused concepts while refreshing the correct concepts.

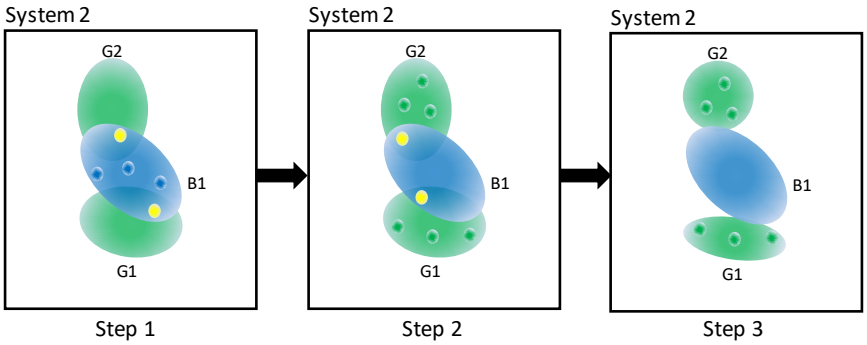

Figure 11: Overwriting of another concept.

Figure 11 illustrates the overwriting of previous knowledge process in System 2. This process is an algorithmic implementation synonymous to the decision-making function of changing a routine. Given the same scenario, System 2 can update an old concept for decision-making to a new concept. If this process is not implemented, old behavioral concepts do not approprietaly deteriate and persistently interfere with newly incoming concepts.

Step 1 shows the process of taking the new concept's (B1) underlying distibution, and testing them against all existing concepts (G1 and G2). The highlighted points are the

B1 underlying points which would trigger a confident guess from an existing concept. Since G1 and G2 both trigger a confident response to at least one underlying point from B1, System 2 tags them as possible existing concepts to overwrite. Step 2 shows the reverse occuring: System 2 tests the existing concepts' underlying points against the new concept. System 2 removes any points from G1 or G2 which B1 confidently responds to.

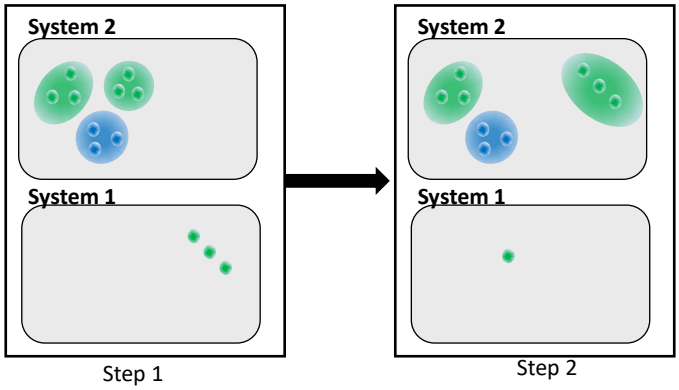

Figure 12: Revoking.

Figure 12 shows the revoking process which retains past experience memory. Assume a windowing value of two. In Step 1, the points in System 1 are able to become a concept. Because they become a new concept in System 2, as illustrated in Step 2, a windowing effect occurs. The windowing effect deposes one of the green concepts. As a result, System 2 tags its underlying points as subject to revocation. In this case, System 2 only revokes one of the underlying distribution points into System 1.

The overall number of points that make up the underlying distribution of a System 2 concept determines the number of points that are revoked. The number of revoked points also depends on the window size (W_s) because if concepts from the same class are quickly being windowed and a lot of points are revoked, it could be possible for a thrashing effect to occur where outdated System 2 concepts create new System 2 concepts through the

buildup of revoked points existent in System 1. Finally, the most central points of the dismantled concept are the points chosen for revocation into System 1.

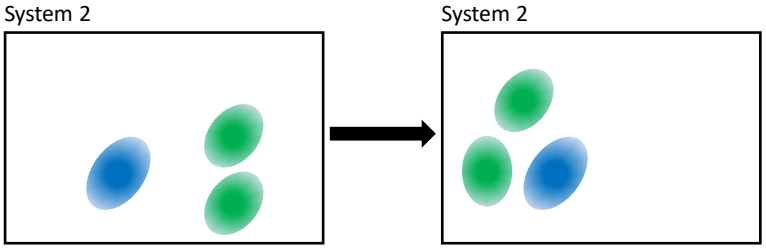

Figure 13: Concept drift.

Figure 13 displays concept drift via the green concepts. Over time, the green concept response is given in different scenarios. System 2 intrinsically detects the concept drift through windowing. The detection of concept drift is important because humans learn. Their ideas and behavior are not static. Concept drift indicates a change of behavior in a person. With most learning tasks, people begin with a rudimentary understanding and strategy. As time passes, they will learn about the problem and update their strategy to become more optimal.

3.3 DPM-MN Behavior Examples

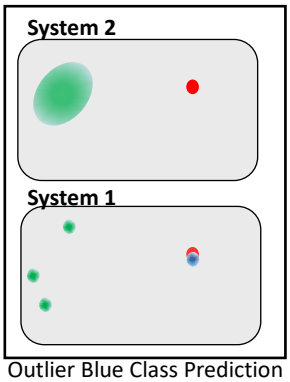

Figure 14: Novelty detection example.

Figure 14 is a simple example of DPM-MN's ability for novelty detection. The red point represents a new input that needs a prediction. In this scenario, the blue class infrequently occurs so it only shows up in System 1. However, a nearly similar situation to the previous blue class response's situation has occurred. An effectively zero confidence prediction is given by System 2, but System 1 provides a highly confident blue class prediction since this anomalous situation has happened before.

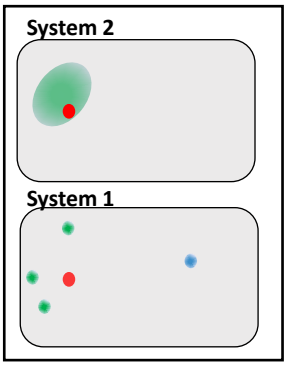

Figure 15: Abstract answer example.

By creating concepts, System 2 can determine situations where an intuitively unknown answer arises. Figure 15 is an example that shows System 2 providing a concept-based answer to the alternatively unknown situation. Even though System 1 previously held datapoints that would have been close to the new datapoint, those System 1 datapoints create the System 2 concept distribution. As a result, System 1 is less crowded and allows for more precise novelty detection and better prediction competition. Furthermore, System 2 represents a generalized prediction mapping of the problem.

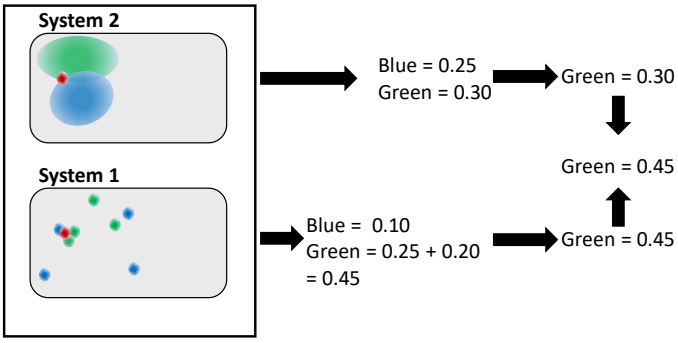

Figure 16: Inference.

Figure 16 illustrates the inference of points during validation and testing. The red datapoint indicates a new input with an unknown truth value. System 2 outputs two different confidence readings from two concepts with different classes. System 1 also outputs two competing confidence readings, but the green class involves two significant green class influencers. Each influencing point in System 1, or concept in System 2, provides its own confidence value. DPM-MN adds the confidence values from the same class to form a full confidence value for the class prediction. Once the System 1 and System 2 predictions with their associated class are provided, DPM-MN compares the confidence values from each system. The higher confidence level determines the final prediction. In this case, the red point is predicted (with 0.45 confidence) as a green class response.

3.4 Summary

This chapter presented the DPM-MN algorithm via illustrations and description. The individual functionalized cognitive functions are independently introduced. A high-level perspective of DPM-MN and an overview of the interconnection between processes is shown. With an awareness of the DPM-MN functionality, the following chapters are set in context.

Chapter 4. Methodology

The Dual-Process Model using multivariate normal probability density functions (DPM-MN) is expected to at least exhibit the dual-process learning characteristics of concept drift, overwriting of previous concepts, windowing, and refreshment of frequently needed concepts. A notional dataset specifically designed with these characteristics in mind is tested to determine effectiveness. The optimal hyperparameters empower the desired capabilities.

A second test evaluates DPM-MN on a human user dataset with data gathered through the *Space Navigator* experiment. *Space Navigator* is proposed as a shared mental model (SMM) system to specifically evaluate the performance of DPM-MN to learn the participants' mental models. DPM-MN performing well on the *Space Navigator* trajectory predictions is akin to proper mental modeling of the players. The baseline tests are also attempting to map the players' mental models. The straight-line predictor and the medoid predictor act as simplistic SMMs. The *Space Navigator* tests are conducted in two ways: the individual player test and the generic player test.

4.1 Notional Dataset Experiment

The experimental goal of the notional dataset is to ensure DPM-MN can handle learning situations that involve the dynamic temporal dual-process learning characteristics of concept drift, overwriting of previous concepts, windowing, and refreshment of frequently needed concepts. The first evaluation of DPM-MN leverages a notional dataset specifically designed to test the dual-process learning cognitive functions within the DPM-MN architecture.

4.1.1 *Notional Environment/Data.*

The notional dataset is made of multiple conceptual steps. Each step contains multiple class groupings synthesized through normal distributions. Within every step, the individual datapoints are randomly shuffled to imitate response variation. The notional dataset is two-dimensional so it easily can be visualized for understanding.

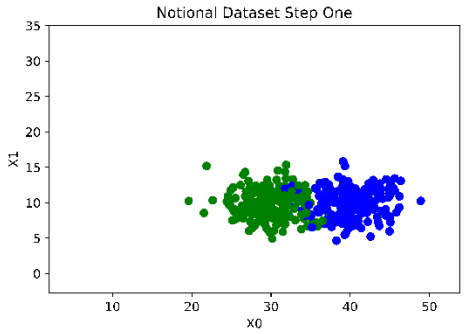

Figure 17: Notional dataset step one.

Step one in the notional dataset creates two mostly separated classes. DPM-MN will quickly make a concept of these two distributions.

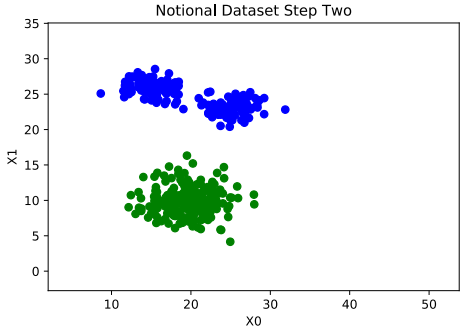

Figure 18: Notional dataset step two.

Step two is a concept drift of the blue class. The blue class moves from the right of the green class to above the green class. DPM-MN will either remember both blue class distribution locations or reject the first blue class distribution via the window functionality in favor of the new blue class concept.

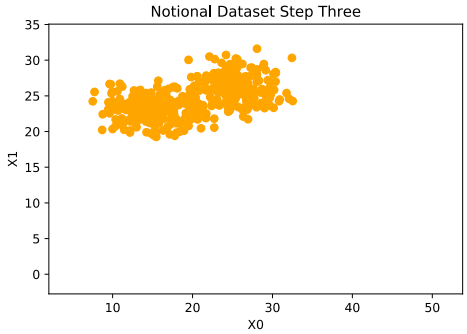

Figure 19: Notional dataset step three.

Step three eventually forces the orange class to become the predominant class at the location shared with the newest blue class concept. If necessary, the orange class overwrites the blue class.

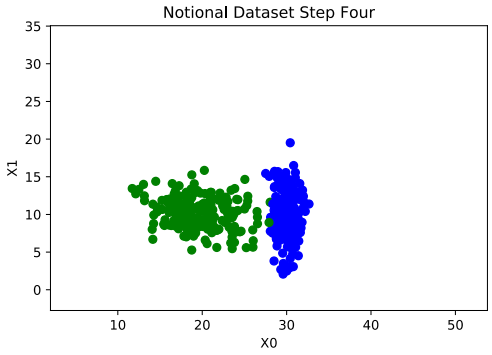

Figure 20: Notional dataset step four.

The blue class returns to its original location once the orange class removes it from its current location.

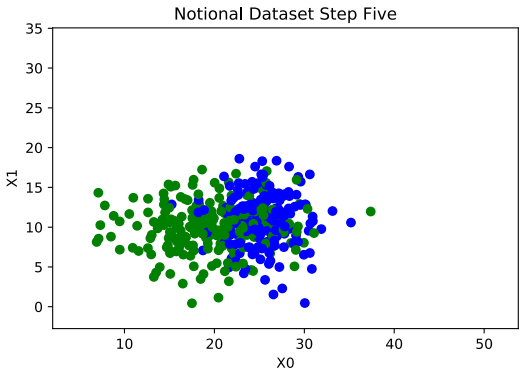

Figure 21: Notional dataset step five.

Finally, the blue and green classes start mixing closer together to provide any machine learning algorithm with more difficulty. The DPM-MN algorithm will adapt to the narrowing of space between the means of the green class and the blue class.

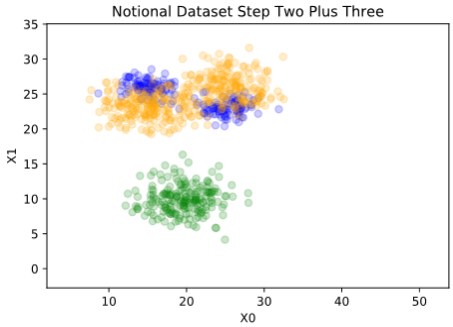

Figure 22: Notional dataset step two and three.

Figure 22 illustrates steps two and three combined. The orange class and blue class overlap is shown. The different class distributions are not completely on top of each other, but they are extremely close.

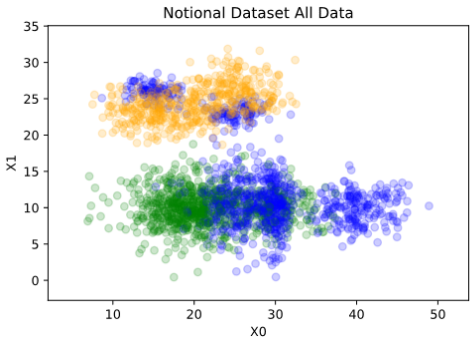

Figure 23: Notional dataset all data.

All of the steps together show a difficult classification problem. If the data is not processed temporally, it is a more difficult task than if characteristics such as concept drift are considered.

4.1.2 *Notional Test Strategy.*

DPM-MN and a support vector machine (SVM) train, validate, and test on the notional dataset. DPM-MN, a temporal algorithm, has an advantage over the SVM because the SVM trains on all of the data at once.

Bayesian optimization determines the hyperparameters for both DPM-MN and the SVM. The DPM-MN Bayesian optimization search space is the same as the search space for *Space Navigator* to allow comparison of optimal hyperparameters between the notional dataset and the *Space Navigator* dataset. The Bayesian optimization algorithm searches between 0.001 and 20 for the SVM penalty parameter *C* of the error term. Bayesian optimization also searches between 0.001 and 20 for the kernel coefficient *gamma*. Finally, the search algorithm also optimizes a decision between using a linear kernel or a radial basis function (rbf) kernel.

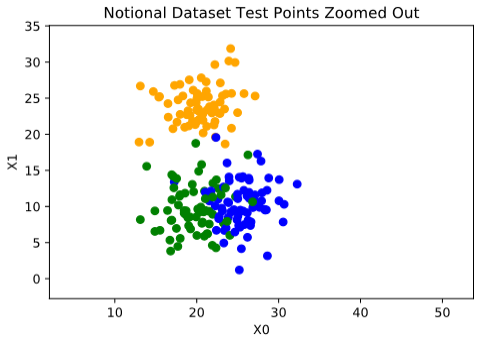

Figure 24: Test points zoomed out.

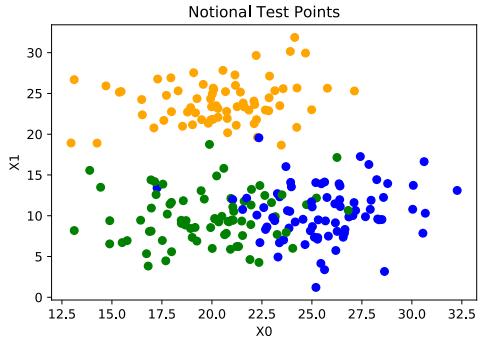

Figure 25: Test points zoomed in.

Table 1: DPM-MN parameter optimization.

Parameter	**Value Range**
C_t	0.001 to 3.0
E_t	0.05 to 1.0
N_c	2 to 50
S_t	2 to 50
S_{one}	0.1 to 2.0
S_{two}	2.0 to 10.0
W_s	2 to 50
Number of trajectory classes	20
K-folds	5
Train/Validation and Test data split	90%/10% (of all the data)

Train and validation data split	75%/25% (of the train/validation data)
Bayesian optimization hyperparameters	10 random search points followed by 100 calculated search points

The training/validation/testing split of the notional dataset is the same as the *Space Navigator* experiment (Table 1). The points per batch iteration is switched to a more precise value of three, instead of ten, since the computational complexity is less for the notional dataset. Figure 24 and Figure 25 show the test points from two different perspectives. The ideal final concept model determined the test points. At the end of DPM-MN, there will exist a System 2 concept rule at each of the three unique distributions corresponding to the correct class.

4.1.3 *Notional Measures.*

Classification accuracy is the test metric for the notional dataset experiment. It is a percentage out of 100%. A 100% is perfect classification accuracy. Because of the synthesized class groupings, each algorithm will test closer to 100%. However, the intentional overlap of distributions at the end will be difficult for both algorithms. Additionally, for the SVM algorithm, the orange class and blue class overlap will cause a prediction difficulty.

4.2 Space Navigator Dataset

This experiment reveals the success of DPM-MN in using cognitive learning functions to better predict human responses via an SMM. *Space Navigator* is a strategic game environment where a decision-making mental model is developed. DPM-MN will determine the mental model of the players. The mean average coordinate distance (ACD) test metric correlates to the extent DPM-MN correctly maps the mental model because

SMM can be tested through output similarity. DPM-MN takes advantage of dual-process theory learning characteristics to produce an SMM for more accurate user trajectory predictions.

4.2.1 *Space Navigator Data.*

The raw data comes from an experiment completed in Bindewald, et al. (Bindewald, et al., 2015). Thirty-five players were tested in sixteen levels each on a game called *Space Navigator*. The player draws trajectories from a spaceship to a corresponding planet of the same color while avoiding obstacles and picking up bonuses. The recorded data includes the trajectory drawn along with the location of the destination planet, bonus points, and obstacle areas.

The data is preprocessed into a nineteen-feature input describing the state space (Appendix E) and an output class mapped to a trajectory response (Bindewald, et al., 2015), (Appendix J). The test metric is the average Euclidean distance per point error. The average Euclidean distance per point error is also called the Euclidean error or the average coordinate distance (ACD). See Appendix H for details on the error calculation given a predicted trajectory and a true trajectory. See Appendix E for an in-depth walkthrough of the data preprocessing steps.

The experiment was conducted on a tablet computer. Participants used their finger to draw trajectories. The game itself was created in *Unity* game engine version 2017.1.0f3.

The data processing and analysis happened in *Jupyter Notebook*. The hyperparameter search and model building was conducted using *Eclipse* version 20171108 on Windows 7 operating system. The hardware was an intel Core i5 CPU at 2.60GHz and

8GB RAM. The most prominent libraries included *NumPy* and *Scikit-learn.* For the computer language, *Python 3.6* was used.

Two tests are performed. The first test treats each player as an individual dataset to train, validate, and test. The second test groups all of the players' data together to train, validate, and test as a collection. Both tests include a comparison to results in Bindewald, et al. (Bindewald, et al., 2017).

4.2.1.1 *Irreducible Error in Space Navigator.*

Irreducible error are data errors that are due to inherent variability of the data and for which other measurement or fitting could not remove. There are two sources of irreducible error in the Space Navigator data. The first source of irreducible error comes from the trajectory representation for each class. Because the medoid of each class cluster becomes the trajectory representation when that class is predicted in DPM-MN, error will still exist even if the DPM-MN class accuracy is one-hundred percent. This irreducible error can be pre-measured though by taking each observation and determining the Euclidean error relative to the matching class representative trajectory.

The second source of irreducible error is due to the fact that humans are non-deterministic. Their behavior may suddenly change for no apparent reason whatsoever if revisiting a state. If the *Space Navigator* screen is the exact same during two different instances, a person may decide to draw an up-curve during the first encounter and a down-curve during the second encounter. If the same person the DPM-MN model was trying to mimic were to make every trajectory prediction, they would still achieve a certain amount of error. This irreducible error exists, but there still exists an underlying pattern. It is the

reason people have attributable personalities rather than everyone existing as randomized robots.

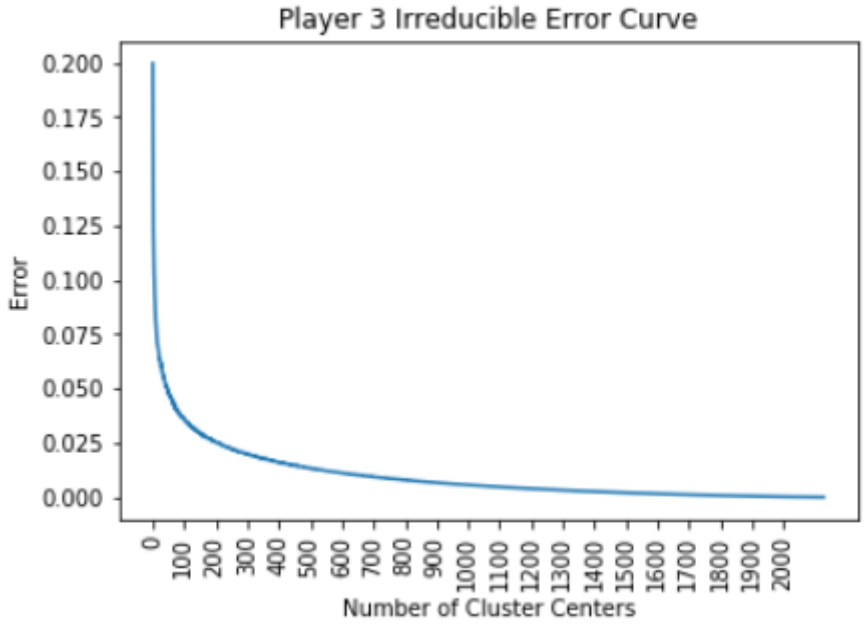

Figure 26: Example of irreducible error curve.

DPM-MN uses classes that correspond to a trajectory representation. When DPM-MN predicts a specific class, the response supplies the trajectory representation. The number of class trajectory representations is manually set. For each experiment, there were twenty different trajectory classes. The number of trajectory classes needs to be selected to reduce irreducible error, but at the same time allow DPM-MN to effectively learn. A high number of trajectory classes greatly reduces the irreducible error and introduces a diverse sample of trajectory representations, but it becomes more difficult to learn System 2 concepts because similar trajectory representations will have different class labels.

Assuming a perfect class prediction accuracy finds the irreducible error. Inherently, there will exist some irreducible error since DPM-MN can guess the correct class, but the true trajectory will still slightly be different from the trajectory representation. Finding the

irreducible error at each point of a parameter sweep of the number of cluster (N_c) centers for finding the trajectory representations creates Figure 26. The N_c centers, in this case, is the same as the number of trajectory class representations. Figure 26, created from an individual player, helps to understand the relationship between irreducible error and the number of trajectory representations. The relationship determines a point that fulfills the necessary balance. As seen by Figure 26, the exponential relationship converges to zero error once the number of trajectory representations equals the number of trajectories present.

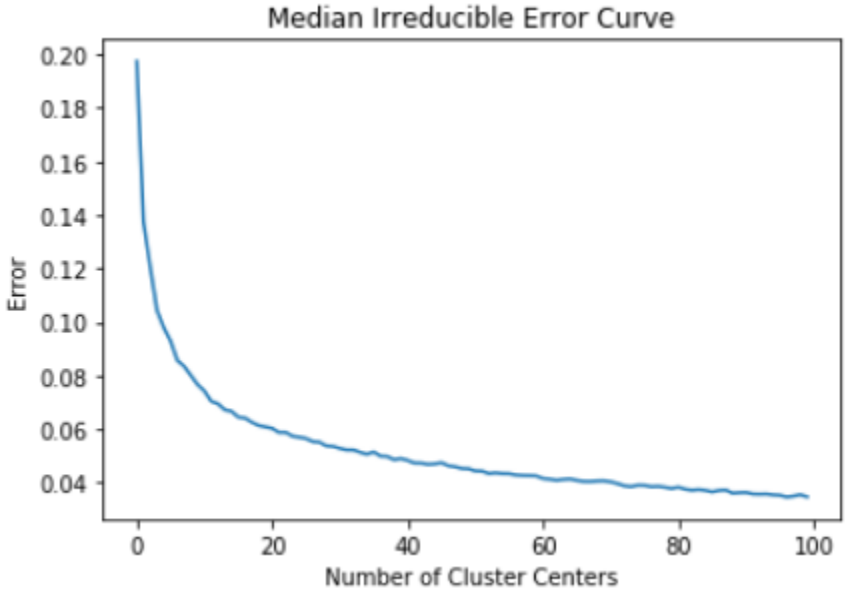

Figure 27: Median irreducible error curve of all players.

Figure 27 shows the median irreducible error curve. It only goes up to one-hundred because after about fifteen classes, the irreducible error reduced per additional cluster center drastically reduces.

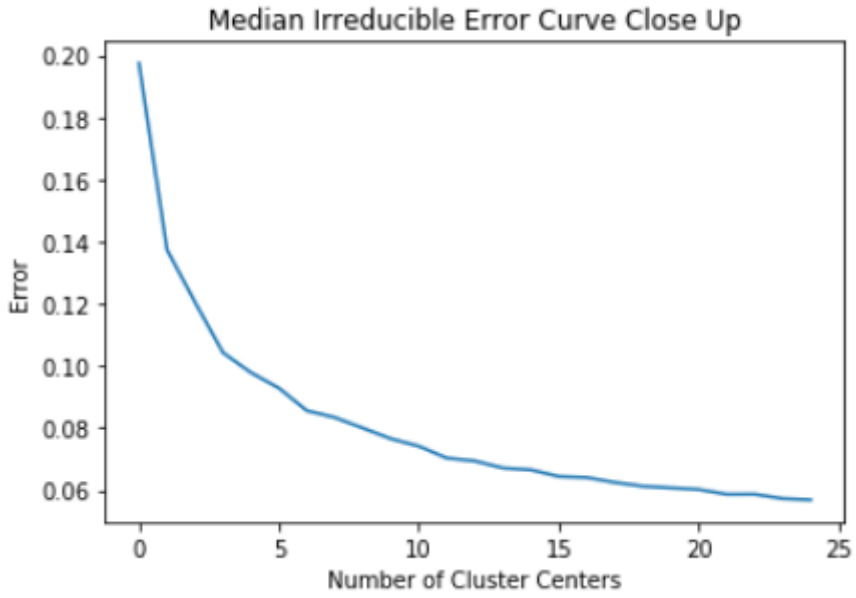

Figure 28: Close up of median irreducible error curve.

Figure 28 zooms in on the beginning values of Figure 27. The initial, and predominant, drop-off of error happens within the first five classes. This indicates a pattern of low variance in drawn trajectories. A handful of trajectory representations account for the irreducible error introduced through using a classification method.

4.2.2 *Hyperparameter and Validation Process.*

1) Number of clusters (N_c) during System 1 group sizing – When System 1 searches for large groupings, it is completed through clustering. This determines how many clustering groups are used in System 1 to split up the totality of points.

2) Size threshold (S_t) for System 1 groups to enter System 2 – To get into System 2 as a concept, the clustered groups must meet the size threshold to be determined as “big enough”.

3) Entropy threshold (E_t) for System 1 groups to enter System 2 – After System 2 determines “big enough” groups, the percentage of points of the same class must meet the entropy threshold.

4) Window size (W_s) for System 2 – The System 2 concepts of the same class can become too stale. Therefore, W_s determines how many concepts of one class

can exist in System 2 at the same time. If System 2 has six class one concepts, and the W_s is five, then System 2 revokes the oldest class one concept.

5) System 1 influence (S_{one}) – This is the smoothing factor of the multivariate normal probability distribution function. It determines the reach from a point. Because the state space is sparse in many areas if the influence is too small, increasing the influence will most likely improve the rate of guess acceptance from whichever system's influence DPM-MN raises.

6) System 2 influence (S_{two}) – This is the same idea as the S_{one} hyperparameter, but for System 2.

7) Confidence threshold (C_t) – This is one of the most important hyperparameters. It is the arbiter of the prediction confidence level. C_t determines if a new query should be added into System 1 or not. At zero, DPM-MN only places the wrong System 2 guesses into System 1. At a very high number, DPM-MN places almost all of the new queries into System 1. The confidence level also plays a part in deciding if a new System 2 concept overlaps an existing System 2 concept. A new System 2 concept uses the confidence as a way to measure which sublevel points overlap each other.

It would be better if DPM-MN determined C_t as a percentage out of one-hundred. Other possible alternatives included major problems. The infinite number of distribution possibilities disallows a static base number of the theoretical highest value. The maximum value always moves. It is possible to know the absolute max value when retrospectively analyzing existing data, but it is not an option when online data that is continuously arriving is meant to eventually become part of the process. An option is to scan all of the System 2 problem space to get a dynamic max value for every class. However, it is too computationally intensive – especially when talking about nineteen dimensions, many different classes, and an unknown boundary.

Another option is to judge the confidence based on the first closest point of the same class and the first closest point of a different class. This does not behave as intended when a new query has a high prediction value from a class because of the summation from

multiple distributions. It also becomes relative based on the distance between the two points being used to judge the confidence. Both of the judging points could be very far away, but because one class is relatively closer than the other, the confidence of the new point could return as high confidence even though it should be near zero.

The intention of the last two paragraphs is to show that a consistent percentage scale was diligently sought after. It would be easier to explain the algorithm functionality if it could be said, as an example, that new queries get placed into System 1 if they do not have a guess above a fifty percent confidence rating. Because of the mixture model format, the confidence level is relative to the problem space. Thus, C_t is a hyperparameter and a raw number rather than a percentage between zero and one-hundred.

There are three viable options to discern between: a restricted grid search, a Bayesian optimization search (Snoek, et al., 2012), and a genetic algorithm search. Bayesian optimization was ultimately selected to leverage the harmony between exploration and exploitation. Bayesian optimization can efficiently navigate the hyperparameter search space to counter the long computational time for a single hyperparameter testing of DPM-MN.

The Bayesian optimization attempts to guide the hyperparameter search in a correct direction depending on the previous hyperparameter search. Each hyperparameter search reveals information about the hyperparameter optimization probabilities. For example, imagine that C_t was set to ten and the following model returned a hypothetical error of one-hundred. Now, everything else is kept the same but C_t is set to five and the following model returned a hypothetical error of fifty. It is safe to assume that the next best hyperparameter guess would be to continue lowering C_t. Bayesian optimization also tries to balance the

benefits of both exploration and exploitation. The primary difficulties of Bayesian optimization are the challenging code implementation and the requirement to greatly narrow the search space so the exploration can be taken advantage of rather than turning into a random search.

During Bayesian optimization, a range of values must be supplied for the algorithm search space. If a non-integer value is chosen when the DPM-MN algorithm requires an integer value, such as size threshold because you cannot count half of a datapoint, the non-integer value is rounded to the nearest integer before being input into the algorithm.

Bayesian optimization carefully considers each hyperparameter's range to allow for various algorithm behaviors to emerge. One example is guaranteeing that S_{one} will be equal to or less than S_{two}. Since System 2 acts as a generalizer and System 1 acts as the anomaly finder, it does not seem intuitive for the System 1 points' kernel probability density estimations to be more smoothed than the System 2 points' kernel probability density estimations.

Another example is the N_c parameter during the System 1 group sizing. The range maxes at fifty because it is a little over twice the number of existing trajectory classes (twenty). The reason the N_c groups exists, is to find groups of individual observations that share the same class. If the data is strongly separated by class, the value of N_c should theoretically be the number of unique classes in System 1. In this testing instance, N_c should be between zero and twenty. The search is between two and fifty, inclusive, because it allows space on each margin of twenty (the number of unique trajectory classes possible) to explore possible algorithm behaviors. The Bayesian optimization algorithm may find it is best to have a quick flow of points between System 1 and System 2. In this case, the N_c

parameter is low, the entropy threshold (E_t) parameter is low, and the size threshold (S_t) parameter is low.

DPM-MN interleaves the supervised training and evaluation. This method has to do with the online training capability. Realistically, once DPM-MN is online training, there must be an interruption where the evaluation occurs. It is improper to test the model with data used for training, so the process separates these two steps. In addition, the temporal aspect of DPM-MN is better fit for interleaved evaluation and training periods.

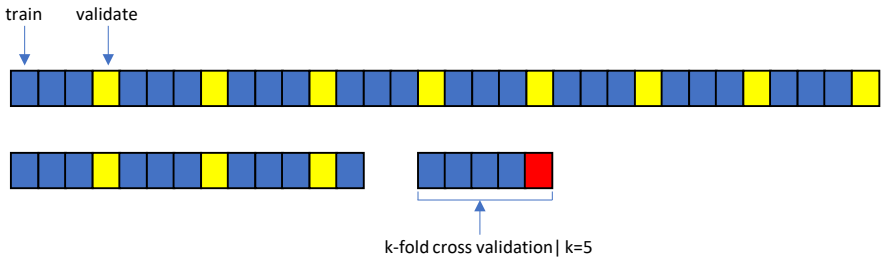

Figure 29: K-fold cross validation.

Figure 29 is an image of the training, validation, and testing process. Each cell in the figure is a batch. The proportions in the image match with the actual proportions from the experiment. Training and validation use 90% of all data. Of that 90%, training uses 75% and validation uses 25%. During the training and validation sequence, training uses three batches, and then a single batch takes the validation measurement. This interleaved process occurs until all the training and validation data is completed. After a model is ready to be tested, the k-fold cross validation (k=5) occurs with the test data. Each time the search algorithm assigns new hyperparameters to a model, the data must process temporally to appropriately build the hidden state. Once the model exhausts the observation data, the final accuracy measurement uses the ACD.

4.2.3 *Individual Player Test.*

The first test individually evaluates each of the thirty-five players. Previous experiments on the *Space Navigator* data, such as the experiment conducted by Bindewald (Bindewald, et al., 2015), indicated an improvement in performance when learning on specific players rather than learning on the data as a whole. Coagulating all the data together masks the idiosyncrasies of individual players and forms general concepts common among players.

For each player, their data executes the training, validation, and testing pipeline. None of one player's data mixes with another player's data. As a result, thirty-five different tests will use thirty-five separate DPM-MN models and their optimal hyperparameters.

Each player's data consists of roughly two-thousand datapoints. Because of the relatively low number of datapoints, the batch size is set to ten.

4.2.4 *Generic Player Test.*

After the completion of the unique tests of each individual, the average hyperparameter values of all the players create a DPM-MN model. This test groups all the player data together since the hyperparameters originate from the average of all the individual player hyperparameter sets. This test explores the volatility of the hyperparameters.

The next test places all of the player data together for training and validation in addition to testing. The resultant DPM-MN mental model captures generalities among all players.

Interleaving the individual player data by the *Space Navigator* level constructs the generic player database's temporal order. For instance, player one, level one data forms the

head of the dataset. Next, the master dataset adds player two, level one data. Once the master dataset contains all the players' level one data, the same pattern occurs except with level two data. This continues until all of the data is added to the master dataset. It could have been possible to add all of each player's data at once (player one, level one; player one, level two; …; player thirty-five, level sixteen), but since DPM-MN is mapping a learning process of a human player, it is better to start with easier levels and end with the harder levels. Also, constantly interleaving the different players, the last player will have less of a chance to greatly impact the final DPM-MN result through the possible importance of recent data.

The entire dataset consists of 68,500 points. For this test, each batch contained two-thousand points. Depending on the hyperparameters, this many points per batch could have a considerable impact on the building of the DPM-MN model. For one, windowing in System 2 will probably occur more often.

Future tests concerned with bootstrapping can use the optimal generic player DPM-MN. Loading a model with the generic player DPM-MN model, and then learning from that point may cause faster learning. Theoretically, the generic player model bootstrap could supply the general behaviors before any learning specific to an individual transpired.

Each test keeps most of the design the same. Most notably, the tests do not change the hyperparameter search space. By maintaining consistency of the search space, the two tests will vary less which allows a fairer comparison between the two outcomes.

DPM-MN competes against other baselines: Bindewald, et al.'s (Bindewald, et al., 2017) Clustering-Based Online Player Modeling, a neural network (Appendix L), and a straight-line baseline. The Bindewald baseline comes from the same experiment where all

the data used in this experiment was collected. It had an average individual error of 0.2036. The neural network leverages a long short-term memory (LSTM) neural network that produces trajectories given a state.

The straight-line baseline is a straight trajectory from each ship to the destination planet. Outperforming the straight-line trajectory baseline indicates better than blind results.

4.2.5 *Space Navigator Measures.*

The individual player test and the generic player test both use the same metrics for evaluation. Using the same metrics permits comparisons between the tests. The test metric is the average coordinate distance (ACD) (Appendix H). The various statistics used to analyze the results include the D'Agostino and Pearson's Normality Test, a Student's t-Distribution to find a confidence interval, the mean and standard deviation, and the Wilcoxon Rank-Sum Test (WRST).

For comparison purposes, the test metric is the ACD (Bindewald, et al., 2015). The ACD describes the average difference between the true trajectories and the predicted trajectories.

The ACD also corresponds to the degree of similarity between the artificially created mental model and the human's actual mental model. It acts as the performance measure for the claimed SMM. An abstract concept like the SMM cannot be directly measured. A few eminent SMM researchers have correlated the mental model similarity to the situational response similarity in some way (X. Fan, et al., 2011; Jonker, et al., 2010; Perelman, et al., 2017). Likewise, the DPM-MN mental model accuracy corresponds to the ACD performance metric since the ACD measures the average difference between the DPM-MN response and the human response to the given situation.

As for statistical decisions, the D'Agostino and Pearson's Normality Test is used to concretely determine if the distribution of result values is normal or not. This discernment guides which types of statistical tests are utilized. The results are not normally distributed according to Appendix K and preliminary testing.

Non-normally distributed results can still use a Student's t-Distribution because it is robust if the number of observations is high. Since roughly 6,800 observations exist, the Student's t-Distribution for finding the confidence interval is considered robust. Additional testing using a nonparametric confidence interval finder such as the bootstrapped confidence interval confirms the robustness of the Student's t-Distribution confidence interval.

The investigation of the two-sided p-value for each hypothesis test uses WRST. The statistics test applies a two-sided p-value over a one-sided p-value because it does not assume that DPM-MN's results perform better than a compared algorithm's results. WRST is a nonparametric algorithm. It is primarily advantageous when the samples are non-normal and when the samples are unpaired. In this research, WRST is a worthwhile hypothesis testing algorithm.

For the individual test results, the statistics are analyzed by grouping all the tested datapoints from each individual. This allows a comparison to Bindewald, et al.'s (Bindewald, et al., 2015) results which display the average ACD across all players. The individual test still tests each player separately, but the analyzed results are grouped together for a more complete depiction of the final results.

4.3 Summary

The notional dataset tests the applied algorithm's characteristics. If DPM-MN's asserted dual-process theory motivated functions exist, it will perform well on the notional dataset.

With each of the *Space Navigator* experiments, the goal is to figure out if the DPM-MN algorithm performs better or worse than each baseline. DPM-MN tries to create the best model so it can successfully map an individual's mental model. DPM-MN consists of seven hyperparameters that harmonize for the most effective mental model learner. The ordering and selection of the data also plays an important role. It determines the mental model DPM-MN learns. The individual player test and the generic player test both elucidate the performance and characteristics of DPM-MN.

Chapter 5. Results

The experimental results reveal the success of the Dual-Process Model using multivariate normal probability density functions (DPM-MN) algorithm as a cognitive architecture for generating a shared mental model (SMM). Concluding the evaluation hypothesis achieves the success determination. Is DPM-MN better or worse at building an SMM than the baseline models? The many human-robot teams pervading society rely on the development of an SMM.

This chapter presents experimental results evaluating DPM-MN's ability to develop a user mental model. Presented first are results from the notional dataset comparing DPM-MN to a support vector machine (SVM). Next, a Bayesian optimization of optimal hyperparameters is performed. Following this, are comparison results of DPM-MN versus three baseline algorithms on individual users. Finally, evaluation of DPM-MN versus the same three baseline algorithms on the entire dataset concludes the findings.

5.1 Notional Dataset Results

The notional dataset test determines the ability of DPM-MN to handle learning situations that involve dual-process motivated learning characteristics of concept drift, overwriting of previous concepts, windowing, and refreshment of frequently needed concepts. A notional dataset requires the desired learning behaviors for success. The notional dataset test compares a model that utilizes the learning characteristics (DPM-MN) to a model that processes the dataset as a whole (support vector machine).

Table 2: SVM optimized hyperparameters.

Hyperparameter	*Optimized Value*
C	19.6390
gamma	0.2382
Kernel	rbf

Table 3: DPM-MN optimized hyperparameters.

Hyperparameter	*Optimized Value*
confidence threshold (C_t)	3.0
entropy threshold (E_t)	1.0
number of clusters (N_c)	40.66
size threshold (S_t)	2.0
System 1 influence (S_{one})	2.0
System 2 influence (S_{two})	2.0
window size (W_s)	2.0

Table 2 and Table 3 show the parameterized optimization values for each model type. The SVM hyperparameters allow for a moderately soft-margin and a nearly linear

gaussian decision boundary. These parameters align with a dataset created from normal distributions. All of the DPM-MN hyperparameters except one are either the possible maximum or possible minimum value. The extreme hyperparameter choices probably correspond to the nature of the synthesized data. The notional dataset contains mostly separated class groupings, and the dataset stages the data in discrete steps.

Table 4: Notional dataset accuracy results.

Algorithm	*Accuracy*
DPM-MN	0.896
SVM	0.769

As expected, the DPM-MN algorithm outperformed the SVM algorithm. DPM-MN's learning characteristics allows it to properly process the data in a temporal manner to end with an SMM that reflects the relative dataset behavior. SVM is not situationally aware of the learning process embedded in the notional dataset.

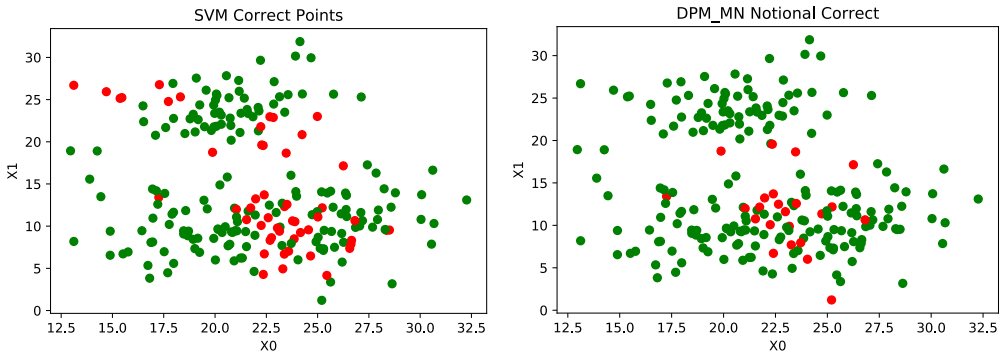

Figure 30: SVM and DPM-MN correct points.

The red points are incorrectly predicted test points and the green points are correctly predicted test points. In the SVM picture, the edges of the orange class and the mixing between the green and blue classes are the most difficult areas. DPM-MN also struggled

with the blue class and green class mixing zone, but it performed much better than the SVM on the orange class test points.

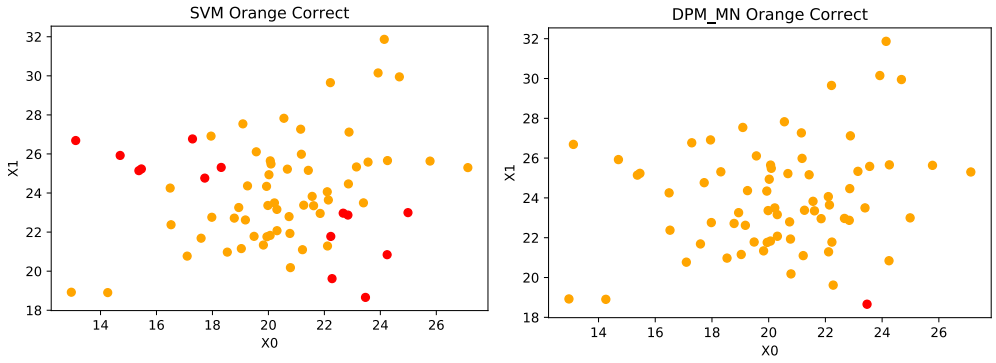

Figure 31: SVM and DPM-MN orange class correct.

SVM encountered trouble with the orange points on the edge of distribution. These misclassified points exist in areas near a blue class distribution source. The orange class conflicts with the blue class distribution source because SVM is unaware of the concept drift event. DPM-MN only misclassified a single orange class test point. DPM-MN credits its success to the concept drift operation.

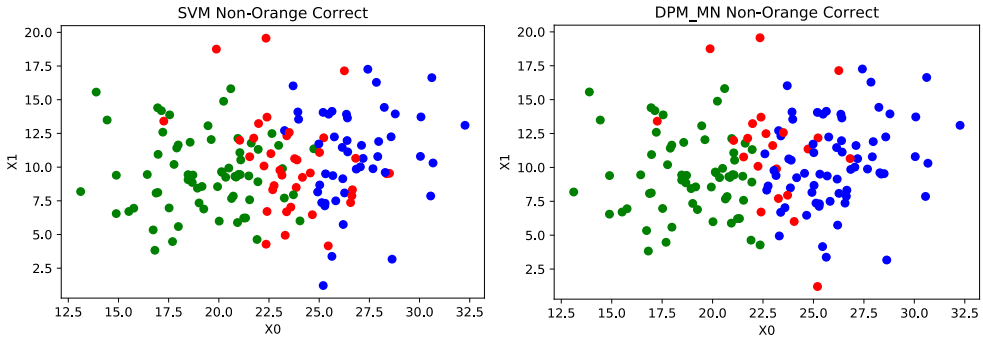

Figure 32: SVM and DPM-MN non-orange class correct.

As stated before, trouble exists between the blue class and the green class in the mixing zone. If the mixing zone contained points where the state space correlated to the response, DPM-MN may have successfully classified those points as well through novelty detection. However, the mixture of the blue class and green class is randomized so it is an

area of irreducible error. Even still, DPM-MN outperformed the SVM classifier in the mixed zone. DPM-MN utilized concept drift to strengthen the boundary between the blue and green classes. Sometimes SVM misclassifies non-outlier points within each distribution while DPM-MN only misclassifies most of the randomized outliers.

5.1.1 *Additional DPM-MN notional dataset analysis*

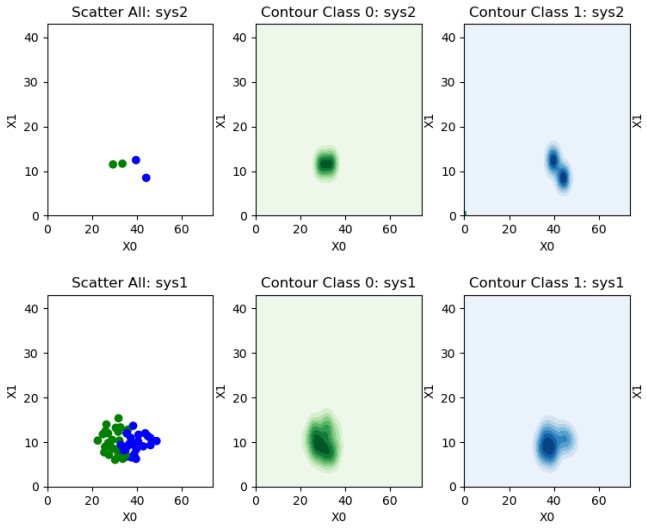

Figure 33: Immediate class groupings.

DPM-MN quickly conceptualized the green and blue classes in System 2 during the first dataset step.

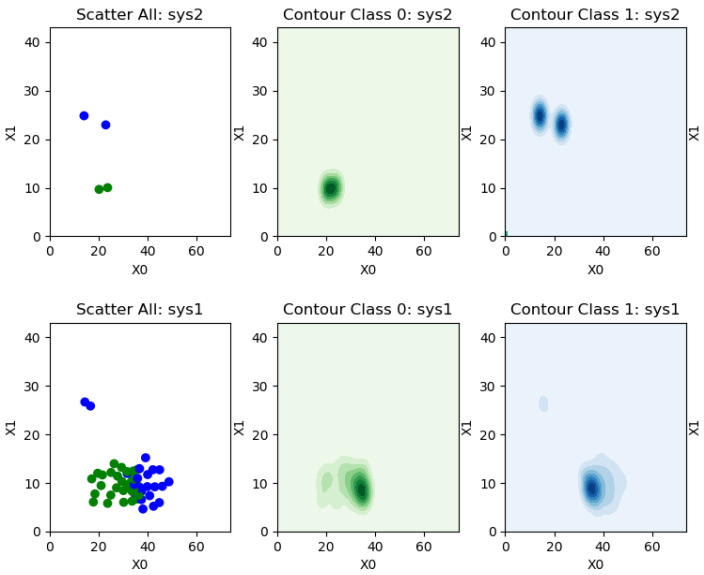

Figure 34: Blue class concept drift.

DPM-MN performed the concept drift operation of the blue class when the blue class moved from the right of the green class to above the green class.

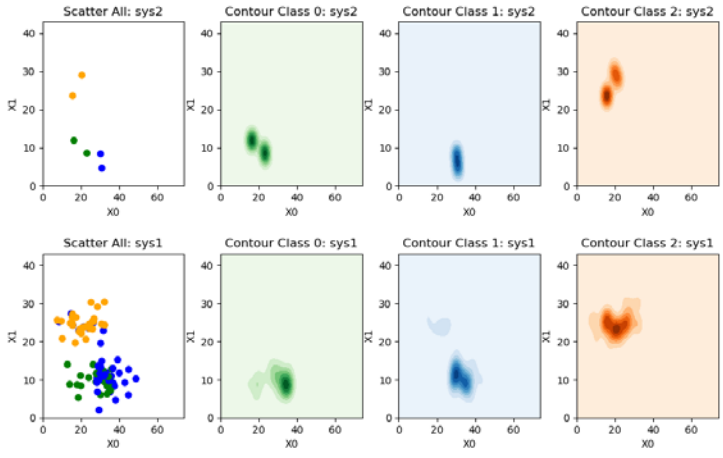

Figure 35: Orange class predominance and blue class return.

The orange class eventually predominated the area previously taken by the blue class. Furthermore, the blue class successfully returned back to its original location.

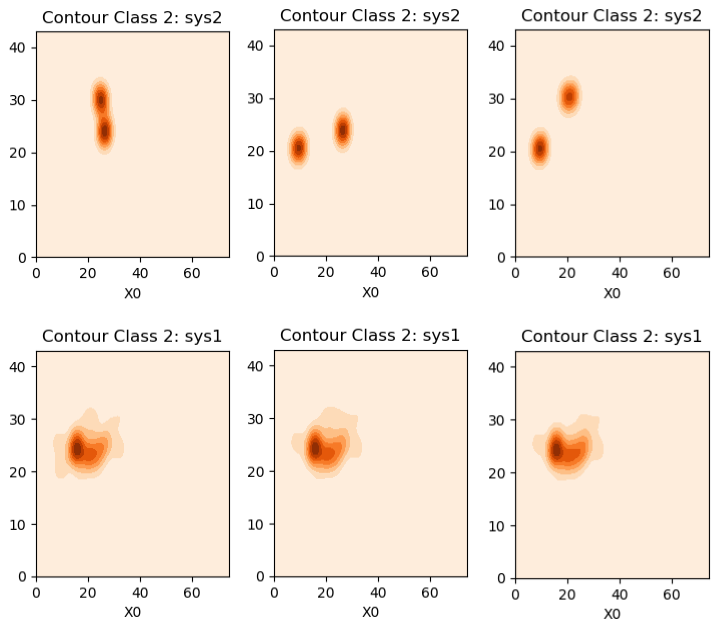

Figure 36: Windowing of orange concept.

The left image shows the beginning position for the orange class concept. In the middle image, because W_s = 2, the top orange concept moves to the left of the bottom orange concept. From the middle image to the right image, the previously moved orange concept remains in place while the older concept moves further up.

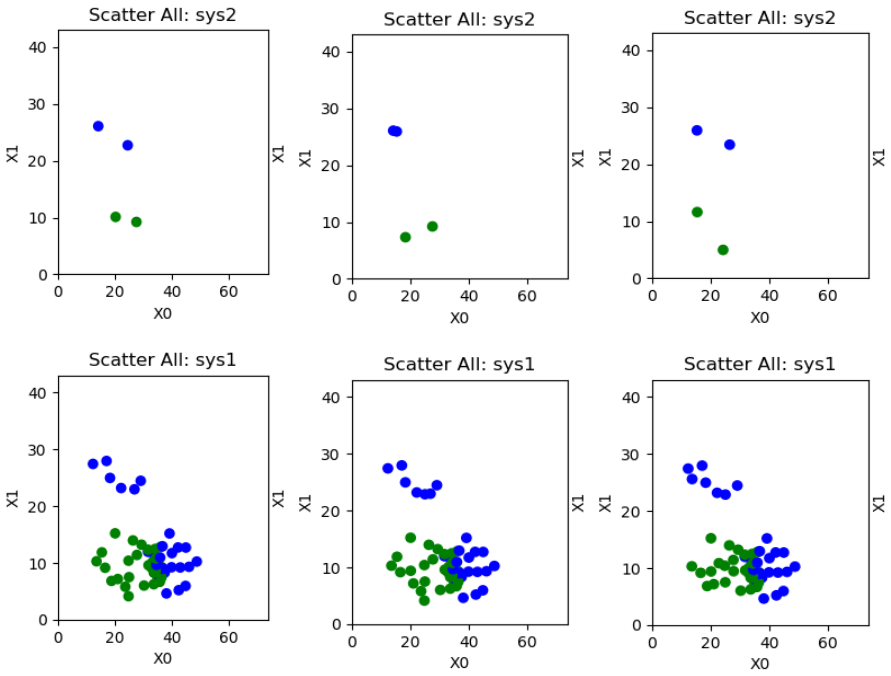

Figure 37: Refreshing of blue concept.

Looking at System 2, one of the blue concepts remains in place despite the occurrence of two concept shifts and a W_s of 2. This happens because System 2 refreshes the leftmost blue concept which allows it to stay since it is important.

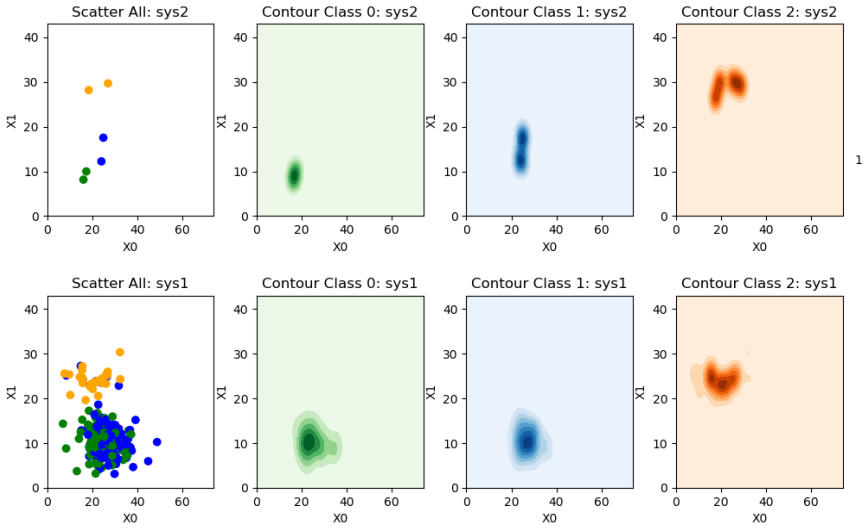

Figure 38: Final DPM-MN notional dataset trained model.

The three distinct class concepts are visualized in the final model. The orange class, the blue class, and the green class are all conceptually located where they are supposed to exist as predefined by the creation of the notional dataset.

5.1.2 *Notional Dataset Hypothesis Closure.*

The notional dataset represents a synthesized mental model of a human that experiences abstract learning during their state-response task. This dataset specifically provides an advantage to algorithms that utilize cognitive learning characteristics such as concept drift, overwriting of previous concepts, windowing, and refreshment of frequently needed concepts. Therefore, algorithms that build an SMM through learning characteristics outperform algorithms that exclude an SMM during notional dataset training.

The notional dataset experiment confirms that DPM-MN can handle learning situations that involve dynamic temporal data containing characteristics. DPM-MN performed significantly better than the high-performing SVM algorithm. The learning characteristics of DPM-MN cause the performance increase. Also, figures visually demonstrate the learning characteristics of concept drift, overwriting of previous concepts, windowing, and refreshment of frequently needed concepts.

5.2 Space Navigator Dataset Results

5.2.1 *Hyperparameter Search Results.*

5.2.1.1 *Individual Player Hyperparameter Results.*

Table 5: Hyperparameters selected for each player and individual results.

	confidenceThresh	entropyThresh	numClusters	sizeThresh	sys1Influence	sys2Influence	windowSize
1	2.912596	0.193673	49.473978	35.360302	0.389941	9.161333	37.987968
2	0.970502	0.636099	46.403306	49.728811	0.729775	9.928823	22.375848
3	0.583844	0.653468	17.267672	49.519895	0.550686	2.618788	49.879614
4	0.258760	0.306581	2.347282	10.624987	1.345394	9.371256	2.044606
5	0.157821	0.976438	3.275739	13.675994	0.458448	9.875178	3.905177
6	1.210109	0.760325	49.224046	2.302689	1.855860	9.150387	23.190683
7	1.626474	0.404619	30.034966	29.838884	1.943949	9.986304	2.342319
8	0.333915	0.884597	3.107800	2.320212	0.602014	9.758279	49.877167
9	1.005711	0.196728	24.268392	47.982973	0.433159	5.931174	47.042418
10	0.357987	0.165900	28.567435	33.654499	1.844108	9.509025	49.558294
11	0.285343	0.418206	2.723365	3.015580	1.999027	9.781991	34.714652
12	2.134009	0.170463	3.253147	3.466975	0.583111	5.766622	45.175847
13	0.164394	0.256734	8.764092	25.623707	1.939765	9.666493	49.804501
14	2.978500	0.909096	48.806766	15.835319	0.548704	3.166230	49.809233
15	0.089323	0.875702	15.430234	4.516370	0.217691	3.632015	20.478288
16	2.938666	0.705028	2.424768	27.596620	0.485521	2.032399	3.490609
17	2.124968	0.203047	2.128505	13.479004	0.417919	2.173913	47.260669
18	0.020947	0.962549	39.333894	20.783235	0.420024	9.831563	2.516673
19	2.639444	0.383253	48.787737	17.975996	1.767604	2.830903	5.801780
20	0.048219	0.994977	49.867345	25.321371	0.458380	2.944883	36.241841
21	0.016846	0.216591	2.000780	2.065650	0.431764	6.706382	26.059582
22	2.831506	0.729390	47.720427	2.079415	0.364223	5.731248	2.852550
23	0.157814	0.249094	22.895037	23.227583	0.557465	3.187737	49.790726
24	0.658011	0.412889	41.240129	31.498864	0.409972	5.638044	44.065783
25	1.272970	0.641689	28.592208	10.325816	0.861829	3.198105	18.165958
26	0.111258	0.966788	17.360744	49.566479	0.545163	4.088431	49.271940
27	0.725694	0.876720	2.002422	39.539398	0.236744	9.964111	49.152012
28	2.096984	0.278202	20.440232	43.038980	0.407014	4.628466	9.170155
29	1.286721	0.231886	5.855163	19.908096	1.435707	3.544551	26.925999
30	2.230593	0.980742	10.685288	28.353760	0.223712	2.024697	26.084577
31	0.001000	1.000000	50.000000	27.287066	0.441827	2.000000	2.000000
32	2.756058	0.782767	27.952955	48.727860	0.964412	9.647879	27.953709
33	0.463122	0.219368	29.855894	32.817427	1.203497	8.527019	37.101749
34	1.648991	0.395145	11.276565	18.570271	1.736402	7.152085	4.001430
35	2.172727	0.422828	39.008297	23.464853	1.455438	2.610746	14.255188

Table 5 shows the hyperparameter values per individual during their best performing DPM-MN model. The most optimal hyperparameter settings found through the Bayesian optimization process describe the learning style of each player. For example, player 31's idiosyncrasies and learning pattern is best functionalized with a low C_t of 0.001, the highest E_t of 1.0, the highest N_c used in the k-means clustering process in System 1 of

50.0, the lowest S_{two} of 2.0, and the lowest W_s of 2.0. This combination of parameters explains a dependence on a few initial points in System 1. Player 31's play patterns were quickly defined with an initial set of points that are difficult to change. They played with a more reactive behavior since the System 2 concept structure did not need to mature. Conversely, player 1 heavily depends on the deliberate System 2 concept structuring with a high C_t of 2.91, a low E_t of 0.19, a high S_{two} of 9.16, and a large W_s of 37.96. These variable values allow a considerable System 2 concept-set to form.

Even though the Bayesian optimization parameters can be analyzed to determine general learning patterns for each player, it is important to remember the complexity between the seven parameters and the overall behavior of the model. Sometimes it is difficult to connect the parameter values to a concrete learning theme.

5.2.1.2 *Averaged Player Hyperparameter Results.*

The average of each hyperparameter in the individual player hyperparameter results creates the averaged player hyperparameter results.

Table 6: Average of variables across players.

Variable	*Average Value Across Players*
C_t	1.1792
E_t	0.5560
N_c	23.7822
S_t	23.8027
S_{one}	0.8648
S_{two}	6.1648

W_s	27.7243

Table 6 distinguishes the average parameters to obtain a general sense across all the players after individually testing them.

5.2.1.3 *Generic Player Hyperparameter Results.*

Table 7: Hyperparameters selected for the generic test.

	confidenceThresh	entropyThresh	numClusters	sizeThresh	sys1Influence	sys2Influence	windowSize
0	1.601999	0.594311	49.886312	3.822416	0.220841	9.472414	49.47468

The Table 7 values can be compared to the average individual player values in Table 6. C_t and E_t are extremely close. The rest of the parameters greatly differ. It would be surprising if everything matched up because the average values for the individual players experiment are calculated from the hyperparameters of thirty-five unique individuals. The hyperparameters also do not act independently, so the unique hyperparameter sets include interdependent influence. If more closely analyzed, second or third level functions probably correlate to a greater extent. For example, it could be true that either a combination of low S_{one} and high S_{two} or high S_{one} and low S_{two} is typical. These higher order relationships may reveal the learning process that a specific human experiences when creating their own mental model.

5.2.2 *Individual Player Test Results.*

Table 8: Individual players results (lower is better).

Algorithm	Mean (ACD) ± SD	Confidence Interval (99%)	Normality Test	p-value

DPM-MN (unique)	0.181 ± 0.170	(0.176, 0.186)	0.0	N/A
DPM-MN (averaged player)	0.188 ± 0.173	(0.182,193)	0.0	0.0081
DPM-MN (generic)	0.191 ± 0.176	(0.186,0.197)	0.0	0.0006
Straight-Line	0.197 ± 0.178	(0.192,0.203)	0.0	4.507e-10
Medoid	0.185 ± 0.151	(0.180, 0.190)	0.0	2.982e-18
Bindewald, et al., 2015	0.204	(0.202,0.206)	N/A	N/A

The DPM-MN (unique) algorithm results are from thirty-five unique individual hyperparameter tests. The individual player test results in Table 8 show that DPM-MN performs better than the straight-line, medoid, and Bindewald baselines with statistical significance. However, the DPM-MN performance increase is relatively minor compared to the medoid baseline performance. The normality tests show all the tested distributions as being non-normally distributed.

Table 9: Individual player DPM-MN (unique) additional data.

	irreducibleError	test result	straightLineDiff	medoidDiff	uniquePred
1	0.073736	-0.208001	-0.218620	-0.198986	4.0
2	0.053910	-0.136762	-0.149124	-0.139418	3.0
3	0.066284	-0.143869	-0.152198	-0.155893	2.0
4	0.070324	-0.183229	-0.188043	-0.207495	1.0
5	0.051951	-0.137526	-0.155210	-0.139011	3.0
6	0.056928	-0.166978	-0.184814	-0.177520	1.0
7	0.072682	-0.189338	-0.188727	-0.193090	1.0
8	0.059975	-0.185037	-0.193646	-0.187092	3.0
9	0.055250	-0.109861	-0.113056	-0.106936	2.0
10	0.057369	-0.160206	-0.165079	-0.164405	2.0
11	0.070665	-0.173841	-0.182347	-0.182643	1.0
12	0.053942	-0.158883	-0.173283	-0.178485	3.0
13	0.074201	-0.208453	-0.292951	-0.204467	1.0
14	0.065598	-0.160893	-0.166891	-0.157244	4.0
15	0.076800	-0.224454	-0.238537	-0.248080	6.0
16	0.077248	-0.219797	-0.226497	-0.216354	4.0
17	0.066582	-0.218786	-0.321393	-0.204669	4.0
18	0.054892	-0.177474	-0.187000	-0.182356	2.0
19	0.070406	-0.210570	-0.204162	-0.216667	1.0
20	0.057793	-0.172892	-0.171309	-0.184374	1.0
21	0.051664	-0.257679	-0.284270	-0.236943	6.0
22	0.064972	-0.224801	-0.231790	-0.229506	5.0
23	0.071710	-0.232637	-0.253446	-0.213926	3.0
24	0.057831	-0.168462	-0.184709	-0.171607	3.0
25	0.071781	-0.161176	-0.163834	-0.166932	2.0
26	0.047218	-0.153560	-0.157369	-0.157694	4.0
27	0.054001	-0.161638	-0.178371	-0.178028	7.0
28	0.067376	-0.235477	-0.224449	-0.255536	5.0
29	0.064344	-0.173090	-0.185942	-0.176404	1.0
30	0.044721	-0.189840	-0.238375	-0.232204	9.0
31	0.059834	-0.141464	-0.144742	-0.154841	4.0
32	0.070148	-0.147738	-0.145863	-0.138629	1.0
33	0.050714	-0.182682	-0.197358	-0.172047	2.0
34	0.060793	-0.171123	-0.206885	-0.163810	1.0
35	0.051447	-0.172877	-0.170995	-0.171946	1.0

Table 10: Average of variables across players additional data.

Variable	*Average Value Across Players*
irreducibleError	0.0621
test result	-0.181
straightLineDiff	-0.196
medoidDiff	-0.185

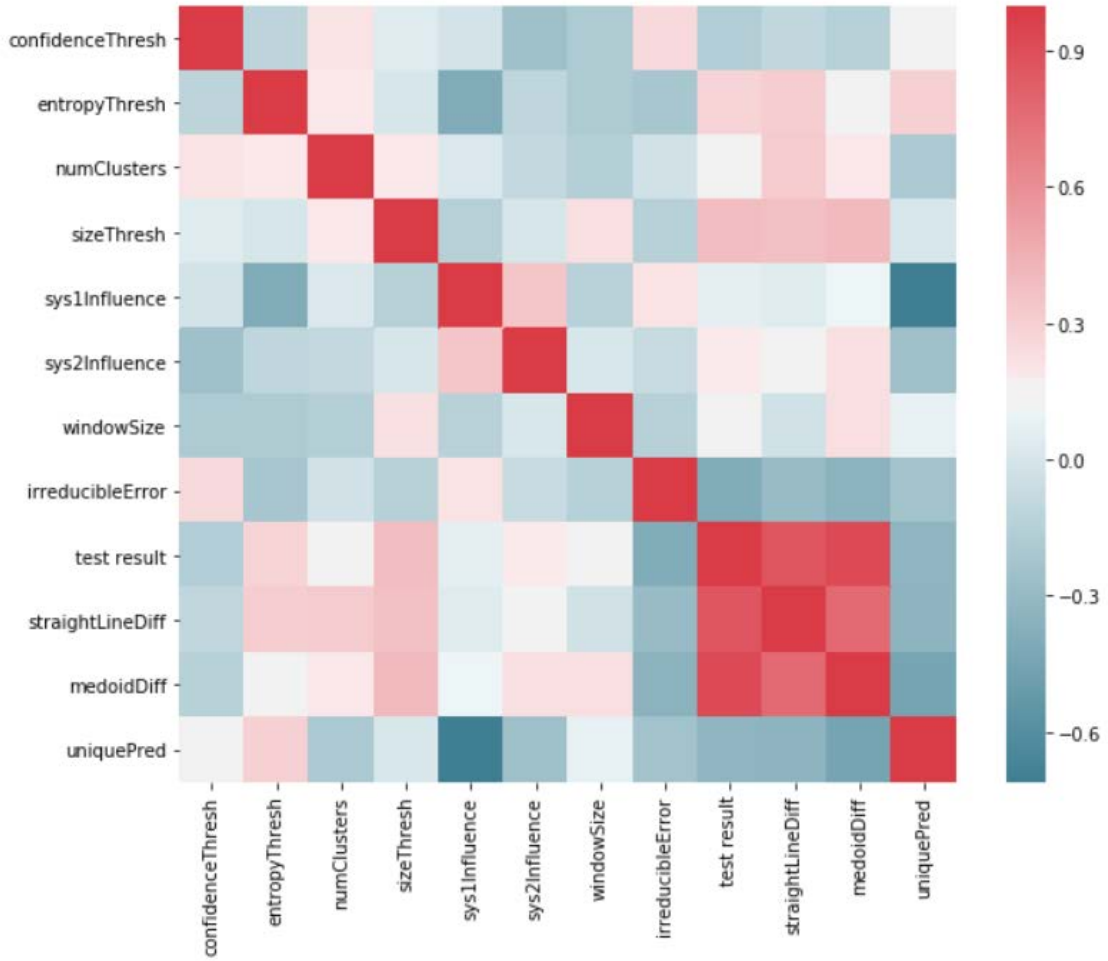

Figure 39: Correlation heatmap.

Figure 39 is a correlation heatmap of certain variables including the Bayesian optimization values and the various results. A few interesting correlations can be pointed out. Aligning with intuition, each of the result types are positively correlated with each other. The irreducible error present has a sizable negative correlation with the result types. When an inherent handicap is present, the results end up suffering. Most of the Bayesian optimization variables have little to no correlation with each other. The harmonizing of hyperparameters to characterize the learning process is nonintuitive. DPM-MN is needed to find the delicate balance of hyperparameters specific to each person.

The most peculiar correlation is between the number of unique predictions made during the test (uniquePred) and S_{one}. These two variables are strongly negatively correlated. The cause of this could have been the anomaly detection provided by a smaller area of influence. A low S_{one} causes points in System 1 to cover less space, but they have a stronger effect in their localized space due to less of a smoothing factor. Unique, or possibly one-off, predictions are more likely to occur with a low S_{one} if a similar unusual state-space is encountered more than once. If the localized, but non-smoothed, points are activated multiple times, they have more of a chance of giving a more confident response than the System 2 response which enables the diversity of predictions. Thus, DPM-MN manipulates the value of S_{one} to produce a high performing number of unique predictions.

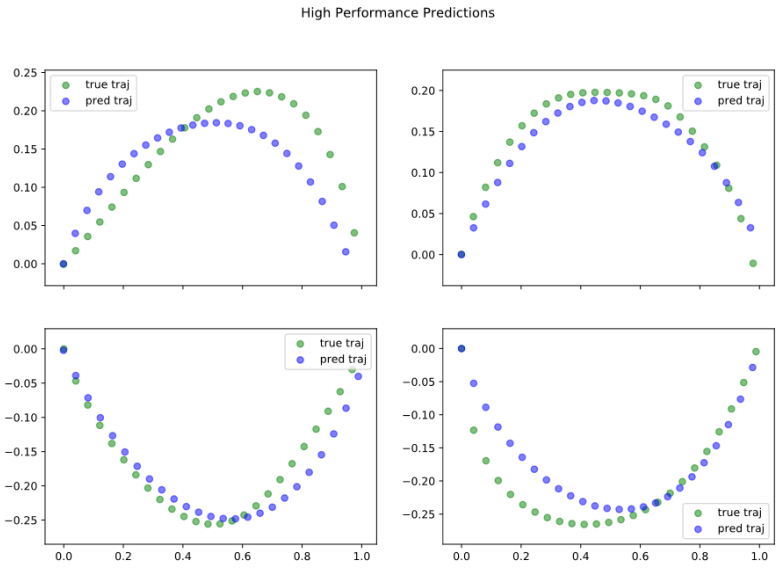

Figure 40: High performance predictions.

Figure 40 displays some of the predictions that performed really well. In these examples, the y-axis is significant; these are actually heavily curved trajectories.

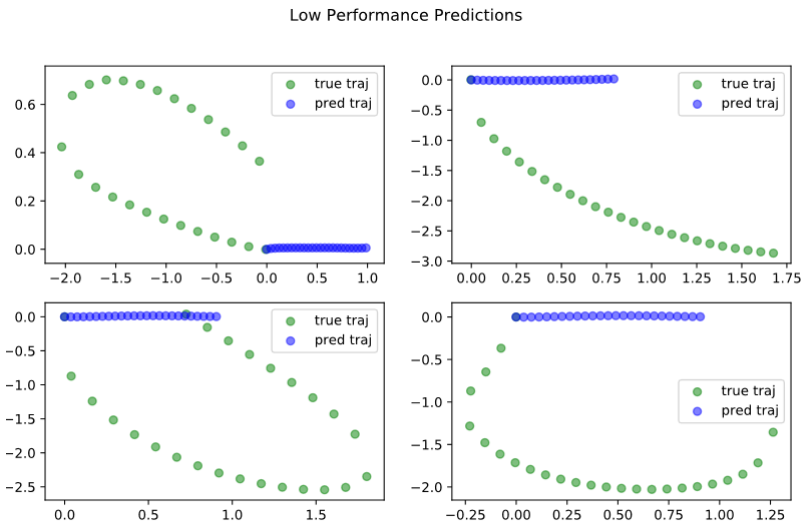

Figure 41: Low performance predictions.

Figure 41 displays some of the lowest performing predicted trajectories. As seen, low performance usually occurs when the true trajectory is extremely unique. Unless the mapping between the specific game state and the specific response is very clear in order to predict the true trajectory through anomaly detection, these true trajectories are not going to be predicted through a generalization method. The poor performance also can be attributed to the extreme distances the points reach. The top right image in Figure 41 has a trajectory going almost twice the distance needed to reach the x-value of the destination planet.

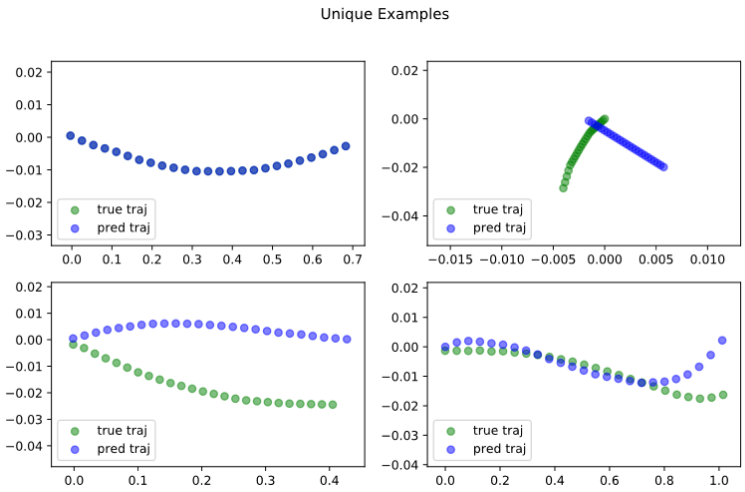

Figure 42: Unique examples.

Figure 42 show some unique occurrences during trajectory prediction. The top left trajectory highlights the instance where the class medoid representation is encountered in the test and correctly predicted. The predicted and true trajectories match up exactly. The top right image is an example of the case where the player draws a stunted line. Both the x-axis and y-axis are on an extremely small scale. The player most likely clicked on the selected ship, and then changed their mind about drawing a trajectory. The bottom left trajectories highlight the importance of distance in the x-axis. Even though they diverge on the y-axis, sometimes the ACD more greatly depends on where the predicted trajectory ends on the x-axis. The bottom right image is another example of two straight lines even though on a smaller scale they look curvy.

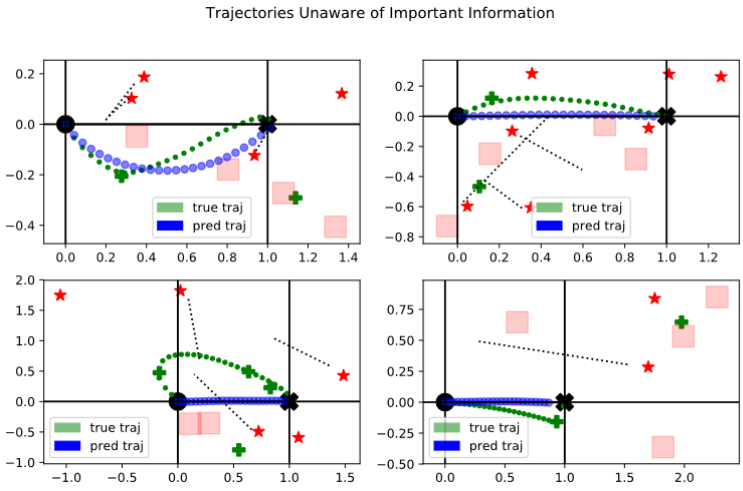

Figure 43: Trajectories unaware of important information.

Figure 43 is about examples where the prediction needed to consider the surrounding important information but could not because of limitations due to the selection of recorded features or the trajectory representation process. The top left predicted trajectory was close to the true trajectory, but the true trajectory goes through the nearby bonus and avoids the no-fly zone. The top right image's true trajectory and predicted trajectory both take a direct path to the destination planet, but the true trajectory collides with the bonus on the way through a very slight path alteration. The bottom left and bottom right images show the true trajectory prioritizing the bonuses instead of directly moving to the destination planet.

5.2.3 *Generic Player Test.*

Table 11: Generic player results (lower is better).

Algorithm	**Mean (ACD) ± SD**	**Confidence Interval (99%)**	**Normality Test**	**p-value**
DPM-MN (generic)	0.170 ± 0.174	(0.165, 0.175)	0.0	N/A

DPM-MN (averaged player)	0.173 ±0.174	(0.167,0.178)	0.0	1.0
Straight-Line	0.174 ± 0.175	(0.169,0.180)	0.0	0.01404
Medoid	0.184 ± 0.153	(0.179, 0.188)	0.0	1.297e-62
Bindewald, et al., 2015	0.2186	(0.217,0.220)	N/A	N/A
Grimm LSTM (Appendix L)	0.22	N/A	N/A	N/A

The generic player test combines all of the player data before training and evaluation. Table 11 consists of the results from the generic player experiment. A couple notable observations are apparent. First, DPM-MN (generic) performed the best out of all the algorithms. Second, the generic player DPM-MN model performed better than the individual player DPM-MN (unique) model. The second observation seems unintuitive. Possible explanations are given in the discussion section. Both the DPM-MN models, generic and averaged player, tested with the grouped generic data statistically performed equally (p=1.0).

Table 12: DPM-MN (generic) test additional data.

	irreducibleError	test result	straightLineDiff	medoidDiff	uniquePred
0	0.068583	-0.169958	-0.174423	-0.183567	5.0

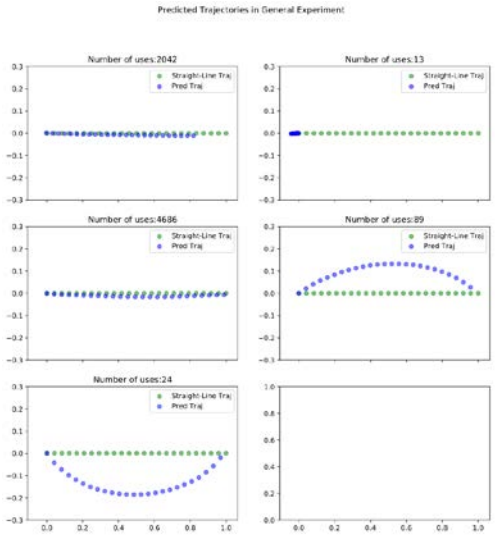

Figure 44: Predicted trajectories in the generic experiment.

Since the generic player DPM-MN model is a generic model, it is compelling to illustrate the predicted trajectories use during the tests. Figure 44 shows which types of responses are given once the *Space Navigator* generic mental model is finished. The titles of each sub-graph illuminate the number of times they are utilized. It is interesting to see that basically a straight line is used in most cases. The generic categories of trajectories can be described as “straight-line”, “extremely short line”, “curve-up”, and “curve-down”. The most effective mental model primarily uses straight-lines but knows when to sometimes use the alternative trajectories.

5.2.4 *Space Navigator Results Initial Discussion.*

Out of all the DPM-MN models, DPM-MN (unique) improved the most with respect to its corresponding straight-line predictor. DPM-MN (unique) successfully implemented its learning characteristics to align its model with the relevant player’s SMM. It is important to target the decision-making of individual players instead of masking the idiosyncrasies during training by grouping player data together. This effect is also

noticeable when looking at the individual player test medoid predictor. Each unique player independently influences the medoid predictor. Subsequently, the medoid predictor outperformed the generic-based hyperparameters in the individual player test. However, the DPM-MN (unique) model still performed the best out of all the models in the individual player test.

In the generic test, the averaged player DPM-MN model performs as well as the generic DPM-MN model. The resultant distributions are statistically the same. Unfortunately, the generic player *Space Navigator* test reduced to mostly straight lines as demonstrated by the straight-line predictor results. Consequently, the averaged player DPM-MN model learned to mostly predict straight lines. Because the averaged player DPM-MN model performed as well as the generic DPM-MN model, the average hyperparameters of the individual player unique tests result in a satisfactory model.

The LSTM neural network model poorly performed, but it resulted in the most unique predictions. The ACD test statistic declares other models as better, but unique predictions might make the autonomous agent seem more human-like. The LSTM model is more advantageous in scenarios where the test metric does not solely rely on the output difference.

5.2.5 *Space Navigator Dataset Hypothesis Closure.*

DPM-MN overall performs the best when compared to the baselines by a statistically significant margin. The DPM-MN model more closely predicts the response of a human player in every experimental instance. The straight-line predictor, the medoid predictor, and the Bindewald, et al. (Bindewald, et al., 2015) model did not perform as well as DPM-MN.

Since the DPM-MN model outperforms the baseline predictors in *Space Navigator*, experimental evaluation demonstrates that DPM-MN builds a better shared mental model of a human teammate. DPM-MN gained an advantage by utilizing cognitive learning functions. As a result, the dual-process theory provides a successful motivation for creating a human's mental model.

5.3 Summary

The notional dataset intended to demonstrate that DPM-MN equips dual-process theory motivated cognitive learning characteristics to build an SMM. The *Space Navigator* dataset intended to demonstrate that DPM-MN performs better than the baseline tests in the proposed learning environment. DPM-MN successfully achieved both intentions.

Chapter 6. Conclusions

Chapter 6 provides an overall discussion of each test, suggestions for future work, and future application recommendations. The overall discussions focus on additional examination of the findings. The future work explains the problems concerning DPM-MN. Finally, the future application section proposes areas where DPM-MN might be useful.

6.1 Discussion

6.1.1 *Overall Space Navigator Results Discussion.*

The results overall are an improvement on previous attempts. DPM-MN's sensitivity to outliers and dual-process theory design materialized an engineering advantage. Within the DPM-MN tests, the generic player DPM-MN model performed better than the individual player DPM-MN model. The vast majority of instances in all tests involve drawing a straight-line.

The speculated reason the generic player DPM-MN model outperformed the individual player model is because the test consisted of more trajectories approximate to a straight-line. Because of the way the generic player database structures the data (from level one data to the last level data), the test trajectories only included trajectories from the last level. The last level in *Space Navigator* is extremely hectic and difficult compared to the beginning levels. As a result, perhaps players drew straighter trajectories to ease their mental workload.

The idea that the generic player DPM-MN model performed well because of the increased number of straight-lines is supported by multiple pieces of evidence. First, the generic player straight-line baseline predictor performed better than the individual player straight-line baseline predictor. Second, the generic player DPM-MN model performed

2.3% better than its respective straight-line predictor while the individual player DPM-MN model performed 8.1% better than its own straight-line predictor. Last, the performance during the validation stage is 0.195. This points towards a more difficult dataset during the earlier stages.

Even though all the tests do not perfectly match up with each other, the tests overlap enough to learn from the results. The methodology for the tests in this experiment is reliable and robust since it focused on testing for the main goal of building the best human mental model via DPM-MN learning characteristics. Even though the generic player experiment contained easier test trajectories, the dataset maintained the learning intention. It makes sense to build the dataset by level rather than by player or some other measure. The basic assumption is that the players will learn better strategy over the course of the experiment. Regardless of the imperfections between experiments, each experiment can also relate using the performance increase from the shared baselines.

The generic player DPM-MN experiment, the individual player DPM-MN experiment, and the Bindewald, et al. (Bindewald, et al., 2015) experiment all performed better than a straight-line baseline. The generic player DPM-MN model performed 2.3% better (mean ACD) than the straight-line predictor, the individual player DPM-MN experiment performed 8.1% better than the straight-line predictor, and Bindewald, et al. performed 12.2% better than the straight-line predictor. Each straight-line predictor used for calculations corresponded to the same test. Bindewald, et al.'s straight-line predictor performed relatively poorly at 0.2319 mean ACD. Bindewald, et al. has the largest increase from the straight-line predictor, but it also has the most room to improve. Additionally, a more assuredly revealing test would involve more strategy.

The Grimm LSTM (Appendix L) performed the worst out of all models including every baseline predictor except Bindewald, et al.'s straight-line predictor (Bindewald, et al., 2015). Although the Grimm LSTM model performed the worst, it arguably predicted the most unique trajectories. By using a neural network, the trajectory is built by determining each individual point. Consequently, the predicted trajectories do not belong to a representative class but are instead each unique. Though, as demonstrated by the other trajectory predicting models, sometimes the best answer is the simple one.

6.1.2 *DPM-MN Notional Dataset Results Discussion.*

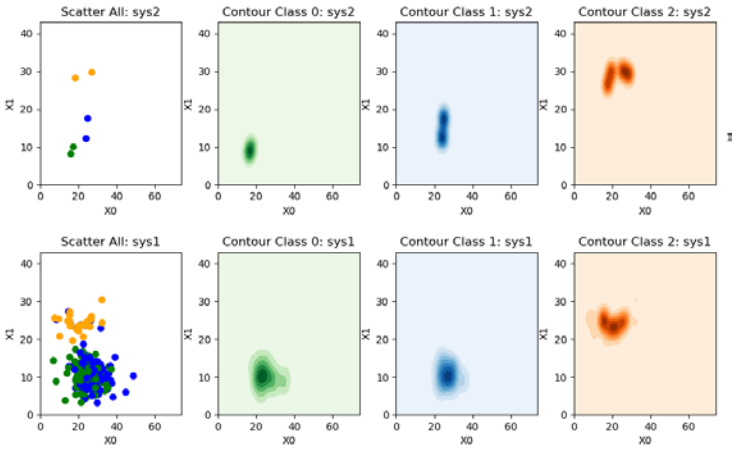

Figure 45: Notional dataset final model revisited.

Figure 45 introduces a couple interesting visual observations. First, the green concept and the blue concept in System 2 are relatively close to each other rather than covering an area closer to their respective true class distribution mean. This is because System 2 acts with high volatility due to a small W_s and a high C_t. With a small W_s, System 2 acts as a short-term memory storage. Since the training set ends with the mixing of the blue and green classes, System 2 reflects the border distribution. With a high C_t, datapoints continuously add to System 1. System 2 rarely is satisfactory in its composition. Also,

System 2 concepts rarely refresh since the concept distributions have trouble meeting the necessary refresh threshold which is based on C_t.

The blue and green classes can border each other and still achieve a high-performance rate because they act as a shield to the true distribution center existing behind the displayed class concepts. For example, any blue class test points located around an "X0" value of 35+ will still be correctly predicted as blue, even though the prediction confidence value is really low, because the green and orange concepts are even further away. At the moment, DPM-MN only cares about accuracy as a performance metric. Intuitively, a cognitive mental model may value confidence in the predictions. To enhance DPM-MN, the dual-process accounts of reasoning (J. S. B. T. Evans, 2003) should be further applied. Further DPM-MN enhancements need to take advantage of the dual-process learning mechanisms.

Next, it may seem unusual for the System 1 points to reflect the concepts in System 2. If System 2 covers the concept, why would points in the area covered by the concept need to be added to System 1 as outliers? In this case, C_t is high, so it is difficult to verify System 2 as a sufficient model. In some dual-process theory inspired algorithms, a reflection of concepts between System 1 and System 2 is intended (Helie, et al., 2011). In DPM-MN, a reflection of System 2 concepts in System 1 may happen when there exists an oversaturation of points in an area, a volatile System 2, and a high C_t. In the notional dataset, it makes sense for the reflection to occur since the class groupings are straightforward. DPM-MN primarily utilizes the System 2 concepts for a quick concept drift advantage to have the performance edge over other hyperparameter sets.

One primary advantage of DPM-MN is its flexibility in the creation of an SMM. The problem space, or the specific human DPM-MN is imitating, likely requires a change in the learning parameters. DPM-MN offers a robust foundation for dual-process learning, but it needs to better utilize ideas from the dual-process accounts of reasoning to enhance mental model mapping accuracy

6.2 Future Work

DPM-MN excels as a cognitive architecture for human behavior imitation, but it can benefit from some possible improvements. DPM-MN can be used as an additional tool in the near future where multiple algorithms are utilized to come up with a fuller solution. It reveals certain properties about symbolic representation and processing of psychological functions such as human learning, memory, and decision-making. DPM-MN also brings about discussion of appropriate cognitive architecture structure. In this case, the dual-process theory motivated the overall algorithm. The algorithmic implementation of the dual-process theory raises further speculative discussion about the correct approach. The version of DPM-MN in this research is a prototype, but it holds a lot more potential with future iterations or future applications.

6.2.1 *Algorithm Optimizations.*

DPM-MN's computational cost hinders extensive training or a vast hyperparameter search space. There are a few considerations which could be explored to improve the computational cost. These suggestions include parallelizing the algorithm, bettering the sequential methods, and implementing scaling solutions.

The first coding iteration of DPM-MN is initially concerned with properly coding the functionalities. However, a more parallelized implementation would help the training

computation time. With parallelization, GPU's can be utilized to simultaneously run portions of the code. Because of the temporal nature of DPM-MN, it may be difficult to find parallelizable parts of the code.

Even if some parts could be parallelized, it would be worth improving the sequential parts. The slowest part of DPM-MN is the constant k-means clustering of System 1 points. With every new input, the System 1 points are clustered. The execution time is slow and suffers even more when the hyperparameters allow a tremendous amount of points to exist in System 1. For example, if the size threshold and the entropy threshold (E_t) are both high, many points will gather in System 1. Some suggestions for improving the sequential timing are to use a more intrinsically efficient clustering algorithm, to save the previous k-means centroids for a starting point of the next k-means clustering iteration, to only use k-means clustering for new System 2 concepts once a certain number of new points enter System 1, or to run this process semi-offline on a different processor.

The computation cost becomes most apparent when the number of datapoints grows. Maybe scaling solutions can speed up the DPM-MN algorithm when it most needs a boost. A couple possibilities include using a fraction of the available input values to create the concepts and raising the number of input points per iteration. Using a larger batch size may have side effects with how DPM-MN learns, but it may be worth it for the speedup. Preliminary testing shows a possibility for a slight relationship between the number of points per training iteration and the test error. An official experiment could be beneficial to determine the behavior of altering the number of points per iteration value to improve computation time.

Another possibility is to group hyperparameter behaviors together to shrink the hyperparameter search space. The hyperparameters could represent concepts rather than direct variable changes. For instance, the concept of volatility could encompass W_s and N_c for System 1 k-means clustering. Volatility describes how quick System 1 points and System 2 concepts move. Another concept could be the ratio between S_{one} and S_{two} rather than directly stating them both. Lastly, the concept of System 2 concept size could include the S_t and the E_t. There are various higher-level concepts that could act as hyperparameters to extend the possible search space.

At the moment, a critical weakness of DPM-MN is the time it takes to train. Even if DPM-MN is superior in performance, it may be worth it to use other learning algorithms if the DPM-MN computation time does not reduce. However, there is high confidence that DPM-MN can improve with slight modifications.

6.2.2 *Narrow Application.*

One issue with DPM-MN is the narrow range of application. The state-response mapping is currently meant for a single problem. There needs to be a way to include other problems or a way to create a chain of decision making. Especially because DPM-MN is created to help handle the extremely complex problem of mapping a human's mental model, DPM-MN must have a potential to become broader. Also, the potential responses to choose from is limited to a predetermined amount.

A possible solution to the narrow problem application of DPM-MN could be to create a hierarchy of DPM-MN models. These multi-level DPM-MN models would be nested to allow recursive decision making. For instance, take two DPM-MN models: D-1 and D-2. D-1 decides which activity to do while D-2 decides which sport to play. Given a

scenario, if D-1 decides the person will play a sport, it can then consult D-2 to decide exactly which sport to play. In effect, D-1 decides which lower-level DPM-MN model to use.

Another problem limiting DPM-MN is the inability to explore new responses. DPM-MN learns mental models through experience where the response types are fixed. The set of available responses must be preprogrammed. Thus, DPM-MN's prediction is limited to a specific assortment of actions. There would need to be some way to create explorative behavior to make it a proper artificial general intelligence (AGI). Motor learning is just as important as declarative learning. If an agent using DPM-MN can never explore responses outside of the programmed responses, there can be no progress through exploration and discovery.

6.2.3 *Section Feature Value Collisions.*

As seen in some comparison pictures between predicted and true trajectories, the true trajectory seems more aware of the exact location of items of interest compared to the predicted trajectories. It could be an artifact from the trajectory prediction process, but it also could be an artifact of errored feature engineering. The feature engineering goal should include an ability to 'see' where components exist in a state instance. Using bonuses as an item example, the zone five bonuses feature value is a latent code for information about bonuses in zone five. This information includes the location and number of bonuses in zone five so a trajectory can be drawn through them, or at least near them in a similar way the human subject would have drawn the trajectory.

The issue arises with the inability to exactly pinpoint the location of the bonuses. Information is lost by merely recording a weighted density of objects within each zone.

The current features tell the algorithm the relevance of each type of object in each zone, but there is missing information on the coordinates of each item. For example, as long as a bonus in zone five is the same straight-line distance from the center line connecting the selected ship and the destination planet, the bonus will apply the same value to zone five. The variance of position on the x-axis does not matter. This lack of information explains why sometimes a predicted trajectory will be similar to the true trajectory, but the predicted trajectory will miss the obvious bonus or enter the obvious no-fly zone.

A possible improvement to the features is to manipulate the final trajectory with rules to alter the trajectory so bonuses are on the path if near the predicted trajectory and the predicted trajectory would avoid no-fly zones. However, this assumes that every player will make good trajectories. It restricts the ability for DPM-MN to learn the mental model of horrible players that miss bonuses and go through no-fly zones. Accordingly, the problem is difficult, but it still exists.

6.2.4 *Problems with State-Response True Correlation.*

For DPM-MN to be effective, the human's mental model needs to have some semblance of strategy. If there is little to no correlation between the state space and the human's response, then the mental model mapped by DPM-MN will have little value. DPM-MN will correctly map a seemingly random mental model, but it does not offer an advantage over other model types.

A possible source of 'randomness' within strategic environments is the free will a human has when making decisions. In some cases, given the same state space, a person will choose between multiple response types based simply on a "feeling". An example in *Space Navigator* is the way a person draws a second-degree polynomial curve between the

selected ship and the designated destination. If no significant items are near the pathway, a person may not use strategy when deciding to draw the curve either to the left or to the right of the center line path. The random behavior problem can be remedied by adding additional features to the state space that correlate to a decision-making strategy. Although, sometimes no such additional features exist.

A prime example is the *Space Navigator* experimental domain. It is difficult to make massive improvements on the naïve straight-line predictor model. Most people playing *Space Navigator* simply drew straight lines between the selected source and the destination planet. No clear strategy existed for many of the drawn trajectories. Even further, no clear and consistent strategy prevailed. DPM-MN takes advantage of the little strategy involved in *Space Navigator* as seen by the statistically significant improvement in the results, but DPM-MN would be more beneficial in problems where the human forms a unique strategy.

A person's strategic mental model requires some form of rationalization for DPM-MN to properly find correlations between the state space and response. *Space Navigator* somewhat encouraged a disengaged approach from the participants; in many cases the human player simply drew a straight-line trajectory from the source to the destination. It would be beneficial to explore DPM-MN's capability in a more strategic intense environment.

6.2.5 *System Two Multicollinearity.*

When a concept is created in System 2, a probability density function that closely represents the underlying points needs to be calculated with a non-singular matrix. A singular matrix occurs when there is perfect multicollinearity. In the case of DPM-MN, it

will occur if the number of observations is not greater than the number of dimensions of the state space. The state space is made of nineteen features, so nineteen features need to make up the underlying distribution for any concept in System 2. As a result, there are many different options that could be used to overcome this multicollinearity necessity. Each option has its own associated positives and negatives.

1) Use a probability density function that does not require the covariance matrix – This approach is scalable, quicker to calculate, and less restricting. A probability density function can be determined with as little as a single point. Although, the simplicity comes at a price. The probability density function is less precise in representing the underlying data than other methods.

2) Deal with a probability density function that requires the covariance matrix – The positives include an accurate and precise mapping of the underlying points. The primary, and substantial, negative is the scalability issue regarding the number of dimensions. Since a covariate matrix is necessary, the number of points used to create the probability density function needs to be proportionate to the number of dimensions of the datapoints. With overlapping functionality, this negative greatly affects the model as a whole. System 2 points will be constantly revoked due to not having enough points necessary for the distribution rule. It also creates a problem with choosing the size threshold and E_t. The S_t multiplied by E_t would have to be at least the necessary number of points. Thus, a new hyperparameter constraint is added to the already complex DPM-MN functionality.

3) Revocation on error – If a probability density function requiring a covariance matrix is used, an error frequently occurs when the overlapping functionality occurs. This requires the entire concept to be revoked since a distribution cannot be calculated anymore. By following this method, the System 2 concepts will more closely reflect the comprising points, but the existence of concepts becomes greatly restricted.

4) Dimensionality Reduction of State Space – Instead of revocation, the number of features for each datapoint can be reduced through dimensionality reduction. Less covariance matrix errors would occur, and the scaling problem would be fixed, but a potentially massive amount of information would likely be lost.

5) Adjust algorithm to accommodate covariance matrix error – The DPM-MN algorithm could be constrained in certain areas to create an environment where the error rarely happens. The first suggestion is to enforce a DPM-MN hyperparameter rule. If the number of points that initially make up a System 2 concept is well past the threshold, it is more likely that any overlapping will not instantly create an error.

> Another suggestion is to take out the overlapping functionality. Once a concept is in System 2, it will not have a high chance of creating a covariance matrix calculation error because the satisfactory number of points will be stable. However, the scaling problem still exists and the hyperparameter search is still constrained. It also creates an absence of a vital DPM-MN functionality.

Ultimately, DPM-MN uses the first option. Weighing the positives and negatives of each option, the precision loss of the first option is minimal compared to the plethora of issues related to a probability density function that requires the covariance matrix. In fact, options three to five are all alternatives for dealing with an algorithm that uses the second option. Those negatives add to the initial intimidating negatives of option two by itself. Introducing the possible covariance matrix error causes too many issues that negatively influence current and future algorithm decisions.

6.2.6 *DPM-MN Priority Improvements.*

The resultant DPM-MN model and corresponding framework rests on solid foundation. However, there are many possible improvements like those already discussed in this section. Through analyzation of DPM-MN results, a couple areas of interest emerged that should be first progressed before moving to other additions. These goals are to obtain more fitting state-response data and to further implement the dual-process accounts of reasoning.

Space Navigator provides clear strategic objectives for the state instance and a straightforward response. However, the strategy is seemingly absent. It is difficult for any model to map a learning process if no learning is happening. The straight-line baseline performs well because most of the drawn trajectories are close to the straight-line response. Each experiment, including DPM-MN, that attempts to functionalize the state-response pairing of the *Space Navigator* data correctly predicts mostly straight-lines with relatively

few correct predictions of curvy lines. A game with an evident, perhaps explicitly stated, strategy is better fit for a mental model mapping process such as DPM-MN.

Next, DPM-MN currently possesses the necessary mental model mapping tools for imitating human cognitive functions. DPM-MN adequately develops the dual-process theory of learning, but DPM-MN must place more stress on the dual-process accounts of reasoning. DPM-MN chooses the hyperparameters to simply produce the most accurate responses. This seems desirable at first, but the primary goal is to create the best mental model mapping of a human. While a high response accuracy is a component to finding a correct mental model, it is not the only factor. The goal of DPM-MN is to create a general intelligence in System 2 while supporting novelty detection in System 1. As seen in the optimal DPM-MN model for the notional dataset, the mental model building occurring in System 1 utilizes System 2 more-so as a complex support. This outcome greatly affects the efficacy of outlier detection in System 1 and the effective interaction between the systems. It does not help understanding of the algorithm when the experiments rely on datasets that are either synthesized, like the notional dataset, or saturated with a single response like the straight-line response in *Space Navigator*.

To assist DPM-MN in fulfilling its complete potential, DPM-MN can develop a few apparent changes. A different testing process may help DPM-MN build a better mental model in System 2. Because System 1 is an outlier detector, perhaps the testing of points can exclusively rely on System 2 or perhaps System 1 growth size can receive penalization in some way to force regularization. That way, System 2 must play a critical role in finding the general data distributions of the data. At the moment, System 1 routinely holds too much power and influence. It is not unusual for System 2 to have under ten concept

distributions while System 1 has over five-hundred concentrated, individual points. System 1 ceases its function as an outlier detector and System 2 cannot compete with a relatively low prediction confidence. Also, in a high-dimensional space, since System 2 is more smoothed out, the concept distributions may get spread too thin. Some possible solutions are to provide System 2 rules with a strength boost and to place a higher standard on the addition of new points into System 1 if a System 2 distribution already exists for the class in the prediction state space.

A more learning intensive environment should experiment with DPM-MN. Also, DPM-MN should further consider the dual-process accounts of reasoning. The basic idea is that System 2 should be capable of overriding System 1's prediction if it is likely wrong. In order to achieve this functionality, System 2 needs more competence. Once DPM-MN enhances the dual-process accounts of reasoning, the interrelation unit between the systems can become more complex with a balance in the duality of deductive and inductive reasoning.

6.3 Future Application

DPM-MN possesses a considerable amount of potential for future applications. Since it aids in the ability to map a human's mental model, human-machine teams can develop a shared mental model (SMM). Through an SMM, the computerized agent can utilize its simulation capability to a greater extent. Without an SMM, the motives and behavior habits of the human teammate are not fully leveraged in a computerized agent's prediction simulation. Now, the computerized agent can more optimally structure its own decisions to increase team performance.

The benefits to human-machine teaming will permeate the Air Force and society. The expansion of teams will, and in some cases currently do, greatly integrate a computerized agent teammate. DPM-MN improves performance anywhere a team is present. Teams can positively apply DPM-MN to enhance teamwork in many operating areas including pilots for their AI co-pilot or wingman, the cyber division for intrusion detection, medical professionals for insight on patient mental models regarding their treatment, and air traffic controllers for alleviating workload during high stress situations. With a better SMM, the autonomous agent in a team becomes more aware of the decision-making of its human teammates. The cognitive and learning design of DPM-MN even promotes the Air Force idea that "flexibility is the key to airpower".

DPM-MN does not need to be a standalone model. It would be useful to incorporate it as a module within a larger architecture structure. DPM-MN's primary function is to create a mental model of human teammates. Other architectures that specialize in different functions may be better suited for encountered problems rather than expecting DPM-MN to change to handle the problem as well. For instance, DPM-MN's learning of new procedures is problematic. There could be a separate module that learns the set of available responses and then those responses can be provided to DPM-MN for the creation of a shared mental model. Additional potentially useful modules to synergize with DPM-MN include an attention mechanism module, a deductive module using formal logic, and an action reinforcement module. Another problem that could be solved by cooperating with other modules is the issue DPM-MN has with unexpected queries. DPM-MN assumes that the incoming data will be complete and unchanged from the expected format. Perhaps a preprocessing module can take raw data and transform it into a consistent, embedded

feature space. Mixing DPM-MN with other model types can advance the creation of a holistic cognitive system used in AGI.

6.4 Summary

Teams work more efficiently and perform better with the presence of an SMM. Human-machine teaming is becoming more prominent throughout society. As a result, there needs to be a way for machine teammates to create a mental model of their human teammates. As demonstrated in the *Space Navigator* experiment, DPM-MN is a solution for the formulation of a human's mental model.

The DPM-MN architecture excels in building a human teammates mental model because of its implementation based on the dual-process theory. DPM-MN accounts for various cognitive concepts such as concept drift, sensitivity to outliers, flexibility in the learning method, and the balance between explicit and implicit decision-making. If the Air Force and society want to progress in human-machine teaming, they should analyze the lessons learned from the implementation of DPM-MN and be open to a dual-process theory motivated approach like DPM-MN.

References

Abdulrahman, A., Richards, D., & Mascarenhas, S. (2018). Shared Planning for building Human-agent Therapeutic Alliance. In *Proceedings of the 18th International Conference on Intelligent Virtual Agents - IVA '18* (pp. 335–336). New York, New York, USA: ACM Press. https://doi.org/10.1145/3267851.3267873

Agrawal, S., & Agrawal, J. (2015). Survey on anomaly detection using data mining techniques. In *Procedia Computer Science* (Vol. 60, pp. 708–713). https://doi.org/10.1016/j.procs.2015.08.220

Ahmed, M., Naser Mahmood, A., & Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*. https://doi.org/10.1016/j.jnca.2015.11.016

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*. https://doi.org/10.1037/0033-295X.111.4.1036

Augello, A., Infantino, I., Lieto, A., Maniscalco, U., Pilato, G., & Vella, F. (2017). *Towards A Dual Process Approach to Computational Explanation in Human-Robot Social Interaction.*

Augello, A., Infantino, I., Lieto, A., Pilato, G., Rizzo, R., & Vella, F. (2016). Artwork creation by a cognitive architecture integrating computational creativity and dual process approaches. *Biologically Inspired Cognitive Architectures*, *15*, 74–86. https://doi.org/10.1016/j.bica.2015.09.007

Azarnov, D. A., Chubarov, A. A., & Samsonovich, A. V. (2018). Virtual Actor with Social-Emotional Intelligence. *Procedia Computer Science*, *123*, 76–85. https://doi.org/10.1016/j.procs.2018.01.013

Baxter, P., Lemaignan, S., & Trafton, J. G. (2016). *Cognitive Architectures for Social Human-Robot Interaction*. Retrieved from https://sites.google.com/site/cogarch4socialhri2016/

Bera, A., Randhavane, T., Kubin, E., Wang, A., Manocha, D., & Gray, K. (2018). Classifying Group Emotions for Socially-Aware Autonomous Vehicle Navigation. https://doi.org/10.1109/CVPRW.2018.00151

Bifet, A., Gama, J., Pechenizkiy, M., & Žliobaitė, I. (2011). Handling Concept Drift: Importance, Challenges & Solutions Motivation for the Tutorial. Retrieved from http://www.cs.waikato.ac.nz/~abifet/PAKDD2011/

Bindewald, J. M., Peterson, G. L., & Miller, M. E. (2015). Trajectory generation with player modeling. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9091*, 42–49. https://doi.org/10.1007/978-3-319-18356-5_4

Bindewald, J. M., Peterson, G. L., & Miller, M. E. (2017). Clustering-based online player

modeling. *Communications in Computer and Information Science*, *705*, 86–100. https://doi.org/10.1007/978-3-319-57969-6_7

Bindewald, J. M., Rusnock, C. F., & Miller, M. E. (2018). Measuring Human Trust Behavior in Human-Machine Teams (pp. 47–58). https://doi.org/10.1007/978-3-319-60591-3_5

Blythe, J. (2012). *A Dual-Process Cognitive Model for Testing Resilient Control Systems*. Retrieved from http://info.deterlab.net/sites/default/files/files/blythe_isrcs12.pdf

Borchani, H., Martínez, A. M., Masegosa, A. R., Langseth, H., Nielsen, T. D., Salmerón, A., … Sáez, R. (2015). Modeling concept drift: A probabilistic graphical model based approach. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 9385, pp. 72–83). https://doi.org/10.1007/978-3-319-24465-5_7

Chen, M., Tian, Y., Fortino, G., Zhang, J., & Humar, I. (2018). Cognitive Internet of Vehicles. *Computer Communications*, *120*, 58–70. https://doi.org/10.1016/j.comcom.2018.02.006

Choi, A., De Melo, C. M., Khooshabeh, P., Woo, W., & Gratch, J. (2015). Physiological evidence for a dual process model of the social effects of emotion in computers. *International Journal of Human Computer Studies*, *74*, 41–53. https://doi.org/10.1016/j.ijhcs.2014.10.006

Choi, D., & Langley, P. (2018). Evolution of the ICARUS Cognitive Architecture. *Cognitive Systems Research*, *48*, 25–38. https://doi.org/10.1016/j.cogsys.2017.05.005

Colman, A. M. (2008). *A Dictionary of Psychology* (3rd ed.). Oxford University Press. Retrieved from http://www.oxfordreference.com/view/10.1093/acref/9780199534067.001.0001/acref-9780199534067-e-2515

Dautenhahn, K. (2007). Socially intelligent robots: dimensions of human-robot interaction. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *362*(1480), 679–704. https://doi.org/10.1098/rstb.2006.2004

De Ruiter, J. (2006). *Strong AI and the Chinese Room Argument, Four views*. Retrieved from https://pdfs.semanticscholar.org/af84/befa07d28a2bedd27f4be3d16b265f5b293d.pdf

De Winter, J. C. F., & Hancock, P. A. (2015). ScienceDirect Reflections on the 1951 Fitts list: Do humans believe now that machines surpass them? https://doi.org/10.1016/j.promfg.2015.07.641

Deng, C., Wu, C., Cao, S., & Lyu, N. (2017). Modeling the effect of limited sight distance through fog on car-following performance using QN-ACTR cognitive architecture. *Transportation Research Part F: Psychology and Behaviour*. https://doi.org/10.1016/j.trf.2017.12.017

Dennis, A. R., & Minas, R. K. (2018). Security on Autopilot : Why Current Security Theories Hijack our Thinking and Lead Us Astray. *ACM SIGMIS Database: The DATABASE for Advances in Information Systems*, *49*(April), 15–38. https://doi.org/10.1145/3210530.3210533

Diamond, D. M., Campbell, A. M., Park, C. R., Halonen, J., & Zoladz, P. R. (2007). The temporal dynamics model of emotional memory processing: a synthesis on the neurobiological basis of stress-induced amnesia, flashbulb and traumatic memories, and the Yerkes-Dodson law. *Neural Plasticity*, *2007*, 60803. https://doi.org/10.1155/2007/60803

Djenouri, Y., Zimek, A., & Chiarandini, M. (2018). Outlier Detection in Urban Traffic Flow Distributions. In *2018 IEEE International Conference on Data Mining (ICDM)* (pp. 935–940). IEEE. https://doi.org/10.1109/ICDM.2018.00114

Erfani, S. M., Rajasegarar, S., Karunasekera, S., & Leckie, C. (2016). High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. https://doi.org/10.1016/j.patcog.2016.03.028

Evans, J. S. B. T. (2003). In two minds: Dual-process accounts of reasoning. *Trends in Cognitive Sciences*. https://doi.org/10.1016/j.tics.2003.08.012

Evans, J., & Stanovich, K. E. (2013). Dual-Process Theories of Higher Cognition: Advancing the Debate. *Perspectives on Psychological Science*, *8*(3), 223–241. https://doi.org/10.1177/1745691612460685

Fan, J., Bian, D., Zheng, Z., Beuscher, L., Newhouse, P. A., Mion, L. C., & Sarkar, N. (2017). A Robotic Coach Architecture for Elder Care (ROCARE) Based on Multi-User Engagement Models. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *25*(8), 1153–1163. https://doi.org/10.1109/TNSRE.2016.2608791

Fan, X., & Yen, J. (2011). Modeling cognitive loads for evolving shared mental models in human-agent collaboration. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *41*(2), 354–367. https://doi.org/10.1109/TSMCB.2010.2053705

Frankish, K. (2010). Dual-Process and Dual-System Theories of Reasoning. *Philosophy Compass*, *5*(10), 914–926. https://doi.org/10.1111/j.1747-9991.2010.00330.x

Gama, A., Bifet, A., & Barcelona, R. (2014). A survey on concept drift adaptation. *ACM Comput. Surv*, *46*. https://doi.org/10.1145/2523813

Gavrilets, S., & Vose, A. (2006). The dynamics of Machiavellian intelligence. *Proceedings of the National Academy of Sciences of the United States of America*, *103*(45), 16823–16828. https://doi.org/10.1073/pnas.0601428103

Gnanaprasanambikai, L., & Munusamy, N. (2018). Survey of genetic algorithm effectiveness in intrusion detection. In *Proceedings of 2017 International Conference on Intelligent Computing and Control, I2C2 2017* (Vol. 2018–Janua, pp. 1–5). IEEE. https://doi.org/10.1109/I2C2.2017.8321877

Goertzel, B., Ke, S., Lian, R., O'Neill, J., Sadeghi, K., Wang, D., … Yu, G. (2013). The cogprime architecture for embodied Artificial General Intelligence. In *2013 IEEE Symposium on Computational Intelligence for Human-like Intelligence (CIHLI)* (pp. 60–67). IEEE. https://doi.org/10.1109/CIHLI.2013.6613266

Gonzalez-Jimenez, H. (2018). Taking the fiction out of science fiction: (Self-aware) robots and what they mean for society, retailers and marketers. *Futures*, *98*(February 2017), 49–56. https://doi.org/10.1016/j.futures.2018.01.004

Görür, O. C., Rosman, B., Hoffman, G., & Albayrak, S. (2017). Toward Integrating Theory of Mind into Adaptive Decision- Making of Social Robots to Understand Human Intention. *International Conference on Human-Robot Interaction*, (March). Retrieved from https://www.benjaminrosman.com/papers/hri17_ws.pdf

Grim, K., Rosenberg, D., Svedberg, P., & Schön, U. K. (2016). Shared decision-making in mental health care-a user perspective on decisional needs in community-based services. *International Journal of Qualitative Studies on Health and Well-Being*, *11*, 30563. https://doi.org/10.3402/qhw.v11.30563

Gudwin, R., Paraense, A., De Paula, S. M., Fróes, E., Gibaut, W., Castro, E., … Raizer, K. (2017). The Multipurpose Enhanced Cognitive Architecture (MECA). https://doi.org/10.1016/j.bica.2017.09.006

Hanna, N., & Richards, D. (2018). The Impact of Multimodal Communication on a Shared Mental Model, Trust, and Commitment in Human–Intelligent Virtual Agent Teams. *Multimodal Technologies and Interaction*, *2*(3), 48. https://doi.org/10.3390/mti2030048

Harriott, C. E., Buford, G. L., Adams, J. A., & Zhang, T. (2015). Measuring Human Workload in a Collaborative Human-Robot Team. *Journal of Human-Robot Interaction*, *4*(2), 61. https://doi.org/10.5898/JHRI.4.2.Harriott

Helie, S., & Sun, R. (2011). How the core theory of CLARION captures human decision-making. *Proceedings of the International Joint Conference on Neural Networks*, 173–180. https://doi.org/10.1109/IJCNN.2011.6033218

Hodhod, R., & Magerko, B. (2016). Closing the Cognitive Gap between Humans and Interactive Narrative Agents Using Shared Mental Models. In *Proceedings of the 21st International Conference on Intelligent User Interfaces - IUI '16* (pp. 135–146). New York, New York, USA: ACM Press. https://doi.org/10.1145/2856767.2856774

Ienco, D., Bifet, A., Pfahringer, B., & Poncelet, P. (2014). Change Detection in Categorical Evolving Data Streams. https://doi.org/10.1145/2554850.2554864

Infantino, I., Augello, A., Maniscalto, U., Pilato, G., & Vella, F. (2018). A Cognitive Architecture for Social Robots. In *2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)* (pp. 1–5). IEEE. https://doi.org/10.1109/RTSI.2018.8548520

Jämsä, J., Pieskä, S., & Luimula, M. (2013). Situation awareness in cognitive transportation systems. *Infocommunications Journal*, *5*(4), 10–16. Retrieved from http://www.infocommunications.hu/documents/169298/393366/2013_4_3_Jamsa.pdf

Jonker, C. M., van Riemsdij, M. B., & Vermeulen, B. (2010). Shared Mental Models: A Conceptual Analysis. *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent System*, (May), 10–14. https://doi.org/10.1007/978-3-642-21268-0_8

Klein, G. (2008). Naturalistic Decision Making. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *50*(3), 456–460. https://doi.org/10.1518/001872008x288385

Krakovsky, M. (2018). Artificial (emotional) intelligence. *Communications of the ACM*, *61*(4), 18–19. https://doi.org/10.1145/3185521

Launchbury, J. (2017). *A DARPA Perpsective on Artificial Intelligence*. Retrieved from https://www.darpa.mil/attachments/AIFull.pdf

Lavin, A., & Ahmad, S. (2015). *Evaluating Real-time Anomaly Detection Algorithms-the Numenta Anomaly Benchmark*. Retrieved from https://github.com/numenta/NAB.

Lazzeri, N., Id, †, Mazzei, D., Cominelli, L., Cisternino, A., & De Rossi, D. E. (2018). Designing the Mind of a Social Robot. https://doi.org/10.3390/app8020302

Lee, K. M., Peng, W., Jin, S. A., & Yan, C. (2006). Can robots manifest personality?: An empirical test of personality recognition, social responses, and social presence in human-robot interaction. *Journal of Communication*, *56*(4), 754–772. https://doi.org/10.1111/j.1460-2466.2006.00318.x

Lehman, F., Laird, J., & Rosenbloom, P. (2006). A GENTLE INTRODUCTION TO SOAR, AN ARCHITECTURE FOR HUMAN COGNITION. *Science*, (0413013), 1–37. Retrieved from https://web.eecs.umich.edu/~soar/sitemaker/docs/misc/GentleIntroduction-2006.pdf

Lemaignan, S., Warnier, M., Sisbot, E. A., Clodic, A., & Alami, R. (2017). Artificial cognition for social human–robot interaction: An implementation. *Artificial Intelligence*, *247*, 45–69. https://doi.org/10.1016/j.artint.2016.07.002

Lemoine, M. P., Millot, P., & Debernard, S. (2002). Human-machine cooperation in air traffic control. In *Control Engineering Practice* (Vol. 10, pp. 2114–2119). https://doi.org/10.1109/icsmc.1997.635177

Lieto, A., Bhatt, M., Oltramari, A., & Vernon, D. (2018). The role of cognitive architectures in general artificial intelligence. *Cognitive Systems Research*, *48*, 1–3. https://doi.org/10.1016/j.cogsys.2017.08.003

Lieto, A., Lebiere, C., & Oltramari, A. (2018). The knowledge level in cognitive architectures: Current limitations and possible developments. *Cognitive Systems Research*, *48*, 39–55. https://doi.org/10.1016/j.cogsys.2017.05.001

Lieto, A., Radicioni, D. P., & Rho, V. (2017). Dual PECCS: a cognitive system for conceptual representation and categorization. *Journal of Experimental and Theoretical Artificial Intelligence*, *29*(2), 433–452. https://doi.org/10.1080/0952813X.2016.1198934

Lindstrom, P., Delany, S. J., & Namee, B. Mac. (2010). Handling Concept Drift in Text Data Stream Constrained by High Labelling Cost. *Florida Artificial Intelligence Research Society Conference (FLAIRS). Florida*, 19–21. https://doi.org/10.21427/D7B022

Lohani, M., Stokes, C., Mccoy, M., Bailey, C. A., & Rivers, S. E. (2016). Social interaction moderates human-robot trust-reliance relationship and improves stress coping. *ACM/IEEE International Conference on Human-Robot Interaction*, *2016–April*, 471–472. https://doi.org/10.1109/HRI.2016.7451811

Losing, V., Hammer, B., & Wersing, H. (2017). SAM: How to Deal with Diverse Drift Types. In *International Joint Conference on Artificial Intelligence*. Retrieved from https://www.ijcai.org/proceedings/2017/0690.pdf

Meyer, D. (2017). Vladimir Putin Says Whoever Leads in Artificial Intelligence Will Rule the World. Retrieved January 24, 2019, from http://fortune.com/2017/09/04/ai-artificial-intelligence-putin-rule-world/

Moulin-Frier, C., Fischer, T., Petit, M., Pointeau, G., Puigbo, J. Y., Pattacini, U., … Verschure, P. F. M. J. (2017). DAC-h3: A Proactive Robot Cognitive Architecture to Acquire and Express Knowledge About the World and the Self. *IEEE Transactions on Cognitive and Developmental Systems*. https://doi.org/10.1109/TCDS.2017.2754143

Park, E., Jin, D., & Del Pobil, A. P. (2012). The law of attraction in human-robot interaction. *International Journal of Advanced Robotic Systems*, *9*. https://doi.org/10.5772/50228

Pereira, G., Prada, R., & Santos, P. A. (2015). Towards social power intelligent agents. In *Proc. Int. Joint Conf. Auton. Agents Multiagent Syst., AAMAS* (Vol. 3, pp. 1857–1858). https://doi.org/10.1242/jeb.024927

Perelman, B. S., Mueller, S. T., & Schaefer, K. E. (2017). Evaluating path planning in human-robot teams: Quantifying path agreement and mental model congruency. In *2017 IEEE Conference on Cognitive and Computational Aspects of Situation Management, CogSIMA 2017* (pp. 1–7). IEEE. https://doi.org/10.1109/COGSIMA.2017.7929595

Picard, R. W. (1995). Affective Computing. Retrieved from http://www.media.mit.edu/˜picard/

Pimentel, M. A. F., Clifton, D. A., Clifton, L., & Tarassenko, L. (2014). A review of novelty detection. *Signal Processing*, *99*, 215–249. https://doi.org/10.1016/j.sigpro.2013.12.026

Potamianos, A. (2014). Cognitive Multimodal Processing. In *Proceedings of the 2014 Workshop on Roadmapping the Future of Multimodal Interaction Research including Business Opportunities and Challenges - RFMIR '14* (pp. 27–34). https://doi.org/10.1145/2666253.2666264

Radovanović, M., Nanopoulos, A., & Ivanović, M. (2015). Reverse nearest neighbors in unsupervised distance-based outlier detection. *IEEE Transactions on Knowledge and Data Engineering*, *27*(5), 1369–1382. https://doi.org/10.1109/TKDE.2014.2365790

Rodríguez-Lera, F. J., Matellán-Olivera, V., Conde-González, M. Á., & Martín-Rico, F. (2018). HiMoP: A three-component architecture to create more human-acceptable social-assistive robots Motivational architecture for assistive robots. *Cognitive Processing*, *19*, 233–244. https://doi.org/10.1007/s10339-017-0850-5

Russell, S., Hauert, S., Altman, R., & Veloso, M. (2015). Robotics: Ethics of artificial intelligence. *Nature*, *521*(7553), 415–418. https://doi.org/10.1038/521415a

Salam, H., Celiktutan, O., Hupont, I., Gunes, H., & Chetouani, M. (2017). Fully Automatic Analysis of Engagement and Its Relationship to Personality in Human-Robot Interactions. *IEEE Access*, *5*, 705–721. https://doi.org/10.1109/ACCESS.2016.2614525

Sarkar, S., Araiza-Illan, D., & Eder, K. (2017). Effects of Faults, Experience, and Personality on Trust in a Robot Co-Worker, 1–33. Retrieved from http://arxiv.org/abs/1703.02335

Saucer, T., & Crossman, J. (2018). Robust hierarchical reasoning over sensor data with the Soar cognitive architecture. In M. C. Dudzik & J. C. Ricklin (Eds.), *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything* (Vol. 10643, p. 30). SPIE. https://doi.org/10.1117/12.2304981

Schaul, T., Togelius, J., & Schmidhuber, J. (2011). *Measuring Intelligence through Games*. Retrieved from https://arxiv.org/pdf/1109.1314.pdf

Schuller, D., & W. Schuller, B. (2018). The Age of Artificial Emotional Intelligence. *Computer*, *51*, 38–46. https://doi.org/10.1109/MC.2018.3620963

Schutte, P. C. (2015). How To Make the Most of Your Human: Design Considerations for Single Pilot Operations. Retrieved from https://ntrs.nasa.gov/search.jsp?R=20160006283

Sheridan, T. B. (2016). Human-Robot Interaction. *Human Factors*, *58*(4), 525–532. https://doi.org/10.1177/0018720816644364

Siau, K. (2018). *Building Trust in Artificial Intelligence, Machine Learning, and Robotics Supply Chain Management View project*. Retrieved from www.cutter.com

Sinclair, J., & Lee, I. (2017). A generic cognitive architecture framework with personality and emotions for crowd simulation. In *2017 12th International Conference on*

Intelligent Systems and Knowledge Engineering (ISKE) (pp. 1–6). IEEE. https://doi.org/10.1109/ISKE.2017.8258798

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. https://doi.org/2012arXiv1206.2944S

Strannegård, C., Von Haugwitz, R., Wessberg, J., & Balkenius, C. (2013). A cognitive architecture based on dual process theory. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 7999 LNAI, pp. 140–149). https://doi.org/10.1007/978-3-642-39521-5_15

Sun, R. (2004). Desiderata for cognitive architectures. *Philosophical Psychology*, *17*(3), 341–373. https://doi.org/10.1080/0951508042000286721

Sun, R. (2015). Interpreting psychological notions: A dual-process computational theory. *Journal of Applied Research in Memory and Cognition*, *4*(3), 191–196. https://doi.org/10.1016/J.JARMAC.2014.09.001

Tarola, C. L., Hirji, S., Yule, S. J., Gabany, J. M., Zenati, A., Dias, R. D., & Zenati, M. A. (2018). Cognitive Support to Promote Shared Mental Models during Safety-Critical Situations in Cardiac Surgery (Late Breaking Report). In *Proceedings - 2018 IEEE International Conference on Cognitive and Computational Aspects of Situation Management, CogSIMA 2018* (pp. 165–167). IEEE. https://doi.org/10.1109/COGSIMA.2018.8423991

Van Der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, *9*(Nov), 2579–2605. https://doi.org/10.1007/s10479-011-0841-3

Vaughan, S. L., Mills, R. F., Peterson, G. L., Grimaila, M. R., Rogers, S. K., Oxley, M. E., & Patterson, R. E. (2016). A dual-process Qualia Modeling Framework (QMF). *Biologically Inspired Cognitive Architectures*, *17*, 71–85. https://doi.org/10.1016/j.bica.2016.07.001

Vityaev, E. E., & Demin, A. V. (2018). Cognitive architecture based on the functional systems theory. *Procedia Computer Science*, *145*, 623–628. https://doi.org/10.1016/j.procs.2018.11.072

Wang, Y., & Liu, G. (2009). Research on Relationships Model of Organization Communication Performance of the Construction Project Based on Shared Mental Model. In *2009 International Conference on Information Management, Innovation Management and Industrial Engineering* (pp. 208–211). IEEE. https://doi.org/10.1109/ICIII.2009.57

Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., Petitjean, F., Webb, G. I., … Petitjean, F. (2016). *Characterizing Concept Drift*. Retrieved from https://arxiv.org/pdf/1511.03816.pdf

Weber, K., Ritschel, H., Lingenfelser, F., & Andre, E. (2018). Real-Time Adaptation of a

Robotic Joke Teller Based on Human Social Signals.

Weiss, W., Fuhrmann, F., Zeiner, H., & Unterberger, R. (2017). Towards an Architecture for collaborative Human Robot Interaction in Physiotherapeutic Applications. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction - HRI '17* (pp. 319–320). https://doi.org/10.1145/3029798.3038393

Wiltshire, T. J., Warta, S. F., Barber, D., & Fiore, S. M. (2017). Enabling robotic social intelligence by engineering human social-cognitive mechanisms. *Cognitive Systems Research, 43*, 190–207. https://doi.org/10.1016/j.cogsys.2016.09.005

Woźniak, M., Ksieniewicz, P., Cyganek, B., & Walkowiak, K. (2016). Ensembles of heterogeneous concept drift detectors - Experimental study. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 9842 LNCS, pp. 538–549). Springer International Publishing. https://doi.org/10.1007/978-3-319-45378-1_48

Ye, P., Wang, T., & Wang, F.-Y. (2018). A Survey of Cognitive Architectures in the Past 20 Years. *IEEE Transactions on Cybernetics*, *48*(12), 3280–3290. https://doi.org/10.1109/TCYB.2018.2857704

Yen, J., Fan, X., Sun, S., Wang, R., Chen, C., Kamali, K., & Volz, R. A. (2003). Implementing Shared Mental Models for Collaborative Teamwork. In *The Workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments in the EEE/WIC Intelligent Agent Technology Conference, Halifax, Canada*.

Zimek, A., Schubert, E., & Kriegel, H. P. (2012). A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, *5*(5), 363–387. https://doi.org/10.1002/sam.11161

Appendix A: Research Community Goals

AI first started out as handcrafted software for narrow problems which was useful for finding a reasonable answer to a new query. An example would be linear regression. The next wave of AI began to incorporate feedback, so the AI models could "learn" from new observations. This second wave is where we currently progressed to with Deep Neural Networks and Reinforcement Learning (Launchbury, 2017). The third wave, which a lot of research is currently directed, will partake in widening the scope of possible problems to solve and learn from. In short, researchers wish to endow AI with better critical thinking and problem-solving skills when coming across a new problem.

A proper starting point in solving the narrow scope of current AI would be to determine if anything exists that currently exhibits excellent generalized problem-solving skills. There might be something that uses past experiences or relatable problem sets to solve a new problem existing in a completely different task space. It just so happens that such a phenomenal object exists: the human brain. Several propositioned AI algorithms, including DPM-MN, attempt to imitate biological solutions. A couple examples include genetic algorithms (Gnanaprasanambikai, et al., 2018) and neural networks. The new frontier for biological imitation is the brain/mind. DPM-MN specifically targets the mental model of a human teammate to boost teamwork.

Cognitive architectures try to copy the functions of the mind. It is a field of research which is hopeful in solving the narrowness of problems a single AI agent can handle. Currently, an AI can be programmed and can learn a contained problem space such as figuring out the best move in a game of chess. However, if that same AI were tasked with deciding the best move in a game of checkers instead, it would fail tremendously. Suddenly,

all the rules have changed, and the AI may not even be able to process the situation, let alone figure out an optimal choice.

A few of the concepts surrounding cognitive architectures are ways to learn new problems, how to produce a solution if there is no “right” choice, and the combining of different algorithms to figure out an answer. These concepts relate to the operation of a person. When encountering any problem, multiple processes are occurring at the same time. For example, if a person needs to kick a soccer ball to their teammate, they are: thinking about the best teammate to kick it to, deciding if it is better to keep dribbling instead, using their eyes to determine where the ball is located, keeping proper balance to perform the kick, and paying attention to possible opposing players challenging the kick. All of these separate processes ultimately determine the final action.

Improving cognitive architectures is a first step in the creation of a general AI. AGI is the sci-fi fantasy of creating a human-like robot such as R2D2 or C3PO from Star Wars. AGI is “aiming to build agents that encompass the whole breadth of human intellectual faculties and more”(Schaul, et al., 2011). If the end goal is to make a human-like robot, then a cognitive architecture can be used to make decisions that are similar to how a human would decide. This increases the similarity between an AGI agent and a human. There is skepticism as to whether or not an AGI, sometimes referred to as “strong AI”, is fully possible (De Ruiter, 2006). However, with research development in cognitive architectures, there may exist a close enough solution.

DPM-MN is categorized as a cognitive architecture. It may possibly push the boundaries that AI researchers as a whole consider important such as the realization of the

third wave of AI, cognitive architectures, and AGI. These three primary concepts are motivation for DPM-MN from a research community perspective.

DPM-MN advances the third wave of AI by discovering the mental model of a person. Once a shared mental model is obtained, an autonomous agent can understand an individual human through a teamwork viewpoint. The autonomous agent should be able to predict what the human teammate will do in a given situation. It should also be able to imitate the human from which the mental model was obtained. DPM-MN will have correctly copied a person's mental model when the prediction of the autonomous agent is measurably similar to the actual response the person would have given in the same scenario.

Appendix B: Dual-Process Theory Influence on DPM-MN

The following is a list of knowledge sources supporting why DPM-MN is a dual-process theory-based cognitive architecture:

1) Dual-Process Theory attempt – A dual-process cognitive architecture should attempt to capture the implicit versus explicit framework of decision making, even though it is difficult to concretely define these ambiguous concepts (Sun, 2015). DPM-MN uses two distinct systems in parallel to aid in decision making. These two systems interact with each other and have their own unique prediction process corresponding to previous researchers' ideas about the dual-process theory. The primary point is that the dual-process theory is philosophically based so there is no exactly true definition of System 1 or System 2. Although, that does not preclude a growing understanding of the concepts.

2) Computational complexity difference – System 1 represents an automatic, or implicit, system while System 2 represents an explicit system, or rather a system utilizing working memory (Frankish, 2010)(Wiltshire, et al., 2017). DPM-MN's System 1 directly adds observations. There is no prior process involved to determine the representation of an observation in System 1. However, System 2 uses working memory through an eager learning scheme. The prediction querying of new observations may be similar in System 1 and System 2, but the System 2 concepts are built through calculating an appropriate distribution through a Gaussian kernel prior to the addition of a new concept.

3) Reasoning difference – The System 1 processes are more associative, or similarity-based, while the System 2 processes are more rule-based (Frankish, 2010). In DPM-MN, System 1 predictions derive from new observations being close to former individual observations. System 2 predictions come from new observations being close to concepts created over time through a build-up of individual observations.

4) Sequentiality – The model learns over time. Human learning and behavior is temporal in nature (Sun, 2004). DPM-MN reads new observations on a timeline. In order to learn behaviors, the observations should be seen by the model in sequential order.

5) Trial-and-error – Humans learn reactively to routines and they adapt according to the empirical results (Sun, 2004). DPM-MN maintains the model structure when new predictions are highly confident and correct.

6) Synergistic interaction – A dual-process model should synergistically interact the explicit and implicit processes (Sun, 2004). Instead of simply being two different models, the two components should harmonize their functionality to aid one another. DPM-MN contains several functional features that cause interaction

between the two systems. For example, when a System 2 concept is retracted, some of the underlying distribution are sent back to System 1.

7) Bottom-up learning – The research among psychologists indicates a tendency for humans to build implicit knowledge first before explicitly creating concepts (Sun, 2004). This describes the "a posteriori" knowledge pattern. DPM-MN creates its concepts in System 2 from the implicit knowledge built through System 1. A top-down approach contends with the bottom-up learning. This perspective focuses on the idea that basic concepts are first intuitively consulted, and then the specific situation is considered if an incomplete answer is provided. This is similar to other dual-process approaches that first consult the obvious solution before considering the in-depth analytical solution. Although, the top-down approach is different because, in relation to the rule-making process, it is assumed that the top-level concepts are already known. Therefore, the top-down learning perspective is dual-process related because of the inference computational cost rather than because of the process for accumulating knowledge. It is worth noting that Sun (Sun, 2004) acknowledges a top-down dual-process approach is prevalent in dual-process implementations. However, it requires "a priori" explicit knowledge of some form. Implicit knowledge can be gained through explicit knowledge and vice-versa. The scope of DPM-MN is currently narrowed to dual process learning characteristics.

8) Modularity – The cognitive architecture should have functional modularity (Sun, 2004). The model should be capable of evolving through the addition of important functionalities. Functional modularity of simple functions facilitates the emergence of higher-ordered functions. In DPM-MN, learning volatility can be seen as the interaction between the System 2 window size (W_s) for a class and the requirements for System 1 points to become a System 2 concept distribution. Another example is shown in CLARION where the unpacking principle and ascertainment bias are revealed through the basic interactions of lower-ordered functions (Helie, et al., 2011).

9) Minimalism – Minimalism can refer to either minimal initial structure or minimal knowledge representations (Sun, 2004). DPM-MN utilizes both to try and keep a minimalist approach. The model starts out blank and begins to learn once new observations are trained on. The knowledge representations are comparatively minimal as well since the learned experiences are represented as state-space mapping through a Gaussian kernel.

10) Confidence levels – Similar to human thinking, DPM-MN contains an element of confidence. As also observed in CLARION, confidence levels allows for an element of uncertainty (Helie, et al., 2011).

11) Memory – There are many different types of memory that a cognitive architecture may capture. These include declarative memory (further sub-divided into episodic memory and semantic memory), procedural memory, and working memory

(Goertzel, et al., 2013). In DPM-MN, episodic memory and working memory are primarily applied since the model attempts to functionalize experiences and the created concepts of System 2 maintain a working memory of the individual experiences that make up any given concept.

Appendix C: DPM-MN Dual-Process Theory Functions

Concept Drift.

Learning and reacting to situations develops over time. People gain new experiences, learn from the past, and explore new options. It would be asinine to assume that people will forever react to a situation the same way as the way they acted during their first experience. People will evolve their understanding of a problem and as a result develop their reaction. A person may learn that a state-response pair in the early stages of a new problem may cause a better outcome if the response was actually given during a different situation. Alternatively, it could be possible that a state instance is not encountered for such a long period of time that it seems like new again when revisited.

When first hired, a person may start taking the bus to work every time they are running late. They begin to learn that if they take the bus to work when they are running late, they will never make it to work on time. As a result, the person starts only taking the bus when they are running early with their schedule.

Overwriting Previous Knowledge.

Concept drift deals with the change of a state location for a certain response. Overwriting previous knowledge deals with the change of a response for a certain state location. Given the same state, people will usually evolve their response over time. A common cause of this effect is learning. Usually a non-optimal response will be given when someone first encounters a problem. Over time, a different and more optimal response becomes the norm for the problem.

Relating the overwriting of previous knowledge to the analogy, the person first started off taking the bus to work every time they were running late. Once they learned that

they would not give to work on time, they instead substitute the bus response with a different one. The person instead may decide to start taking their personal car when they are running late. They are rewarded for finding a better response to the running late situation, and now they are sometimes on time for work when running late.

A new response has taken the place of an old response for a given state. It is important for the new response to overwrite the old response rather than merely adding to it. If the old response is not erased, it starts to cause irreducible error in the response. The stale reaction creates randomness between multiple decisions given a state. Now imagine the effect over a long period of time. Many different responses would overlap each other, and the model would become no better than random chance.

Retaining Past Experience Memory.

Dual-Process Model using multivariate normal probability density functions (DPM-MN) parameterizes the memory capacity of the model even though it is artificially limiting the capability of the computer. Why force the computer to 'forget' certain concepts or experiences when it possibly has the capacity to remember every experience – either individualized in System 1 or aggregated somewhere in System 2? To achieve the goal of human behavior, rather than rational optimality, these artificial limitations are meant to mimic the limitations of the human behavior. By limiting capacity, it is a way to enhance the importance of relevancy.

Even though some memory capacity is limited, there are parts which try to keep certain memories for functional purposes. The first example is the perpetual outlier. An outlier in System 1 will stay in System 1 if no other identical class responses are placed near it. This captures the idea of one-off scenarios. People may remember singular

instances of certain experiences. Many people can answer if they have ever traveled to a location better than how many times they have traveled to a location. The former highlights the ability to remember outlier situations while the latter shows the memory leakage of remembering specifics about everything in their life.

Another concept related to the retainment of past experiences is the revocation of System 2 knowledge concepts. If a System 2 concept, represented by a multivariate normal probability density function, is erased, a fraction of the underlying points which make up that erased concept are sent back to System 1. This keeps the harmony between updating knowledge concepts and holding onto memories of past experiences. Only a fraction of the points is sent back to System 1 to reduce thrashing between both systems.

Aggregation of Experiences to Form a General Concept.

System 1 represents the individual experiences and System 2 represents the general concepts of knowledge gained from the conglomeration of individual experiences. When growing the knowledge base, if a response to a handful of similar situations occurs enough times, it then moves from System 1 and into System 2. The new System 2 knowledge concept is created through processing all of the individual responses moving from System 1 to System 2. The System 1 points influence close-by queries. On the other hand, because the System 2 concepts are mapped by multiple System 1 points, the System 2 concepts cover a larger area which allows for more generalization; the new query states do not have to be extremely close to a previous point for System 2 to provide a confident response.

A specific concept can be refreshed if it provides confident and correct responses to new information. This ensures the most relevant concepts to stay in System 2 while the outdated concepts are the ones that get revoked. System 2 concepts also determine if DPM-

MN needs to learn the incoming information. If System 2 cannot provide a correct response with high confidence, the new input is placed into System 1. When a new input does not need to be learned, it acts as an indication that the System 2 generalizations are currently correct. Once enough points of the same class are in System 1, a new concept can be created to cover the newly found pattern.

Outlier Sensitivity.

When encountering a new situation, a person either has to recall on past experiences or discern an action based on applicable life experiences. The second time a person encounters a new situation, they have at least the first time to recall. DPM-MN wants to capture that first experience to possibly use in the second encounter. As a result, observations which are seldom encountered are still stored in System 1 for later use. If a new observation is close enough to that previous outlier situation, then it will provide the same response learned from before. The advantage of outlier sensitivity is to allow for variance in responses when the setting is appropriate. DPM-MN uniquely prevents an onslaught of new observations from causing the decision-making process to trend away from that one-off scenario. It usually is an appropriate choice to make since in most cases, the one-off scenario will not be visited again. However, DPM-MN allows the reuse of the response from the outlier experience if the scenario is nearly similar. The scenario has to be extremely close in order to activate the observation in System 1, or else it will be generalized on by System 2.

Along with the behavior of outliers in System 1, DPM-MN prioritizes the population of outliers in System 1. When a new observation point is queried, System 2 is first consulted to determine if a confidence answer can be given. If System 2 cannot

confidently generalize on a new observation, the new observation is placed into System 1. This process allows outliers to almost always make an appearance in System 1 even if it near-randomly receives a correct prediction from DPM-MN.

Online Learning.

The final form of a DPM-MN model should process live observations in real-time. After it has been trained on previous data, it can accept datapoints as they arrive. A continuation of learning and model updating takes it from a functionalization of the past and turns it into a functionalization of the present.

Bootstrapping the model with previous information is important to get a baseline and to quicken the whole process. Even though the data is temporal, or the data is processed according to a timeline, the previous information will still capture ground truths and recent behaviors. It is also this model characteristic that allows the possible alteration from a general model using all available data into a specific model using a specific person's data alone. The generalized model would first capture common state-response pairs shared by everyone, and then the specific person's data would capture idiosyncrasies.

Overall, the most important aspect of online learning is the passive continuation of the model in real-time. DPM-MN has the capability to learn as time continues through every new observation. The model is flexible enough to allow this online learning; DPM-MN can slowly change shape over time to reflect knowledge that prioritizes recent observations. It acts like a sliding window function that fades out most of the information from the distant past while keeping key outlier observations.

Balance Between Implicit and Deliberate Systems.

When a person participates in decision-making, they have many alternative choices. There are two primary, and competing, decision-making processes: the implicit process and the deliberate process. These two processes are otherwise known as System 1 and System 2 (or Type 1 and Type 2) in the dual-process theory (J. Evans, et al., 2013). Depending on the circumstances, one process overcomes the other when coming to an ultimate decision. Dividing the decision-making process into two different and distinct processes is a trademark of the dual-process theory and subsequentially DPM-MN.

In DPM-MN, System 2 symbolizes the deliberate thought process and System 1 symbolizes the implicit thought process. There is a balance between each of these systems. Given a new query, each system returns their own prediction and a number indicating the confidence of their prediction. Because the knowledge representation of the System 2 concepts and System 1 points are created through the use of multivariate normal probability density functions, granularity is provided. It is extremely unlikely that System 1 and System 2 tie in their confidence numbers. The balance between System 1 and System 2 is controlled by a hyperparameter which determines the influence reach of all the points in a particular system. If System 1's influence reach is high, then more points in System 1 will be confident on predicting a response for a new observation. The probability density function of each point smooths out to reach a larger area.

Appendix D: Bias-Variance Tradeoff in Experimental Design

The dataset includes thirty-five different players. When the goal is to create a model that maps the behavior of a specific individual, noise is introduced in the idiosyncrasy detection when using thirty-five different players' data. At the same time, generalized behavior becomes the dominating quality in the model. The behavior common to all the players will be reinforced while the peculiar actions of a single player will get concealed by the rest of the data.

All of the data will be tested as a relative statistic, but more success will most likely come from weighting the single player data in some way. Since the goal is to map an individual's behavior, it is better to prioritize detecting idiosyncrasies. The weighting of a single player can be done in two different ways: only the single player data is used, or the single player's data is weighted as more important than the other behavior. Technically, only using a single player's data is weighting that data as one-hundred percent and the rest of the data as zero percent. If only the single player's data is used, the model will directly reflect the chosen player's state-response pairs. Although, a lot of the data will be essentially thrown out.

One method to utilize all the available data while still focusing on the single player would be to find generalized behaviors through everyone else's data, and then focus on the single player's data. Because recency is important in DPM-MN when training, simply looking at the chosen player's data last should allow idiosyncrasies to emerge. The generalized behavior should agree with the chosen player and the idiosyncrasies that disagree with the starting model should move out any other state-responses occupying similar state locations as long as the model is volatile enough. This method may also

quicken training in general since the model is being bootstrapped with common behaviors. Another possibility is to give extra attention to the single player by running through their data twice. The chosen player's data will be accentuated because of multiple encounters.

Appendix E: Data Preprocessing

The state-response data comes from a *Space Navigator* experiment conducted in 2015 by Major Bindewald. The study participants simply played *Space Navigator*. Whenever they drew a path between a planet and the intended goal, various game state features and the points which make up the trajectory were recorded.

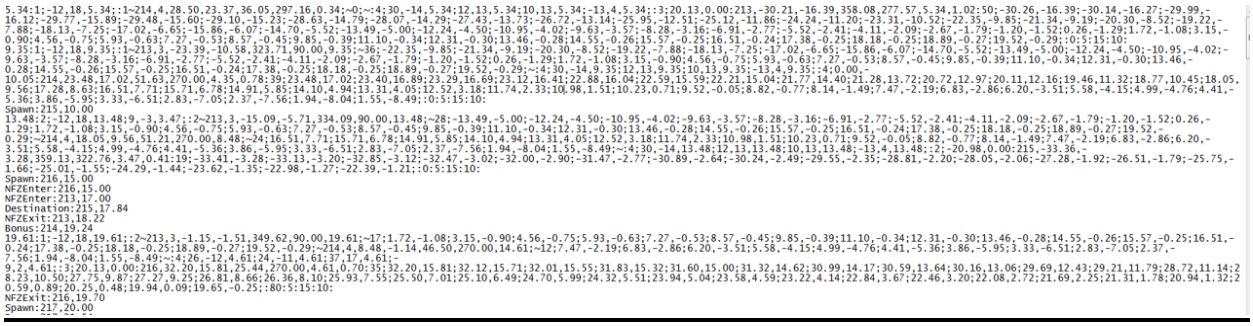

Figure 46: Raw data input example.

After capturing the data from an experiment, Figure 46 is what the raw data looks like. 35 players completed 16 levels each and generated a total of 68,538 different trajectory instances. Every line starting with a number is a single example of a drawn trajectory given the state instance.

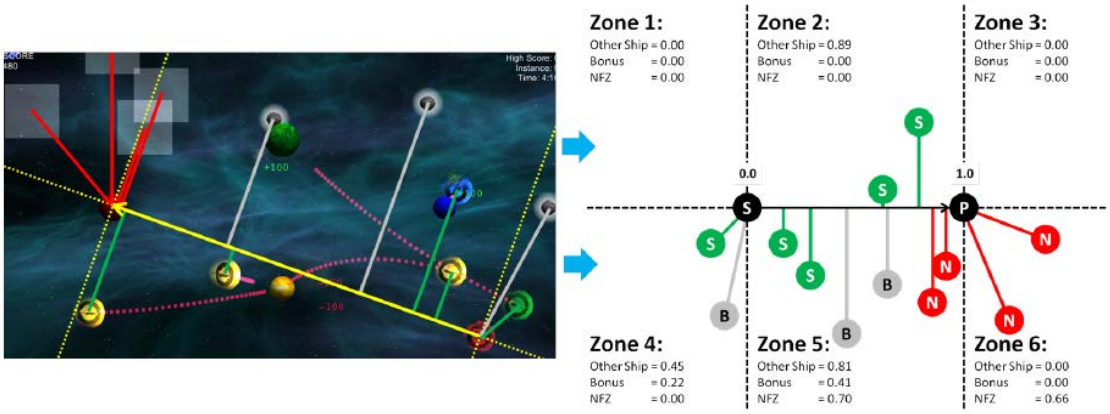

Figure 47: A Space Navigator screen capture highlighting important game objects.

Algorithm 1 State-space feature vector creation algorithm.

input: L = straight-line trajectory from spaceship to destination planet.
initialize: $\eta \in [0.0 \ldots 1.0)$ = weighting variable; s = empty array (length 19); $zoneCount = 0$
Translate all objects equally s.t. the selected spaceship is located at the origin.
Rotate all objects in state-space s.t. L lies along the X-axis.
Scale state-space s.t. L lies along the line segment from $(0, 0)$ to $(1, 0)$.
for each object type $v \in (OtherShip, Bonus, NFZ)$ **do**
    $relevance = 1.0$
    for each zone $z = 1 \rightarrow 6$ **do**
        $zoneCount = zoneCount + 1$
        for each object o of type v in zone z **do**
            if (type $v ==$ $OtherShip$) **then**
                $relevance$ = calculate new relevance ▷ See Appendix B
            d_o = the shortest distance of o from L
            $w_o = relevance \cdot (e^{-(\eta \cdot d_o)^2})$
            $s[zoneCount] = s[zoneCount] + w_o$
$s[19]$ = the non-transformed straight-line trajectory length
return s, normalized between $[0, 1]$

Algorithm 2: State-space feature vector creation.

After the data is gathered, it needs to be preprocessed to become suitable for training (Figure 47). Each feature value in the data becomes normalized between zero and one. Without normalizing the features, the straight-line trajectory length (s[19] in Algorithm 4) will overshadow the other features because of the large scale it exists on.

In addition to Algorithm 2, the Gaussian weight function is multiplied by the relevance variable for the other ship object types. This was added to the algorithm because it is important to capture which direction the other ships are going. Intuitively, if the other ships are moving away from the direct path between the selected ship and the destination planet, it poses less of a threat than other ships moving straight toward. If a non-selected ship is moving straight towards the area of interest, its Gaussian weight function is multiplied by two. If a non-selected ship is moving away from the area of interest, its

Gaussian weight function is multiplied by zero. All of the values in between towards and away act on a continuous function between two and zero. For instance, a ship in zone two moving parallel to the line of interest has its Gaussian weight function multiplied by one. For more information on this additional metric, see Appendix F.

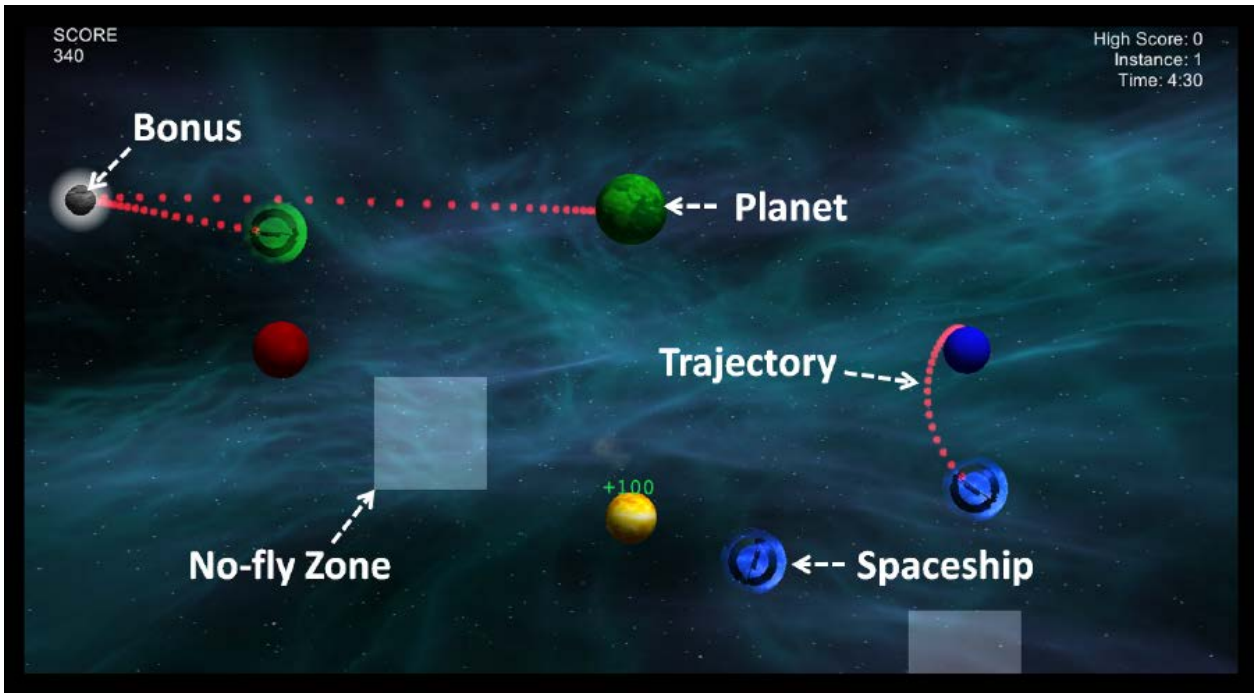

Figure 48: Important parts of Space Navigator.

There are 68,538 observations. Each state includes nineteen features. Algorithm 2 shows how these features are computed. Figure 48 shows what each state object looks like. The bonuses give extra points, the no-fly zones take away points whenever a ship exists within it, and a lot of points are given when a spaceship reaches a planet of the same color. Each of the six zones contains three features. That accounts for eighteen of the nineteen features. The last feature is the non-transformed straight-line trajectory length. This last feature represents the straight-line distance between the spaceship and the planet.

The output variable will be a trajectory sequence. The accuracy performance measurement will be the average Euclidean distance between the trajectory points guessed by the AI and the state transformed trajectory points actually drawn by the experiment participant. Part of the trajectory normalizing process includes standardizing the number of points in each trajectory by interpolating the points. Every trajectory is interpolated to contain twenty-five points. See Appendix G for possible problems with this interpolation method.

A graphical example of the state capture and trajectory is shown below. The red squares are the no-fly zones. The green plus signs are the bonus points, the red stars are the other ships, a trajectory coming from a red star is a straight-line trajectory for that ship, the black “X” is the destination planet, the black “O” is the selected ship, the black line connecting the two is the straight-line trajectory, and the blue line is the user-drawn path.

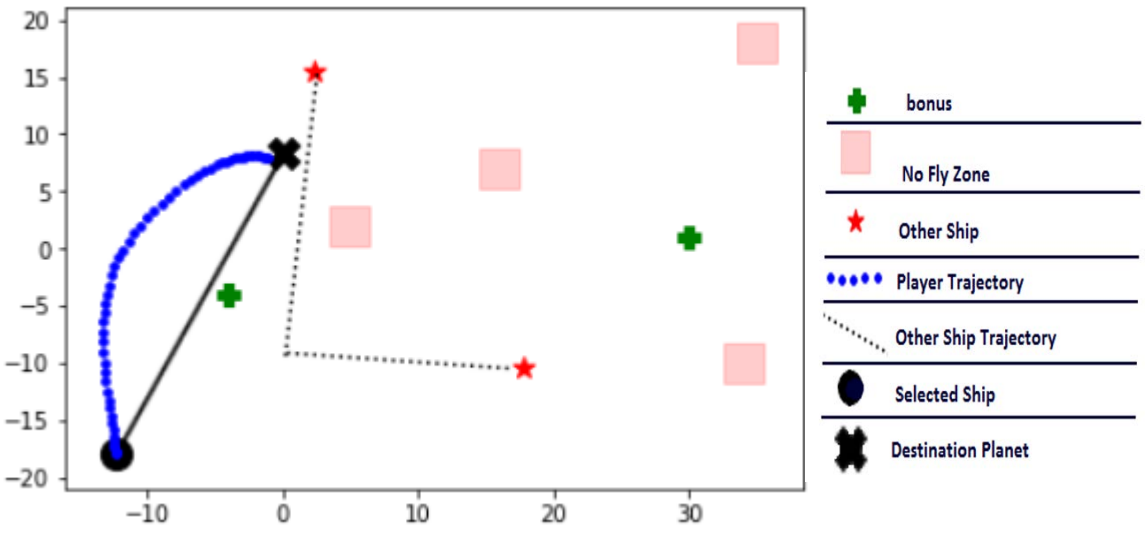

Figure 49: Initial data capture and graphical representation.

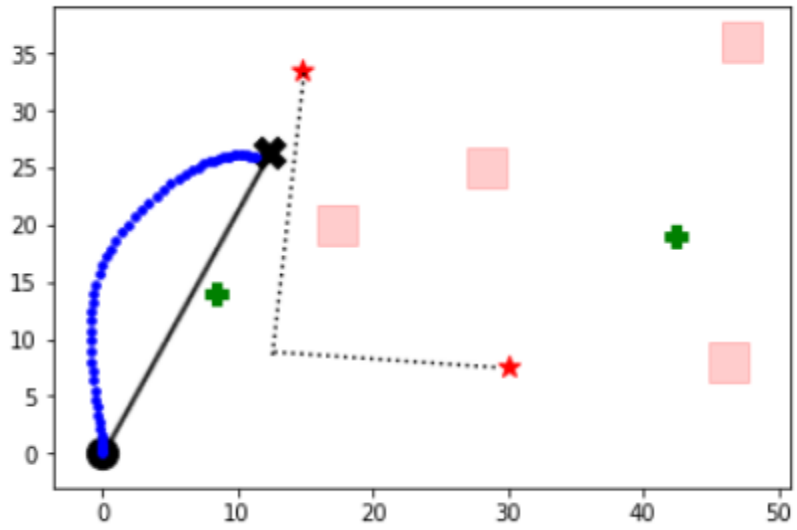

Figure 50: Translation of objects.

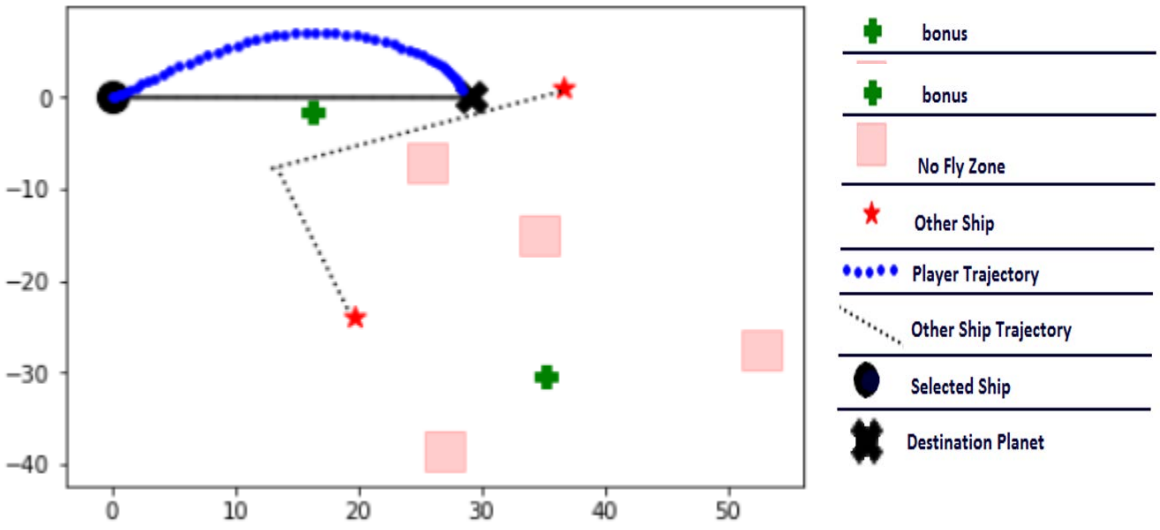

Figure 51: Rotation of objects.

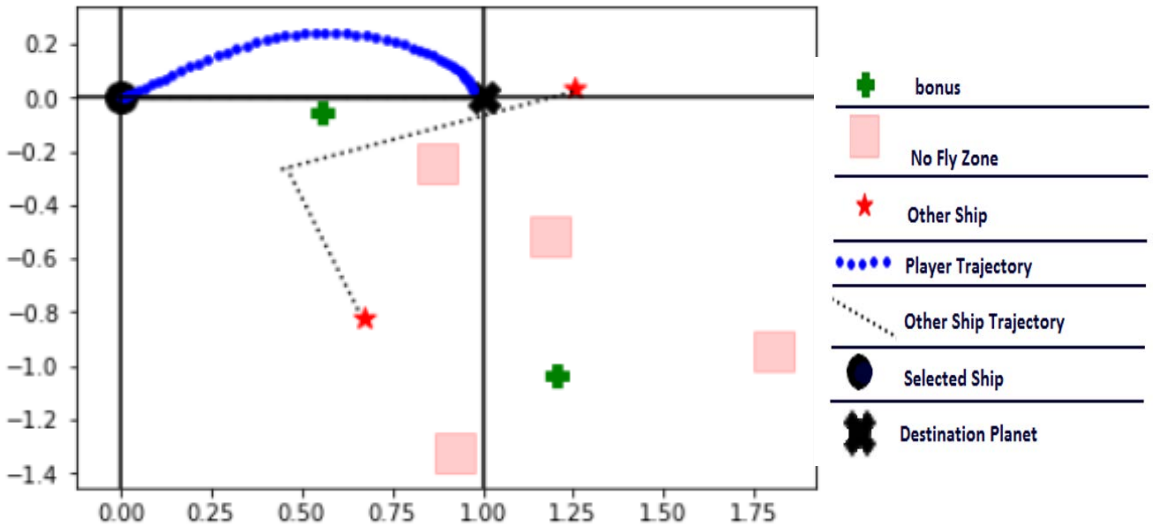

Figure 52: Scaling of objects and addition of zone lines.

The initial data state is first translated so the selected ship is at coordinate (0,0). Then, all objects are rotated so the selected ship and the destination planet are on the same Y-coordinate plane. Finally, everything is scaled so the distance between the selected ship and the destination planet is equal to one. This transformation is pictured with Figure 49 to Figure 52.

```
Player: 029
Level: 5
        os1       os2  os3  os4       os5       os6  bon1  bon2  bon3  \
4  0.856252  3.233753  0.0  0.0  0.195841  1.45491   0.0   0.0   0.0

       bon4  bon5  bon6  nfz1  nfz2  nfz3      nfz4  nfz5      nfz6     linlen
4  0.965102   0.0   0.0   0.0   0.0   0.0  0.986463   0.0  0.893812  22.540719
```

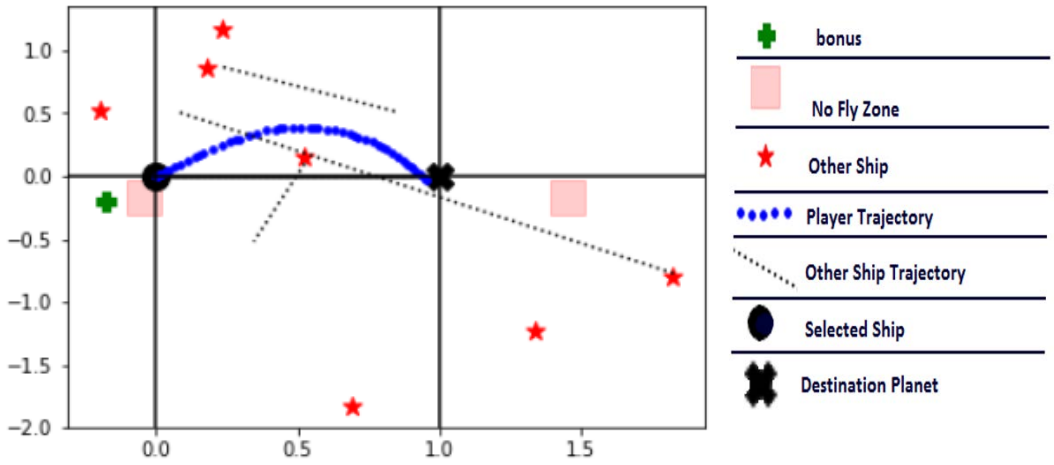

Figure 53: Example of non-normalized feature values for a normalized state space.

Figure 53 is an example of the non-normalized values for a random state. The weighting variable was at 0.7. “os1” stands for “other ships in zone 1”. “bon” is short for “bonuses”. “nfz” is short for “no-fly zone”. Finally, “linlen” is the original straight-line length between the selected ship and destination planet. Looking at random examples along with their variables allows a quick sanity check on the algorithm to see if the values are in the general vicinity of expectations. This example from player 29 on level 5 contains a few ships in zone 2, with one of them extremely close to the line-of-interest and also heading towards it. It is expected that “os2” will have a high value and as seen above, it does. There are no bonuses in zones 1,2,3,5, or 6, so they are expected to be at a value of zero -- which they are.

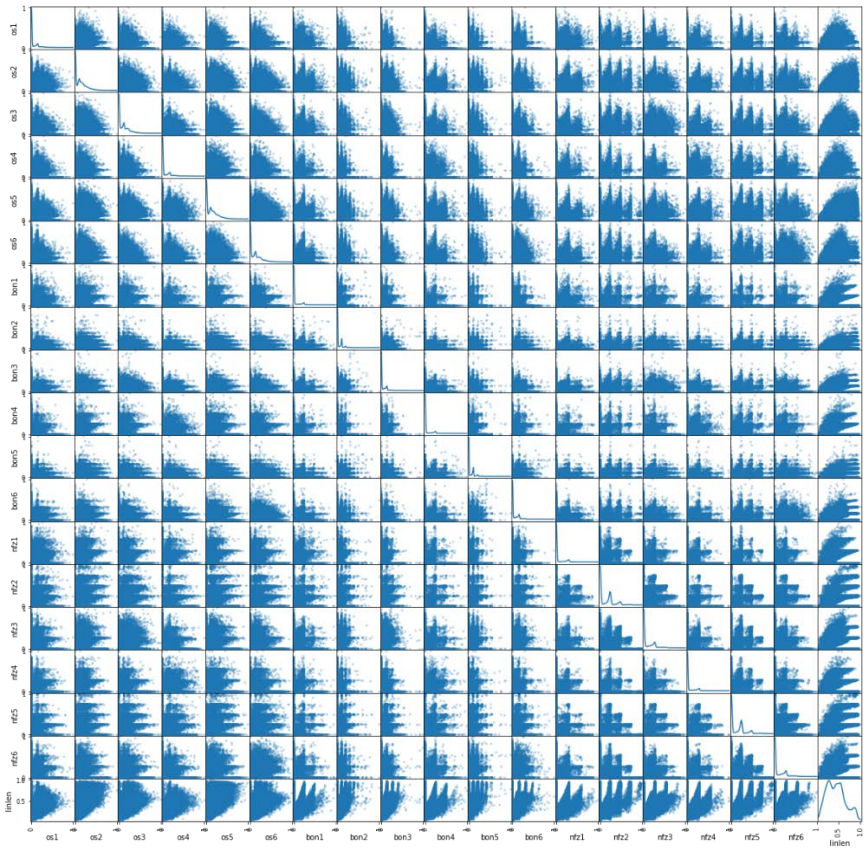

Figure 54: Scatter matrix of features.

Because of the number of features, Figure 54 is best viewed on a computer so zooming in is possible. As seen above, most of the features have near zero correlation. This is expected since the features are virtually independent of each other. A bonus in zone two has no effect on the likelihood of a no-fly zone spawning in zone five. An interesting feature of the data is the sparsity of high values. It is most likely for each feature, except the straight-line length, to have a near zero value.

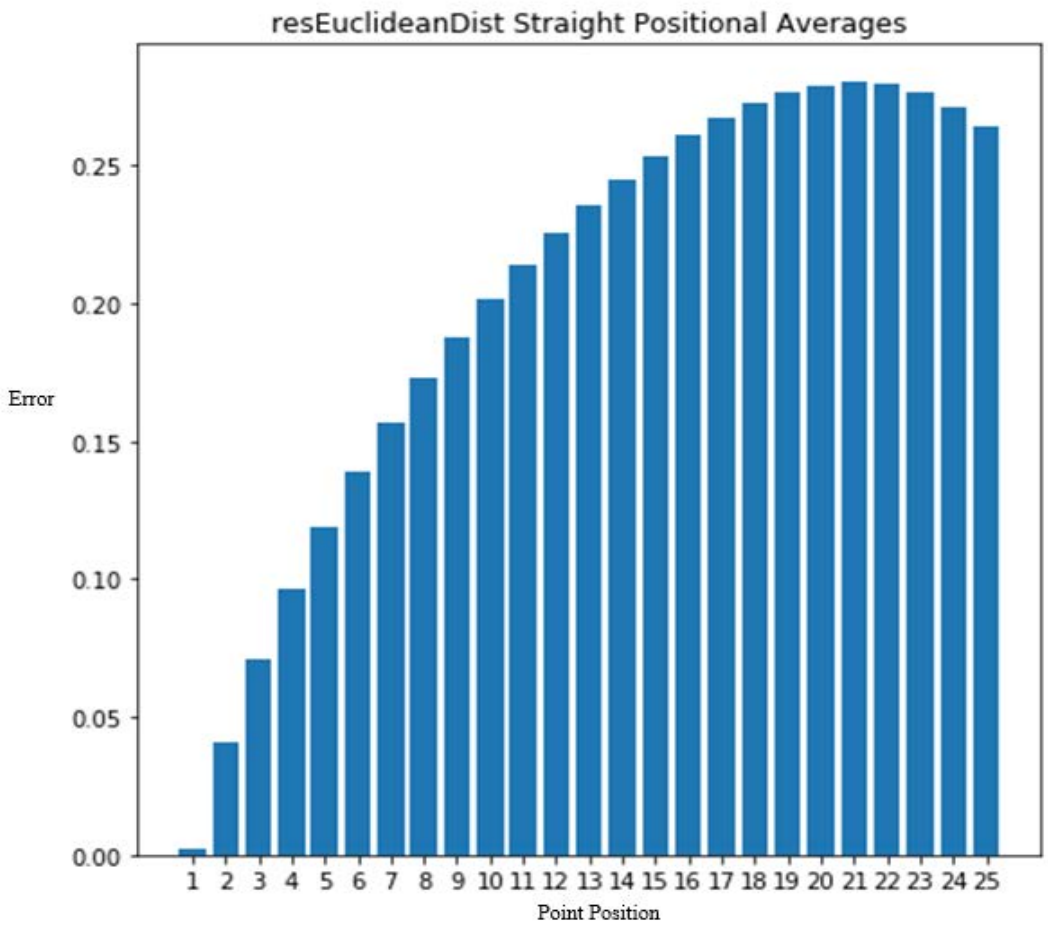

Figure 55: Data Euclidean distance.

Figure 55 shows the full data trajectory average Euclidean distance from the center line. The first trajectory point is almost always completely accurate. The trajectories begin on the selected ship so that makes sense. The distance grows as the points progress until a slight dip at the very end. The error in the trajectory points compound over the space. If point one is messed up, that will also mess up point two, and so on. The end dips a little because most trajectories end at the destination planet. However, a substantial number of trajectories are nowhere near the destination planet, so it still has a high average point distance.

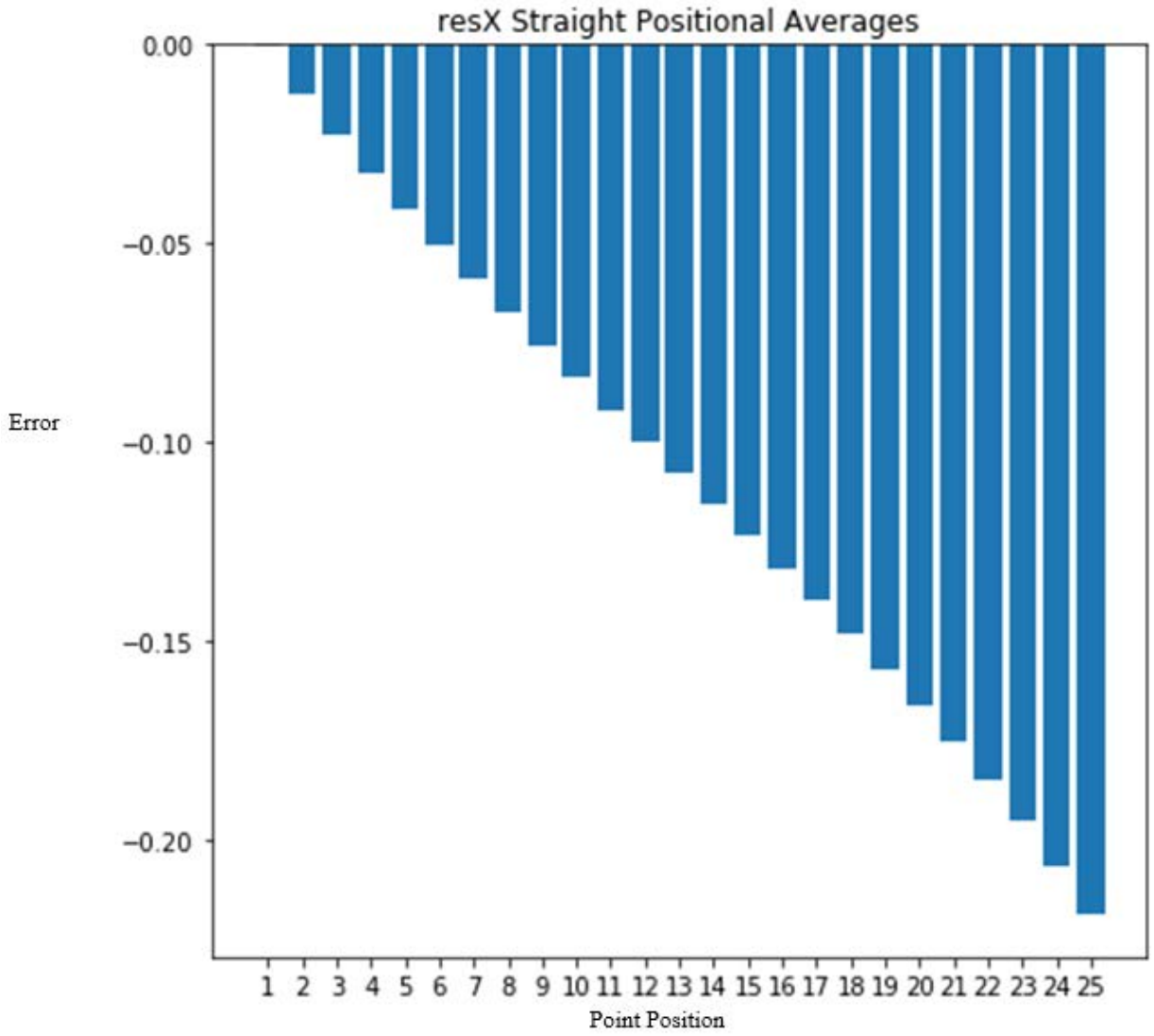

Figure 56: Data trajectory X value residual error averages.

The residual error is calculated as the truth minus the prediction. In this case, the prediction is always a straight line from the selected ship to the destination planet. Because of this standard trajectory, the trajectories can be analyzed across the whole dataset. As seen in Figure 56, the true trajectories, on average, are shorter than the middle line. The x-values for each trajectory point are not progressing across the x-axis as fast as the straight-line.

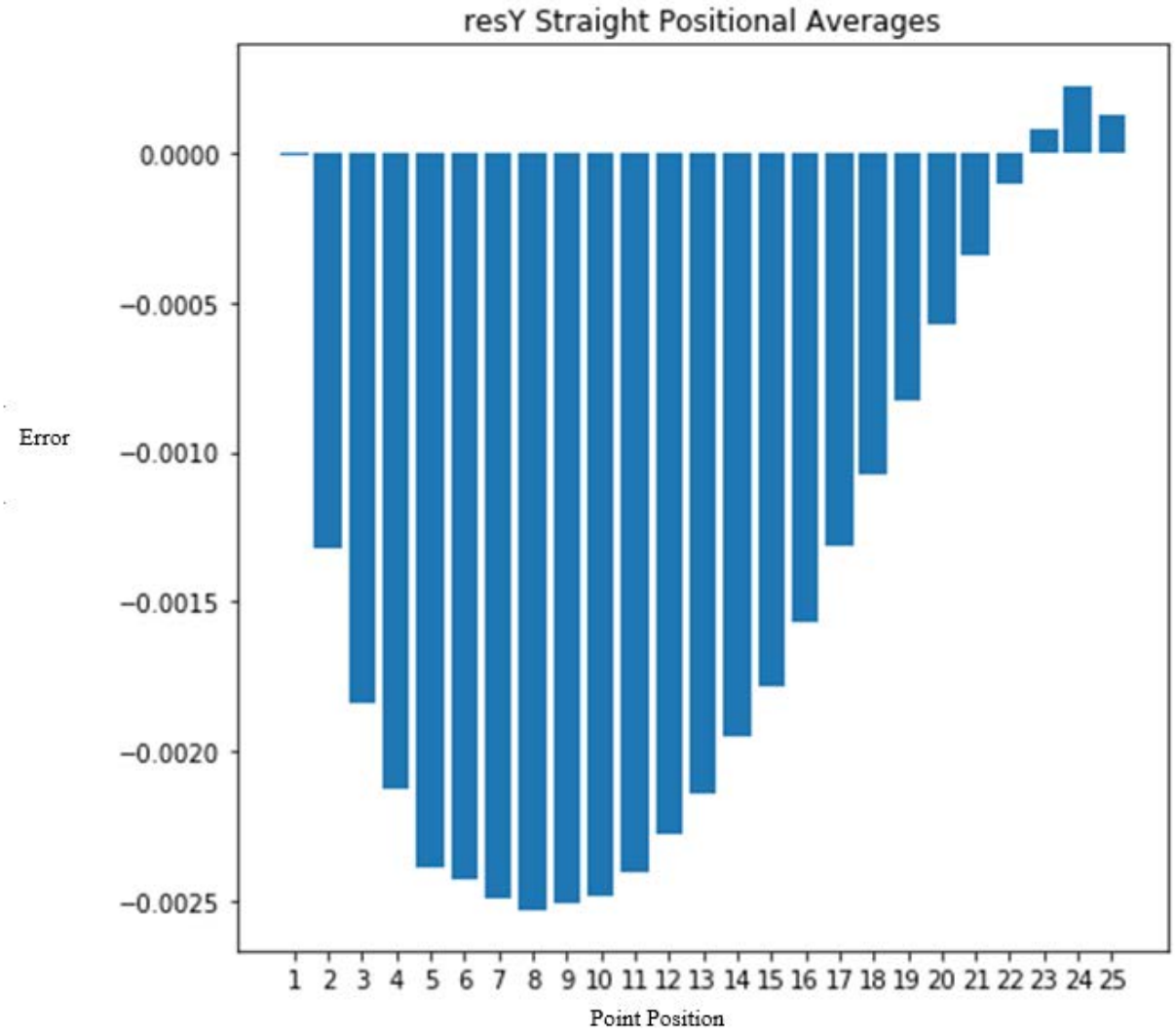

Figure 57: Data trajectory Y value residual error averages.

Looking at Figure 57,the true trajectories are placed, on average and for most of the point locations, underneath the middle line. In fact, there even exists a parabola-like shape indicating a fall and rise in the trajectory

Appendix F: Other Ship Gaussian Weight Function Multiplier

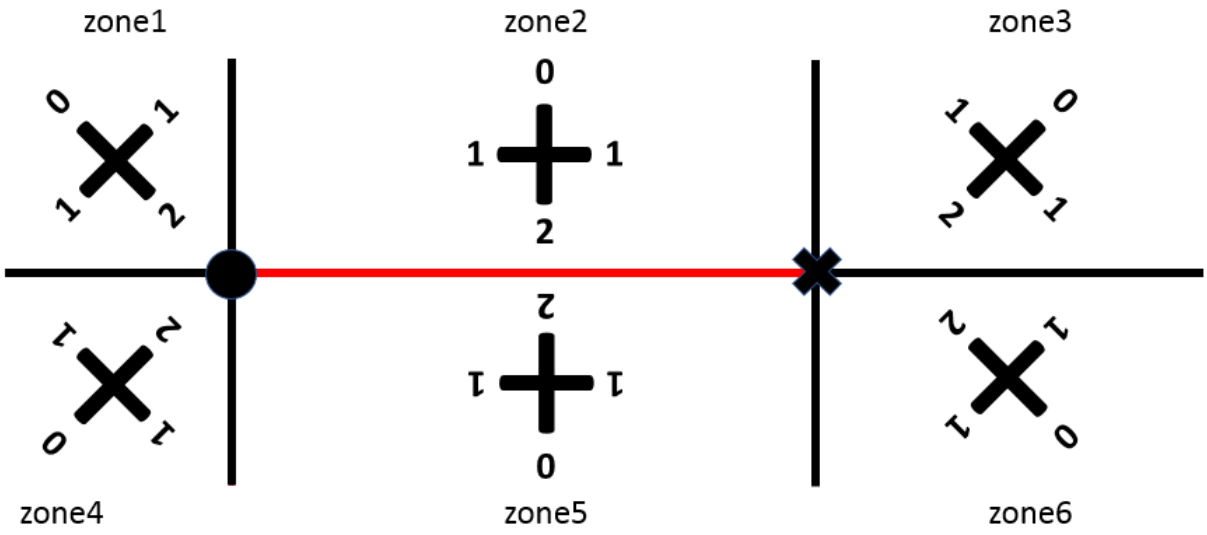

Figure 58: Relevance factor calculation.

The red line symbolizes the line of interest. The black circle is the selected ship and the small black 'X' is the destination planet.

The main idea in Figure 58 is to illustrate the relevance weighting depending on which zone the other ship is located. Basically, if the ship is headed towards the line of interest, its zone value contribution is multiplied by two. If the ship is headed away from the line of interest, its zone value contribution is multiplied by zero. Every area space in-between these two points is interpolated on a continuous scale. For instance, if a ship is in the center of the "+" in zone 2 and is headed in a 45-degree angle (where straight towards the line of interest, or where the "2" is printed, is 0 degrees), the relevance factor will be a 1.5 for that ship.

This part of the algorithm exists because a player will most likely dismiss the effect of another ship's trajectory if it is headed away from the line of interest. Another ship

heading away from the line of interest mainly is relevant only if the player wants to collect a bonus situated off-course from the line of interest. However, players primarily attempt to send the selected ship towards the destination planet, so it is safe to assume the relevance factor is a beneficial algorithm item.

Appendix G: Interpolation Method

Each initial trajectory is resampled to achieve a specific number of points per trajectory while maintaining the original trajectory shape. The interpolation method evenly spaces out the twenty-five trajectory points across the X-axis. When sampling trajectories during the experiment, the mouse speed was captured. If the player held the mouse down at the selected ship, and then quickly swiped to the destination planet, the majority of the sampled points would be cluttered around the selected ship with a couple points between the clutter and the destination planet. By resampling evenly across the X-axis, the mouse speed characteristic is lost. This is an attempt to produce less noisy trajectories by eliminating a seemingly unnecessary characteristic which causes extra difficulty. By evenly spreading out the trajectory points, the shape of the trajectory is saved while also becoming more robust with a fewer number of points. If the mouse speed was maintained in the trajectory, it would take more points to capture the original trajectory.

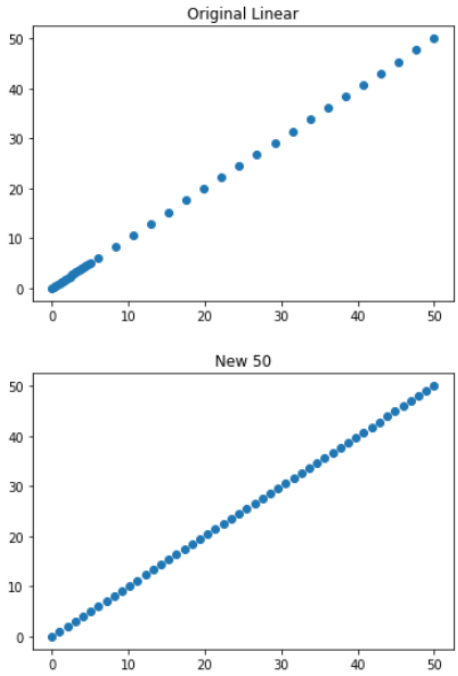

Figure 59: Interpolation example.

Figure 59 shows an example of the interpolation method. The top image represents a trajectory drawn by a user. The bottom image is the transformed, interpolated trajectory. The interpolation function basically evenly spreads out the trajectory points to eliminate the speed effect during the drawing of a trajectory.

Within Dual-Process Model using multivariate normal probability density functions (DPM-MN), the interpolation gives an inherent advantage since the standardization of points and the elimination of the speed effect causes less types of trajectories to exist. As long as the trajectory follows the same path, two trajectories differentiated by speed will be classified as the same trajectory. DPM-MN performance is influenced by the number of classes present. Outside of DPM-MN, the interpolation's effect is dependent on the algorithm used. It could be the case that interpolating the trajectory creates more error. The spreading out of a trajectory's individual points may

cause each individual point to be further away from the true value than if the trajectory was left untouched. Other experiments using the same trajectory dataset exploit other preprocessing methods such as eliminating the most eccentric trajectories altogether. The results of this experiment are still comparable to other experiments in the *Space Navigator* problem domain because the same straight-line predictor baseline is used. Any extreme differences in preprocessing can be inferred through the straight-line predictor results.

Table 13: Generic straight-line results (lower is better).

Algorithm	***Mean (ACD) ± SD***	***Confidence Interval (99%)***
DPM-MN (generic straight-line)	0.174 ± 0.175	(0.169,0.180)
Bindewald, et al., 2015 (generic straight-line)	0.232	(0.223,0.234)

Table 13 shows the difference between two *Space Navigator* results when using a naïve straight-line predictor. DPM-MN most likely has an advantage since most lines drawn are straight lines from the source to the destination. Also, the DPM-MN generic test used the final level data. This level is more difficult than the rest and players probably became overwhelmed by the hectic level. As a result, most people used the most efficient line: a straight-line. With a straight-line, the source is at least heading to the destination, and no strategy needs to be involved. If time is spent thinking, ships will probably collide, and points will be lost. The two experiments can still be compared with each other, and the two experiments can also compare within the experiment through baseline tests. DPM-MN

still consistently outperformed both the straight-line and medoid baselines which also used the interpolated data. Furthermore, DPM-MN outperformed the LSTM experiment (Appendix L) as well.

The proclaimed reason for keeping the speed effect in the trajectory dataset is the idea that it better captures a human-like response. This false assumption is the motivation for using an interpolation method. The *Space Navigator* player imitation goal is split into two sub-goals: the trajectory path and the trajectory drawing. The experimental data collection method allows the former sub-goal to be explored, but not the latter. A multitude of problems disconnects the data collection with the pragmatic application. First, a user could hold their finger down on the selected ship before actually drawing the desired trajectory. The collected data would show an intentional speedup between the source and destination. However, the true intention of the user is to draw a trajectory with uniform speed. Second, the data collection occurs when the user lifts their finger. This process tacitly implies an instantaneous decision and drawing of the trajectory. In reality, the players make their trajectory choices in a temporal space. Finally, different trajectory sampling speeds will give different trajectory speed characteristics. Even the trajectories with the speed effect do not properly characterize the changes of speed during the trajectory drawing. Eliminating the trajectory speed effect allows the experiment to focus on the trajectory path. The way the computerized agent draws the determined path is important, but it is a separate experimental question altogether; it is not concretely determined through this experiment. The spatial and temporal trajectory aspects should be separated.

Appendix H: Trajectory Euclidean Average Algorithm Computed

Algorithm 2 Average error calculation between a single true trajectory and a single predicted trajectory

input: T_p = predicted trajectory made up of points $T_{p_1}, T_{p_2}, \ldots, T_{p_{n-1}}, T_{p_n}$;
T_a = actual trajectory made up of points $T_{a_1}, T_{a_2}, \ldots, T_{a_{n-1}}, T_{a_n}$
initialize: q = empty array (length n)
Each point is a 2D coordinate. For example, $T_{a_1} = (X_{a_1}, Y_{a_1})$
for i in $1 \ldots n$ **do**
    $eDistance$ = Euclidean distance between T_{p_i} and T_{a_i}
    $q[i] = eDistance$
$meanDistance = \frac{\sum_{j=1}^{n} q[j]}{n}$
return $meanDistance$

Algorithm 3: Trajectory difference calculation.

The above algorithm is applied to finding the difference between one test observation and the corresponding prediction. The final values displayed in the results chapter are the average of the error when the above algorithm was applied to all the test observations and all the corresponding predictions. The final values more concretely represent the average Euclidean distance between a test trajectory point and the corresponding predicted trajectory point. The meanDistance output of Algorithm 3 is also referred to as the average coordinate distance (ACD).

Appendix I: Parameters Assumed

The assumed DPM-MN parameters limit the number of hyperparameters that are searched. Because the parameters are assumed, a bias is introduced. This introduces bias but allows for a finer search granularity of the parameters anticipated as being more output sensitive.

1) Number of points that make up a trajectory – The trajectories are all set to twenty-five points. This allows for each trajectory to keep the original trajectory's shape, without being too computationally expensive. The trajectories also maintain even spacing on the X-axis to control the total number of dependent variables for each trajectory. See Appendix F for a discussion on this decision.
2) Number of trajectory classes – The number of trajectories chosen was twenty. When grouping the trajectories during the data processing to get the class representatives, the number of clusters (N_c) matters. If one cluster was chosen, there would only exist one usable prediction trajectory. This would stabilize the DPM-MN algorithm class-wise but would be awful for prediction since there exists no variation in predictions. On the other hand, if there was a cluster for every trajectory, the DPM-MN algorithm would not be very reliable, and the trajectory predictions would be very specific. The DPM-MN algorithm would end up getting 100% of the class predictions wrong since every class would be different. There is a clear bias-variance tradeoff occurring with the choosing of this parameter. This variable needs to be high enough to support many unique types of trajectories, but low enough to allow DPM-MN to learn the class representations. The tradeoff is further discussed later in this chapter in the perspective of irreducible error.
3) Number of points in training and evaluation iterations – For the experiment, the number of points per batch depended on the total amount of data per hyperparameter iteration. A 0.75/0.25 split was chosen for the training and validation batches. During the training/validation stage, three batches of data would be used for training and then a batch would be used for validation. This cycle would continue until the appropriated data was exhausted.
4) Revocation Amount from System 2 – The revocation amount is positively correlated with the size threshold hyperparameter and negatively correlated with the W_s parameter. When a System 2 concept is erased, a portion of the underlying points (in the sublayer of the concept) that agglomerated to make that concept are revoked back into System 1. As discussed before, this is an attempt to functionalize keeping remnants of past experiences when they become outdated. The size threshold positive correlation allows for a proportional number of underlying points to be revoked back into System 1. The negative correlation to W_s disincentivized new concepts from being created solely through many System 2 concept

revocations happening in a short time span. To reduce thrashing between the systems, only a small subset of the points should return to System 1. Imagine if one-hundred percent of the points returned to System 1. They would turn around and end right back into System 2 since there would be a large enough amount to gain access into System 2 as a concept again. However, there should exist a disparity between concepts with a large sublayer of agglomerated points, and concepts with fewer points in the sublayer to stay consistent with the size of the System 2 concept.

Appendix J: Trajectory Representations

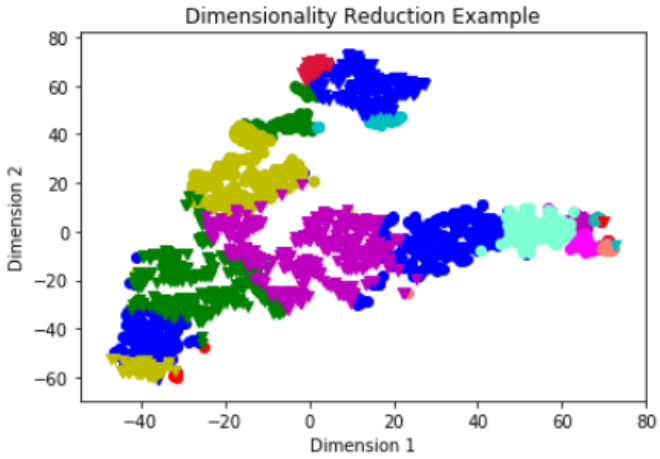

Figure 60: t-SNE trajectory class visualization.

Each trajectory exists in a 50-dimensional space (X1, Y1, X2, Y2, …, X25, Y25). The Figure 60 visualization is a clustered dimensionality reduction of the trajectory space. A k-means clustering algorithm, where ‘k’ equals twenty, is used to find the trajectory clusters in the 50-dimensional space. Afterwards, each trajectory is dimensionally reduced to a 2-dimensional space through t-SNE (Van Der Maaten, et al., 2008). The trajectory classes are then displayed using different colors and shapes. Overall, the clustering and dimensionality reduction seems consistent. There are twenty classes in Figure 60, but a few unequally represented classes contain most of the trajectory data values.

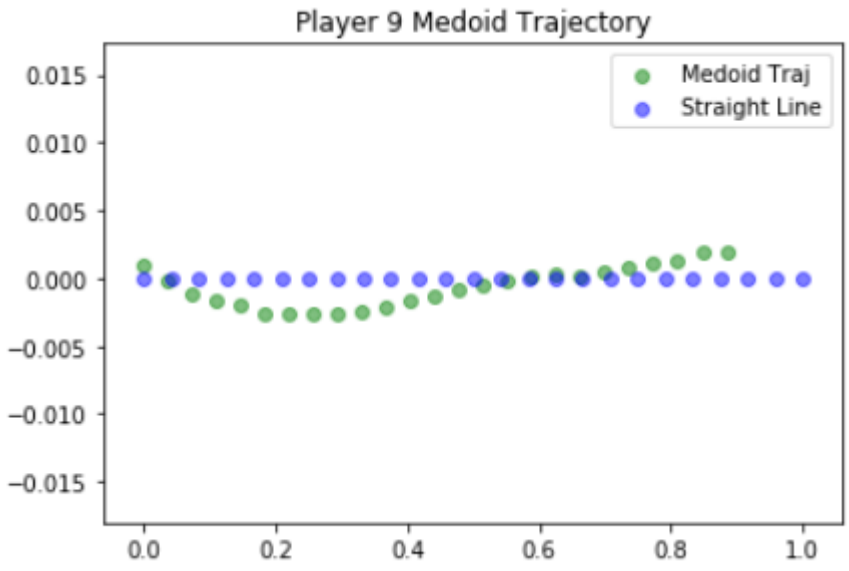

Figure 61: First example of characteristic medoid trajectory.

Figure 61 is the medoid trajectory representation from player 9 where only trajectory class exists. In other words, it is the medoid of all of player 9 trajectories. This medoid trajectory is an example of the medoid trajectory used in finding the medoid prediction difference. Player 9, along with two other players, are prominent examples of recognizing playstyle trends through visualizing the medoid trajectory. Player 9's medoid is the most common type of trajectory drawn by the thirty-five players. The usual trajectory is a simple straight line from the selected ship to the destination planet. It is crucial to notice the scaling of the y-axis and the x-axis. Even though the medoid trajectory looks somewhat curvy, the y-scaling illuminates the truth. If seen on a computer screen during a playthrough, this medoid would be perceived as a straight line.

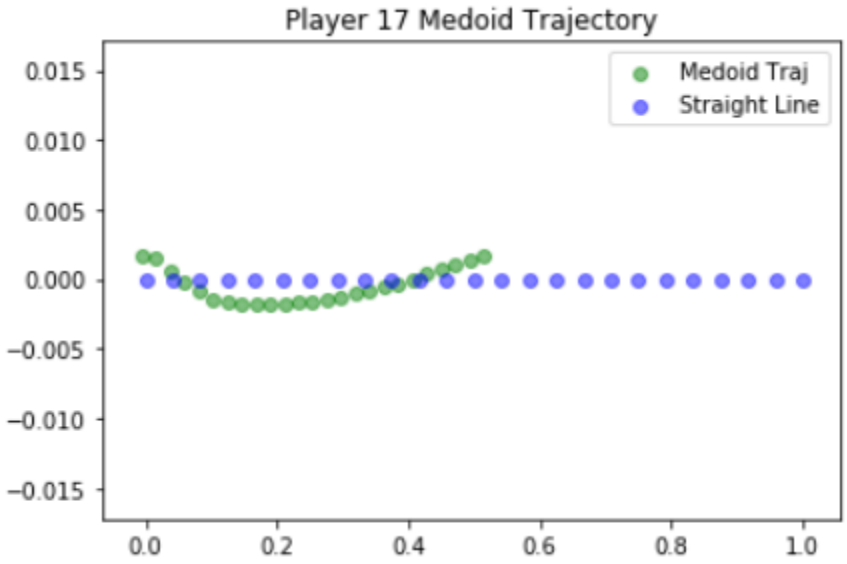

Figure 62: Second example of characteristic medoid trajectory.

Player 17's medoid trajectory in Figure 62 shows a routine of drawing short trajectories. Because the selected ship continues to follow the trajectory on a straight path after the end is reached, this user probably felt it was unnecessary to connect the selected ship with the destination planet. Even though the two are not connected with the drawn

trajectory, the selected ship will continue a straight path towards to destination planet as though a straight line was drawn between the selected ship and the destination planet.

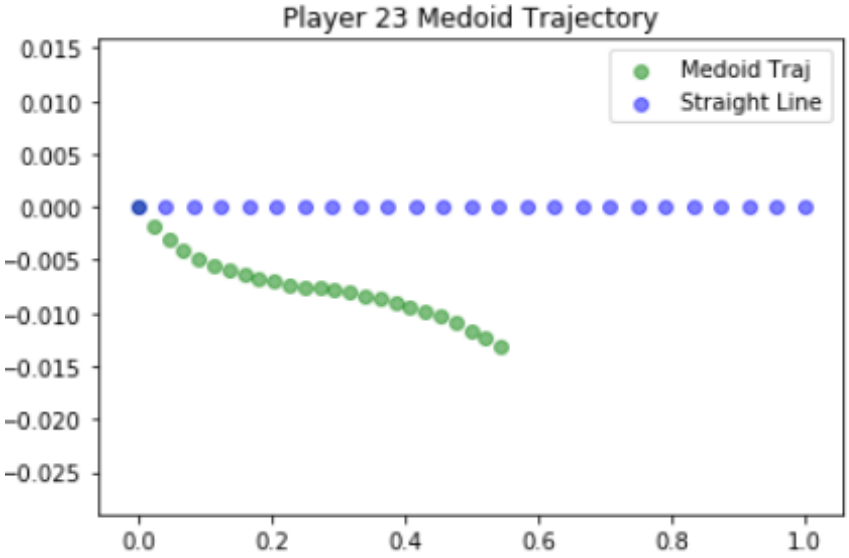

Figure 63: Third example of characteristic medoid trajectory.

Figure 63 illustrates player 23 as someone that more often pursues points other than matching the selected ship with its designated destination planet. The medoid trajectory sends the selected ship on a path away from the destination planet. It can only be assumed that the player was attempting to achieve a different strategy.

Appendix K: Data Distribution Non-Normality

The following graphs are mostly representative of all the average coordinate distance (ACD) tests as far as the identity of the results distribution.

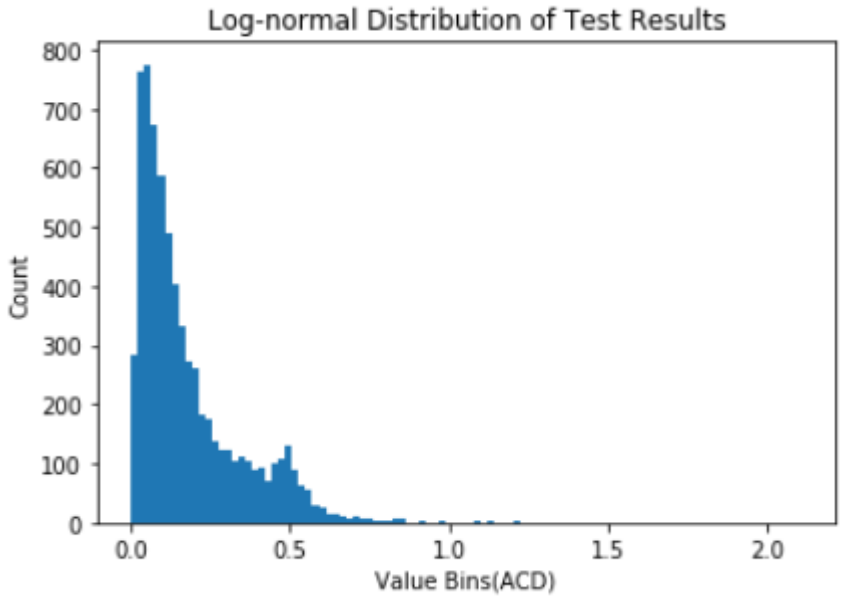

Figure 64: Log-normal distribution of test.

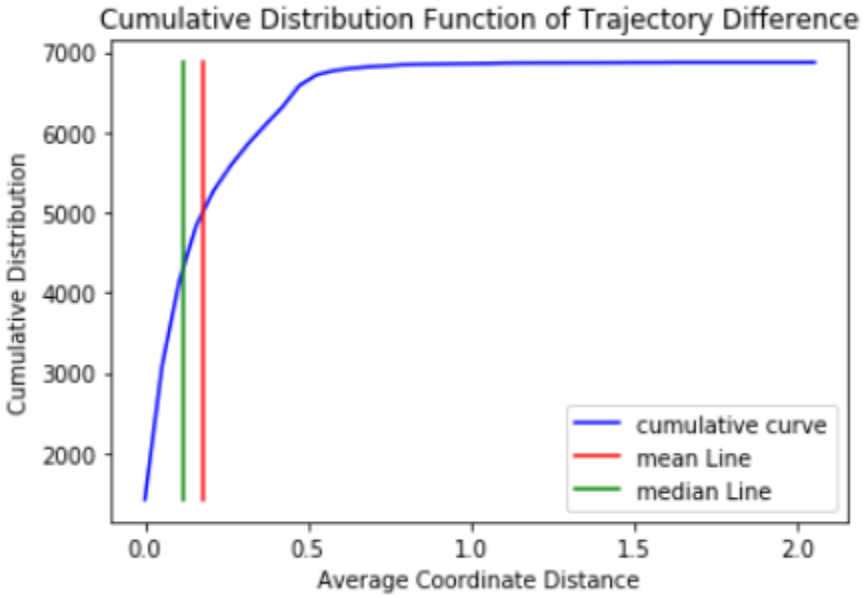

Figure 65: Cumulative distribution function.

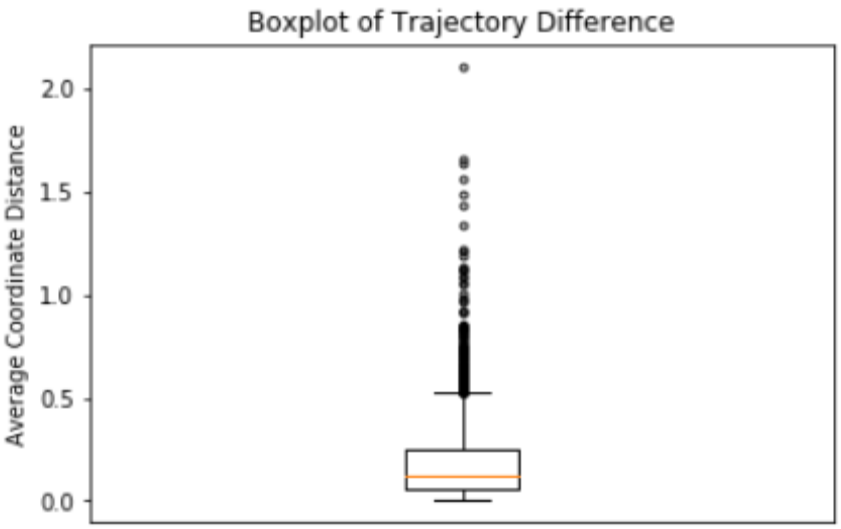

Figure 66: Boxplot of trajectory difference.

Appendix L: Grimm LSTM Machine Learning Project

Abstract:

Human-robot teaming is becoming more prevalent in the Air Force and in society. To enhance the trust and effectiveness in the team environment, it is necessary to develop algorithms enabling the robot to predict and imitate human behavior. The game environment *Space Navigator* [1] provides a state-response scenario for data collection. Deep learning is proposed and tested as a solution for human-like playing. The deep learning algorithm's objective is to create a trajectory in response to a given *Space Navigator* state instance similar to what a human would have created. When the deep learning architecture learned from the data gathered from multiple players, it did not perform better than the baseline simple response. In this case, a perfect performance would be making the exact same trajectory the human player made in the same scenario.

Introduction:

The research domain will be a video game called *Space Navigator*. Users must guide spaceships towards designated planets while avoiding obstacles and collecting bonuses. The *Space Navigator* source code is accessible which allows for detailed data gathering. Specific details such as the exact location and number of objects are used to capture features that describe the *Space Navigator* state at any time.

The specific goal of this research is to create an artificial intelligence (AI) which plays the game like a human would play it. In the game, players draw trajectories to control the movement of the spaceships. The targeted behavior for the AI to imitate is the drawing of trajectories. Given the game state, the AI should draw a trajectory in a human-like way instead of exclusively in an optimal manner. The performance measure is the

distance between the AI response and the human response to a perceived game state. It is predicted that an AI model can outperform the baseline model.

The expanded purpose is to develop AI that act similar to humans and that predict a person's actions. The larger domain is a state-response situation which could possibly occur in a variety of settings. Human-robot teams are becoming increasingly important in the Air Force and society. These teams can better perform if the robotic teammate excels in understanding the human's thoughts.

Deep learning artificial neural networks are the chosen tool for this attempt at functionalizing a human-like AI. Specifically, a long short-term memory (LSTM) architecture will be the primary applied component and supervised training will be used to learn the regression problem. LSTMs are often used for sequential outputs. The generated trajectories in *Space Navigator* are made up of many points which follow one another. As a result, the inherent ordering of points that make up the trajectory allows the location of a previous point to be exploited for guessing the location of the current point. The estimation error emerges from the difference in distance between the predicted trajectory and the true trajectory.

A one-to-many recurrent neural network is frequently used as one of the primary topologies for sequential data along with many-to-many and one-to-one topologies. An image captioning LSTM architecture inspired the *Space Navigator* proposed solution. For image captioning, a single input (a picture) and a sequential output (a string of words) make up the input and output format. The *Space Navigator* LSTM also has a single input (the state instance) and a sequential output (multiple points that make up a trajectory).

The results show poor AI performance when the training data includes multiple players' data. However, when a single player's data is trained on, the AI performs better than the baseline straight-line trajectory competitor. This reveals an importance of the idiosyncrasies of each player. Each player's unique playstyle is not only a contributing factor but a primary element in creating an AI which acts like a human.

The related work section will provide an opportunity to learn about previous endeavors in similar machine learning tasks. Next, the data gathered from *Space Navigator* will be explained and the custom LSTM architecture will be examined. The results will show what overarching themes were learned from the experiment. The conclusion and future work section will describe the mixed findings and give suggestions for future possible directions.

Related Work:

Bindewald et al. first worked with imitating individual players in the space navigator domain [1]. They gathered the trajectory data through initial experimentation. Bindewald utilized a cluster-based algorithm to retrieve responses given a specific state. A generic player model was trained offline with the majority of the data, and then specific players achieved their individual player models through online training. The generic player model acted as a base model, and the online training updated the generic player model to reflect the specific player's idiosyncrasies. The individual player models achieved a mean Average Coordinate Distance (ACD) of 0.2036 which improved on the straight-line generator mean ACD of 0.2319. Specified units are not attached since the comparisons are calculated after the state space and features are normalized. However,

the original state space units depended on the screen resolution. It is best to think of the state space domain as a generic grid.

From chapter 10 of the Deep Learning textbook [2], recurrent neural networks are explained as useful in sequential problems. The main idea is to not only use individual inputs for each timestep but also to use the previous output for current decisions. An LSTM architecture is a progression of the recurrent neural network (RNN) architecture. The LSTM format modifies the RNN model to enhance the possible context influence. LSTMs are traditionally used instead of RNNs when a larger influencing context window is desired. Dense layer gates are added to the cell to influence a memory bus which transmits throughout each timestep. This allows the neural network to remember important features as time passes instead of only utilizing immediately previous points.

Karpathy et al. developed a method for generating captions from images [3]. This problem represents a one-to-many RNN architecture; a single input and a sequential output both exist.

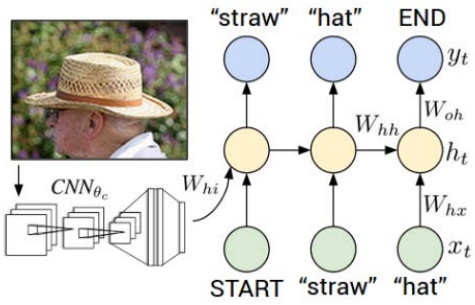

Image 1: Image caption generator architecture

The basic idea of Karpathy's architecture [4] is to take an image and condense it down into a single vector (Image 1). This vector (along with the START tag) make up the input for the first RNN timestep. When the state vector is added to the START tag, the

state vector will be produced. The training phase takes advantage of teacher forcing. The input of each timestep is the output from the previous timestep. Thus, the training phase assumes the correct previous output at each timestep to optimize learning. During testing, each timestep input uses the predicted output from the previous timestep until the END token is reached.

Approach/Methodology:

The state-response data comes from a *Space Navigator* experiment conducted in 2015 by Major Bindewald. The study participants simply played *Space Navigator*. Whenever they drew a path between a planet and the intended goal, various game state features and the points which make up the trajectory were recorded.

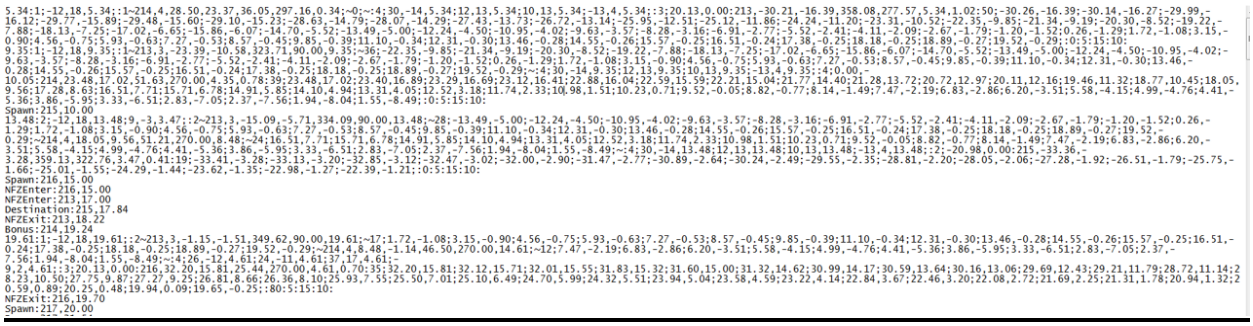

Image 2: Raw data input example

After capturing the data from an experiment, Image 2 is what the raw data looks like. 35 players completed 16 levels each and generated a total of 68,538 different trajectory instances. The highlighted part is a single example of a drawn trajectory given the state instance.

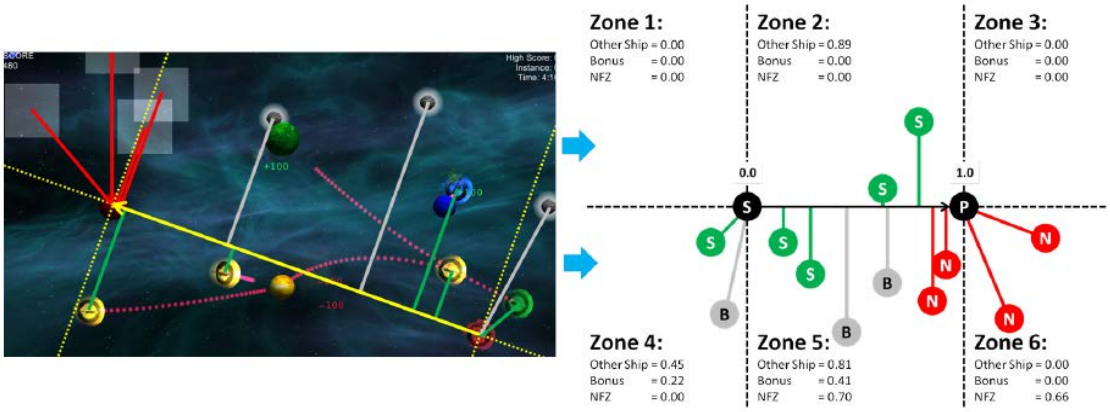

Image 3: A Space Navigator screen capture highlighting important game objects

Algorithm 1 State-space feature vector creation algorithm.

input: L = straight-line trajectory from spaceship to destination planet.
initialize: $\eta \in [0.0 \ldots 1.0)$ = weighting variable; s = empty array (length 19); $zoneCount = 0$
Translate all objects equally s.t. the selected spaceship is located at the origin.
Rotate all objects in state-space s.t. L lies along the X-axis.
Scale state-space s.t. L lies along the line segment from $(0, 0)$ to $(1, 0)$.
for each object type $v \in (OtherShip, Bonus, NFZ)$ **do**
    $relevance = 1.0$
    for each zone $z = 1 \rightarrow 6$ **do**
        $zoneCount = zoneCount + 1$
        for each object o of type v in zone z **do**
            if (type $v ==$ $OtherShip$) **then**
                $relevance$ = calculate new relevance ▷ See Appendix B
            d_o = the shortest distance of o from L
            $w_o = relevance \cdot (e^{-(\eta \cdot d_o)^2})$
            $s[zoneCount] = s[zoneCount] + w_o$
$s[19]$ = the non-transformed straight-line trajectory length
return s, normalized between $[0, 1]$

Algorithm 4: State-space feature vector creation

After the data is gathered, it needs to be wrangled to become suitable for training (Image 3). Each feature value in the data becomes normalized between zero and one.

Without normalizing the features, the straight-line trajectory length (s[19] in Algorithm 4) will overshadow the other features because of the large scale it exists on.

In addition to Algorithm 1, the Gaussian weight function is multiplied by the relevance variable for the other ship object types. This was added to the algorithm because it is important to capture which direction the other ships are going. Intuitively, if the other ships are moving away from the direct path between the selected ship and the destination planet, it poses less of a threat than other ships moving straight toward. If a non-selected ship is moving straight towards the area of interest, its Gaussian weight function is multiplied by two. If a non-selected ship is moving away from the area of interest, its Gaussian weight function is multiplied by zero. All of the values in between towards and away act on a continuous function between two and zero. For instance, a ship in zone two moving parallel to the line of interest has its Gaussian weight function multiplied by one. For more information on this additional metric, see Appendix B.

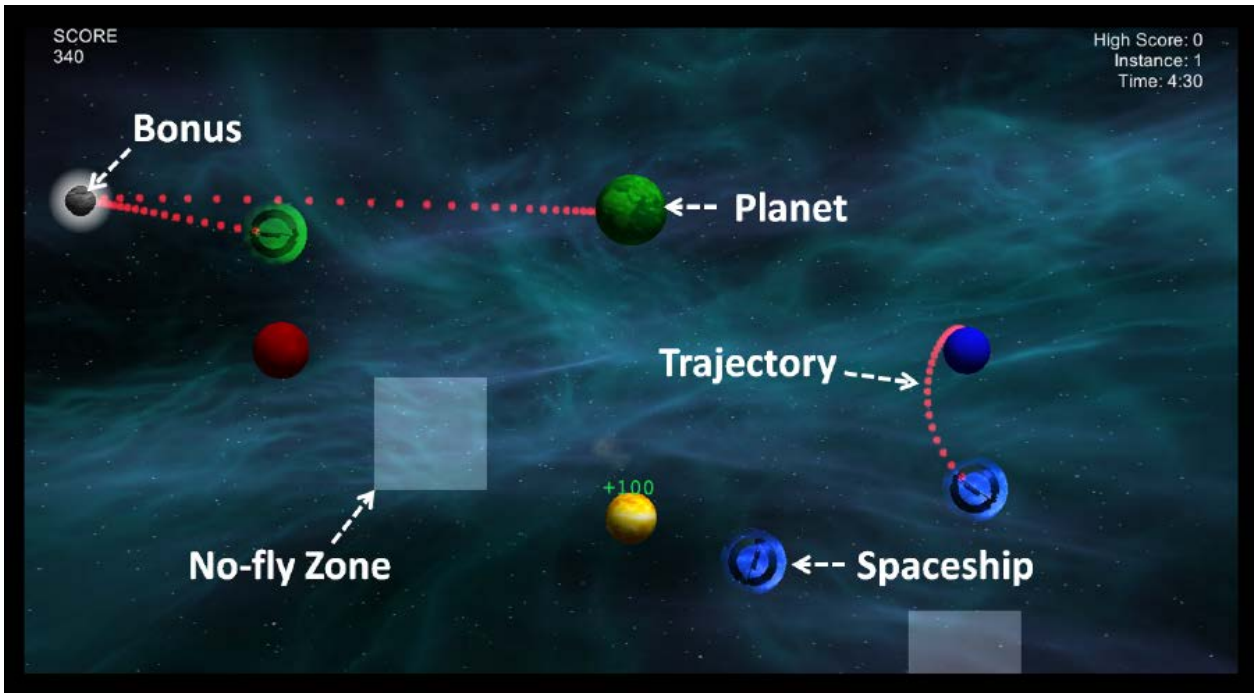

Image 4: Important parts of Space Navigator

There are 68,538 observations. Each state includes nineteen features. Algorithm 1 shows how these features are computed. Image 4 shows what each state object looks like. The bonuses give extra points, the no-fly zones take away points whenever a ship exists within it, and a lot of points are given when a spaceship reaches a planet of the same color. Each of the six zones contains three features. That accounts for eighteen of the nineteen features. The last feature is the non-transformed straight-line trajectory length. This last feature represents the straight-line distance between the spaceship and the planet.

The output variable will be a trajectory sequence. The accuracy performance measurement will be the average Euclidean distance between the trajectory points guessed by the AI and the state transformed trajectory points actually drawn by the experiment participant. See Appendix C for more detail. Part of the trajectory

normalizing process includes standardizing the number of points in each trajectory by interpolating the points. Every trajectory is interpolated to contain twenty-five points. See Appendix A for possible problems with this interpolation method.

A graphical example of the state capture and trajectory is shown below. The red squares are the no-fly zones. The green plus signs are the bonus points, the red stars are the other ships, a trajectory coming from a red star is a straight-line trajectory for that ship, the black “X” is the destination planet, the black “O” is the selected ship, the black line connecting the two is the straight-line trajectory, and the blue line is the user-drawn path.

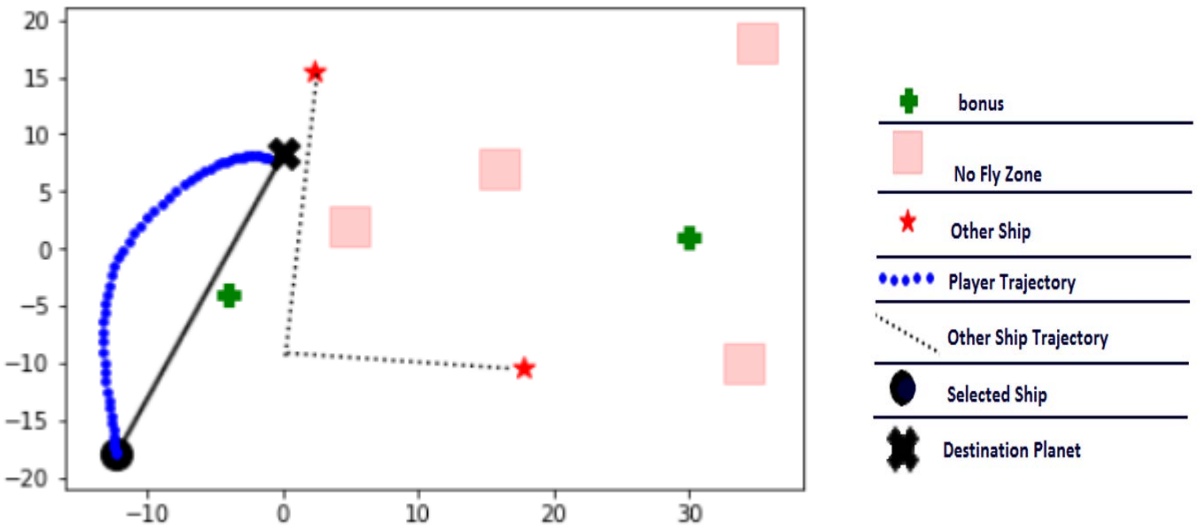

Image 5: Initial data capture and graphical representation

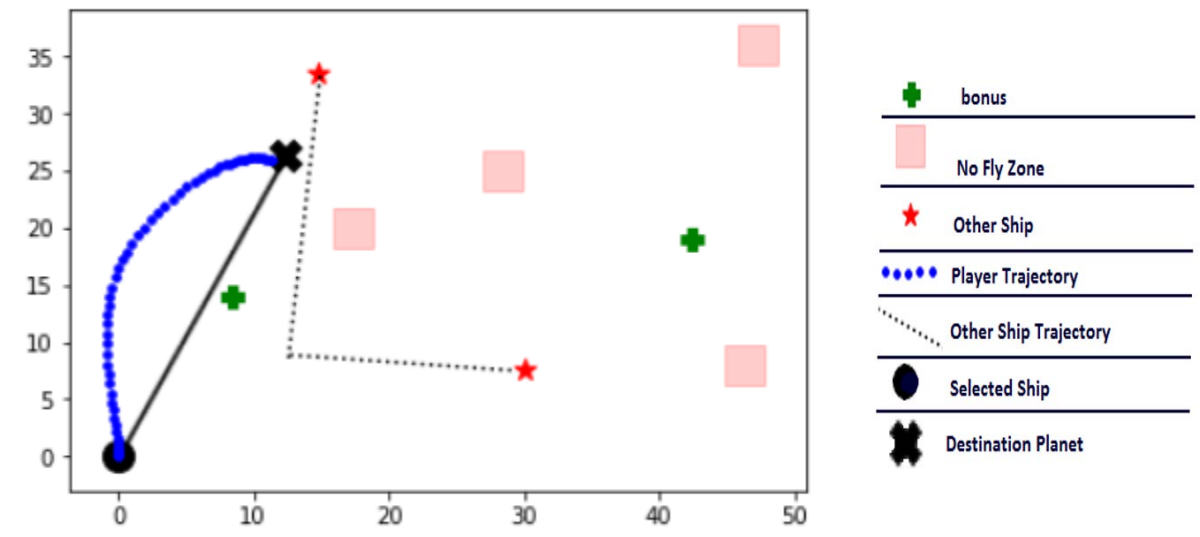

Image 6: Translation of objects

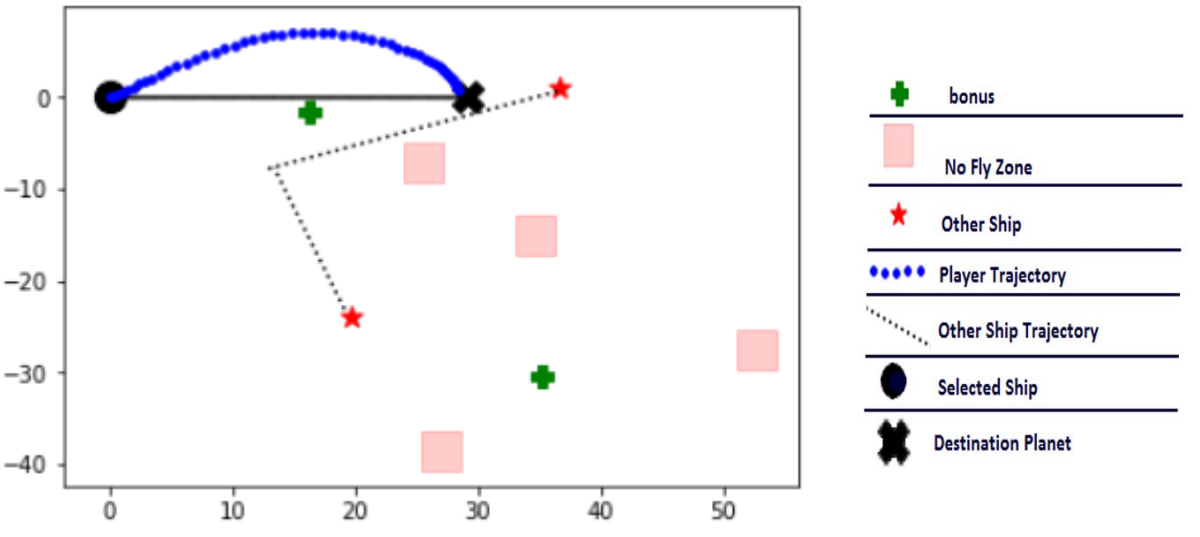

Image 7: Rotation of objects

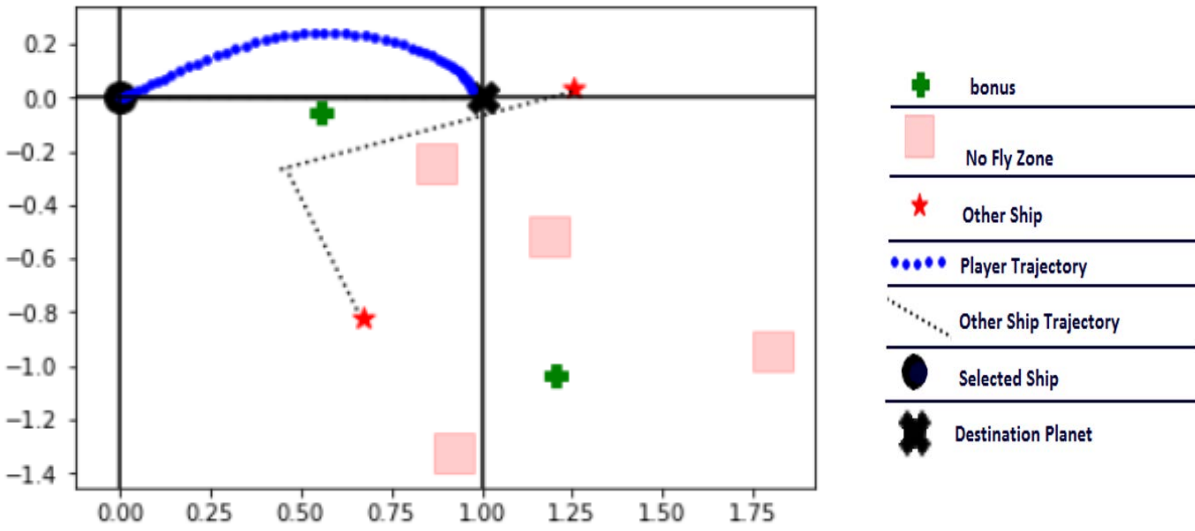

Image 8: Scaling of objects and addition of zone lines

The initial data state is first translated so the selected ship is at coordinate (0,0). Then, all objects are rotated so the selected ship and the destination planet are on the same Y-coordinate plane. Finally, everything is scaled so the distance between the selected ship and the destination planet is equal to one. This transformation is pictured with Image 6 to Image 8.

```
Player: 029
Level: 5
        os1       os2  os3  os4       os5      os6  bon1  bon2  bon3  \
4  0.856252  3.233753  0.0  0.0  0.195841  1.45491   0.0   0.0   0.0

       bon4  bon5  bon6  nfz1  nfz2  nfz3      nfz4  nfz5      nfz6     linlen
4  0.965102   0.0   0.0   0.0   0.0   0.0  0.986463   0.0  0.893812  22.540719
```

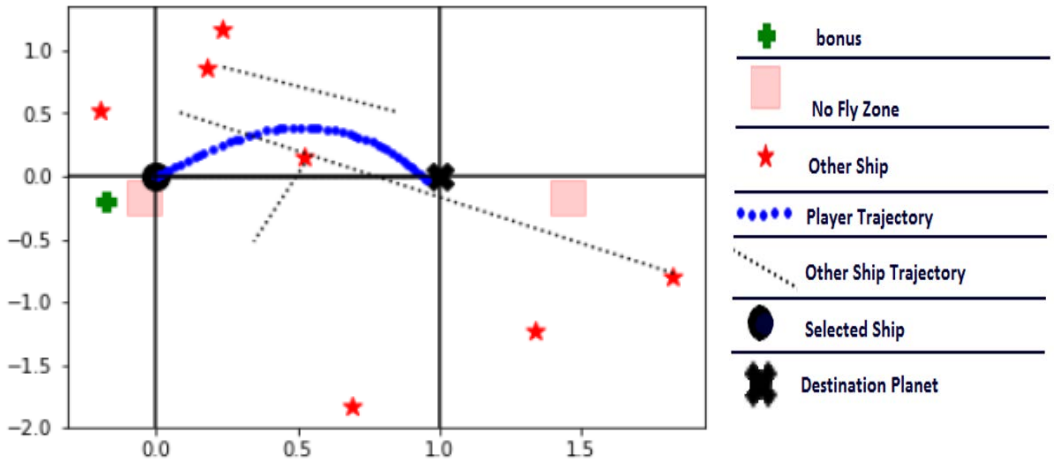

Image 9: Example of non-normalized feature values for a normalized state space

Image 9 is an example of the non-normalized values for a random state. The weighting variable was at 0.7. “os1” stands for “other ships in zone 1”. “bon” is short for “bonuses”. “nfz” is short for “no-fly zone”. Finally, “linlen” is the original straight-line length between the selected ship and destination planet. Looking at random examples along with their variables allows a quick sanity check on the algorithm to see if the values are in the general vicinity of expectations. This example from player 29 on level 5 contains a few ships in zone 2, with one of them extremely close to the line-of-interest and also heading towards it. It is expected that “os2” will have a high value and as seen above, it does. There are no bonuses in zones 1,2,3,5, or 6, so they are expected to be at a value of zero -- which they are.

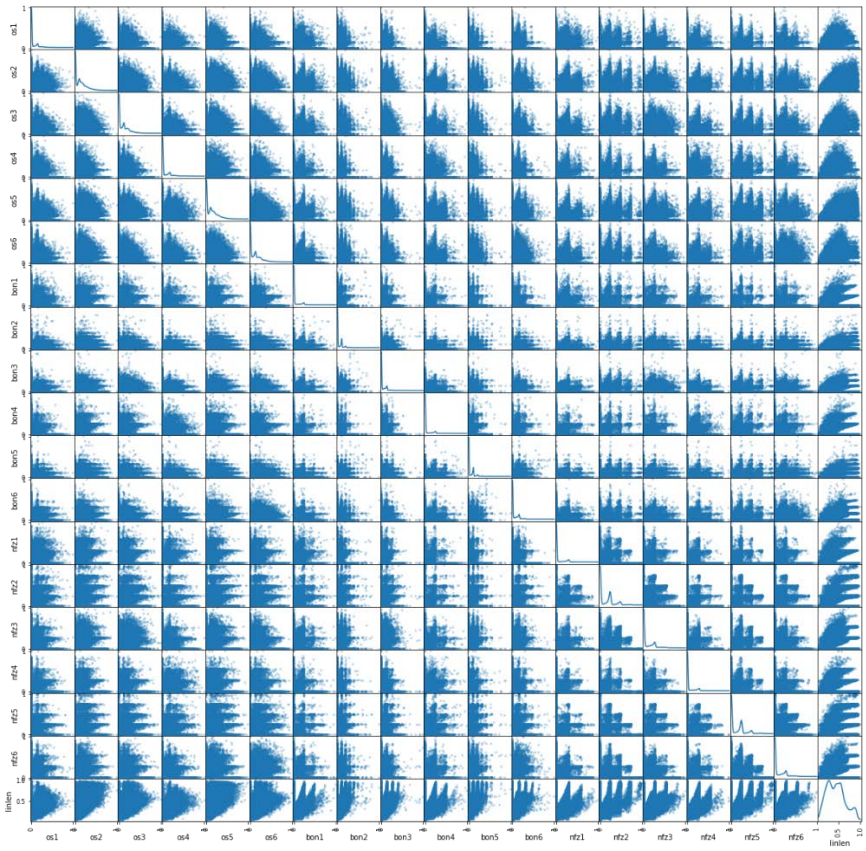

Figure 1: Scatter matrix of features

Because of the number of features, Figure 1 is best viewed on a computer so zooming in is possible. As seen above, most of the features have near zero correlation. This is expected since the features are virtually independent of each other. A bonus in zone two has no effect on the likelihood of a no-fly zone spawning in zone five. An interesting feature of the data is the sparsity of high values. It is most likely for each feature, except the straight-line length, to have a near zero value.

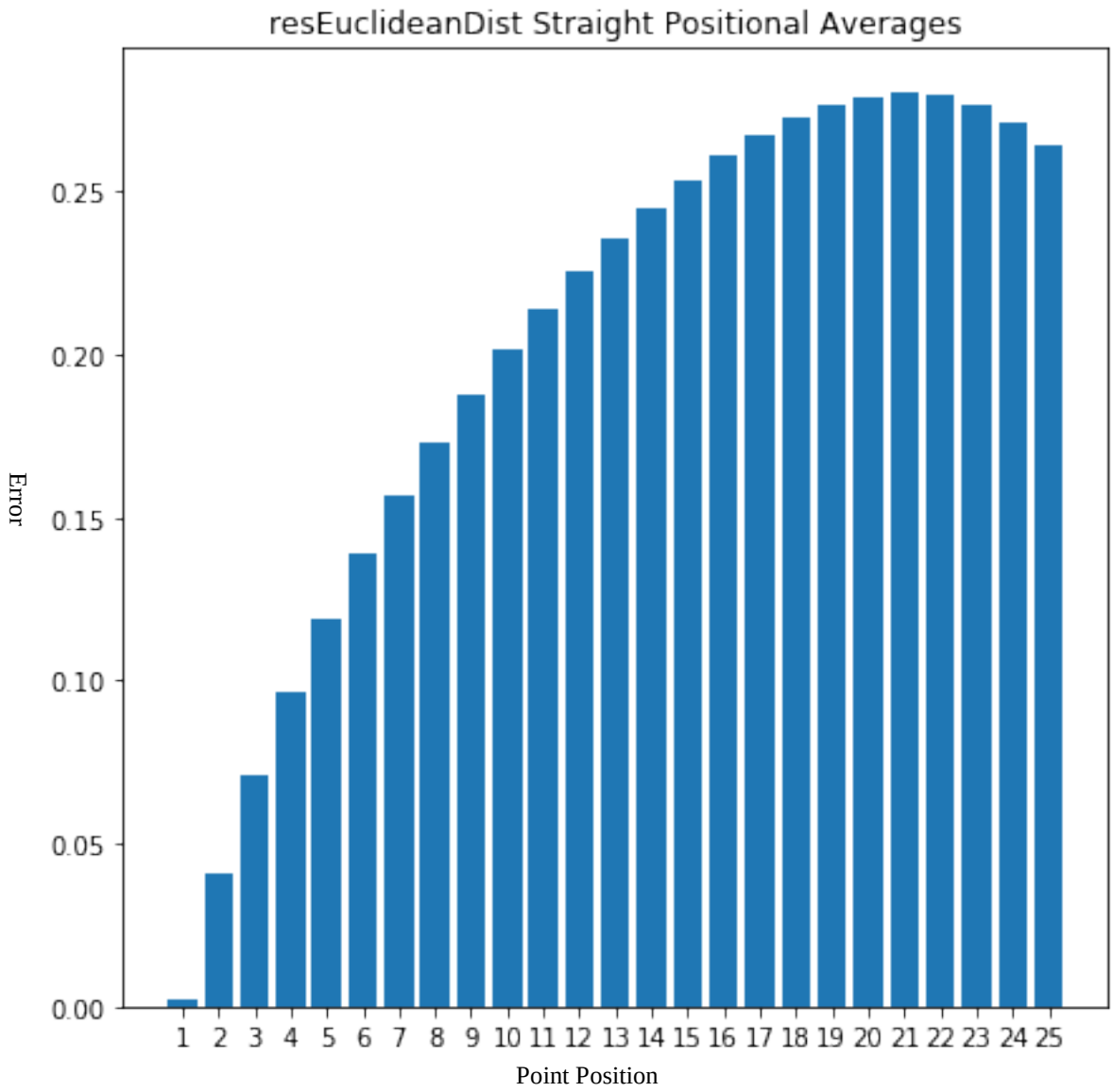

Figure 2: Data Euclidean distance

Figure 2 shows the full data trajectory average Euclidean distance from the center line. The first trajectory point is almost always completely accurate. The trajectories begin on the selected ship so that makes sense. The distance grows as the points progress until a slight dip at the very end. The error in the trajectory points compound over the space. If point one is messed up, that will also mess up point two, and so on. The end dips a little because most trajectories end at the destination planet. However, a substantial

number of trajectories are nowhere near the destination planet, so it still has a high average point distance.

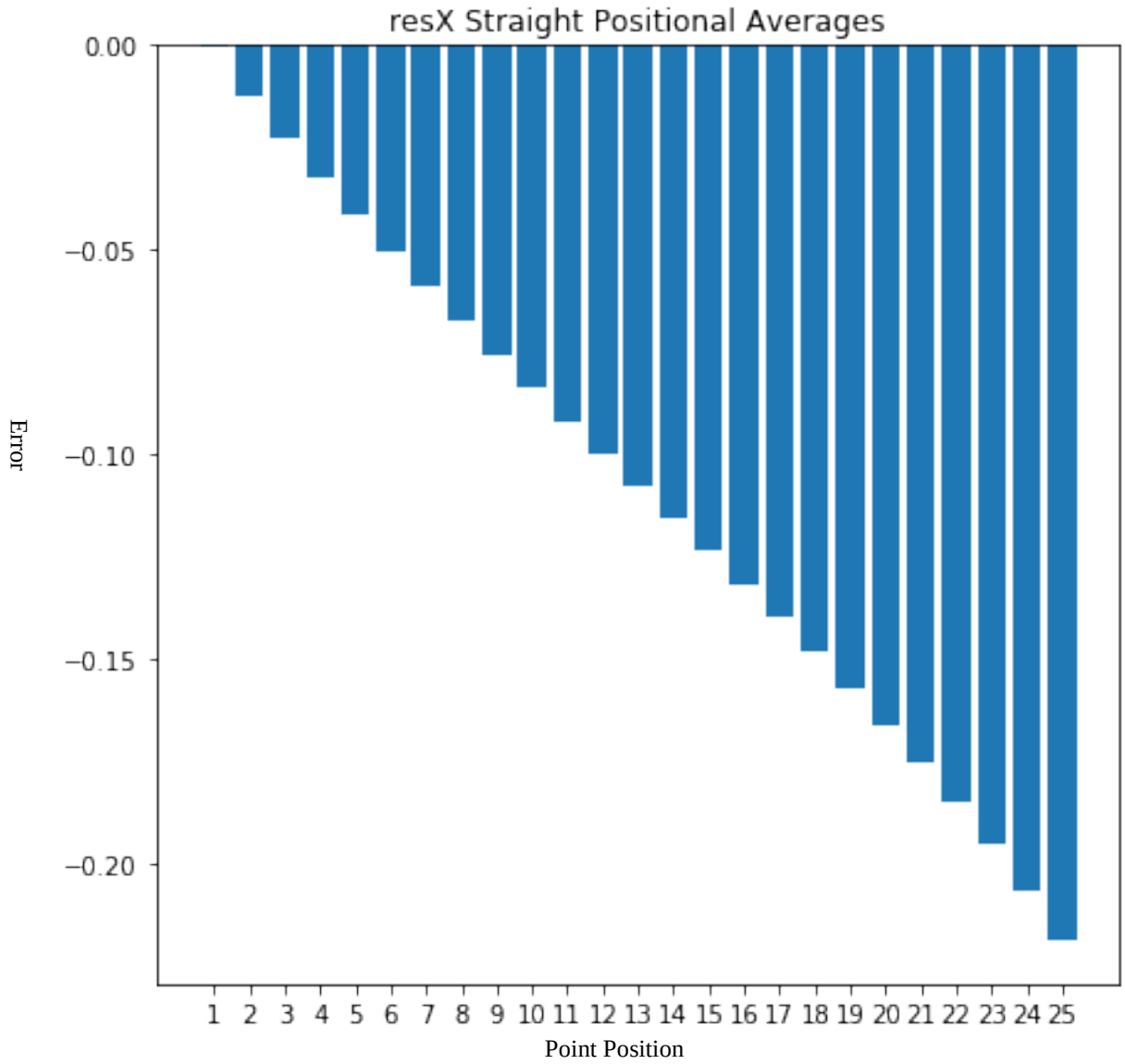

Figure 3: Data trajectory X value residual error averages

The residual error is calculated as the truth minus the prediction. In this case, the prediction is always a straight line from the selected ship to the destination planet. Because of this standard trajectory, the trajectories can be analyzed across the whole dataset. As seen in Figure 3, the true trajectories, on average, are shorter than the middle

line. The x-values for each trajectory point are not progressing across the x-axis as fast as the straight-line.

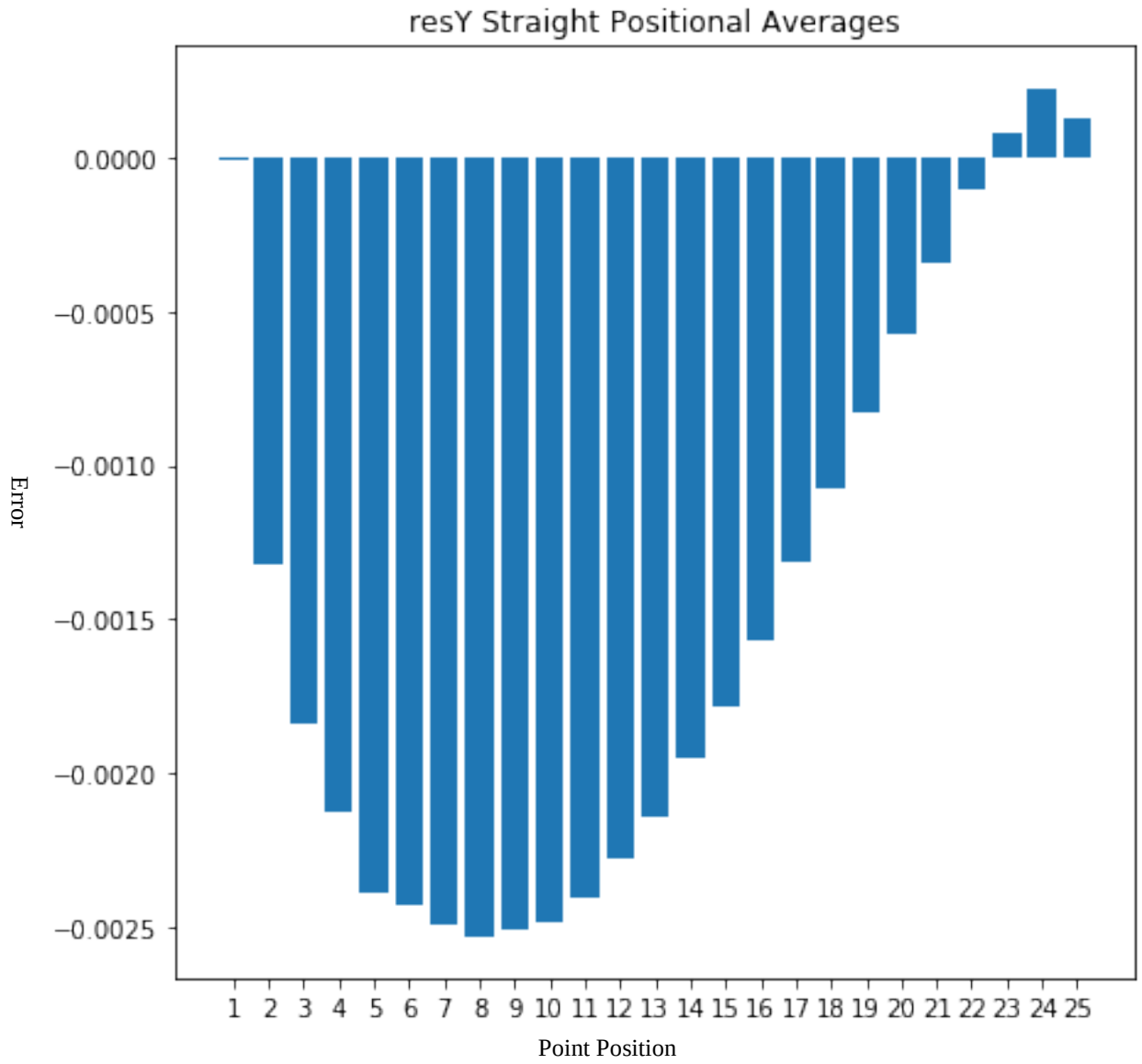

Figure 4: Data trajectory Y value residual error averages

Looking at Figure 4,the true trajectories are placed, on average and for most of the point locations, underneath the middle line. In fact, there even exists a parabola-like shape indicating a fall and rise in the trajectory.

A simple validation set is used because of computation constraints. A limited grid search is conducted on the learning rate, capacity, generalization rate, and optimizer type. Twenty training epochs with early stopping was implemented to save on time. Ten percent of the data will be set aside for validation purposes and ten percent of the data will be set aside for testing purposes. The performance results are based on the average Euclidean distance between each predicted trajectory point and the ordinally matched actual trajectory point drawn by the player. The baseline will be a prediction of all straight-line trajectories. Major Bindewald's implementation results can also be used as a baseline [5].

Hyperparameter	Options
Learning Rate	0.001, 0.01, 0.1
Capacity	32, 128
Generalization Rate	0.2, 0.5
Optimizer	RMSProp, Adam

Figure 5: Hyperparameters searched

The hyperparameters searched are shown in Figure 5. Every possible combination of the above hyperparameters was tried. The learning rate and optimizer are self-explanatory. The capacity determines how wide the dense layers and LSTM layers are while the generalization rate determines the dropout rates. Only a single validation set

was administered due to computational constraints. A k-fold cross-validation where k is greater than one would take too much time.

Hyperparameter	Best Single Model
Learning Rate	0.001
Capacity	128
Generalization Rate	0.2
Optimizer	Adam

Figure 6: Hyperparameters picked

The best hyperparameters are a low learning rate, a high capacity, a low generalization rate, and the Adam optimizer (Figure 6). See Appendix E for a detailed decomposition of the validation process.

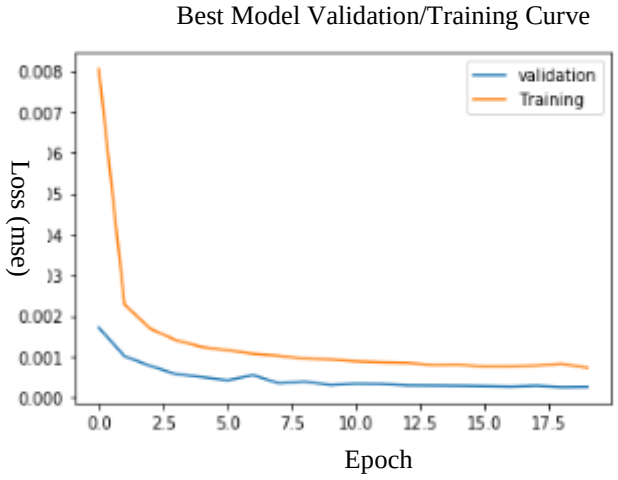

Figure 7: Best model validation and training

The model improved most in the first couple epochs (

Figure 7). After that, the improvement slightly decreased until it leveled off. An early stopping callback was used with a patience of five, so the model definitely continued improving after the easily seen initial drop-off.

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 1, 19)	0	
dense_5 (Dense)	(None, 1, 128)	2560	input_3[0][0]
dropout_4 (Dropout)	(None, 1, 128)	0	dense_5[0][0]
dense_6 (Dense)	(None, 1, 128)	16512	dropout_4[0][0]
dense_7 (Dense)	(None, 1, 2)	258	dense_6[0][0]
input_4 (InputLayer)	(None, 24, 2)	0	
concatenate_2 (Concatenate)	(None, 25, 2)	0	dense_7[0][0] input_4[0][0]
reshape_2 (Reshape)	(None, 128)	0	dense_6[0][0]
lstm_3 (LSTM)	(None, 25, 128)	67072	concatenate_2[0][0] reshape_2[0][0] reshape_2[0][0]
dropout_5 (Dropout)	(None, 25, 128)	0	lstm_3[0][0]
lstm_4 (LSTM)	(None, 25, 128)	131584	dropout_5[0][0]
dropout_6 (Dropout)	(None, 25, 128)	0	lstm_4[0][0]
time_distributed_2 (TimeDistrib	(None, 25, 2)	258	dropout_6[0][0]

Total params: 218,244
Trainable params: 218,244
Non-trainable params: 0

Image 10: Deep learning LSTM architecture

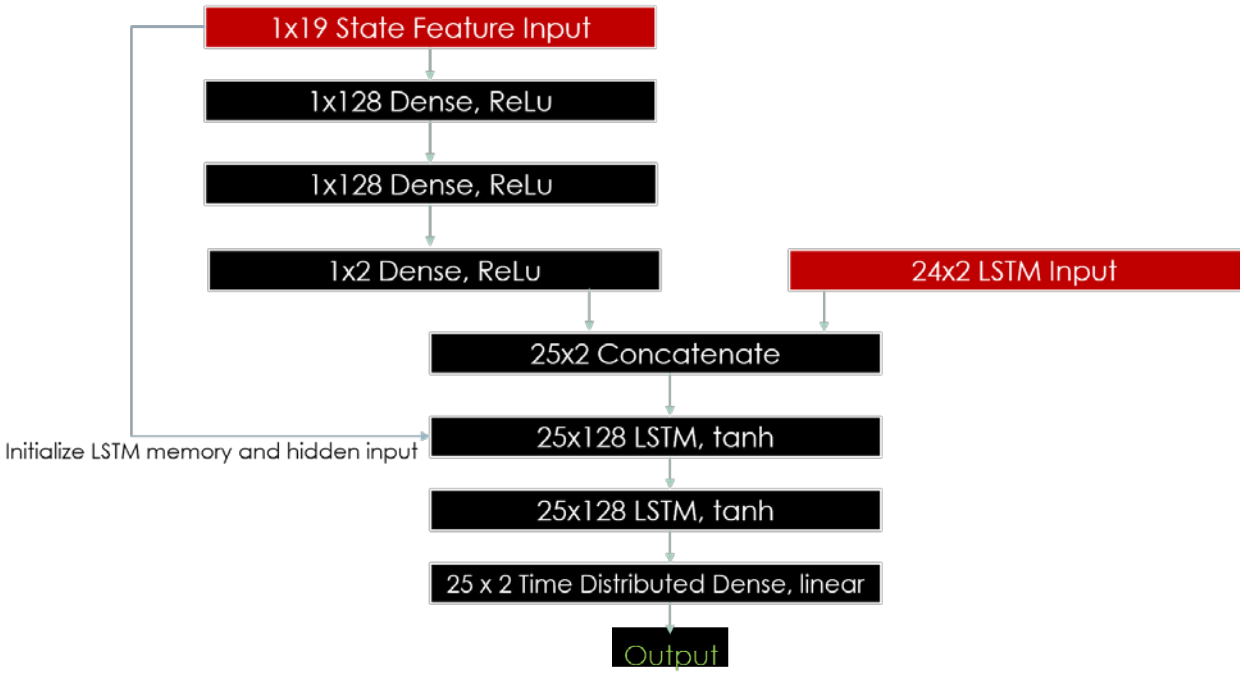

Image 11: Deep learning LSTM architecture – graph form

The target sequence for the LSTM will be the state transformed trajectory. The LSTM input will be the state space features. The LSTM is a one-to-many architecture. An approach similar to that of the image caption generator is used [3]. Due to computation capacity restrictions, the LSTM is limited in the number of parameters and layers. Although, it should not matter too much since the output is a sequence of 2D points. Most of the relative attributes of a trajectory will most likely be found at a low level.

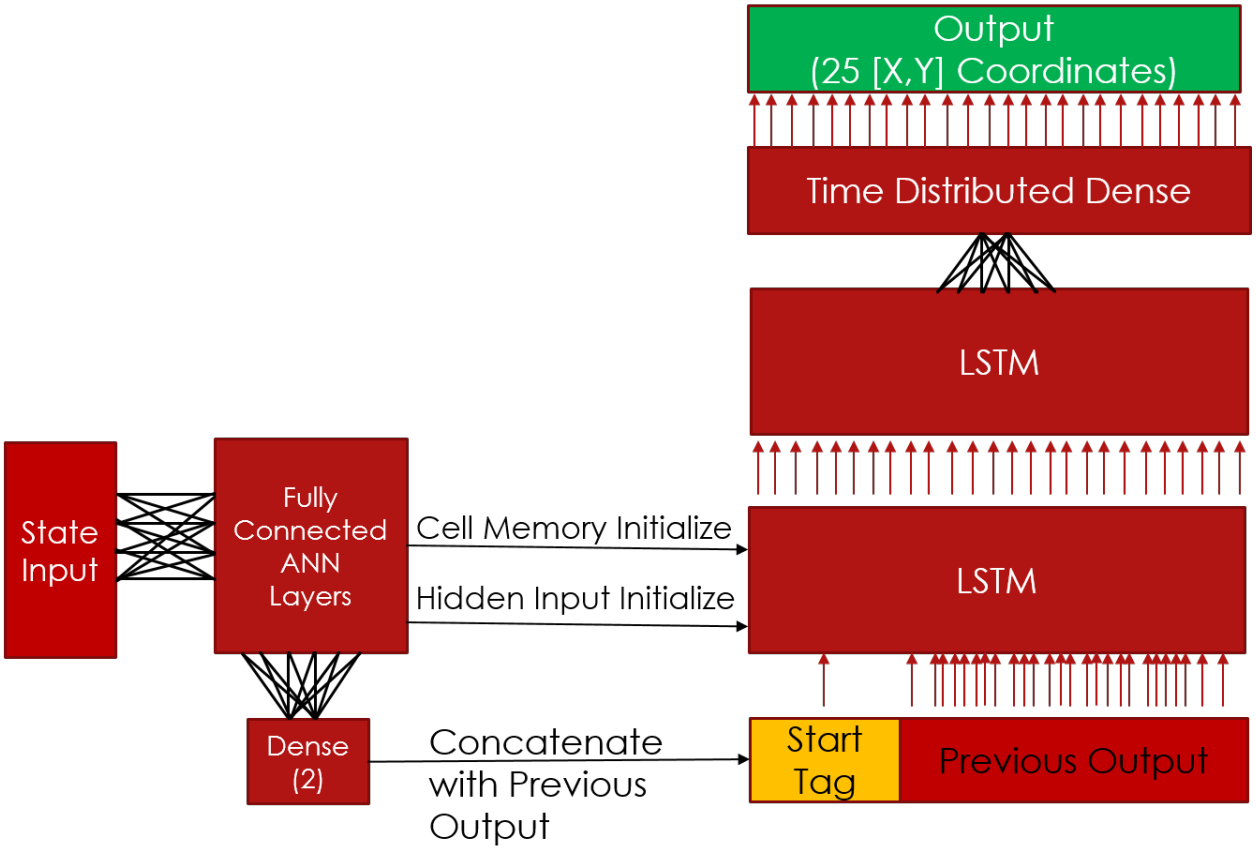

Image 12: Deep learning LSTM architecture – higher level visual form

Image 10, Image 11, and Image 12 are all visual representations of the LSTM architecture producing trajectories for *Space Navigator*. For this problem, a start tag is not readily apparent. The null-valued start tag cannot be (0,0) since that coordinate maps to an actual data point that may be predicted. In this case, an additional assumption would need to be taken if the start tag was (0,0). That would be assuming every trajectory's start point as (0,0). That is a close guess, and all the trajectories start very close to (0,0), but it would not be exact to assume that. As a result, the start tag is learned using an additional dense layer. It is of length two because it needs to be concatenable with the previous output which is twenty-four two-dimensional (X, Y) coordinates.

The time distributed layer is a dense layer that independently acts on each timestep output. In this situation, that means each second layer LSTM output goes through the same dense layer before becoming an official trajectory point output.

To take advantage of the state input, it goes through a couple of dense layers before becoming the first layer LSTM's initial hidden state input and initial cell memory. A similar initialization method is exploited in the image caption generator architecture [4].

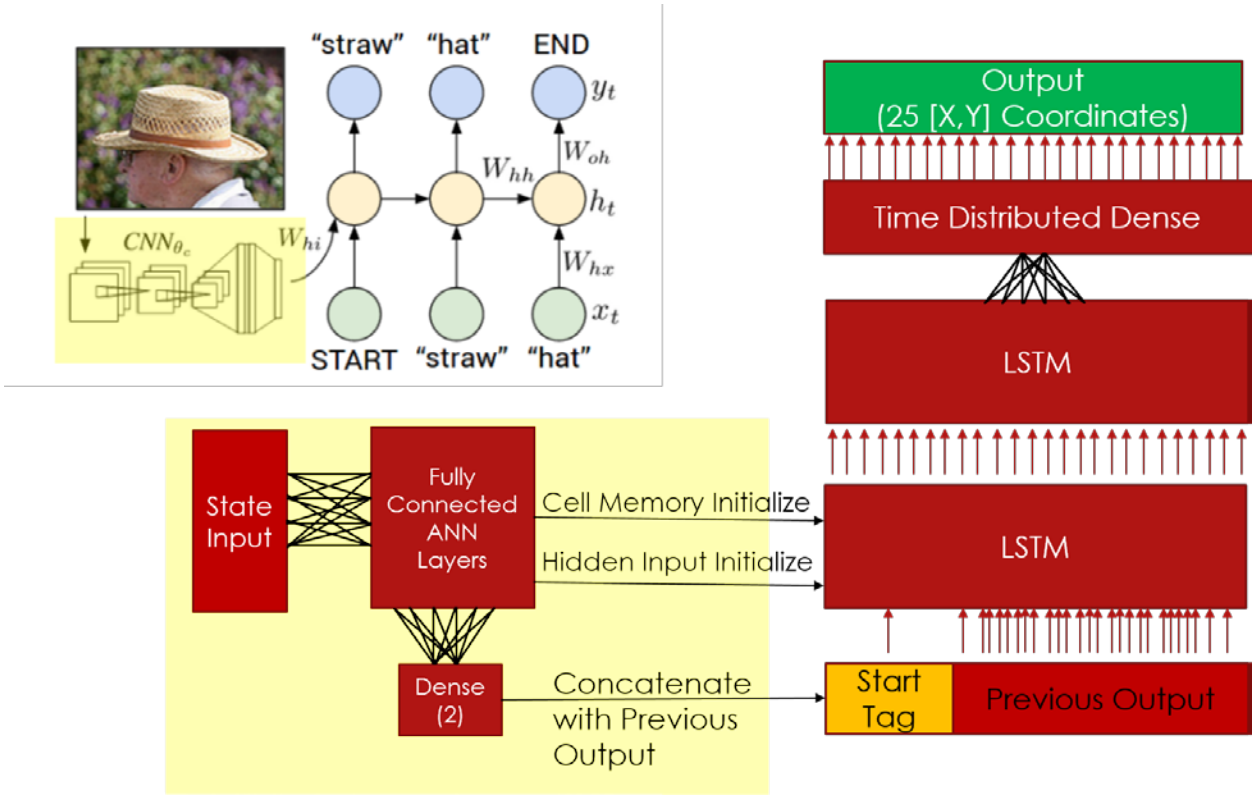

Image 13: Inspirational architecture analogous functionality -- input

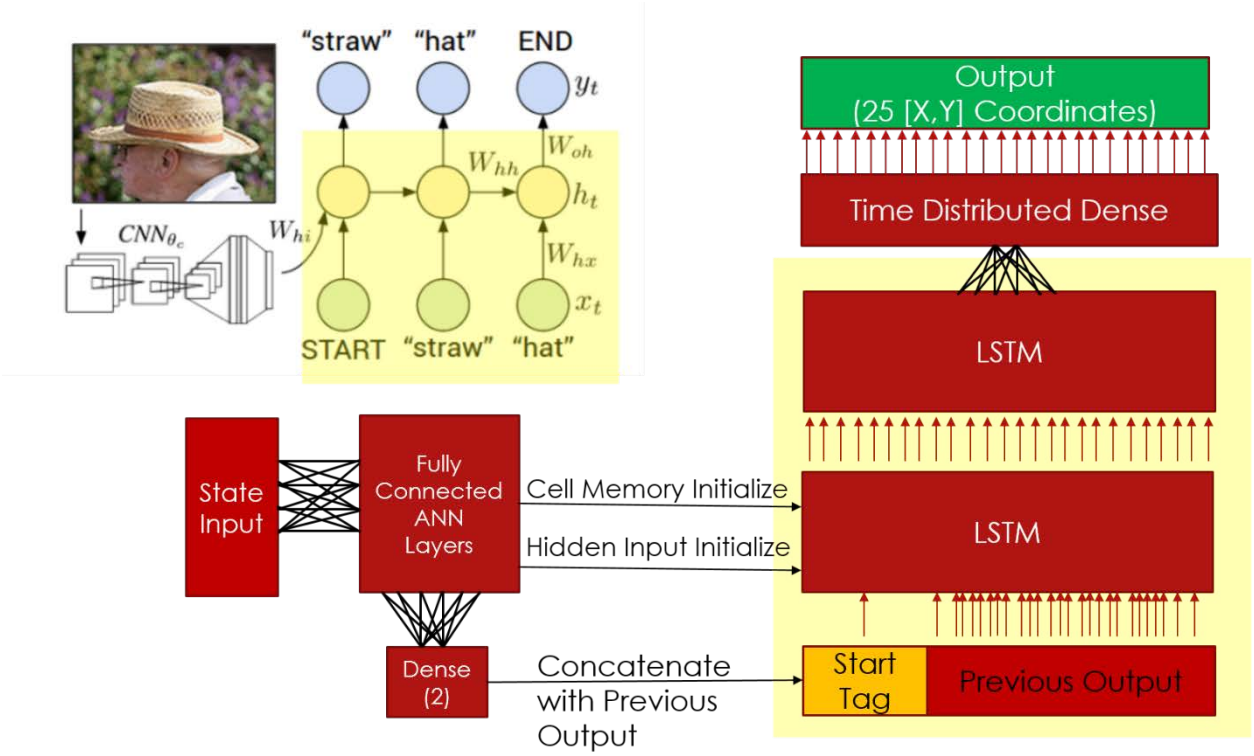

Image 14: Inspirational architecture analogous functionality – LSTM layers

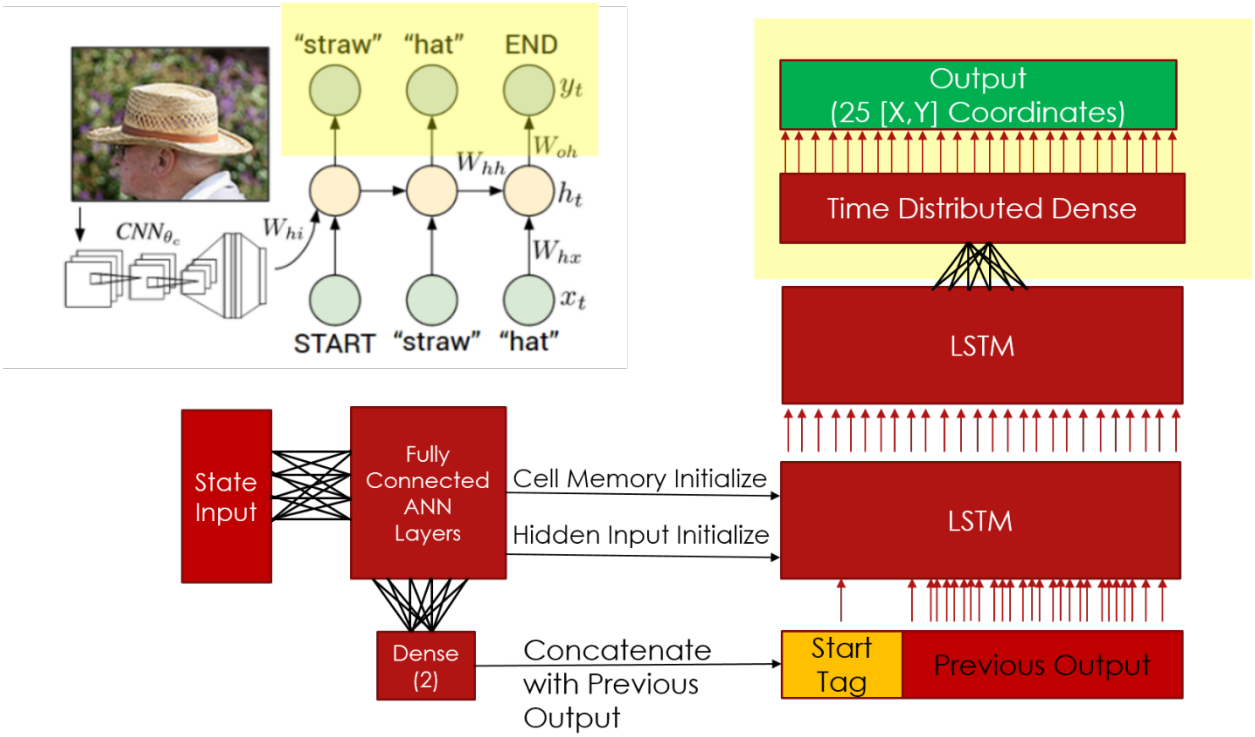

Image 15: Inspirational architecture analogous functionality – output

Image 13 to Image 15 placed the inspiring RNN image captioning architecture next to the *Space Navigator* LSTM architecture. The corresponding functionality is highlighted in each image to better understand how the new architecture was influenced.

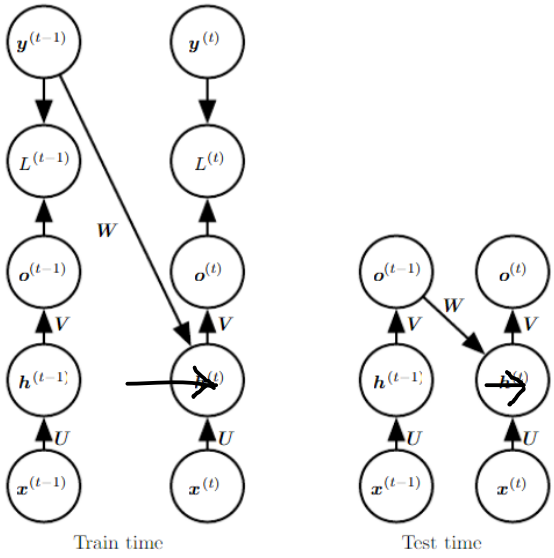

Image 16: Training and testing process

Image 16 illustrates teacher forcing. This will allow the model to be trained with the correct input vector size. In reality, all of the "x(t)" inputs will be the "y(t-1)" value. Teacher forcing lets the model learn as if it obtained the correct guess at each point. Incorrect guesses can possibly throw off the learning because errors can compound over time. During the prediction stage, the input trajectory slowly grows with each timestep. If the test is on timestep five, the prediction will occur all the way through the end, but only the first five timesteps are kept and entered in as the new input trajectory for the next timestep. The next timestep will give the sixth predicted point. This will go on until twenty-five points are retrieved. After the twenty-fifth point, a trajectory has been produced given a state. Perhaps a more customizable library such as PyTorch would allow for a more straightforward implementation of training and testing, but this workaround is needed for Keras.

Results:

The testing phase used ten percent of the data after it was shuffled. A single test iteration provided the final results. It would have been advantageous to get more reliable results through a k-fold test process, but that necessitates more data and a longer computation time.

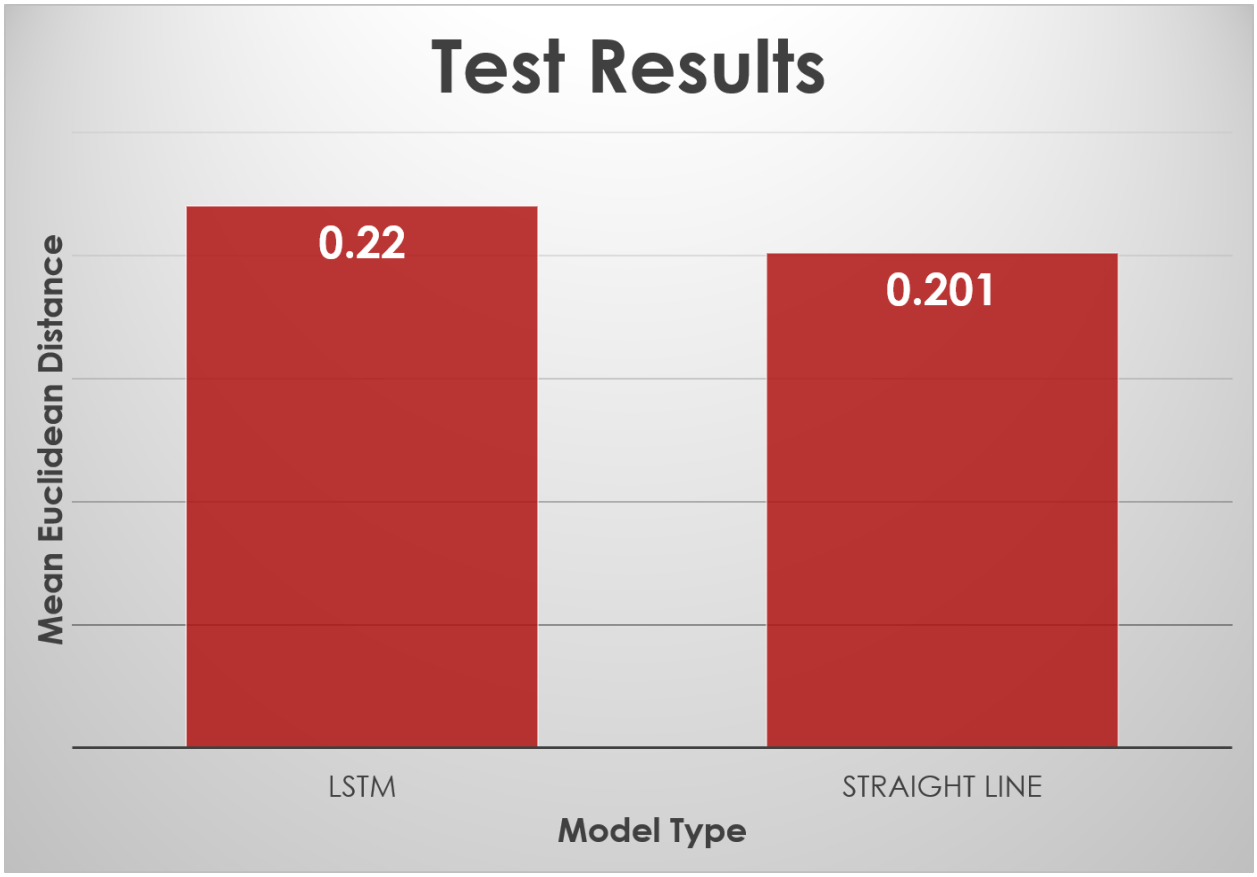

Figure 8: Test results

Figure 8 shows the test results of the LSTM architecture against the baseline. The value represents the mean Euclidean Distance an estimated trajectory point is from the

corresponding true trajectory point. As seen above, the LSTM architecture performed worse than simply drawing a straight line every time.

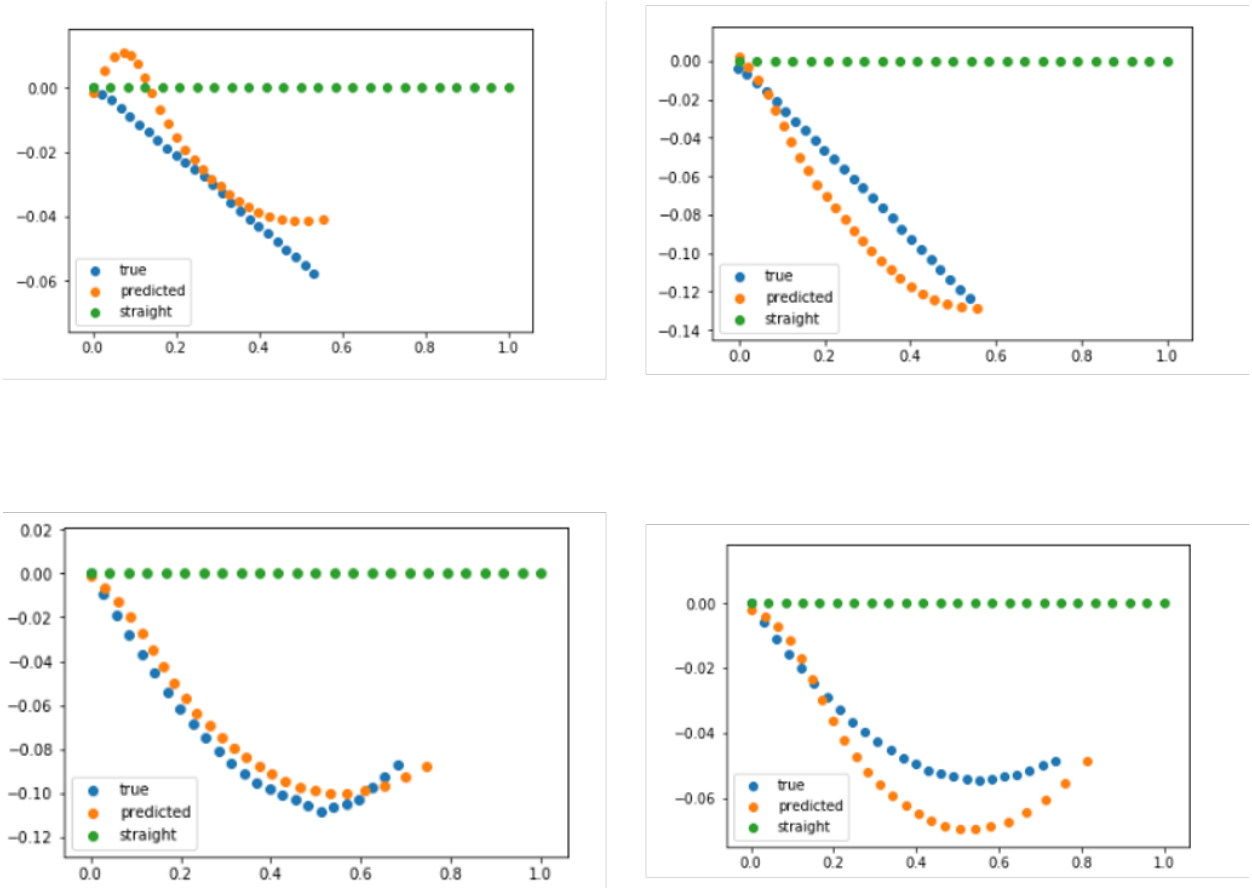

Image 17: Best predictions

The best predictions made by the LSTM architecture are shown in Image 17. This level of accuracy was initially expected from all of the predictions, but that was not the overall case.

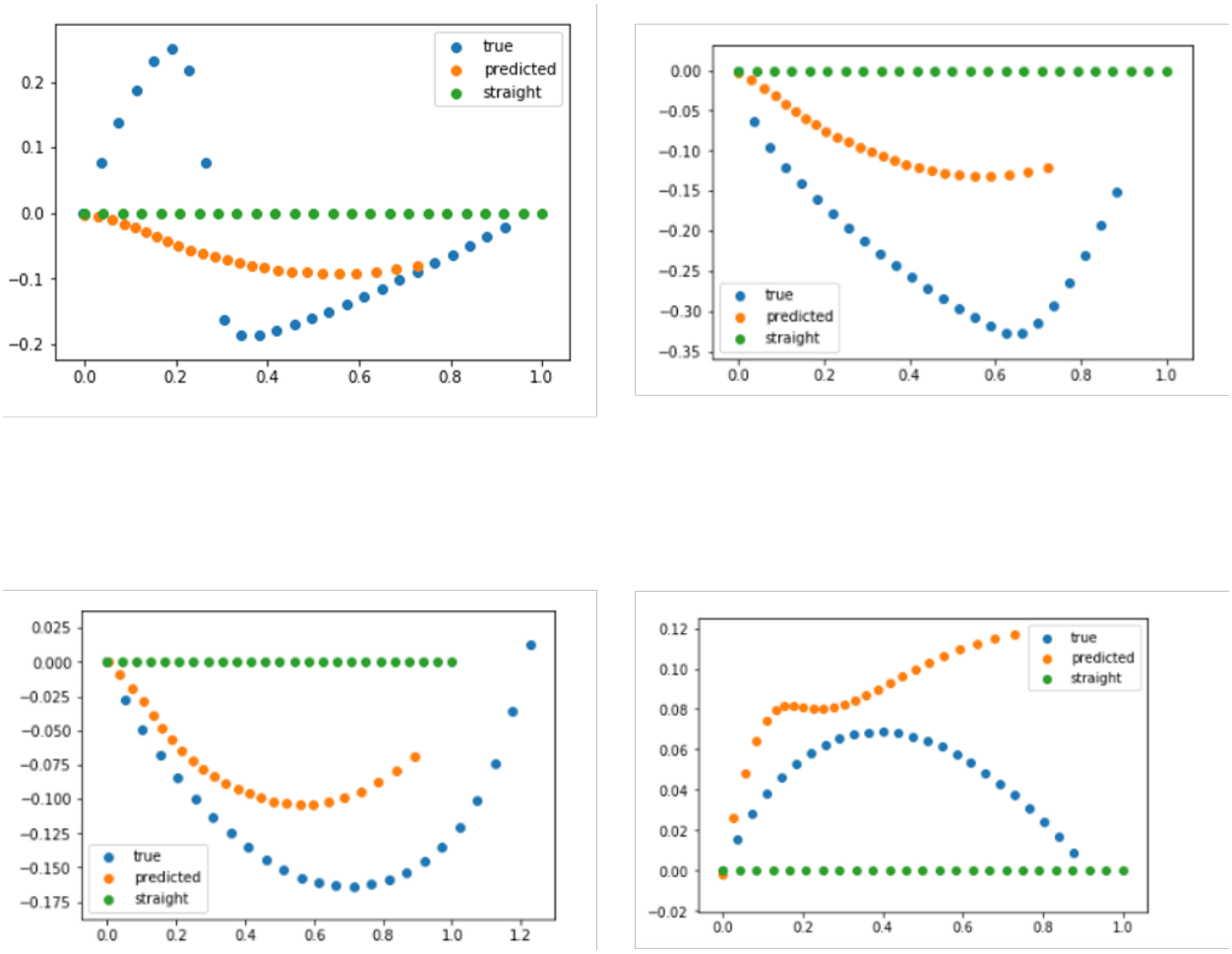

Image 18: Average predictions

The average performance predictions are showcased above. A generalization problem becomes noticeable. The top left prediction in Image 18 best exemplifies this. The true trajectory is curvy, but the prediction still scores satisfactorily by predicting through the middle. Eccentric curves will be difficult to accurately predict. Most of the predictions are third-order curves at most.

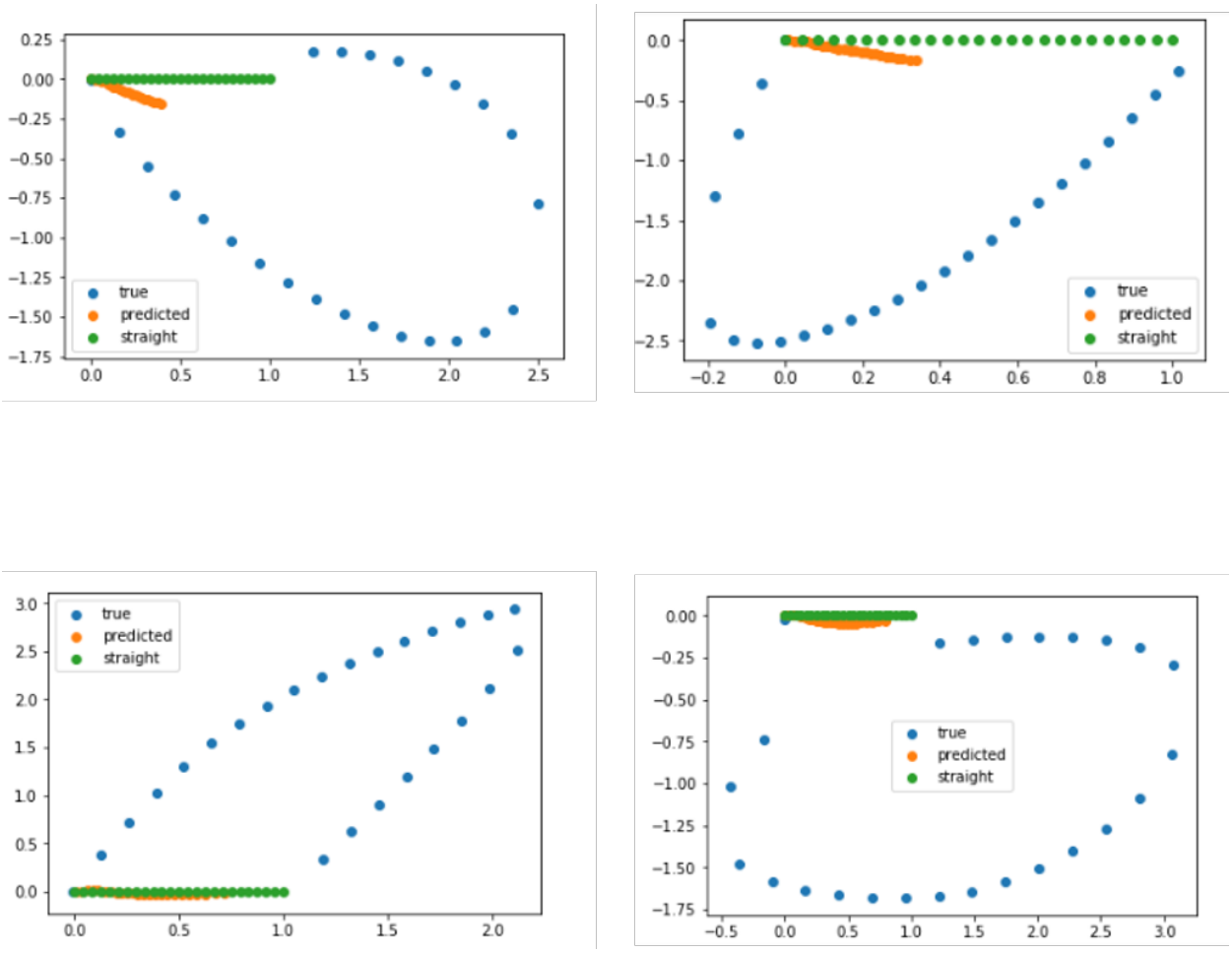

Image 19: Worst predictions

These are instances of the worst-case predictions. As seen in Image 19, if the user draws a very unique trajectory that spans a large distance, the neural network has a tough time predicting it.

This highlighting of player idiosyncrasies illustrates why the neural network did not perform better than the baseline. It is difficult enough to handle outlier trajectories from a single player, but now every player has the possibility to conflict with each other in their responses. As a result, the neural network has a tough time learning how a human

would draw since each player has their own characteristic playstyle. I decided to delve further with a quick look at using a dataset made with only a single player's data.

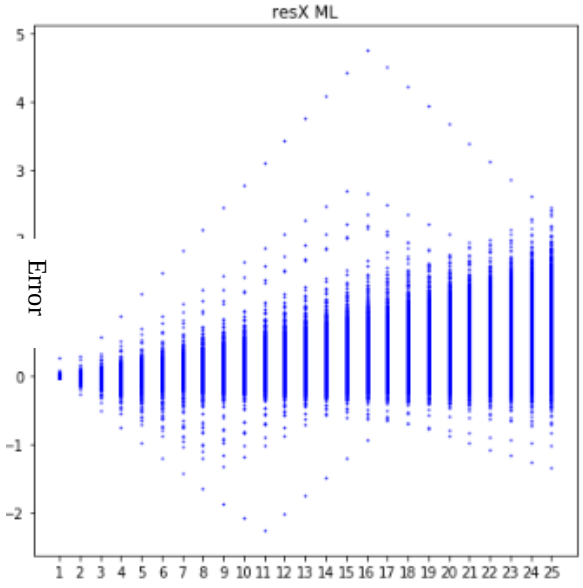

Figure 9: LSTM predicted residuals on the x-axis

Figure 9 is a plot of the residuals at each positional location from all of the trajectories.

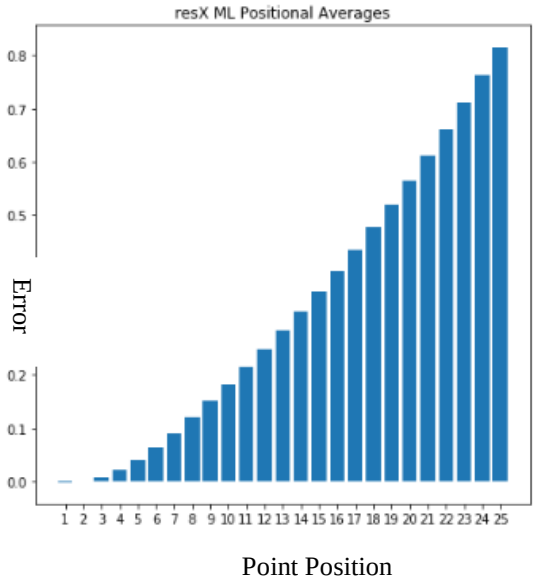

Figure 10: LSTM predicted average residuals on the x-axis

Figure 10 is the average of all the values at each point location in Figure 9.

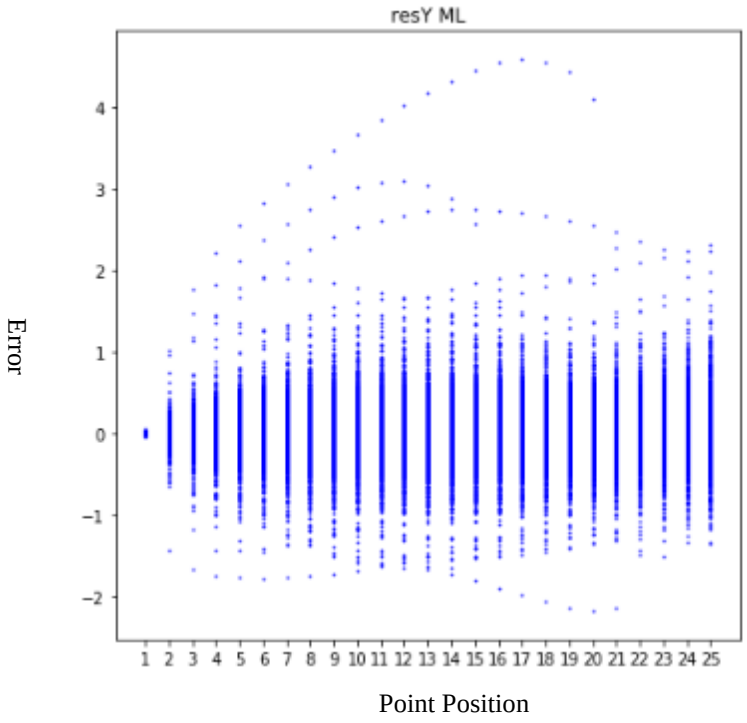

Figure 11: LSTM predicted residuals on the y-axis

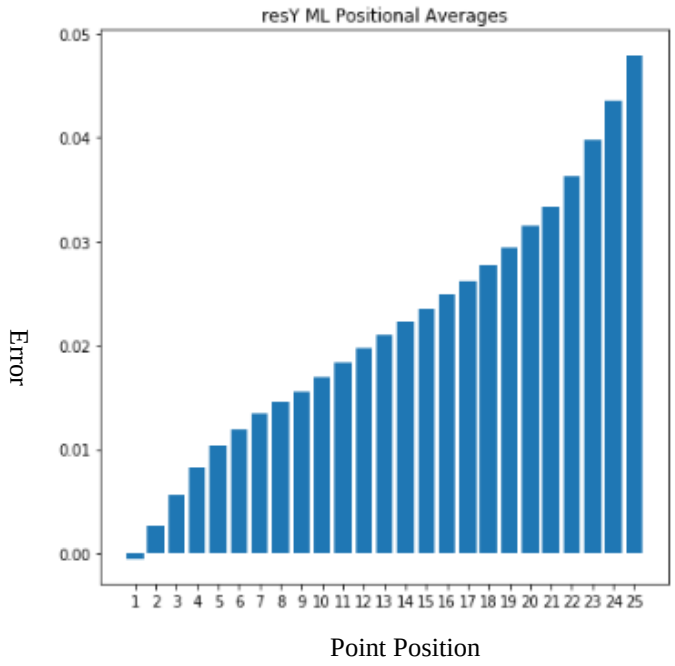

Figure 12: LSTM predicted average residuals on the y-axis

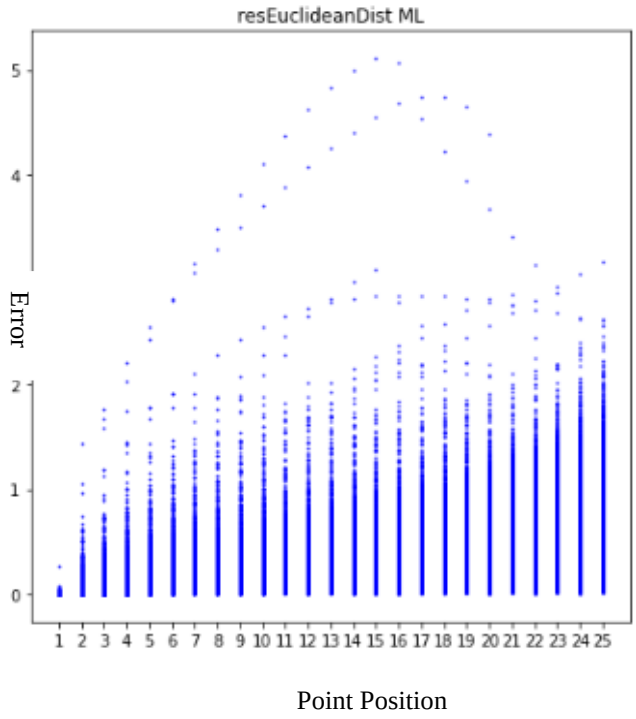

Figure 13: LSTM predicted Euclidean error

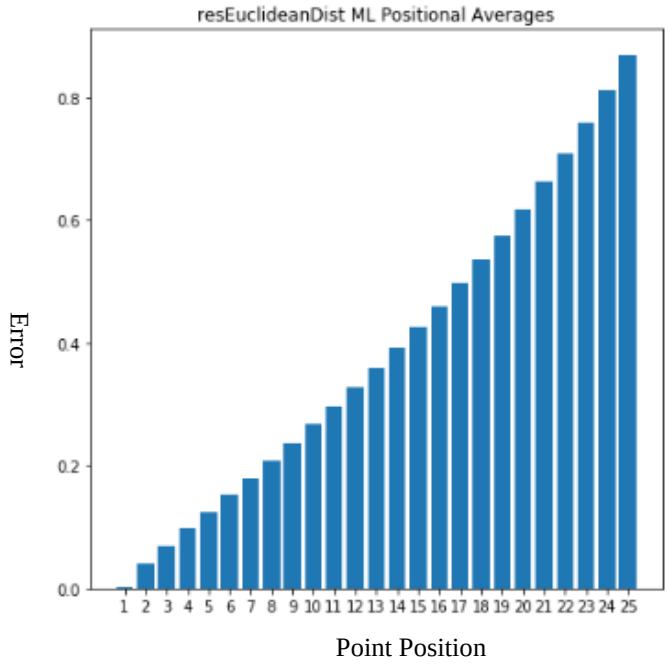

Figure 14: LSTM predicted average Euclidean error

The two primary characteristics from Figure 11 to Figure 14 are the strictly monotonic increasing and the almost fully positiveness of each averaging plot. The strictly monotonic increasing represents the error compounding over time. The predicted trajectory will add error on top of all the previous error made before. This mostly occurs because of difference in length of trajectories. The almost fully positiveness results indicate a shorter and lower predictor trajectory on average compared to the true trajectory.

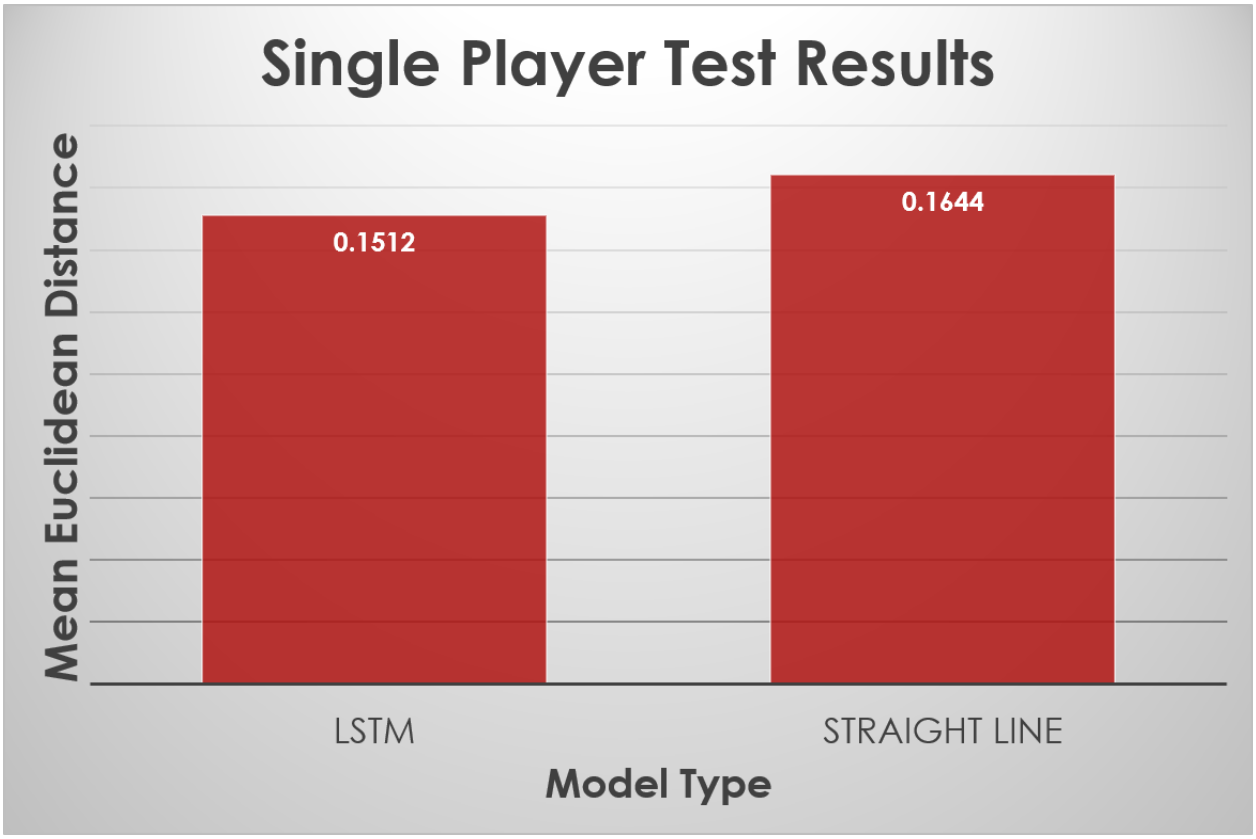

Figure 15: Single player test results

Figure 15 reveals successful results through a new process. Instead of grouping all the data together and having the neural network learn, improvement occurs when a single player's data is learned on and tested against. See Appendix D for result comparisons with additional models.

Conclusion & Future Work:

A neural network which learned unique predictions was successfully built. The predicted trajectories that the LSTM produces look like plausible trajectories that a human would draw. The individual points sequentially flow, and the trajectory aims from a selected ship towards a destination planet. Many different looking trajectories are produced given different states. This behavior signifies a sensitivity to the input features.

Even though the LSTM generated trajectories look human-like, they initially did not perform well. It was difficult for the AI to frequently draw a trajectory similar to what the human player actually drew. Although, the result was not awful considering the difficulty of the problem when grouping every player's data together. Multiple completely different trajectories could be produced for the same state instance. Different players usually give different responses. Even the same player might give two different responses to the same state input due to effects from learning the game.

The inherent difficulties of copying human behavior now have been exposed. The problem space is demonstratively unusual with the conflicting personalities of different players. The comparison with the baseline straight-line prediction displays the conflict of personalities. The heavily trained LSTM architecture performed worse than a straight-line prediction when all players' data were grouped together, but the LSTM architecture performed better than a straight-line prediction when only a single player's data was learned and tested. Moreover, the LSTM neural network was lightly trained on the single player data and a validation process was not administered to reoptimize the hyperparameters.

An LSTM that can achieve a better performance than the baseline with the full original dataset was not created. However, it was not due to a shortage of capacity or a failure in finding an optimal tradeoff between bias and variance in the model. The original dataset was too noisy when combining all the players' data. Too much irreducible error, specific to the engineered features and the LSTM model, was present.

As shown by the quick single player data test which easily outperformed the baseline, it is necessary to prioritize the difficulty presented by variation in responses due

to human factors. Simply addressing this issue determines success while optimizing the LSTM model on the original dataset resulted in failure. The differences between players were originally thought to be minimal, but it actually is a major concern.

The next research step should involve further exploring the previously touched on single player dataset solution. A faster computer should be used to eliminate the computational constraint when optimizing. This research was conducted with limited resources so the searchability of hyperparameters and unique augmentations was vastly restricted. Faster computation would allow a widening of the grid search space which has a better chance of enhancing optimization of the incorporated model.

Other solutions also exist for differentiating the players' behavior. Perhaps adding more features would allow the neural network to recognize which player drew in the observed state space. Maybe taking a similar direction to Bindewald et al. [1] could prove fruitful where most of the data is used as a bootstrap model to learn general tendencies, and then a single player's data heavily augments the baseline model to target a specific player's behavior. The answer could be to create a cognitive architecture which is sensitive to the recency of each data observation. This method would also focus attention on the idiosyncratic behavior of the last player's observed data. All of these are possible answers that would be worth taking a look at in future research.

References

[1] Bindewald, J. M., Peterson, G. L., & Miller, M. E. (2017). Clustering-based online player modeling. *Communications in Computer and Information Science, 705*, 86–100. https://doi.org/10.1007/978-3-319-57969-6_7

[2] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. Retrieved August 13, 2018, from http://www.deeplearningbook.org/

[3] Lindstrom, P., Delany, S. J., & Namee, B. Mac. (2010). Handling Concept Drift in Text Data Stream Constrained by High Labelling Cost Handling Concept Drift in a Text Data Stream Constrained by High Labelling Cost. *Florida Artificial Intelligence Research Society Conference (FLAIRS). Florida*, 19–21. https://doi.org/10.21427/D7B022

[4] Karpathy, A., & Fei-Fei, L. (2015). *Deep Visual-Semantic Alignments for Generating Image Descriptions*. Retrieved from https://arxiv.org/pdf/1412.2306.pdf

[5] Bindewald, J. M., Peterson, G. L., & Miller, M. E. (2015). Trajectory generation with player modeling. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9091*, 42–49. https://doi.org/10.1007/978-3-319-18356-5_4

Appendix A: Interpolation Method

Each initial trajectory is resampled to achieve a specific number of points per trajectory while maintaining the original trajectory shape. The interpolation method evenly spaces out the twenty-five trajectory points across the X-axis. When sampling trajectories during the experiment, the mouse speed was captured. If the player held the mouse down at the selected ship, and then quickly swiped to the destination planet, the majority of the sampled points would be cluttered around the selected ship with a couple points between the clutter and the destination planet. By resampling evenly across the X-axis, the mouse speed characteristic is lost. This is an attempt to produce less noisy trajectories by eliminating a seemingly unnecessary characteristic which causes extra difficulty. By evenly spreading out the trajectory points, the shape of the trajectory is saved while also becoming more robust with a fewer number of points. If the mouse speed was maintained in the trajectory, it would take more points to capture the original trajectory.

Even though this interpolation method is useful, it assumes that capturing the mouse speed during each portion of the trajectory is unnecessary. After production of the algorithm, it was discovered that users actually enjoyed a robotic teammate that drew trajectories with the mouse speed conserved. It added a more human-like element when the AI was actually integrated into *Space Navigator*. For now, the interpolation method will be kept the same, but it may be revamped in the future to recapture the original mouse speed trait.

Appendix B: Other Ship Gaussian Weight Function Multiplier

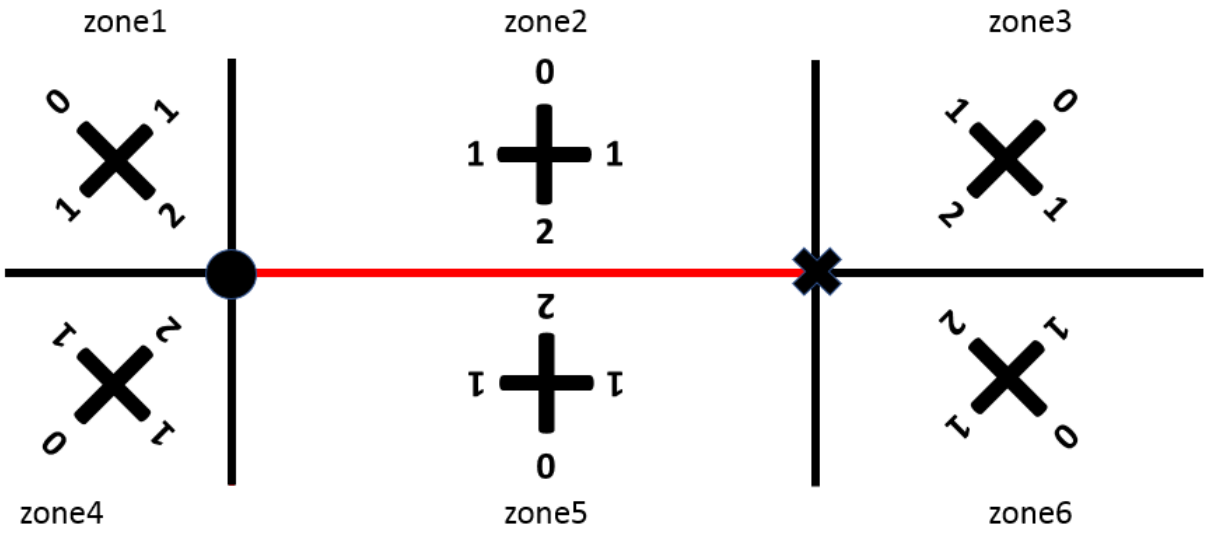

Image 20: Relevance factor calculation

The red line symbolizes the line of interest. The black circle is the selected ship and the small black 'X' is the destination planet.

The main idea in Image 20 is to illustrate the relevance weighting depending on which zone the other ship is located. Basically, if the ship is headed towards the line of interest, its zone value contribution gets multiplied by two. If the ship is headed away from the line of interest, its zone value contribution gets multiplied by zero. Every area space in-between these two points is interpolated on a continuous scale. For instance, if a ship is in the center of the "+" in zone 2 and is headed in a 45-degree angle (where straight towards the line of interest, or where the "2" is printed, is 0 degrees), the relevance factor will be a 1.5 for that ship.

This part of the algorithm exists because a player will most likely dismiss the effect of another ship's trajectory if it is headed away from the line of interest. Another ship heading away from the line of interest mainly is relevant only if the player wants to collect a bonus situated off-course from the line of interest. However, players primarily attempt to send the selected ship towards the destination planet, so it is safe to assume the relevance factor is a beneficial algorithm item.

Appendix C: Trajectory Euclidean average algorithm computed

Algorithm 2 Average error calculation between a single true trajectory and a single predicted trajectory

1: **input:** T_p = predicted trajectory made up of points $T_{p_1}, T_{p_2}, \ldots, T_{p_{n-1}}, T_{p_n}$;
   T_a = actual trajectory made up of points $T_{a_1}, T_{a_2}, \ldots, T_{a_{n-1}}, T_{a_n}$
2: **initialize:** q = empty array (length n)
3: Each point is a 2D coordinate. For example, $T_{a_1} = (X_{a_1}, Y_{a_1})$
4: **for** i in $1 \ldots n$ **do**
5: $\quad eDistance$ = Euclidean distance between T_{p_i} and T_{a_i}
6: $\quad q[i] = eDistance$
7: $meanDistance = \frac{\sum_{j=1}^{n} q[j]}{n}$
8: **return** $meanDistance$

Algorithm 5: Trajectory difference calculation

The above algorithm is applied to finding the difference between one test observation and the corresponding prediction. The final values displayed in the results section is the average of the error when the above algorithm was applied to all the test observations and all the corresponding predictions. The final values more concretely represent the average Euclidean distance between a test trajectory point and the corresponding predicted trajectory point.

Appendix D: Extensive Results Comparison

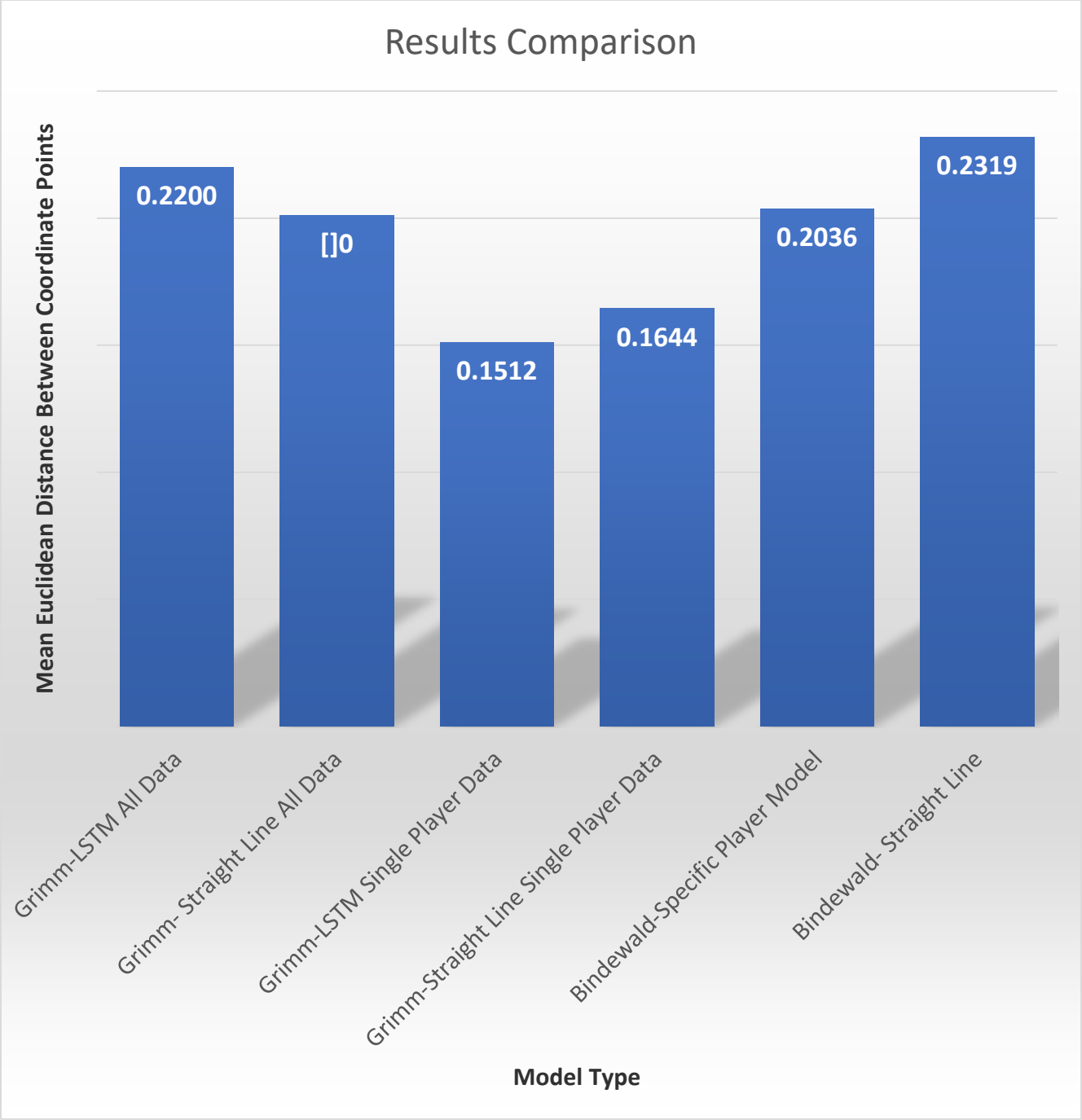

Figure 16: Comparison of results

Figure 16 at face value shows the “Grimm-LSTM Single Player Data” model performing the best out of all the models. It also shows different results for the “Grimm-Straight Line All Data” and “Bindewald- Straight Line [All Data]” even though it was on the same set of data. The difference can be at least partially attributed to testing on different parts of the data, and the resampling method of trajectories.

The “Grimm – LSTM Single Player Data” performed very well. However, the isolated single player did not draw very complicated trajectories as seen by the corresponding straight-line model achieving the second-best performance.

The Bindewald models also involved their own unique advantage. During the preprocessing stage, 25% of possible responses deemed as outliers are pruned. The “Grimm” LSTM model still has the possibility of providing an outlier-like response.

Another primary difference is that the “Bindewald Specific Player Model” creates a model for each specific player and averages the performance of each. The “Grimm LSTM Single Player Data” focuses on one player.

Because of the differences in methods, Figure 16 is not the ultimate distinguisher of different model types, but an early predictor. Further research is necessary to more evenly compare each model such as the “Grimm LSTM Single Player Data” model not isolating a relatively easily learned dataset.

Appendix E: Validation Details

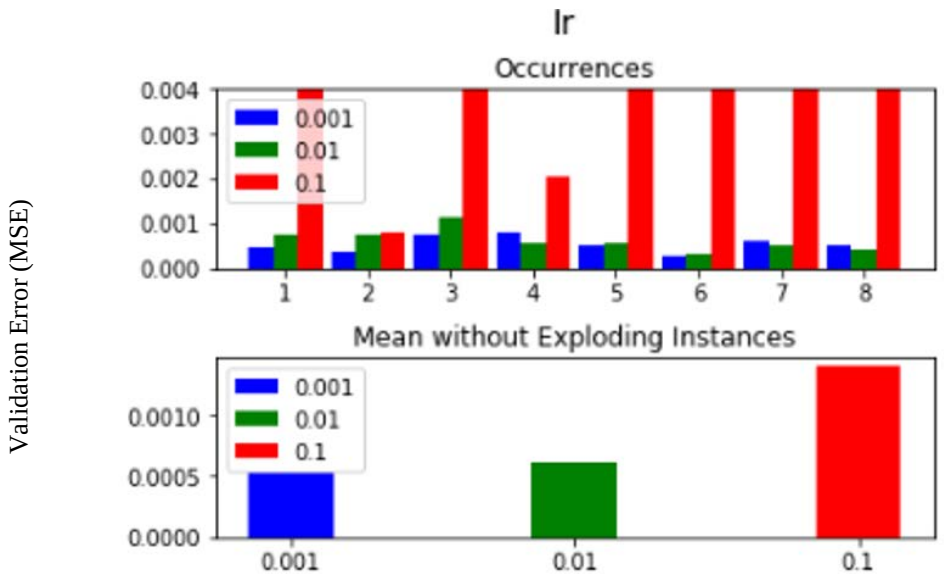

Figure 17: Learning Rate Comparison

The top graph shows a comparison of each hyperparameter instance. It is a way of looking at how each hyperparameter performed in general. The bottom graph shows the mean of the values to get a higher-level overview. However, extreme outliers are excluded so the averages are not greatly skewed if one of the instances involved an exploding gradient. This description of the layout for Figure 17 will be the same as the layout for Figure 18, Figure 19, and Figure 20.

The learning rate of 0.001 in Figure 17 usually outperformed the other two learning rates on a consistent basis. The 0.1 learning rate never performed very well.

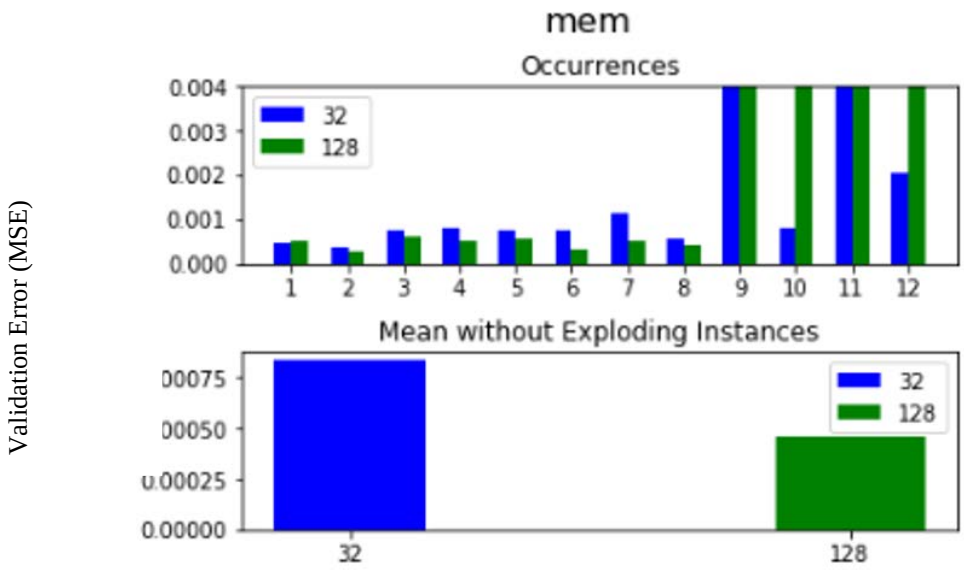

Figure 18: Capacity Comparison

Figure 18 has both 32 and 128 performing well; however, a capacity of 128 reliably is a little better besides a few instances. The average for 128 is much better since the exploding gradients are not included.

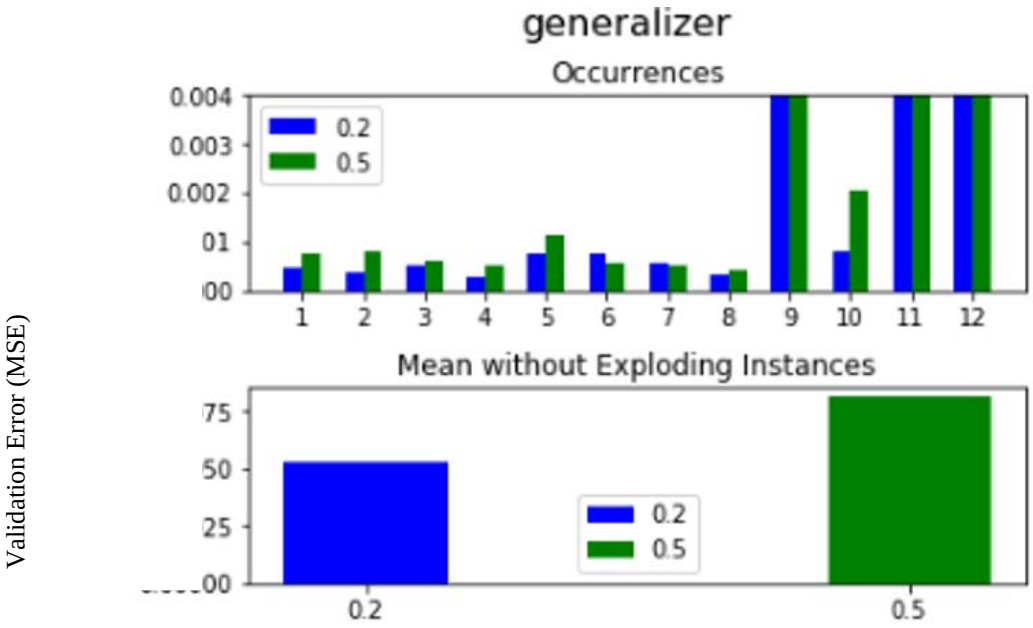

Figure 19: Generalization Comparison

The generalization rates in Figure 19 indicates a low generalization rate as usually better. A higher variance in the learned model leads to a higher performance in the validation set.

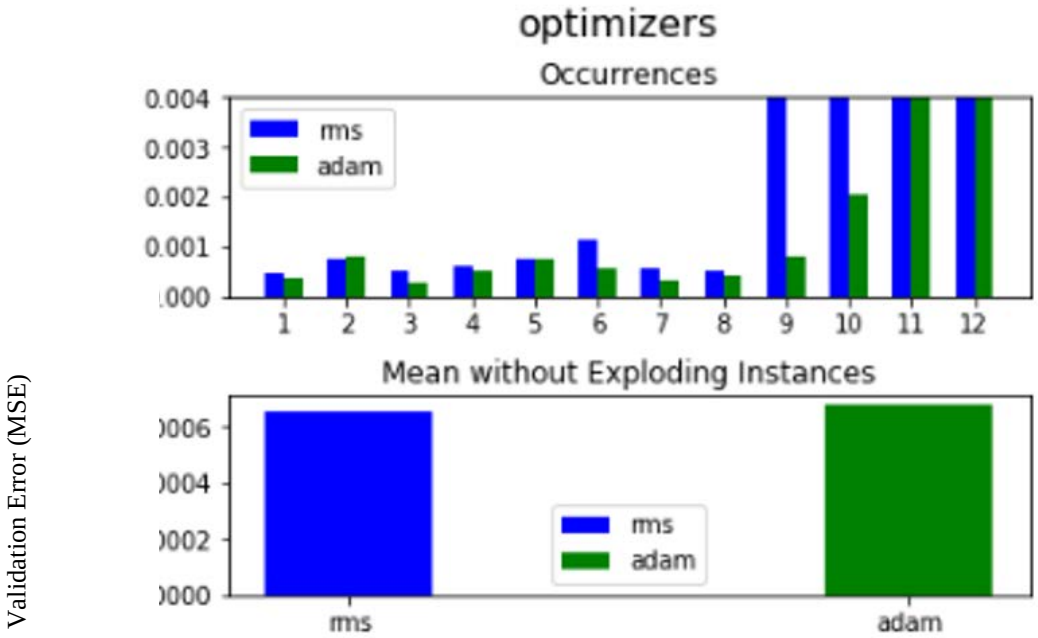

Figure 20: Optimizers Comparison

Figure 20 has the Adam and RMSProp optimizers as almost equal in the averages graph. Though, Adam was less likely to result in an outlier due to poor performance.

Overall, these general views of how each hyperparameter performed in each validation model instance just gives a survey of performance. In the end, the single model with the best performance became the chosen model. There was no individual selection of hyperparameters, but rather a selection of the single best combination of hyperparameters. Each hyperparameter's performance depends on the other temporarily selected hyperparameters in this case since the hyperparameter performances are not independent of each other.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
21-03-2019	Master's Thesis	Sept 2017 - March 2019

4. TITLE AND SUBTITLE

Imitating Human Responses Via a Dual-Process Model Approach

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Grimm, Matthew A, 2d 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
Wright-Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-19-M-030

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Autonomy Capability Team 3
Dr. Steven Rogers, AF Senior Scientist for Autonomy
2241 Avionic Circle, WPAFB, OH 45433
Email: steven.rogers@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RY

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Statement A. Approved for Public Release;Distribution Unlimited

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Human-autonomous system teaming is becoming more prevalent in the Air Force and in society. Often, the concept of a shared mental model is discussed as a means to enhance collaborative work arrangements between a human and an autonomous system. The idea being that when the models are aligned, the team is more productive due to an increase in trust, predictability, and apparent understanding. This research presents the Dual-Process Model using multivariate normal probability density functions (DPM-MN), which is a cognitive architecture algorithm based on the psychological dual-process theory. The dual-process theory proposes a bipartite decision-making process in people. It labels the intuitive mode as "System 1" and the reflective mode as "System 2". The current research suggests by leveraging an agent which forms decisions based on a dual-process model, an agent in a human-machine team can maintain a better shared mental model with the user. Evaluation of DPM-MN in a game called Space Navigator shows that DPM-MN presents a successful dual-process theory motivated model.

15. SUBJECT TERMS

Artificial Intelligence, Shared Mental Models, Cognitive Architecture, Dual-Process Theory, Human-Machine Teaming

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. Gilbert L. Peterson, AFIT/ENG
U	U	U	UU	218	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3636 x4281 Gilbert.Peterson@afit.edu

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18