Trajectory Generation with Player Modeling[*,†]

Jason M. Bindewald, Gilbert L. Peterson, and Michael E. Miller

Air Force Institute of Technology, Wright-Patterson AFB, OH, USA
{jason.bindewald, gilbert.peterson, michael.miller}@afit.edu

Abstract. The ability to perform tasks similarly to how a specific human would perform them is valuable in future automation efforts across several areas. This paper presents a k-nearest neighbor trajectory generation methodology that creates trajectories similar to those of a given user in the *Space Navigator* environment using cluster-based player modeling. This method improves on past efforts by generating trajectories as whole entities rather than creating them point-by-point. Additionally, the player modeling approach improves on past human trajectory modeling efforts by achieving similarity to specific human players rather than general human-like game-play. Results demonstrate that player modeling significantly improves the ability of a trajectory generation system to imitate a given user's actual performance.

Keywords: Human-Computer Interaction; Player Modeling; Clustering; Trajectory Generation

1 Introduction

The ability to perform tasks similarly to how a human would perform them is valuable to future automation efforts. For example, in adaptive automation (AA), where a human-machine team (HMT) achieves a goal by varying the allocation of tasks between human and machine entities during system operation, the actions of the releasing entity up to that point in time can positively or negatively impact the HMT's future performance [3]. Therefore, the similarity of the machine's task performance to that of the human affects the overall human-machine team's performance.

One specific arena that can benefit from similar action performance is automated trajectory generation. Given a specific state, the ability to generate a trajectory that is similar to a trajectory generated by a unique user would provide opportunities in many areas. In object motion tracking, generating similar

[*]The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense, or the United States Government.

[†]This work was supported in part through the Air Force Office of Scientific Research, Computational Cognition & Robust Decision Making Program (FA9550), James Lawton Program Manager.

trajectories to those of a specific performer allows for prediction of future movements. In Air Traffic Control environments, generating trajectories similar to those of a specific person might enable collision avoidance in crowded areas.

This paper contributes an automated "full maneuver" trajectory generation system that creates trajectory responses similar to those of a given user in the *Space Navigator* tablet computer game [3]. Through player modeling it improves on past efforts in two ways. First, the trajectory generator produces trajectories as whole entities rather than creating them point-by-point. Second, the player modeling approach achieves significantly better similarity to specific human players over past human trajectory modeling efforts that mimic general human-like game-play.

This paper proceeds as follows. Section 2 reviews related work in the fields of trajectory generation and player modeling. Section 3 presents the specific user trajectory generation system methodology through the *Space Navigator* testbed, a $\underline{k}$-$\underline{n}$earest $\underline{n}$eighbor (k-NN) trajectory generation process, and a clustering-based technique for player modeling. Section 4 gives experimental results showing the clustering-based player modeling method's improvements. Section 5 summarizes the information presented and proposes potential future work.

2 Related Work

Most existing trajectory generation and prediction methods create trajectories in a piecemeal fashion using various methods such as trajectory libraries [11] or Gaussian mixture models [15]. As states are recognized, the trajectory generator predicts the next point on the trajectory. It then either returns this single point as the prediction or recursively finds further points until completing a full trajectory. Generating a trajectory one point at a time is unlike the "full maneuver"-based manner humans apply when generating trajectories for flight tasks [9]. Additionally, there is little research that imitates the behavior of a *specific* operator generating trajectories in a dynamic environment.

Within other application domains, player modeling has provided the ability to distinguish a single user from among several others. There are two main types of player models: model-based and model-free [16]. In a model-based player model the player types are pre-defined according to some set of features identified by the practitioner. Examples include the use of supervised neural networks [5], trait theory to pre-determine player types [2], and defining strategy groupings based on game design features [14].

Model-free player modeling learns player types that arise from the collected data and then defines player types from the groupings. A model-free method finds frequently occurring player groupings within the feature data. Previous research relies on clustering methods such as hierarchical clustering [6], evolutionary algorithms [12], and self-organizing maps [6]. Lazy learning methods, such as k-NN [11], provide model-free player modeling methods that are useful when a large number of training examples are available [1]. Within a database of example state-action pairs, the system is presented a state and a k-NN search

returns the k states that most closely resemble it. The system generates a response by using the k responses associated with each state in some way. Methods of this type have been used successfully in imitation learning in robotics [1, 7] and trajectory generation [11].

3 Trajectory Generator with Player Modeling

To generate "full maneuver" trajectories similar to the ones generated by specific users, a cluster-based player modeling method was developed. The trajectory generator performs the steps in Fig. 1 in two sections: trajectory generation and player modeling. Operating in the *Space Navigator* environment, both sections rely on a consistent representation technique for dynamic states and a method for comparing trajectories of differing length. These techniques are used first by a k-NN based trajectory generation algorithm and then a clustering-based player modeling method allows the trajectory generator to create trajectories similar to those of a specific user.

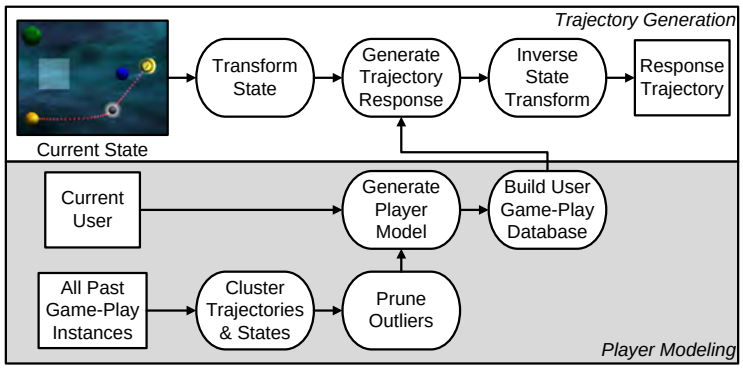

Fig. 1. A player modeling trajectory generator.

3.1 Application Environment Considerations

Space Navigator [3] is a tablet computer-based trajectory generation game, in which spaceships appear from the sides of the screen. The player draws lines on the screen directing each ship to a destination planet. Points accumulate when ships encounter their destination planets or bonuses that appear in the play area. Points decrement when ships collide and when a ship traverses one of several "no-fly zones" (NFZs) that move to random locations at set intervals. The game ends after five minutes, a time that provided a high level of user engagement in a pilot study. *Space Navigator* has only one action available to the player, but maintains enough dynamism that an automation cannot achieve a "best" input. Game-play data was collected from 32 participants. The tablet presented each of four difficulty settings in random order to each participant four

times, resulting in 16 games per person. When the player draws a trajectory, the game captures data associated with the game state (e.g. the drawn trajectory and bonus locations).

To address the enormous state-space of *Space Navigator*, state representations contain only elements that affect a user's score (other ships, bonuses, and NFZs) scaled to a uniform size. The feature vectors transform the state-space by rotating and translating so the selected ship is at the origin with the X-axis as the straight-line trajectory between the ship and destination planet, and scaling it so the ship to planet trajectory is unit length. The feature vectors account for differing element locations by dividing the state-space into the six zones delineated in Fig. 2. To compare inconsistent numbers of objects, each zone accumulates a score in a manner modeled after [7] over each object within the zone. Scores use a Gaussian weighting function on the shortest Euclidean distance from the object to the straight-line trajectory, as shown in Fig. 2. The resulting feature vector consists of 18 positive real-valued scores that can be adjusted using different weighting methods.

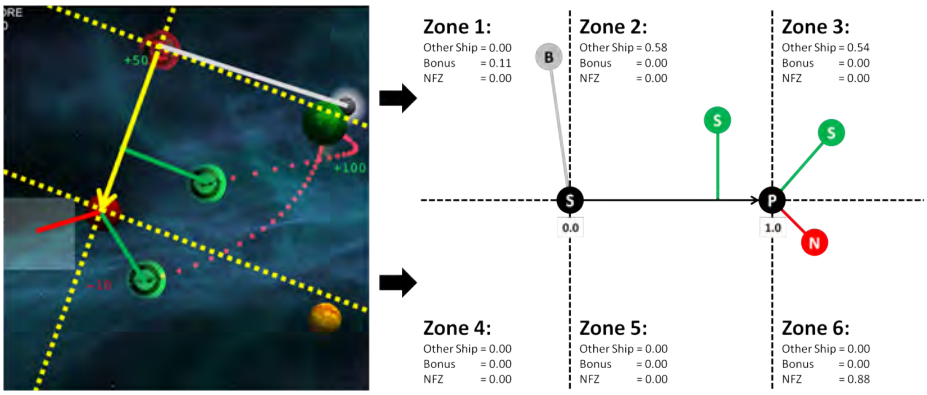

Fig. 2. A *Space Navigator* state and its transformed feature vector representation.

Several steps in the trajectory generator shown in Fig. 1 require trajectory comparison. However, *Space Navigator* trajectories contain differing numbers and locations of points. Trajectory comparison requires both transformation and re-sampling. Trajectories are geometrically transformed in the same manner as their paired states. Then the trajectories are re-sampled to 50 points (approximately the average number of points in a trajectory found during experiments) using trajectory smoothing and linear interpolation [10]. Trajectory comparison uses Average Coordinate Distance (ACD) [8], which is a measure of distance between same-length trajectories based on ordinally comparing pairs of points.

3.2 Trajectory Generation

The trajectory generator portion of Fig. 1 leverages the state representation and trajectory re-sampling methods to create new trajectory responses from a gameplay database. A k-NN search finds the k closest states, according to Euclidean

distance, to a new state out of the user's game-play database. The trajectory generator weights each of the k user response trajectories based on Euclidean distance from the query using a Gaussian Radial Basis Function (RBF) [4]. The RBF gives more weight to states that are closer to and less weight to those that are farther away from the new state's representation. An influence variable can be used to determine how much more weight is given to closer states than distant ones. The response trajectory returned by the generator is a weighted average of the k-NN response trajectories given. Finally, the combined trajectory is normalized over the total weight of all k trajectories.

3.3 Player Modeling

To imitate a specific user's game-play, the trajectory generator implements the clustering based player modeling technique in the bottom of Fig. 1, by modifying the game-play databases that feed the trajectory response generator. Clustering state-trajectory pairs by state and trajectory allows common state and trajectory types to be analyzed. Cluster membership helps prune outlier instances from consideration. The clusters form a player model of state to trajectory cluster mappings representing a player's tendencies, and player models form the basis of a user game-play database population method.

Cluster States and Trajectories. Ward agglomerative clustering [13] provides a way to group instances by state and trajectory type. In agglomerative clustering, each instance begins as its own cluster. Clusters are progressively joined together until reaching a chosen number of clusters. Ward's method was proven effective for clustering in a trajectory creation environment in [9]. The number of clusters, 500, was determined heuristically and has not been tested experimentally. The agglomerative clustering assigns each instance to one state cluster and one trajectory cluster. An instance mapping a state cluster to a trajectory cluster demonstrates a user's proclivity to react with a given maneuver in a specific game situation. The frequency of state to trajectory cluster mappings reveals common responses and outlier actions.

Prune Outliers. A trajectory response from only one user in one instance is unlikely to be replicated by future users, and proves counterproductive in predicting future user action in preliminary experiments. So, outlier instances are pruned from the set of game-play instances available for use in building a specific user's game-play database. Instances with outlier trajectory responses are removed by removing all instances assigned to the least populated trajectory clusters. Instances falling in the bottom 25% of all trajectory clusters according to cluster size are removed. Outlier state clusters are removed in two ways. First, instances that fall in the bottom 25% of all state clusters according to cluster size are removed. Second, instances where the state cluster was encountered by an extremely small subset of users are removed. The number of instances to prune was determined through visual inspection of the resulting clusters.

Generate Player Model. The player model creation method begins with a user and the set of all state to trajectory cluster mappings left after pruning. It first creates a set of counters to help determine how many of the user's state-trajectory pairs belong to each cluster mapping and then processes each of the state to trajectory cluster mappings. For each cluster mapping, the number of instances provided by the user that belong to it is recorded. The player's model is a set of likelihoods that a given state-trajectory pair chosen at random from the pruned game-play database belongs to a specific cluster mapping, normalized by the total number of instances within the pruned game-play database.

Build User Game-Play Database. The trajectory generator then uses the player model to create a new game-play database. The database builder method adds state-trajectory pairs until the user game-play database has reached a desired size. At each iteration, a state to trajectory cluster mapping is selected according to player model probabilities. Then the database builder method checks whether the given cluster mapping contains any more instances. If so, a random instance from the cluster is added to the new game-play database. The resulting user game-play database is returned.

4 Results

An experiment using 10-fold cross-validation tests the effectiveness of the player modeling trajectory generator. Four 10-fold, 1,500 instance user game-play databases are created for each user: (1) a straight-line database where each response trajectory is the line from the ship to its destination planet; (2) 1,500 random instances from the user's original database; (3) a generic player database created with the player modeling method across all user game-play instances; and (4) a specific player database created with the player modeling method on only one user's game-play instances. For each game-play database type, the experiment sets aside one fold (150 instances) as the test set and the remaining (1,350) instances as the game-play database. One trajectory response is generated from each of the 32 single-user game-play databases, for each of the 150 test instances' state representations. For k-NN based methods, a value of $k = 5$ is set based on preliminary experimental results. The experiment records the ACD from the generated response trajectory to the user's actual trajectory. This is repeated over each of the current user's ten folds.

Comparison testing of the game-play databases shows that the trajectories generated using the specific player model database improved specific user imitation results when compared to those generated by the other three game-play databases. Table 1 contains results comparing trajectories generated using each database with the actual trajectory provided by the user, showing the mean ACD and 99% confidence interval across all 1,500 instances and 32 users.

The worst performing user game-play database was the straight line database. The mean ACD of 0.2319 and its 99% confidence interval (CI) are both the largest values for any database. The original user database slightly outperformed

Table 1. Mean and 99% CI of the average coordinate distances (ACD) across all users.

Database	Mean ACD	99% CI
Straight-Line	0.2319	$[0.2299, 0.2339]$
Original User DB	0.2311	$[0.2292, 0.2331]$
Generic Player Model	0.2186	$[0.2169, 0.2204]$
Specific Player Model	0.2036	$[0.2018, 0.2055]$

the straight-line database. The straight-line database performs very well in many instances where the best possible trajectory response is extremely close to a straight line, while the original user database contains all of a user's outlier instances. The generic player model database improves on the straight-line and original user databases, because it maintains the instances where straight-line response trajectories are useful while avoiding the outlier instances of the original user's database. The specific player model provides a significant improvement over the player model at the 99% confidence level. This database improves on the generic database by ensuring a representation of common instances based on a user's game-play habits and strategies in that user's game-play database. The specific player model outperformed all others for 30 of 32 users.

Although there is improvement using the specific player model, further improvements could be gleaned by addressing three issues. First, a more detailed state representation scheme could improve the accuracy of generated trajectories, but would hinder the ability of the computer to generate real-time trajectories. In its current form, there is information present within each state that is not represented, such as the pre-drawn trajectories of other ships. Secondly, a user will not always make a similar decision when presented with same situation. Some players will operate within the game inconsistently, and over time all players' game-play patterns will change. Lastly, user actions are often unpredictable in situations when learning is occurring, which was likely the case over the course of this experiment. As a result, the players' tendencies likely changed over time.

5 Summary and Future work

The results show that the clustering-based player modeling method for user game-play database creation allows a k-NN based trajectory generator to imitate specific user game-play patterns. This research demonstrates that by eliminating outlier game-play instances and ensuring that the game-play database represents a user's demonstrated tendencies, the ability of the trajectory generation system to imitate the user's actual trajectories improves in efficiency. These results show an improvement over incremental trajectory generation techniques by using a "full maneuver" approach, which allows for modeling specific users.

Future work in this research effort will include improvements to the real-time performance of the player modeling method and a follow-up *Space Navigator* data collection. Real-time performance improvement will focus on reducing

"state-to-response" speed and enabling real-time incremental player model learning. To ensure that ACD captures the perceptions of human player, a further data collection will compare objective measures for trajectory similarity (e.g. Euclidean distance, ACD) with subjective measures of trajectory similarity provided by human game players. The resulting similar trajectory generator could allow insights into what makes specific users unique (both good and bad) in a trajectory generation environment. These insights can then be used to aid in areas such as adaptive automation, automated training, and adversary modeling.

References

[1] Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robotics and Autonomous Systems 57(5), 469–483 (2009)

[2] Bateman, C., Lowenhaupt, R., Nacke, L.E.: Player typology in theory and practice. In: Proceedings of DiGRA (2011)

[3] Bindewald, J.M., Miller, M.E., Peterson, G.L.: A function-to-task process model for adaptive automation system design. International Journal of Human-Computer Studies (2014)

[4] Boyd, J.P., Bridge, L.R.: Sensitivity of RBF interpolation on an otherwise uniform grid with a point omitted or slightly shifted. Applied Numerical Mathematics 60(7), 659–672 (2010)

[5] Charles, D., Black, M.: Dynamic player modeling: A framework for player-centered digital games. In: Proc. of the International Conference on Computer Games: Artificial Intelligence, Design and Education. pp. 29–35 (2004)

[6] Drachen, A., Canossa, A., Yannakakis, G.: Player modeling using self-organization in tomb raider: Underworld. In: IEEE CIG 2009. pp. 1–8 (Sept 2009)

[7] Floyd, M.W., Esfandiari, B., Lam, K.: A case-based reasoning approach to imitating robocup players. In: FLAIRS Conference. pp. 251–256 (2008)

[8] Hu, W., Xie, D., Fu, Z., Zeng, W., Maybank, S.: Semantic-based surveillance video retrieval. IEEE Transactions on Image Processing 16(4), 1168–1181 (2007)

[9] Huang, Victor, H.H.S.T., Tomlin, C.J.: Contrails: Crowd-sourced learning of human models in an aircraft landing game. In: Proceedings of the AIAA GNC Conference (2013)

[10] Li, X., Hu, W., Hu, W.: A coarse-to-fine strategy for vehicle motion trajectory clustering. In: ICPR 2006. vol. 1, pp. 591–594. IEEE (2006)

[11] Stolle, M., Atkeson, C.G.: Policies based on trajectory libraries. In: ICRA 2006. pp. 3344–3349. IEEE (2006)

[12] Togelius, J., De Nardi, R., Lucas, S.M.: Towards automatic personalised content creation for racing games. In: IEEE CIG 2007. pp. 252–259. IEEE (2007)

[13] Ward Jr, J.H.: Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association 58(301), 236–244 (1963)

[14] Weber, B., Mateas, M.: A data mining approach to strategy prediction. In: IEEE CIG 2009. pp. 140–147 (Sept 2009)

[15] Wiest, J., Hoffken, M., Kresel, U., Dietmayer, K.: Probabilistic trajectory prediction with gaussian mixture models. In: IEEE Intelligent Vehicles Symposium. pp. 141–146. IEEE (2012)

[16] Yannakakis, G.N., Spronck, P., Loiacono, D., André, E.: Player modeling. Artificial and Computational Intelligence in Games 6, 45–59 (2013)