Archetypical Motion: Supervised Game Behavior Learning with Archetypal Analysis

Rafet Sifa  
Game Analytics and University of Bonn  
rafet@gameanalytics.com

Christian Bauckhage  
Fraunhofer IAIS and University of Bonn  
christian.bauckhage@iais.fraunhofer.de

Abstract—The problem of creating believable game AI poses numerous challenges for computational intelligence research. A particular challenge consists in creating human-like behaving game bots by means of applying machine learning to game-play data recorded by human players. In this paper, we propose a novel, biologically inspired approach to behavior learning for video games. Our model is based on the idea of movement primitives and we use Archetypal Analysis to determine elementary movements from data in order to represent any player action in terms of convex combinations of archetypal motions. Given these representations, we use supervised learning in order to create a system that is able to synthesize appropriate motion behavior during a game. We apply our model to teach a first person shooter game bot how to navigate in a game environment. Our results indicate that the model is able to simulate human-like behavior at lower computational costs than previous approaches.

I. INTRODUCTION

In addition to being a popular source of entertainment and revenues, video games have spawned considerable research in human level artificial intelligence (AI). Modern video games not only provide realistic environments that simulate the real world but also allow for investigating task specific behaviors of players, for instance by means of mining the network traffic between game clients and game servers.

Over the years different computational intelligence techniques have been considered as a means of bridging the gap between human behavior and video game AI. Recent examples include analyzing social behavior in MMORPGs [1], analyzing players’ lifetime values [2] or modeling their experience [3], generating game content [4], or creating believable Non-Player Characters (NPCs) [5]–[7]. For a comprehensive review of the state of the art regarding the latter, we refer to [8].

One of the many challenging tasks in modeling believable NPC behavior is the problem of generating autonomously navigating game bots. Recent games use techniques like finite state machines or the A∗ algorithm to implement navigation behaviors. Such techniques create an illusion of intelligence that vanishes quickly once players get used to a game because the actions of game-bots tend to become stale and predictable.

In this paper, we address this problem and study a behavior learning model that uses recorded human game-play data to create NPCs that act human-like. Our focus is on learning how to navigate the game environment. Similar to how movement is realized in biological organisms, we model player actions in terms of mixtures of primitives. That is, we describe actions as displacement vectors that move game agents from one location to another and assume a finite number of action primitives that are linearly combined to represent any possible action that the game bot can perform.

Our computational model is composed of two parts: first, we extract action primitives from large amounts of in-game data, and second, we learn how to mix action primitives based on the state history of a game bot. For the first part, we identify action primitives using Archetypal Analysis [9] to represent motions as convex mixtures of primitives. For the second part, we build a predictive model that learns mixing coefficients in a supervised manner so that, during the game, our system can determine suitable motions and synthesize them automatically. To the best of our knowledge, this is the first study that applies archetypal analysis to game play data in order to create movement behavior for game-bots.

We test our behavior learning model in the context of Unreal Tournament 2004® (UT 2004), a fast paced first person shooter game by Epic Games (see Fig. 1). For the work reported here, we access the decision making modules via the middle-ware library Pogamut 3 [10].

Next, we review earlier work on supervised learning in video games and briefly discuss action primitives. Then, in Section III, we provide an introduction to Archetypal Analysis and explain the Simplex Volume Maximization Algorithm to efficiently determine archetypes from data. In Section IV, we elaborate on our model and present experimental evaluations and comparisons to other behavior learning techniques in Section V. Finally we conclude our contribution in Section VI.
II. MOTIVATION AND RELATED WORK

In this section, we briefly review previous approaches that used Machine Learning and Pattern Recognition in order to create more believable NPCs in video games. Furthermore, we introduce the notion of movement primitives from the fields of biology and neuroscience and discuss examples of their use in robotics and game AI.

Learning the behavior of a video game character can be cast as estimating a function \( f \) that maps from a player’s state space to the corresponding action space [5]. In particular, given a history of player states \( s_t, s_{t-1}, \ldots \), where \( t \) defines the current time, and the current state of the game environment \( e_t \), the function \( f \) is supposed to determine an appropriate action \( a_{t+1} \), that is

\[
a_{t+1} = f(s_t, s_{t-1}, \ldots, s_{t-k}, e_t) \quad \text{and} \quad (1)
\]

\[
s_{t+1} = s_t + a_{t+1}. \quad (2)
\]

The above model has been adapted in many studies on learning human-like behavior in video games. Using recorded tournament data sets Zanetti and Rhalibi [11] realized the function \( f \) by means of Multilayer Perceptrons (MLPs) to learn aiming, shooting, weapon selection, and navigation. Thurau et al. [6] used a hybrid, neural network based method that combines Self Organization Maps and MLPs to learn human-like behavior from previously recorded demo data for the FPS game Quake II. The same authors also considered Bayesian learning methods based on state transition probabilities in a way point map created using a Neural Gas Algorithm [12]. Similar learning approaches have been adopted to other game genres as well. Chaperot and Fye [13] used MLPs in a motocross game to teach game bots how to ride a motorcycle from recorded human game-play data. Gemine et al. [7] used MLPs to learn production strategies in Starcraft II.

Though generally successful, all these approaches are rather ad-hoc in that they did not systematically address aspects of information representation or coding. In particular, questions pertaining to suitable topologies or parameterizations of neural architectures have not been studied in the above contributions and, consequently, issues such as computational efficiency or memory consumption have been largely ignored.

Aspects of efficient neural representations are studied in biology and neuroscience. With respect to the learning of movement behaviors in video games, we may thus assume another point of view on how to model the generation of behaviors. One of the currently dominant theories addressing the production of motion behavior relies on the notion of action primitives. Action primitives are understood as finite sets of elementary action generators whose outputs are combined in order to solve complex tasks. For instance, having analyzed the forces generated in the limbs of frogs through stimulating their spinal cords, Bizzi et al. [14] showed that certain force fields can act as primitives which, when linearly combined, create other force fields that are in the limb’s workspace. Considering this mechanism as a weighting scheme, the weights are believed to be determined by the nervous system [15]. Contributions like these support the idea that elementary motor primitives are not unique for every action the body performs but rather that combinations of general motor primitives provide the set of behaviors (or repertoires) that biological bodies can perform [15].

The notion of movement primitives has already been used in robotics, for instance to learn human arm movements. Taking four principle components of an arm trajectory data set as primitives and linearly combining them, Park and Jo [16] described an optimization method to calculate the mixture coefficients that realize complex motions using the primitives. A similar representation has also been studied in the context video games. Thurau et al. [17] proposed to learn human-like movement behavior in Quake II and extracted elementary movement vectors using Principal Component Analysis (PCA). They then determined action primitives as the most frequently occurring actions and used probabilistic models to select appropriate action vectors based on a player’s state history.

With the work reported here, we extend these previous contributions. In particular, we introduce convexity constraints in the determination of primitives. This causes the extracted motion primitives to be extreme motions rather than average ones and, as we will show, leads to more efficient representations for behavior synthesis.

III. ARCHETYPAL ANALYSIS AND SIMPLEX VOLUME MAXIMIZATION

Given a data set the main goal of Archetypal Analysis [9] is to determine extreme entities in the data called archetypes. Once archetypes have been determined, every available data entity can be represented as a convex combination of these extremes (see Fig. 2).

Formally, the problem is as follows: Given an \( m \) dimensional data set of \( n \) entities \( X = \{ x_i \in \mathbb{R}^m \mid i = 1, \ldots, n \} \), Archetypal Analysis attempts to find \( k \ll n \) archetypes \( W = \{ w_j \in \mathbb{R}^m \mid j = 1, \ldots, k \} \) and \( n \) stochastic coefficient vectors \( H = \{ h_i \in \mathbb{R}^k \mid i = 1, \ldots, n \mid \| h_i \|_1 = 1 \} \) to approximate each data vector as a convex combination of the
archetypes. Thus, once archetypes are available, we may write

\[ x_i \approx x'_i = \sum_{j=1}^{k} w_j h_{ij} \quad \text{where} \quad h_{ij} \geq 0 \land \sum_{j=1}^{k} h_{ij} = 1. \quad (3) \]

Various ways have been proposed to determine suitable Archetypes. Cutler and Breiman [9] proposed an alternating least squares algorithm to simultaneously identify archetypes and coefficients. Unfortunately, this constrained convex optimization problem scales cubically with the number of data and hence does not allow for efficient learning from large data sets.

Thurau et al. [18] therefore introduced several methods to speed up archetypal computation. In particular, they proposed the Simplex Volume Maximization (SIVM) algorithm as a provably linear time approach to determine archetypes by means of fitting a simplex of maximum volume to a given data set. In contrast to the approach in [9], SIVM constrains the archetypes \( w_j \) to coincide with certain extremal data points \( x_i \). The approach is based on notions from distance geometry and considers the volume of the Cayley-Menger Determinant in order to determine suitable archetypes; the essential steps of the procedure are summarized in Algorithm 1.

\begin{algorithm}[htb]
\caption{Simplex Volume Maximization}
\begin{algorithmic}
\STATE Select \( x_i \) randomly from \( X \)
\STATE Choose the first simplex vertex:
\hspace{1em} \( w_1 = \arg\max_x \text{dist}(x_1, \arg\max_x \text{dist}(x_i, x_j)) \)
\FOR{index \( i \in [2, k] \) \DO
\STATE Let \( S_{i-1} \) be the current simplex with \( i - 1 \) vertices.
\STATE Find the vertex that maximizes:
\hspace{1em} \( w_i = \arg\max_q \text{Vol}(S \cup x_q) \)
\STATE Update \( S_i = S_{i-1} \cup w_i \).
\ENDFOR
\end{algorithmic}
\end{algorithm}

IV. ARCHETYPAL MOTION

In this section we describe our hybrid biologically inspired model to learn movement behavior in video games. We adapt the idea of Mussa-Ivaldi and Bizzi [15] to video games; instead of directly mapping from the game state space to the action space we aim to represent actions of players as weighted combinations of previously determined action primitives. Our goal is to obtain a learning model that is not only biologically adequate but also efficient in terms of learning.

Dividing our model into two parts, we first compute a representation of primitives that highly matches with biological mechanisms. Second, we aim to generalize this model in a supervised manner. Fig. 3 shows an overview over the workflow of our system for learning action primitives and building a model to generalize the mixing coefficients.

A. Archetypal Analysis to Identify Action Primitives

With respect to the learning of motion behaviors, we define a primitive as an archetypal action that moves the body of the game player from one location to another. Having recorded \( n \) displacement vectors \( D = \{ d_i \in \mathbb{R}^3 \mid i = 1, \ldots, n \} \) from the game environment as our training action vectors, we determine action primitives by means of extracting \( k \ll n \) archetypes.

\[ P = \{ p_j \in D \mid j = 1, \ldots, k \} \] from \( D \). Suitable mixing coefficients \( h_{ij} \) are then determined from minimizing

\[ \min_{h_{ij}} \sum_{i=1}^{n} \left\| d_i - \sum_{j=1}^{k} h_{ij} p_j \right\|^2 \quad (4) \]

subject to \( h_{ij} \geq 0 \land \sum_{j=1}^{k} h_{ij} = 1 \)

which can be accomplished using common solvers [19]. Subsequently, we can represent each displacement vector \( d_i \) as a convex combination of archetypal movements

\[ d_i = \sum_{l=1}^{k} h_{il} p_l. \quad (5) \]

This representation directly matches the notion of action primitives described in [15]. Additionally, since we consider convex combinations, our model also copes with the fact that action primitives cannot be combined arbitrarily but have to comply with limitations (i.e. work-spaces of body parts [15]) when generating movements. In the next step, we show how we can generalize our model and learn the mixing coefficients in order to control the navigation behavior of an Unreal Tournament 2004\textsuperscript{®} game bot.

B. Learning Mixing Coefficients

In order to use the above model to generate actions from human action data in real time, we need to determine how to find the mixing coefficients to combine the primitives.

Formally, using the primitives \( P \), the coefficient vectors \( H \), and histories of state vectors \( S \), our main goal is to learn a function

\[ f_{\text{mixture}} : S \rightarrow \mathbb{H} \quad (6) \]

where \( \mathbb{H} \) denotes the domain of all possible mixing vectors \( h \). Considering the state history as in (1), our task is to find the
mixing coefficients $h^t$ at each time step $t$ and then use them to weight the combination of the primitives to determine the next action $a_{t+1}$. That is, we compute $h^t$ and $a_{t+1}$ as follows

$$h^t = f_{\text{mixture}}(s_t, s_{t-1}, \ldots, s_{t-q})$$

(7)

$$a_{t+1} = \sum_{j=1}^{k} h^t_j p_j.$$  

(8)

Any function approximation technique can be used to learn $f_{\text{mixture}}$. We choose to learn $f_{\text{mixture}}$ using Multilayer Perceptrons (MLPs) as they are considered to be the universal function approximators. Additionally, MLPs are in line with our idea of biologically adequately generated motions since they compute mixing coefficients using neural mechanisms.

Since our training data consists of $n$ consecutive data samples containing pairs of states and actions, we can train our MLP histories using histories of states as input. Following the model in (1), we thus set a time delay parameter $q$ to denote the length of sequence of states and build an input data set

$$S = \{ \sigma_t = (s_t, s_{t-1}, \ldots, s_{t-q}) \mid t = n, n-1, \ldots, q \}$$

(9)

for training. Similarly, we define the output as the set that is composed of the corresponding actions observed in our recordings

$$A = \{ a_t \mid t = n, n-1, \ldots, q \}.$$  

(10)

Since our aim is to interpret every action as a combination of the previously determined primitives, we replace the actions in $A$ with the corresponding coefficients that maps to the actions. Namely, we map the actions $a_j \in A$ to their corresponding coefficient vector $h_j \in H$ and obtain

$$O = \{ h_t \mid t = n, n-1, \ldots, q \}.$$  

(11)

Finally, after training an MLP with the input data set $S$ and output data $O$, we can generate mixing coefficients at run time. Combining the primitives weighted by the coefficients output by the trained MLP provides a game bot with the action to be taken at the current state of the game.

V. RESULTS AND EVALUATIONS

In this section we present results of experiments conducted to evaluate our model. Here, we focus on our model’s learning capability in two baseline settings and consider the following imitation tasks to be learned: running a circular or a crossed path. Then, having shown that our model is capable of learning these tasks, we compare it to different previous models to demonstrate that our model facilitates learning and that it generates plausible movements from only a rather small number of action primitives.

A. Experiment Settings

In order to determine mixing coefficients during runtime of a game, we used Time Delayed Neural Networks (TDNNs) which are a type of MLPs where input vectors contain a history of previous states. We trained our model with data extracted at 10 Hz, from the UT 2004 map DM-TrainingDay. Figure 4 shows examples of two dimensional projections of recorded in-game data.

As the number of states in the history vectors is a crucial parameter when considering the complexity of learning task, we evaluated its impact on performance. Extensive experiments showed that histories of 3 and 8 state vectors for circular and crossed motion, respectively, yielded the best results in terms of smoothness of the motion performed by the bot. Therefore, we confine our presentation of results to cases observed for these settings.

Similar to [12], we clustered the data set using a Neural Gas algorithm to learn the topology of the map and to provide a way to discretize the state space, and to be able to directly
B. Learning Circular and Crossed Motion

As a starting point, we trained our model with data sets containing 1493 and 5727 samples from tours of a human player running on a circular and crossed paths respectively. In a first step we extracted the movement primitives using SIVM. Figure 5 shows a two-dimensional visualization of how the actions observed in the corresponding data are distributed over six automatically determined archetypal motion vectors. Note that the archetypal motion vectors shown in the figure are indeed three-dimensional. Variations along the \( x \) and \( y \) directions are immediately obvious; variations along the \( z \) direction are encoded using colors: cyan represents a downwards displacement, for instance observed whenever the player did run down a flight of stairs; red represents an upward motion, for instance observed during jumps or when walking up stairs; vectors of magenta color represent planar motion only.

Having trained MLPs to determine mixture coefficients for activity synthesis, we measured an MSE of 0.0015 for the training data and an MSE of 0.0016 for the test data on circular motions. For motions along a crossed path, we observed a training MSE of 0.0018 and a test MSE of 0.0019. In either case, using the resulting MLPs in the decision making module of UT 2004 to synthesize motion within the game environment, we visually observed the resulting game bot to behave very similar to the human player whose data had been used for training.

In order to analyze the effect of the number of neural gas centroids for topology representation on behavior learning, we conducted experiments with varying numbers of Neural Gas Centroids. For the configurations described above, we repeated the training process 30 times for an increasing number of
neural gas centroids and analyzed the training and testing errors for circular and crossed paths. Figures 6 and 7 show our results for the two tasks considered. (Note that here and in the following figures, the $y$ axis is scale to a maximum of 0.01 to ease comparisons.) The results summarized in these figures suggest that the appropriate number of Neural Gas clusters is likely to be 60 for the task of circular motion data as this yields the lowest average testing error. Additionally, we observe that choosing centroid numbers larger than about 20 does not significantly impact the performance of our behavior learning module. For the task of learning to run along a crossed path, we observe similar behavior; the error values do not differ significantly if the number of Neural Gas centroids is increased beyond 80. Yet, the smallest average mean error is found for 160 centroids.

### C. Evaluations

Having tested the learning capability of our model, we conducted two additional sets of experiments in order to investigate the impact of representing actions as combinations of primitives and the merits of using Archetypal Analysis to determine primitives.

In the first type of experiments we examine the benefits of adding the intermediate step of learning primitives and describing a players’ actions as combinations of action primitives. To this end, we compare the learning results of our model to those of a model that learns a direct mapping from the state space to the action space.

Figure 8 shows the resulting average errors on our test data for circular motion where the whiskers indicate the corresponding standard deviations. Using the direct mapper, our model with 6 and 8 action primitives. We observe that, when using only six action primitives, our approach always yields better results than the direct mapper. We performed the same type of experiments to learn crossed motion using either a direct mapper or our model with 6, 8, 10, or 12 action primitives. The resulting average errors are shown in Fig. 9. Again we find that our model, which learns stochastic coefficient vectors to synthesize actions, outperforms the approach where state histories are mapped to actions directly.

Therefore, action primitives determined by means of Archetypal Analysis seem to simplify the problem of context sensitive learning of complex activities. This is indeed well in line with biological findings and corresponding theories as to the utility of action primitives. In other words, our results suggest that the task of learning to perform appropriate actions is simplified if complex actions are being synthesized from elementary ones rather than being represented explicitly.

In the second set of experiments we attempt to validate the usefulness of our representation of action primitives in terms of extreme data points. To this end, we compare our model to an approach where action primitives are assumed to be average data points. That is, we compare our model to a behavior learning method based on defining action vectors in terms of primitives that occur frequently. The latter strategy has been proposed by Thrun et al. [17] and we follow their suggestion in that we we cluster our training set of player actions using $k$-means clustering [21] in order to determine action primitives. Subsequently, we used MLPs of the structure and parametrization as above to learn mappings from the state space to a space of coefficient vectors which are used to synthesize actions.

Using the same color coding as above, Fig. 10 displays the results of $k$-means clustering of our action data sets for circular and crossed motions. Comparing these results to the ones in
In this paper, we addressed the problem of learning believable game-bot behavior from recordings of human game play. To this end, we proposed a model that adapts the notion of action primitives from biology and neuroscience. In contrast to earlier related work, we explored the merits of using extreme data points rather than average ones to represent action primitives. Our resulting model thus consists of two stages.

In the first stage, we use Archetypal Analysis by means of the SIVM algorithm in order to extract action primitives from a data set of observed human activities. In the next stage, we apply supervised learning using Multilayer Perceptrons in order to map histories of game states to coefficient vectors which then allow us to synthesize actions in terms of convex combinations of archetypal actions. The resulting framework was shown to produce believable, i.e., human-like, behaviors for NPCs.

In addition to being biologically adequate, extensive experiments using Unreal Tournament 2004 as a test-bed revealed the following beneficial properties of our approach:

- Convex combinations of archetypal actions are able to mimic a variety of more complex activities which are necessary to allow a game bot to perform extended tasks.
- Convex combinations of archetypal actions show very good quantitative and qualitative performance; quantitatively, our approach yields consistently lower error rates on test data than related approaches that attempt to learn actions directly; qualitatively, our approach was observed to produce in-game behavior that closely resembles that of human players.
- Convex combinations of archetypal actions require fewer primitives in order to produce appropriate behaviors than approaches based on linear combinations of average or frequent actions; we attribute this to the fact that the MLP which learns to associate state histories with coefficient vectors has to deal with constraint domain; that is, in contrast to learning architectures that have to learn arbitrary combinations of primitives, the mixing coefficients in our model obey convexity constraints $h_{ij} \geq 0$ and $\sum_j h_{ij} = 1$. In other words, the coefficient vectors $h_i$ reside in a $k$ dimensional simplex so that the range of the output of our MLP is naturally bounded. This yields higher numerical stability in training, faster convergence to solutions, and apparently good generalization capabilities.

To the best of our knowledge, synthesizing behaviors from archetypal action primitives has not been attempted before, neither in biological research, nor in robotics or game AI. Given the favorable performance of the proposed model in rather simple tasks, the obvious next step is to apply the approach to more demanding settings. This will include capabilities for, say, aiming and shooting and likely require more elaborate hybrid architectures such as in [6], [11], [13]. In future work we will thus explore to what extend the approach proposed in this paper can be tailored towards hierarchical systems consisting of dedicated expert networks that are activated in a context-dependent manner.

In fact, as this idea is akin to the idea of mixtures of expert architectures [22] where there are gating networks whose output layers weight the contributions of individual experts, we recognize a similarity to the idea of behavior synthesis from action primitives. We therefore have reasons to be optimistic that this approach will yield useful behaviors for game bots.

Fig. 10: Action primitives resulting from $k$-means clustering where wet set $k = 6$.

Fig. 5, we note that primitives resulting from $k$-means tend to represent shorter displacements and appear to cover a smaller variety of displacement angles. Using $k$-means to determine 6 or 8 action primitives, we obtained a bot that manages to run the circular but shows rather abrupt changes of direction. This apparent artificial, i.e., less believable, behavior disappeared after increasing the number of $k$-means centers to 20.

For the scenario where the bot was supposed to learn to run a crossed path, we obtained an even less encouraging result. While our model was perfectly capable to synthesize complex motion from only a small number of primitives, the approach based on $k$-means clustering was not. Having trained the $k$-means based model to learn crossed motion with 6, 8, 10, 12 primitives, we did not obtain working bots but bots that frequently ran into walls or deviated from their path. As stated in [17] increasing the number of $k$-means centers has an obvious impact on the type of the motion and, in our case, increasing the number of primitives helped smoothing the generated motion. Yet, even though for models with 50, 100, 150, and 200 action primitives the motion performed by the bot were noticeably improved, the bot was still hitting the walls from time to time or got stuck in curves. These results indicate that the more complex the task is the larger the number of primitives we need if we use $k$-means to represent action primitives.

Finally, we note that the results of the last set of experiments illustrate the impact of the level of complexity of a task to be learned from human generated data. While our example of learning to perform circular motion around a map can be used as a baseline test-bed to assess the capabilities of a learning architecture, the slightly more difficult task of learning to move along a crossed path already poses challenges that learning frameworks which are too simplistic cannot cope with anymore.

VI. Conclusion and Future Work

In this paper, we addressed the problem of learning believable game-bot behavior from recordings of human game play. To this end, we proposed a model that adapts the notion of action primitives from biology and neuroscience. In contrast to earlier related work, we explored the merits of using extreme data points rather than average ones to represent action primitives. Our resulting model thus consists of two stages.
REFERENCES


