ACSIJ Advances in Computer Science: an International Journal, Vol. 3, Issue 4, No.10 , July 2014
ISSN : 2322-5157
www.ACSIJ.org

ACSIJ
WWW.ACSIJ.ORG

Locomotion Planning with 3D Character Animations by Combining Reinforcement Learning Based and Fuzzy Motion Planners

Peyman Massoudi[1], Alireza Bagheri[23] *

[1]Department of Computer Engineering, Tehran Science and Research Branch, Islamic Azad University, Damavand, Iran
peyman.massoudi@gmail.com

[2]Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran
ar_bagheri@aut.ac.ir

[3]Faculty of Engineering, Tehran North Branch, Islamic Azad University, Tehran, Iran
ar_bagheri@aut.ac.ir

Abstract

Motion and locomotion planning have a wide area of usage in different fields. Locomotion planning with premade character animations has been highly noticed in recent years. Reinforcement Learning presents promising ways to create motion planners using premade character animations. Although RL-based motion planners offer great ways to control character animations but they have some problems that make them hard to be used in practice, including high dimensionality and environment dependency. In this paper we present a motion planner which can fulfill its motion tasks by selecting its best animation sequences in different environments without any previous knowledge of the environment. We combined reinforcement learning with a fuzzy motion planer to fulfill motion tasks. The fuzzy control system commands the agent to seek the goal in environment and avoid obstacles and based on these commands, the agent select its best animation sequences. The motion planner is taught through a reinforcement learning process to find optimal policy for selecting its best animation sequences. To validate our motion planner's performance, we implemented our method and compared it with a pure RL-based motion planner.

Keywords: *Reinforcement Learning, Fuzzy Control System, Motion Planning, Character Animation, Locomotion Planning*.

1. Introduction

With progression of computer hardware, its memory capacity and processing power, 3D applications could be capable of having several different character animations. For example a sports simulation application needs many character animations to become similar to real sports. Organizing several animations needs a lot of work and it is not possible to manually manage it. The simulations which have to work with character animations, usually need motion planning. For this reason, in the recent years, motion planning has been highly noticed in the field of character animation. Motion planning by using premade character animations has its own challenges and it is a little different from robot motion planning because in this type of problem, selecting the best sequence of premade character animations is important to plan a task.

Several works have been done in this field by applying reinforcement learning. RL-based motion planners can find the near optimal sequences of animations for doing each motion task. In an application which owns huge amount of character animations, a manual setup of different animations to fulfill a motion task is nearly impossible. One of the reasons that reinforcement learning based motion planners are noticed is that they can find near optimal sequences of animations for a motion task in an automated process and without manual work but they suffer from some problems that make them hard to be used in practice. One of the problems is the difficulty of defining suitable state representation. A RL-based motion planner needs to have a good knowledge of its working environment. It works with discrete states and needs to sample its states in an environment. The complexity of environment can affect its state representation and count. Designing state representation could be a challenging task while simple state representations can lead the motion planner to not take optimal and correct decisions and too

* Corresponding Author

many parameters on each state can cause high dimensionality problem which causes huge amount of states which is not practical in the case of memory usage and also it is very hard to manipulate [2]. Also this kind of motion planners are usually dependent of the shape of the environment and obstacles within it, so an RL-based motion planner which learned to avoid obstacles and reach a goal in a specific environment can not necessarily avoid obstacles and fulfill its task in a different environment.

On the other hand there exists fuzzy motion planners which are environment independent and can avoid obstacles and reach goals in different environments without facing high degree of complexity or high dimensionality. In this paper we present a method in which a fuzzy motion planner is combined with a RL-based animation controller to let an automated agent navigate through different environments and reach its goal without colliding obstacles. Our Motion planner owns a low dimensional state space. We use a fuzzy motion planner to command an agent to navigate in environment and avoid obstacles and beside it we design a RL-based animation controller to return the best sequence of animation in response of the fuzzy motion planner navigation commands.

The rest of the paper is organized as follows. In the second section we consider the related works. In the third section we present our method. In the fourth section we compare our experimental results with a pure RL-based motion planner and at the end conclusion and future works are given.

2. Related Works

In the field of motion planning with premade character animations the challenging part is to find an optimal sequence of animations to fulfill a motion task. The selected animations are going to be played sequentially so an important thing is that the consecutive animations should not be very different visually so if they are played after each other, no jump in poses or speed of joints be seen. A solution to find similar motions in a motion database was introduced in motion graphs [8]. In motion graphs all of the animations in the database are seen as sequence of poses. Each pose is a snapshot of the character joint transforms in a specific time. The graph is constructed by finding similar poses in a time range specified by user. An edge is drawn between two most similar poses existing in the time range. Going from one animation to another is done by navigating through the graph. Motion graphs are good at finding similar motions but as [4] showed there is a significant delay in switching between animations and it is not suitable for real-time environments which most of the 3D simulation and video games are among. The delay is because, the graph should meet a specific pose to make a transition from one animation to the other.

Parametric animation blending [12, 6] provides a continuous motion space using almost similar animations. The method ensures that by changing parameters, user gets a desired motion. This method is widely used in 3D character animation simulations and video games. [9] Introduced a motion graph over parametric animations while each node represents sampled parameters which returns a pose from different animations belonging to a parametric group, however parametric motion graphs like motion graphs have delay in changing animations and are not suitable for real-time environments. Since motion graphs can provide smooth transitions between animations many works in the field of motion planning have been done based on them. Reinforcement learning based motion planners also used motion graphs to select and weight graph edges to fulfill their motion tasks [2, 13, 14].

Reinforcement learning finds its way to the field of character animation quickly. [10] Trained an automated agent to fight automatically and select its best animations in a simulated boxing match. User control is an important part of 3D simulations and video games. [5] Used reinforcement learning to teach agents to respond to user control by selecting its best suited animations. [1] Introduced a method named motion fields to respond to user control. The method samples different poses of animations existing in motion database and named them motion states. Each motion state is connected to a list of other motion states which have similar poses and speed. It uses reinforcement learning to select sequences of motion states and their corresponding animation parts to fulfill motion tasks. The method responds fast to user control and returns suitable animations but it needs many animations to work well [11].

There are other motion planners that use reinforcement learning to avoid obstacles and seek goals. [4] Introduced a new motion blending technique to avoid animation foot skating and on top of it, reinforcement learning is used to teach agents to avoid obstacles and reaching goals. [3] Used regression trees and parametric animations to improve the performance of the method represented in [4] on both goal reaching and obstacle avoidance and also on grasping objects. Although RL-Based motion planers work well in selecting best animations to fulfill motion tasks but they suffer from the curse of high dimensionality in their state representation and this makes them hard to be used in practice. They can work very well in a specific environment but they can not necessarily work in different environments with different arrangement of obstacles and

the learning phase should be repeated for each different environment.

Our method uses a fuzzy motion planner to coordinate agents in different environments. Fuzzy motion planners are environment independent and do not suffer from curse of high dimensionality however they possibly cannot find the optimal path to the goal but they can find their way to the goal in different environments with different obstacles without any previous knowledge of environment. In our method the fuzzy motion planner commands the agent to navigate through the environment and controls its speed and direction. Based on the navigation commands comes out from the fuzzy system, we used a method almost similar to motion fields to teach the agent to select its best actions (animations) so it can fulfill the navigation tasks.

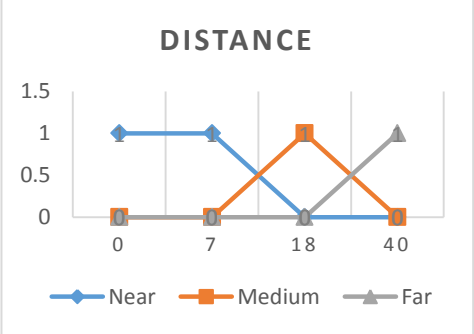

Fig1 (a) Agent's distance to obstacles

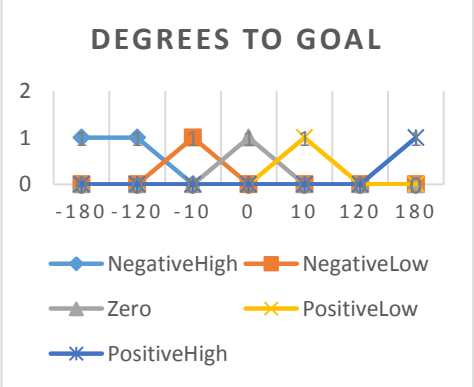

Fig1 (b) Agent's angle difference to goal in degrees

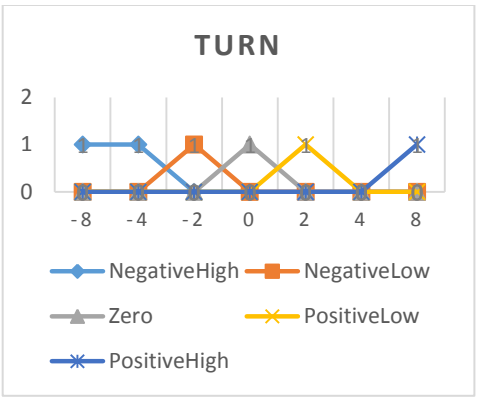

Fig1 (c) Agent's turn value in degrees

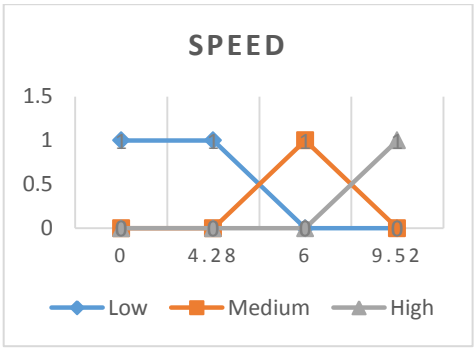

Fig1 (d) Agent's speed in m/s

Fig1. Fuzzy diagrams used in the fuzzy motion planner.

3. Proposed Method

In this section we describe our method in detail. Firstly we show how we use a fuzzy control system to control the agent's navigation and then we show how we designed a RL-based animation controller to react well to navigation commands who comes out of from the fuzzy control system.

3.1 The Designed Fuzzy System

We used a layered fuzzy control system [7, 17] which can avoid obstacles and reach goal in a 2D environment. The fuzzy system controls both speed and direction of the agent. Fig1 shows fuzzy sets belonging to our designed fuzzy control system. The direction is controlled by two layers and the speed is controlled by just one layer. For the direction, the first layer carries out the goal seeking behavior which leads the agent to reach the goal and the second layer is the obstacle avoidance layer which carries out the rules to avoid obstacles. The goal seeking layer is masked partially or completely with respect to the agent's distance to its surrounding obstacles. Since our fuzzy system can compute the obstacles oriented bounding box,

it can avoid any obstacles with any shape and size, however it is better to use obstacles convex hull to avoid them more accurately. The fuzzy system ensures the agent to reach the goal, although it possibly cannot find the optimal and shortest path to the goal but it does not need to know any previous knowledge about the environment and it can find its way through the goal in different environments with different shaped obstacles. The environment independency of our fuzzy control system helps us to create a low dimensional RL-based animation controller on top of our animation database to respond near optimally to the navigation commands comes out from the fuzzy system.

3.2 The RL-Based Animation Controller

The main goal of our research is to bring a motion planner which can reach desired goals by selecting a suitable sequence of its different animations with a low dimensional state space. To reach this goal we designed an animation controller on top of our animation database.

Through our animation database, we select sub animation parts and call them animation intervals. Each animation interval consists of consecutive poses which are playing sequentially in time. These animation intervals are the points that RL-based animation controller can accept navigation commands from the fuzzy control system and make transitions to other animations if needed. For each animation interval, we find similar animation intervals from the animation database. Since we are working on locomotion planning and navigation, most of our animations are related to human locomotion. For each animation, we select intervals manually and we usually select the times in which one foot is planted on the ground or at some points we select double support phase of walking animations where two feet are planted on the ground. Selecting animation intervals in which at least one foot is planted on the ground help us to avoid foot skating while switching between two animations.

For each animation interval we select K nearest neighbors of it by finding the most similar intervals from our animation database. Each animation interval can just switch between its neighbors to make an animation sequence. The neighbors are selected in a preprocessing phase before learning process starts. We need to define a similarity rule for animation intervals so we can apply it to KNN algorithm to find the K most similar neighbors of each animation interval.

3.2.1 Defining Similarity Values

Character animation is based on a hierarchical structure of joints called skeleton. These joints are just containers for 3D transforms. In our research, the root joint has both rotation and translation and the other joints just carry out rotations and no joint has uniform or non-uniform scale. Each joint rotation is represented relative to a constant transform called binding pose. The transform values of joints of a skeleton in a specific time is called a pose of a skeleton. We can find the similarity value of two different poses by using equation (1):

$$PoseDifference(Pose_1, Pose_2) = w_{root} * |p_{root} - p'_{root}| + \sum_{i=1}^{n} w_i * |q_i * (q'_i)^{-1}| \quad (1)$$

Where p_{root} and p'_{root} indicates positions of the root joints of two poses which are being compared. $q_i$and q'_i indicates the rotation of ith joint of the skeleton in form of unit quaternions. We use unit quaternions because they can represent better pose interpolation with less calculations in comparison to other rotation representation methods like Euler angles [16] . w_{root} and w_i shows arbitrary weights of the different joints. Since some joints have more effect on the motion of the skeleton, all joints should not have same weights. For example when the root joint moves, all the hierarchy moves and rotates with it. In overall when a joint moves or rotates, its children move or rotate with it and they save their current transform in their parent space so a joint with higher level in the hierarchy can have higher weight than the others. One other parameter which could have effect on setting weights for a joint in equation (1) is the length of the joints. The larger bone could be seen easier so they need higher weights too.

As mentioned in section 3.2, for each animation interval, we need to find its KNN. Equation (1) shows the similarity value of just two poses, but an animation interval consists of series of poses which are playing sequentially in a time interval. So we need to compare a series of consecutive poses to find an accurate similarity value. Equation (2) shows similarity value of two intervals:

$$IntervalSimilarityValue = \sum_{i=1}^{n} PoseDifference(Interval_{\frac{NT}{i}}, Interval'_{\frac{NT}{i}}) \quad (2)$$

Where NT indicates the normalized time of two intervals. $Interval_{\frac{NT}{i}}$ shows the source animation interval pose at its time $\frac{NT}{i}$ and $Interval'_{\frac{NT}{i}}$ shows the destination

animation interval pose at its time $\frac{NT}{i}$. Intervals may have different time lengths. For example one could be 0.2 second and the other could be 0.4 second. We normalize the time lengths so we can compare both animation intervals in an equal time space. The animation interval is sampled 'n' times and we compare these samples with each other. As we use animations which indicates human motion, there are less differences between consecutive poses of an animation interval and the motions are continuous so sampling animation intervals can give us a good overview of the animation poses. Lower value of IntervalSimilarityValue shows more similar poses.

To switch between two animation intervals, we use cross fade animation blending. By using cross fade animation blending, weight of source animation will go to 0 from 1 and at the same time, the weight of the destination animation goes to 1 from 0 both in a same predefined transition time length. This kind of transitional blending between two animations helps us to achieve faster responses while changing animations. By using cross fade transitional blending, there is no need to wait to meet a special frame to switch between two animations and at the start of the animation interval a cross fade can be made. This can make our work very responsive to animation switching which is crucial in real time simulations and video games. For each K neighbors of an animation interval we assign a weight which shows its degree of similarity value with respect to the other neighbors. Equation (3) shows weight value of each neighbor:

$$W_n = 1 - \frac{IntervalSimilarityValue_n}{\sum_{i=1}^{k} IntervalSimilarityValue_i} \quad where\ 1 \le i \le k \qquad (3)$$

Where W_n represents the weight of nth neighbor of a specific animation interval.

3.2.2 Control Process

Now each animation interval has k another animation intervals which can make transition to. We teach the agent to select a sequence of animation intervals to make near optimal decisions to fulfill its different navigation tasks. We use reinforcement learning to teach our agent to find an optimal policy which let it to select its best animations. Reinforcement learning problems are mostly modeled in a Markov decision process (MDP) which is shown by a tuple with four elements (S, A, P, R):

1. *S*: Set of available states for agent.
2. *A*: An action space depicting all of the possible actions the agent can take.
3. $P: S \times A \times S \rightarrow [0,1]$ Which is a stochastic transition function showing the probability of transitioning to state 's'' from state 's', by selecting action 'a'.
4. $R: S \times A \times S \rightarrow R$ Which is a reward function showing the feedback the agent achieves from environment after transitioning from state 's' to state 's'' by selecting action 'a'.

In our RL-based animation controller, states consist of an animation interval that is playing plus some parameters that show how the motion task is fulfilling. For example these parameters could be the difference of agent direction to the desired direction, or the difference of the agent speed and the desired speed.

Each state has K actions which leads the agent to make a transition to one of its k nearest neighbors. In MDP, the stochastic transition function can make a transition to another state by selecting an action with a probability of Pn. So if there exists m states, by selecting each action, m transition probabilities exists which sum of P1 to Pm should be 1. In our RL-based animation controller by selecting action 'a' from state 's' the agent goes to a specific state 's'' with a probability of 100% and no stochastic function presented in our work.

The reward function shows how much the agent was successful in fulfilling his motion task but this is not sufficient for our work. The beauty and continuity of the successive animations that are going to be played after each other are important too, so in the reward function we apply the weight parameter that is calculated in equation (3). The reward function is a weighted sum of similarity value and the success value of fulfilling desired motion task. Equation (4) shows the reward function:

$$R(s, a_i) = \beta * w_i + \delta * T(s); \ \ where\ 1 \le i \le k \qquad (4)$$

Where $R(s, a_i)$ shows the reward that agent obtains by selecting action a_i in state 's'. β and δ are tunable parameters showing weights of w_i and T(s). w_i is the similarity weight given by equation (3) and T is a function returning the success value of the selected action in fulfilling its motion task.

For the learning process we select some predefined tasks based on the navigation commands coming out from the fuzzy control system. The navigation commands are dealing with agent direction and speed so we sampled the direction and speed parameters as recognized tasks for our animation controller. MDP has a discrete state space, so during learning phase, different states are being sampled. The sampled states are consist of the defined state parameters and the animation interval. The learning

process occurs in the framework of SARSA algorithm [15]. The state action values are calculated based on equation (5).

$$Q(s_{t-1}, a_{t-1}) = (1-\alpha)Q(s_{t-1}, a_{t-1}) + a\big(R(s_{t-1}, a_{t-1}) + \gamma Q(s_t, a_t)\big) \quad (5)$$

At each time step 't', the value of a state-action pair $Q(s_{t-1}, a_{t-1})$ is specified. s_{t-1} is the previous state where agent had been and a_{t-1} is the action it took in that state. In our work each time step is when the animation interval is playing. The $R(s_{t-1}, a_{t-1})$ value in the formula is the immediate reward which the agent receives after transitioning from state s_{t-1} to state 's_t' by selecting action 'a_{t-1}' based on the Markov decision process. $0 \leq \gamma < 1$ Is discount factor which shows the influence of future actions on the value of selected action in the current state. This parameter specifies how much the value of each state-action pair is influenced with respect to it its future outcomes. $0 \leq \alpha \leq 1$ is learning rate which shows the influence of agent's previous experiences on rating each state-action pair. To achieve optimal policy, we always select the most valuable ($Q(s_t, a_t)$) in equation (5) to rate $Q(s_{t-1}, a_{t-1})$.

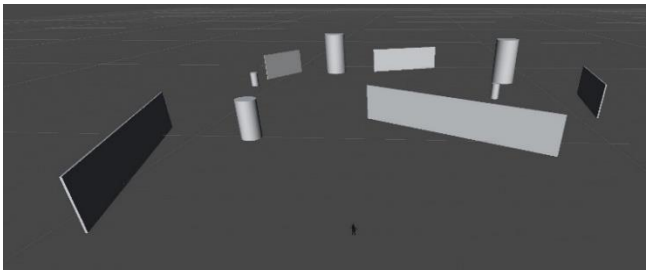

Fig2 (a). Sparse environment

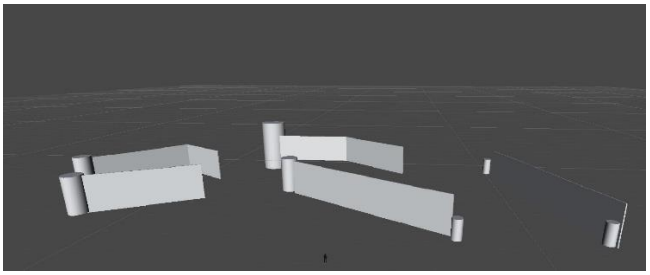

Fig2 (b). Semi cluttered environment

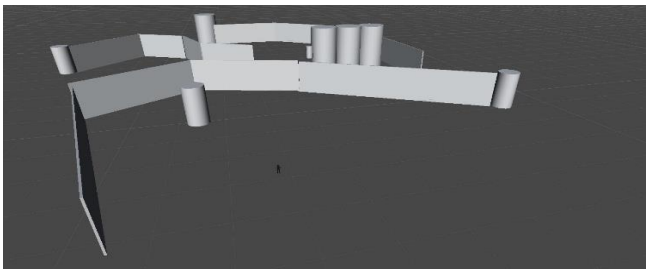

Fig2 (c). Cluttered Environment

Fig2. Different environments with different obstacles. Our method can find the goal in each environment by selecting its best animation sequences with a low dimensional state space and the learning phase does not need to be repeated for different environments but the pure RL-Based motion planners works with high dimensional state spaces and the learning phase should be repeated for each environment.

3.2.3 Near Optimal Decisions

Markov decision process consists of a discrete state space and in learning phase the states are sampled. Except the playing animation interval, the other state parameters the agent needs to work, are continuous and they could take other values than what the agent sampled in its states. To take near optimal decisions at run-time with respect to what agent learned and sampled in its learning phase, we select the state which has the shortest distance to agent's current state parameters from agent's sampled state sets. At run-time, agent can be in a state which is not sampled during learning phase. When this occurs, we find the most similar state to agent's current state from the samples state sets. Since we used a fuzzy control system for agent navigation, we brought out the environment dependency from agent state parameters and we made a low dimensional state space. This low dimensionality can help us to find nearest states to agent's current state with less calculation at run-time. For example our animation controller is just working with character speed and direction.

To select the nearest state to the agent's current state at run-time we separated the sampled states with respect to the animation is playing. The state representation is like S_t = (Animation interval, Current Speed, Current Direction). We organize and save the states who has a specific animation interval in separated tables and assigned a row to each parameter. We find the table which has the same animation interval with agent's current state and from it, we find the state which has the shortest distance to agent's current state and use its most valued action to take the near optimal decision.

4. Experimental Results

We implemented our method to measure its performance and compare it with a pure RL-Based motion planner. The pure RL-Based motion planner is fulfilling its motion tasks just by using reinforcement learning.

As mentioned in section 3.1, we defined a two layered fuzzy control system to compute the direction of the agent and the speed is controlled with respect to agent's distance to obstacles and it's just using one layer. For the direction, the first layer is goal seeking layer and the second one is for obstacle avoidance. The first layer is masked by second layer partially or completely with respect to agent's distance to obstacles. We defined four world sets and different fuzzy sets on these world sets to control both direction and speed of the agent. Fig1 shows the different fuzzy sets we used for our fuzzy control system. As shown in fig1, distance and degrees to goal parameters are the system's input parameters, turn and speed are the system's output parameters.

For our RL-Based animation controller we used 46 different walk, run, sprint and turn animation intervals and we set K to 6 for the KNN algorithm. We defined different motion tasks based on the output of the fuzzy motion planner and started the learning phase with different animation intervals for each task. The motion tasks are different combination of direction and speed values. These combinations are selected manually based on the output of the fuzzy control system. The system samples and rates many states through the SARSA algorithm during learning phase while trying to fulfill its tasks. Our experimental setup is just on the XY plane and is not working with the height of the obstacles. Also all of the animations are moving in the XY plane.
With the same set of animations, we designed a pure RL-Based motion planner which samples the environment to avoid obstacles and reach the goal. The RL based motion planner saves its situation relative to environment within its state parameters. These kind of motion planners should learn to avoid different shaped and sized obstacles separately.

To compare our work with the pure RL-Based motion planner we defined 2 types of obstacles a box and a cylinder, each one with 3 different sizes. Three different environments are also made, the first one is sparse, the second one is more cluttered and the third one is heavily cluttered. Each environment is filled with the different obstacles mentioned above. Fig2 shows the three different environments used in our experimental results.

The defined rules of our fuzzy motion planner can avoid any obstacle with any size and it is independent from the environment density and obstacle arrangement. The learning phase of our RL-Based animation controller occurs with only three state parameters, first the animation interval which is playing, second the agent speed and third the agent direction. The obstacle avoidance rules comes out of the fuzzy system, and animation sequence selection comes out from RL-Based animation controller.

For the pure RL-Based motion planner we should let the agent to go through environments and learn the shape of environment. We just sample the obstacles' relative positions and orientations to the agent if the agent distance is near to them, the agent has to save position and orientation of each obstacle relative to its current transform which leads the agent to own high dimensional states also it should save the shape of the obstacles too so it can avoid them differently. Table 1 shows the state parameters which is used in the pure RL-Based motion planner for learning. As the results show the states become very high dimensional with respect to obstacle variety and count and it is impractical but our method has just 3 state parameters for all environments and it is independent from environment shape and density. Also the learning phase doesn't need to be repeated for different environments. The system is trained once and it can work in different environments with different size and obstacles.

5. Conclusions and Future Works

Fuzzy motion planners present good ways to avoid obstacles in different environments. By combining a fuzzy motion planner and a RL-Based animation controller we achieved a locomotion planner which can seek goals in different shaped environments with a low dimensional state space. Our motion planner is taught once but the pure RL-based motion planners should be taught in each environment separately. Our motion planner might not find the optimal or the shortest path to the goal but it is not environment dependent and it carries low dimensional state space which makes it practical and easy to use.

Table 1- The number of state parameters needed for agent's learning. Our approach can avoid obstacles and reach goal in different shaped environments with just 3 state parameters.

	Pure RL-Based Motion Planner State Parameters	***Our approach***
Environment1	4	3
Environment2	10	3
Environment3	16	3

For future works we want to apply our approach to physically-based animations and compute the blending factors of physically-based animation and premade animations with respect to incoming physical force and torque.

References

[1] Yongjoon Lee, Kevin Wampler, Gilbert Bernstrin, Jovan Popovic, Zoran Popovic, "Motion Fields for Interactive Character Animation", ACM Transactions on Graphics, Volume 29 Issue 6, Article No.138, December 2010.

[2] Wan-Yen Lo, Claude Knaus, Matthias Zwicker, "Learning Motion Controllers with Adaptive Depth Perception", Eurographics/ ACM SIGGRAPH Symposium on Computer Animation, 2012, pp. 145-154.

[3] Wan-Yen Lo, Matthias Zwicker, "Real-Time Planning for Parameterized Human Motion", Eurographics/ ACM SIGGRAPH Symposium on Computer Animation, 2008, pp. 29-38.

[4] Adrien Treuille, Yongjoon Lee, Zoran Popovic, "Near-optimal character control with continuous control", ACM Transactions on Graphics – Proceedings of ACM SIGGRAPH 2007, Volume 26 Issue 3 Article No. 7, July 2007.

[5] James McCann, Nancy Pollard, "Responsive characters from motion fragments", ACM Transactions on Graphics – Proceedings of ACM SGGRAPH, Volume 26, Issue 3, Article No. 6, July 2007.

[6] T.Pejsa, I.S Pandzic, "State of the Art in Example-Based Motion Synthesis for Virtual Characters in Interactive Applications", Computer Graphics Forum, Volume 29, Issue 1, March 2010, pp. 202-226.

[7] Anis Fatmi, Amur Al Yahmadi, Lazhar Khriji, Nouri Masmoudi, "A Fuzzy Logic Based Navigation of a Mobile Robot", World Academy of Science, Engineering and Technology, Vol. 22 , ISSN: 1307-6884 2006, pp. 169-174.

[8] Lucas Kovar, Michael Gleicher, Frederic Pighin, "Motion Graphs", In Proceedings of SIGGRAPH, 2002, pp. 473-482.

[9] Rachel Heck, Michael Gleicher, "Parametric Motion Graphs", Appeared in the ACM SIGGRAPH Symposium on Interactive 3D Graphics conference proceedings, 2007, pp. 129–136.

[10] Jehee Lee, Kang Hoon Lee, "Precomputing avatar behavior from human motion data", In Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2004, pp. 79-87.

[11] Sergey Levine, Jack M.Wang, Alexis Haraux, Zoran Popovic, Vladlen Koltun, "Continuos Character control with low-dimensional embeddings", ACM Transactions on Graphics, Volume 31 Issue 4, Article No. 28, July 2012.

[12] Andrew Feng , Yazhou Huang , Marcelo Kallmann , Ari Shapiro, "An Analysis of Motion Blending Techniques", In Proceedings of the 5th International Conference on Motion in Games (MIG), Rennes, France, 2012, pp. 232-243.

[13] B. J. H. van Basten, A. Egges and R. Geraerts, "Combining path planners and motion graphs", Computer Animation and Virtual Worlds, Volume 22, Issue 1, 2011, pp 59 -78,.

[14] Dan Zong, Chunpeng Li, Shihong Xia, Zhaoqi Wang, "Planning interactive task for intelligent characters", Computer Animation and Virtual Worlds, Volume 23, Issue 6, 2012, pp 547 – 555.

[15] Sutton R. S. and A. G. Barto. "Reinforcement Learning: An Introduction". MIT Press, 1998.

[16] Rick Parent, Computer Animation, Second Edition: Algorithms and Techniques, The Morgan Kaufmann Series in Computer Graphics, 2007.

[17] Jen-Yao Chang, Tsai-Yen Li, "Simulating Virtual Crowd with Fuzzy Logics and Motion Planning for Shape Template", In Proceedings of IEEE International Conference on Cybernetics and Intelligent Systems, 2008, pp. 131 - 136.

Peyman Massoudi received his B.Sc. in computer software engineering from I.A.U. Tehran north branch and received his M.Sc. degree in computer software engineering from I.A.U. science and research branch of Damavand. He has worked on several video game projects for 8 years as animation programmer and technical animator. His research interest includes behavioral-based and physically-based animations.

Alireza Bagheri received his B.S. and M.S. degrees in computer engineering from Sharif University of Technology (SUT) at Tehran. He received his Ph.D. degree in computer science from Amirkabir University of Technology (AUT) at Tehran. Currently he is an assistant professor in the computer engineering and IT department at Amirkabir University of Technology (AUT) at Tehran. His research interests include computational geometry, graph drawing and graph algorithms.