Actuation Strategy of a Virtual Skydiver
Derived by Reinforcement Learning

Anna Clarke * Per-Olof Gutman **

* Technion Autonomous Systems Program, Technion - Israel Institute of Technology, Haifa 32000, Israel (e-mail: anna.clarke@campus.technion.ac.il)
** Faculty of Civil and Environmental Engineering, Technion - Israel Institute of Technology, Haifa 32000, Israel (e-mail: peo@technion.ac.il).

Abstract: An innovative approach of training motor skills involved in human body flight is proposed. Body flight is the art of maneuvering during the free fall stage of skydiving. The key idea is gradually constructing the movement patterns which are the combinations of body degrees-of-freedom that are activated synchronously and proportionally as a single unit, and turning this process into a coaching strategy. The proposed method is iterative: at each skill level an optimal movement pattern is constructed from the basic elements of the current movement repertoire. The free-fall maneuvers of each learning stage can be executed using any one of the basic elements. The construction has two stages: 1. tracking the desired maneuver while the body is actuated by each one of the basic patterns; 2. finding an optimal combination of these patterns to form a new way of body actuation. This hierarchical design resolves stage 2 by Reinforcement Learning with pure exploration and a minimal number of episodes. The method was tested in a Skydiver Simulator and resulted in deriving a movement pattern that showed a superior performance of the studied maneuver. The states and the reward of the Reinforcement Learning algorithm were converted into motor learning aids.

Keywords: autonomous systems, simulators, multiple degrees-of-freedom, reinforcement learning

1. INTRODUCTION

The human body has kinematic redundancy, enabling us to perform a variety of motor activities. It is known that some muscles, segments, and joints are activated in the body synchronously and proportionally, as a single unit. Combinations of such Degrees of Freedom (DOFs) are referred to as movement patterns (MPs). Motor learning is the process during which these MPs emerge (Schmidt and Wrisberg, 2008). First, the MPs are simple (coarse), providing just the basic functionality. As the learning continues, the MPs become more complex (fine), providing adaptation to perturbations and uncertainties, and improved performance (Tani et al., 2014). The objective of this research is to construct MPs involved in body flight - the free-fall stage of skydiving, making it possible to actuate an autonomous skydiver (Clarke and Gutman, 2017), and intended to aid training of novices: a vital and unresolved problem of the skydiving sport.

In sports biomechanics the research on MPs focuses on comparing MPs between experts and amateurs (see e.g. roller skiing in Gloersen et al. (2018), and ice hockey in Robbins et al. (2018)). This identifies what technique elements the amateurs need to change to improve their performance. In some sports simulations of kinematics and dynamics are used to find optimal values for the most important technique elements. For example, for V-style ski jumping the distance can be maximized by finding the optimal angle between the two skis and between the skis and the torso (Seo et al., 2004). Unfortunately, both of these approaches are not applicable for skydiving. In free-fall a great variety of maneuvers is executed by only slight changes in body posture. The same maneuver can be achieved utilizing completely different body DOFs: the MPs highly depend on individual body parameters and type of equipment. Consequently, every experienced skydiver has an individual style not transferable to others, involving multiple equally important DOFs.

Therefore, our method suggests to guide the trainees through the process of developing their individual optimal MPs. Initially, one trains several simple MPs, each involving 1-2 DOFs. Each MP is trained by performing a free-fall maneuver that is easily achievable by this set of MPs. Next, the trainees practice the same maneuver but free to use any combination of the learnt MPs. We hypothesize that the trainee’s body will construct efficient MPs from the previously practiced elements. To accelerate this process, visual cues will be displayed in real time via an augmented reality interface (Clarke and Gutman, 2018). A question is which variables are informative and convertible to visual cues for improvement of the task performance. In order to identify such variables the above learning process is simulated in this work using the skydiving simulator developed in Clarke and Gutman (2017). The text is organized as follows: In section 2 six basic...
MPs are defined and used to perform a given maneuver. In section 3 the performance is improved by combining the basic MPs using reinforcement learning tools. In section 4 an optimal MP for the maneuver under investigation is constructed. Implications of the results on the intended coaching strategy are given in section 5.

2. BASIC MOVEMENT REPERTOIRE

Six basic MPs for turning and side-sliding are proposed, see Fig. 1. The first two MPs involve the internal rotation DOF of the right and left shoulders, respectively. This causes the forearms to press on the airflow and turn to the opposite (of the engaged shoulder) direction. The third and fourth MPs involve the horizontal adduction DOF of the left and right shoulders. This movement causes the upper-arms to press on the airflow and turn/slide in the direction of the engaged shoulder. The fifth and sixth MPs involve the bending DOF of the hip and knee of the right and left legs. This movement will drop the knee down causing the thighs and shins to press on the airflow and turn in the opposite but slide in the same direction of the engaged leg. The first four MPs involve only one single DOF, and the last two leg MPs involve two DOFs each. The eigenvectors for these six basic MPs are:

\[
\begin{array}{c|cccccc}
\text{right forearm} & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 \\
\text{left forearm} & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{right upper arm} & 0 & 1 & 0 & 0 & 0 & 0 \\
\text{left upper arm} & 0 & 0 & 1 & 0 & 0 & 0 \\
\text{right hip} & 0 & 0 & 0 & 1 & 0 & 0 \\
\text{left hip} & 0 & 0 & 0 & 0 & 0.56 & 0 \\
\text{right leg} & 0 & 0 & 0 & 0 & 0 & 0.56 \\
\text{left leg} & 0 & 0 & 0 & 0 & -0.83 & 0 \\
\end{array}
\]

For the tracking purpose three PD controllers were designed:

\[
u = K_P \cdot \text{error} + K_D \cdot \dot{\text{error}}
\]

\[
\text{error} = \tan^{-1}\left(\frac{X_{\text{path}}(t_{\text{LA}}) - X_{\text{pred}}}{Y_{\text{path}}(t_{\text{LA}}) - Y_{\text{pred}}}\right) - \tan^{-1}\left(\frac{V_x}{V_y}\right)
\]

where \(V_x, V_y\) is the current inertial horizontal velocity of the skydiver, \(X_{\text{path}}, Y_{\text{path}}\) is the desired trajectory, \(X_{\text{pred}}, Y_{\text{pred}}\) is the predicted position of the skydiver while the prediction time, and the look ahead time \(t_{\text{LA}}\), are parameters conveying the delay and reaction times. The physical meaning of \(\text{error}\) is the disparity between the current direction of the skydiver and the required direction, assuming that: 1. The same direction will be kept during the ’prediction’ time; 2. It is needed to move towards a point on the desired path located ahead of the skydiver. The look-ahead time represents the time it will take to converge to the desired direction. As found in simulations, typical parameters for skydiving in a belly-to-earth stable pose are: prediction time 1.5 s, and look-ahead time 4.5 s. Large delay and slow reaction time are the reason for using the derivative part \(K_D \cdot \dot{\text{error}}\).

The delay conveys the physics of skydiving: airflow around the moved limb is rearranged in a new pattern, forming new areas of high and low pressure. This changes the aerodynamic forces and moments that induce linear and angular accelerations, which sum up, if the limb retains its new position, generating linear and angular velocities. In turn, these velocities cause the desired change in position and orientation.

The controllers parameters were tuned in simulation:

\[
K_{\text{Parms}} = 0.12 \quad K_{\text{Darms}} = 0.8 \\
K_{\text{Parms}} = 0.3 \quad K_{\text{Dlegs}} = 1.7
\]

which means that all MPs associated with the arms can be controller by using \(K_{\text{Parms}}, K_{\text{Darms}},\) and MPs associated with the legs - by \(K_{\text{Parms}}, K_{\text{Dlegs}}\). Notice that when the controller command is positive (corresponds to a left turn) one of the left-turning MPs is enabled, and for negative commands - one of the right turning MPs is enabled (see Fig. 3). Also, it is assumed that each DOF has a limit on the rate of change of its value: 15 [deg/sec] for DOFs associated with the arms, and 45 [deg/sec] for DOFs associated with the legs.

It is possible to track the trajectory shown in Fig. 2 by the means of each one of the three options: using forearms, upper-arms, or thighs as an aerodynamic surface. However, in each case the position errors and control effort are significant, see Fig. 2, 3. In the next section a learning algorithm is applied in order to combine the basic MPs for improving trajectory tracking accuracy and reducing the control effort.

3. REINFORCEMENT LEARNING

It is desired to learn which combination of the primitive MPs engaged at each instant of time will provide the most accurate trajectory tracking. This problem includes two hierarchical tasks, that we choose to solve separately. The first, high level task is deciding which MPs to engage and quantifying the desired relative effort corresponding to each chosen MP. For example, do 90% of the work by the
system with desired specifications. The actions of the Reinforcement Learning algorithm, i.e., options for $w_i$ combinations, can thus be chosen from these ranges. This way all actions will result in a stable and reasonably accurate trajectory tracking. It remains to learn which options work better than others depending on the situation (state).

3.1 Problem Formulation

The main challenge in solving realistic problems by reinforcement learning methods is formulating the problem in terms of a set of actions and a set of states. In our case, the variables defining the state will be the best candidates for becoming the cues of the training system. The reason is that a successful reinforcement learning process will indicate that the chosen states contain all the information required for describing the situation at each instant of time. Variables used for calculation of the reward should also be used as training cues as they contain information required for choosing the best action in each situation. A conventional application of reinforcement learning assumes simulating the task multiple times and processing hundreds of episodes, while continuously updating the value function. The most frequently adopted policy is choosing the action that maximizes the current estimate of the value function, and once in a while choosing a random action for exploration purposes. In our case, however, it is desired to complete the learning process after a small amount of trials, as each trial means a new parachute jump. Current training methods actually require hundreds of jumps in order to acquire basic skills. The training program we propose aims to reduce this amount by an order of magnitude. Thus, the learning scheme described below uses pure (100%) exploration, meaning that the number of episodes will be equal to the number of actions, while the policy is that a successful reinforcement learning process will indicate that the chosen states contain all the information required for describing the situation at each instant of time. Variables used for calculation of the reward should also be used as training cues as they contain information required for choosing the best action in each situation.

The hierarchical design of the overall system allows to solve each task by the tools best suited and specifically developed for these tasks. Choosing the way of actuating the body is resolved by Reinforcement Learning, and tracking the desired trajectory given the chosen actuators is resolved by Control Theory. Given the plant model (see Clarke and Gutman (2017)) it is possible to determine the ranges of $w_i$ such that the weighted average of commands computed by several linear controllers will provide a stable
defined by three parameters, since these MPs are engaged in pairs: (A,B), (C,D), and (E,F). The following 30 actions are chosen:

\[ w_1 | \begin{array}{cccccccc} 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \end{array} \]

Out of many variables that are related to the current position, orientation, velocity, desired path and motion the following three were found to be most effective states:

1. The mean local radius of the upcoming path segment. The segment starts at the desired path point closest to the current skydiver’s position, and it’s length is 7.5 m. Given three consecutive segment points \((x_1,y_1), (x_2,y_2), \) and \((x_3,y_3)\) spaced at 0.1 m, the local radius \( R \) is computed as follows:

\[ r_1 = \frac{(y_2 - y_1)}{x_2 - x_1}, \quad r_2 = \frac{(y_3 - y_2)}{x_3 - x_2}, \quad \phi = \arctan \left( \frac{r_1 \cdot r_2}{1 + r_1 \cdot r_2} \right) \]

2. Disparity between the desired and actual skydiver’s heading. The desired heading angle \( \Psi_{des} \) is assigned to each path point when the desired path is constructed. For the heading error calculation the desired heading associated with the path point closest to the current skydiver’s position is taken. The actual skydiver’s heading \( \Psi \) is continuously updated by the Skydiver Simulator. The disparity \( H \) is computed as:

\[ H = \Psi_{desired} - \Psi \]

3. Disparity between the desired and actual motion direction. The motion direction is computed from two consecutive desired and actual path points, respectively. The first path point \((X_{des,y_{des}})\) is the one closest to the current skydiver’s position (X,Y). The second desired \((X_{pdes},Y_{pdes})\) and actual \((X_p,Y_p)\) points are the preceding ones. The disparity \( D \) is computed as:

\[ D = \arctan \left( \frac{Y_{des} - Y_{pdes}}{X_{des} - X_{pdes}} \right) - \arctan \left( \frac{Y - Y_p}{X - X_p} \right) \]

Fig. 4 shows these three states during simulations of 30 exploration episodes. Notice that all states are given in their absolute values and are trimmed to certain maximal values. This is preparation for states discretization, summarized in the table below, and resulting in total of 120 states. The range of each of the continuous variables in Fig. 4 is divided into a small number of tiles (n):

<table>
<thead>
<tr>
<th>( R ) [m]</th>
<th>( H ) [deg]</th>
<th>( D ) [deg]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>[3.5, 10.5, 17.5, 24.5]</td>
<td>[2.5, 7.5, 12.5, 17.5, 22.5, 27.5]</td>
<td>[2.5, 7.5, 12.5, 17.5, 22.5]</td>
</tr>
</tbody>
</table>

The states discretization is a step towards developing a simple user interface, as these states will be converted to visual cues.

Expression for the reward that produced best results consists of two parts: a punishment proportional to position error squared, and a constant reward for keeping position error below the threshold of 5 cm:

\[ \text{Reward} = \begin{cases} 3 - (\text{PosE})^2, & |\text{PosE}| < 0.05 \\ - (\text{PosE})^2, & \text{otherwise} \end{cases} \]

where PosE is position error, calculated as the shortest Euclidean distance from the current skydiver’s position to the desired path.

Out of many suitable reinforcement learning algorithms (see Sutton and Barto (2018) for a review) State-Action-Reward-State-Action (SARSA) was chosen. The Q-function or action-value \( Q(s,a) \) is the long-term return (as opposed to the short-term Reward) of the current state \( s \), taking action \( a \) under the current policy. The Q-function at step \( k-1 \) is updated using the Reward and Q-function at step \( k \) as follows:

\[ Q(s_{k-1},a_{k-1}) = Q(s_{k-1},a_{k-1}) + \alpha (R_k - Q(s_{k-1},a_{k-1})) \]

where \( \alpha \) is a learning rate and \( \gamma \) is a discounting factor. The objective is to learn the Q-function (let it converge) and then at each step the action that maximizes the Q-function given the current state can be chosen. It is possible to implement the SARSA algorithm in many variations. In our case, it is advantageous to use SARSA with Linear Function Approximation (LFA), when the Q-function is represented by a linear combination of numerical features \( F_i(s,a) \), \( \ldots \), \( F_n(s,a) \) of the state and the action:

\[ Q(i,s,a) = \sum_{i=1}^{n} \eta_i F_i(s,a) \]

where the weights \( \eta_i \) have to be learned. We define \( n = N_{states}N_{actions} = 120 \cdot 30 = 3600 \) feature functions, while most of them are zero at any given moment. The computation of \( F(i,s,a) \) for state \( i = 1,\ldots, I \), \( N_{states} \) and action \( j = 1,\ldots, J,\ldots, N_{actions} \) where \( I, J \) are the indexes of the state and action at the current moment, is shown in (10), (11).

\[ F(i,s,a) = \begin{cases} e^{-\frac{1}{2}(\Delta_r + \Delta_h + \Delta_d)}, & \text{if } j = J \\ 0, & \text{otherwise} \end{cases} \]

\[ \Delta_r = \left( \frac{R - R_{tiles}(i)}{R_{tiles}(i)} \right)^2 \]

\[ \Delta_h = \left( \frac{H - H_{tiles}(i)}{H_{tiles}(i)} \right)^2 \]

\[ \Delta_d = \left( \frac{D - D_{tiles}(i)}{D_{tiles}(i)} \right)^2 \]

where \( i = (i_3 - 1) \ast N_{rites} + (i_2 - 1) \ast N_{hites} + (i_1 - 1) \ast N_{dites} + 1 \) is the index of state \( s \) and \( i_1, i_2, i_3 \) are the indexes of its three discrete components \( R_{tiles}, H_{tiles}, D_{tiles} \), while the current continuous state is \( R, H, D \); \( \Delta R, \Delta H, \Delta D \) are the tiles’ widths, and \( r, h, d \) are the tuning parameters defining...
the width in each direction of the radial feature function. We used more 'narrow' distributions during exploration \( r = 4, h = 4, d = 4 \) than during exploitation \( r = 0.7, h = 1, d = 1 \). This produced smoother control inputs and less control effort. Radial as opposed to binary feature functions allow a faster convergence of the Q-function, and are similar to the fuzzification stage during fuzzy control design.

Since in our case the number of learning episodes is small and it is desired to increase the efficiency of learning, it is advantageous to combine the SARSA algorithm with an eligibility trace. Eligibility traces are a basic mechanism for temporal credit assignment. When the Reward is computed, only the eligible states-actions pairs are assigned credit or blame for the error, which is weighted according to the eligibility trace. Normally, the state-action pair of the previous simulation step \( s_{k-1}, a_{k-1} \) is considered eligible for the update with the weight of 1, as stated by Eq. 8, and the state-action pairs visited during earlier steps receive fading-out weights. In our case we can utilize our knowledge of the system reaction time: The eligible state-actions pairs will be the ones visited \( t_e = [0.5833, 0.5667, 0.55, 0.5333] \) seconds ago with weights \( W_e = [1, 0.7, 0.4, 0.1] \) accordingly.

Due to the small number of episodes we introduce one more modification to the conventional SARSA algorithm: scaling the updates of the Q-function according to the total number of visits to each state-action pair. Conventional reinforcement learning algorithms assume that the total number of visits to each state-action pair will be sufficient for convergence (infinite for all engineering purposes). In our case, however, it is known that the number of visits to state-action pairs will be small and unequal, i.e. some parts of the Q-function will be updated more often. Scaling introduced below allows to exploit the Q-function that is obtained from a small number of learning episodes before it actually converges.

In summary, our modified version of the SARSA algorithm can be formulated as follows:

**Initialization:**

\[
\begin{align*}
\eta &= \text{zeros}(N_{states}N_{actions}, 1) \\
v_{sa} &= \text{zeros}(N_{states}N_{actions}, 1) \\
\gamma &= \beta = 1
\end{align*}
\]

(12)

where \( v_{sa} \) is the number of visits to an \((s, a)\) pair.

**Simulation Step:**

Compute the vector of features \( F \) for the current state and action (Eq. 11), the immediate Reward (Eq. 7), and the update of \( \eta \) for each of the 4 eligible vectors of features \( \eta_{e(i)} \), i = 1, 2, 3, 4:

\[
\delta = \text{Reward} + \gamma F^T \eta - F^T \eta_{e(i)}
\]

\[
v_{sa, prev} = v_{sa}
\]

\[
v_{sa} = v_{sa} + F_{e(i)} W_e(i)
\]

for each nonzero element \( k \) of \( v_{sa} \):

\[
\eta(k) = \eta(k) v_{sa, prev}(k) + \beta \delta \frac{F_{e(i)}(k) W_e(i)}{v_{sa}(k)}
\]

(14)

The number of Simulation Steps is the number of episodes (30) multiplied by the length of each episode (2820 steps = 47 sec). After this learning process the obtained \( w \) can be used to choose at each instant of time the optimal action:

\[
a = \text{argmax}_a F(s, a)^T \eta
\]

(15)

where the current state \( s \) is composed from its three components \( R, H, D \) and the elements of \( F(s, a) \) are computed according to (11).

### 3.2 Results

A linear combination of three controllers weighted according to the policy stated in (15) produced fruitful results: the trajectory was tracked with the mean position square error below 0.03 [m], and the generated limbs movements were smooth engaging at every instant of time several joints, see Fig. 5, 6.

![Fig. 5. Trajectory tracking during simulations of 30 exploration episodes and one exploitation run](image)

![Fig. 6. Controller weights and limbs movements during the simulation exploiting the learned Q-function](image)

It appears that the learning process resulted in finding an optimal combination of synchronously activated joints in order to solve the desired task - accurately track a particular trajectory. This, however, is the definition of an MP. In other words, it seems that an MP efficient for the given task was constructed in the two design steps described above: 1. designing three controllers that involve three different sets of joints; 2. learning the optimal linear combination of the weights for these controllers.

If the above hypothesis is correct it will be possible to extract this MP from simulations and replace the three controllers and their weighting policy by one controller using the obtained MP for body actuation. We also expect that in the later case the tracking accuracy will be better especially if the task is altered. The reason is that the learned Q-function provides an optimal policy only for the original task. However, if this knowledge is implemented in one MP, a controller driving its input signal will be able to deal with task variations and disturbances.
In the next section we present a simulation of a slightly modified task from which the learned MP is extracted and the above hypothesis is non-falsified.

The main result of this work is justification for the suggested training method. The design process shown in sections 2-4 resembles the natural learning of a motor skill: First, we learn some body movements matching our current skill level. Then, we perform the activity trying out and combining these movements until a new MP emerges. In the presented simulation a new MP, better suited for tracking a given trajectory, has emerged from actuating the body by a weighted combination of primitive MPs, while the weights were found by a reinforcement learning algorithm. Its states and reward can thus be used as visual cues in the training system which we aim to develop next.

REFERENCES