In this paper, we present more complicated and more realistic autonomous virtual creatures (called animated robots: Anibots in this paper) and develop a design tool for them. An animated robot can behave autonomously by using its own sensors and controllers on three-dimensional physically modeled environment. The developed tool enables us not only to design three-dimensional animated robots consists of rigid objects, elastic objects with various joints and controllers, but also to simulate the behaviour of designed animated robots on the physical environment instantly. In order to simulate more realistic world, an approximate fluid environment model is presented, which enables us to reduce the computational costs of simulations. In the developed tool, designed animated robots may have sensors and controllers with neural networks and these controllers can be evolved by means of the genetic algorithm (GA) or the particle swarm optimization (PSO). In this paper, we demonstrate that the salamander model can obtain an autonomous walking behaviors and swimming behaviors on virtual land and water environments. Furthermore, in order to show the validity of the approximate fluid environment, the flight control task of the helicopter model is demonstrated.
2. Animated Robot

An animated robot is a virtual object on three-dimensional physically modeled environment. The animated robot can behave autonomously by controlling its actuators based on its sensor inputs such as the distance and the angle of the light sources, other objects and so on. We can design an animated robot by conjugating some primitive objects such as spheres, cuboids, meshed spheres and meshed cuboids. Figure 1 shows an example of our designed gold beetle and dragonfly robots.

3. Design Tool

3.1. Design of Animated Robots

An animated robot consists of some rigid objects with some joints. The structure of the robot can be stored into the database in the form of graph structures. Nodes in the data structure correspond to the object components such as rigid objects and joints, and links correspond to the joint relation between these objects.

A rigid object is a main component of animated robots. Some primitives such as spheres, cuboids, meshed spheres and meshed cuboids are prepared for designing animated robots. Figure 3 shows an example of various rigid objects.

Particularly, in the process of designing an elastic animated robot, it is very important to know the stability of the robot on the physical environment with gravity, frictions and collisions. By simply switching the mode from “modeling” to “simulation” in the tool and simulating the virtual environment including designed anibots, we can know the behaviors of designed anibots and the stability of it immediately. This feature is novel and can support the modeling process of these virtual creatures.

3.2 Design of Motions

In order to animate designed objects, some kinds of sensors are prepared. In the current version of our developing tool, photo sensors and touch sensors are implemented. However, other sensors such as image sensors, or sonar sensors can be easily implemented.

The photo sensors are attached to rigid objects. Input signals from the sensors are used for motion control.
A motion of an animated robot is created by the rotations of joints, and expansions and contractions of springs. Two types of motion attributes are prepared. The first one is the attribute for a passive motion caused by surrounding forces. The second one is the attribute for an active motion that makes an effect with surrounding forces to control a model. The active motion is controlled by a neural network controller.

The user of the developed design tool can switch the mode between “modeling mode” and “simulation mode” at any time during the design process. In the modeling mode of our designed tool, above explained three-dimensional anibots can be easily designed as well as the typical 3D modeling tools. In the simulation mode, designed objects laid to the physical environment with gravity. We adopt the PhysX engines by NVIDIA Cooperation [7]. Therefore, we can know the behaviors of designed animated robots immediately during the modeling process. It will be very helpful for such kind of design tools. Figure 4 shows a demonstration image of the mode switching. Four kind of objects are designed in the modeling mode (Fig. 4 (a)), and the simulation result is obtained immediately by switching to the simulation mode (Fig. 4 (b)).

![Fig. 4 Design mode (a) and simulation mode (b)](image)

3.3 Design of Physical Environment
As described in the previous subsection, PhysX engine is used for simulating the physical environment. However, it is very difficult to model the fluid effects and compute the force of drag and lift. In particular, the particle method is very time-consuming method, thus it is not appropriate for developing anibots, because many high-speed simulations are required for optimizing their motion controllers. In this paper, an approximate fluid environment is proposed, which can reduce the computational time drastically compared with the particle method.

Suppose that there is a flat board in the uniform fluid environment as shown in Fig. 5. If the flat board moves with the speed $\mathbf{v}$ in the uniform fluid with the velocity $\mathbf{u}$, the force $\mathbf{F}$ of drag to the flat board is calculated in the following equation:

$$F = -\frac{1}{2} \rho A_p C_d v^2 \mathbf{n}, \quad (1)$$

where $\rho$ is the density of the fluid, $A_p$ is the reference area, $C_d$ is the drag coefficient, $\mathbf{n}$ is the unit vector indicating the direction of the velocity (the negative sign indicating the drag is opposite to that of velocity).

![Fig. 5 Force of drag](image)

In order to compute the force of drag more precisely in the case of the flat board moves and rotates, the flat board is divided into the equal size areas and the equation (1) is applied to each area. Also, more complicated meshed object is treated in the same fashion, i.e., each mesh is divided into the small area and is led to compute the force of drag. Note that the drag coefficient is dependent on the size and the shape of the whole object. In general, it is very difficult to determine the precise values of it for any object, thus $C_d$ is set to 1.5 for all objects in this paper.

3.4 Optimization of Motion Control
An animated robot takes the input signals from its sensors and outputs the torques to all controllers (joints) and the behavior of the robot on the physical environment is determined as a result. Of course, the animated robot is affected by other objects, for example, the collisions against other objects. In order to compute these effects, physical engines are very useful.

According to the previous studies, an Artificial Neural Network (ANN) is used for controlling the actuators (joints). However the connection structure of ANN is not limited to the feed forward network in this paper. If random values are assigned to the weight of the neural network, random behaviors are obtained regardless of the input signals.
In order to acquire reasonable behaviors of animated robots, we adopt a genetic algorithm (GA) and a particle swarm optimization (PSO) for optimizing the weight values of each neural controller. Combining ANN and GA (or PSO) is a most promising approach for optimizing this kind of motion control. The obtained behaviors are strongly depending on the predetermined fitness function. The optimization of behaviors will be described in the next section.

4. Autonomous behaviors

In order to make the autonomous behaviors of designed objects more realistic, the neural controllers are optimized so as to maximize the fitness function. We present two demonstrations for acquiring the autonomous behaviors.

4.1 Salamander model

First, we present a salamander model simulation. A salamander model is designed by using our developed modeling tool by one of the authors (see Fig. 6). It consists of 20 meshed primitives, 2 sensors and 13 joint controllers. The density of the model is 300 [kg/m³]. The coefficients of static and dynamic frictions are 0.9 and 0.8, respectively. The restitution coefficient is set to 0.3.

The objective of the salamander model is to reach the light source put on the physical environment. The sensor inputs are the light strength and the difference between the sensor position and the light source. The output torque for each controller is computed in the following equation:

\[ \text{Torque} = R_i \sin(\omega_i t + \phi_i) + B_i \]  \hspace{1cm} (2)

where \( R_i \) is an amplitude, \( \omega_i \) is an angular velocity, \( \phi_i \) is a phase and \( B_i \) is a bias. Although the sine function is used for creating the cyclic forces, it may not be essential in case that the simulation step (time interval of sensor inputs) is relatively small.

In the GA optimization process, some parameters are set to the values as shown in Table 1. A mutation operation is implemented by replacing the values of randomly selected weights to random values. A crossover operation is executed by replacing the output neuron and its all connecting weights with that of randomly selected controllers of randomly selected individuals.

**Table 1 GA Parameters for the salamander model**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>20</td>
</tr>
<tr>
<td>Crossover provability</td>
<td>0.4</td>
</tr>
<tr>
<td>Mutation provability</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Fitness functions are crucial for the evolution process. In this paper, we adopt the weighted sum function consists of distance, energy and direction factors. If the distance \( D \) from the light source to the robot is small during one simulation, the robot obtains better fitness values. If the energy consumption \( E \) is small and if the direction difference \( L \) between sensors and light source is small, the animated robot gets better fitness values. The fitness function is as follows.

\[ f = -aD -bE +cL \quad (a,b,c: \text{constant}) \]  \hspace{1cm} (3)

\[ aD > cL > bE \]  \hspace{1cm} (4)

The direction factor \( (cL) \) is not always required, but it may be necessary for obtaining an animal-like behavior, because the animal tends to direct to the target.

After the evolution with 100 generations, reasonable walking behaviors of salamander model by twisting the body are obtained. Figure 7 shows the snapshot of the evolved walking behaviors of the salamander model. A swimming behavior of the salamander model is also evolved in the water environment (data not shown).
4.2 Helicopter Flight Control

We present the demonstration of the flight control of helicopter model in order to show the validity of the approximate fluid environment presented in section 3.3. The helicopter model as shown in Fig. 8 is inspired by the sketch of Leonard da Vinci. However, it is well known that da Vinci helicopter cannot flight stably.

First, we verify that the flight of da Vinci helicopter is unstable in our approximate fluid environment, and then the revised helicopter model is developed and the flight controller is optimized by using the particle swarm optimization (PSO) [8].

The da Vinci helicopter has the axis with the length 1.2[m] and width 0.1[m]. The wing with 0.6[m] radius winds 1.5 rounds. The density of the axis and the wing is 100.0 [kg/m$^3$] and 85.5 [kg/m$^3$], respectively.

By adding the torque to the axis of the helicopter, the flight simulation is executed. In our simulation result, unstable flight of the da Vinci helicopter is observed (data not shown; c.f. demo video). Although our approximate fluid environment does not consider the force of lift, the affect of wind and the noise, it is shown that our proposed environment model is valid from the above simulation result.

Next, we present the revised simple helicopter model in order to enhance the flight ability of da Vinci helicopter as shown in Fig. 9. The revised model has two propellers that rotate to the opposite directions by giving the torque to the axis. The revised model has the axis with the length 0.6[m] and width 0.1[m]. The wing is the box with the size of 1.8[m]×0.2[m]×(3.0×10$^{-3}$)[m], and fixed to the axis with 35°. The density of the axis and the wing is all 100.0 [kg/m$^3$].

The objective of the flight control is to keep the target altitude. This task is not so difficult, but it is appropriate for verifying that our approximate fluid environment is valid and the optimization method works well.

Artificial neural network (ANN) controllers are used for determining the torque of the axis at time $t$. The sensor inputs of the model are the altitude and the speed to the vertical axis. The number of neurons in hidden layer is 4 and the output of ANN is the torque.

The sigmoid functions $f(u), g(u)$ for hidden layer and output layer are calculated in the followings.

$$ f(u) = \frac{2\alpha_f}{1 + e^{-\beta u}} - \alpha_f $$

$$ g(u) = \frac{2\alpha_g}{1 + e^{-\beta u}} - \alpha_g $$

Particle Swarm Optimization (PSO) is a relatively new collective approach to optimization problems [7]. In order to optimize the torque control for keeping the flight altitude, PSO is applied to the weights of ANN and $\alpha_f, \beta_f, \alpha_g, \beta_g$ in equation (5) and (6).
The population size of PSO is 20 and the number of search (update) iterations is 1000. The value of inertial constant is decreasing from 0.9 to 0.4 linearly and other coefficients are set to 2.0.

The fitness function $f$ is given by the following equations:

$$ f = \sum_{j=1}^{n} V_j $$

$$ V_j = \begin{cases} 
\frac{1}{(1 + |10 - h|)^2} & (h \geq 2) \\
0 & (h < 2) 
\end{cases} $$

(7)

where $h$ is the altitude of the helicopter, and $T$ is the simulation step ($T=1800$).

Figure 10 shows the transition of fitness values in the search iterations. From this figure, we can verify that PSO works well, because the maximum fitness value is increasing as the generation proceeds.

Figure 11 shows the transition of altitude of the helicopter in some typical generations. In generation 15, the descending behavior is observed for the first time. As the generation is proceeding to 1000, an optimal behavior is obtained.

5. Concluding Remarks

In this paper, we present a modeling tool for designing animated robots that can behave autonomously on 3D physical environments.

The main feature of our developing tool is to support users to design anibots by switching between the design mode and simulation mode at any time. Also, an approximate fluid environment is presented in order to simulate the water or the air environments. The neural network of a designed animated robot is optimized and reasonable behaviors of the robot are obtained.

References