TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol.12, No.1, January 2014, pp. 304 ~ 313
DOI: http://dx.doi.org/10.11591/telkomnika.v12i1.3996

Design and Realization of Dynamic Obstacle on URWPSSim2D

Shuqin Li*[1], Xiaohua Yuan[2], Xiao Chen[3]
[1]College of Computer, Beijing Information Science & Technology University, Beijing 100101, China
[2]College of Information, Shanghai Oceanic University, Shanghai 201306, China
[3]College of Computer, Beijing Information Science & Technology University, Beijing 100101, China
*Corresponding author, e-mail: Lishuqin_de@126.com[1], xhyuan@shou.edu.cn[2], 763219676@126.com[3]

Abstract

Simulation system is charged with the strategy validation and dual team meets, and as the 2-dimensional simulation platform for underwater robotic fish game, URWPGSim2D is the assigned platform for Chinese underwater robot contest and Robot cup underwater program. By now on URWPGSim2D, there is only static obstacles, thus short of changeableness. In order to improve the changeableness and innovation of robotic fish contest, to extend the space for the programming of contest strategy, and to increase the interest, this paper study the design of dynamic obstacles on URWPGSim2D, and design and implement two kinds of dynamic obstacles, which are the evadible dynamic obstacle and the forcing dribbling obstacle.

Keywords*: Robotic fish, URWPGSim2D, Dynamic obstacle, robot game*

1. Introduction

Chinese robot contest is a new competition program. Similar with land football which taking the intelligent and bionic robot fish as the members, robot contest carry on some intense against competition. Since in robot contest there are both strong technical challenge and high esthetics, thus it is the perfect combination of research and popular science. In the underwater robot contest, there are competition of global vision teams, autonomic vision teams, and 2-dimensional simulation teams. In the simulation competition, the main programs include the 2 vs. 2 grab the ball, the Synchronized Swimming, the Sri Lanka water ball, 2-dimensional simulation of collaboratively passing through the hole, 2-dimensional simulated polo, 2-dimensional relay with the ball, and et al. [1].

In Chinese underwater robot contest, the assigned platform is URWPGSim2D (Underwater Robot Water Polo Game Simulator 2 Dimension Edition [2], developed by the intelligent control laboratory of Peking university), which can truly simulate the bionic robot fish, in respect of the pose change of all robot joints, the motion changes, and the contest process. By URWPGSim2D, we not only can well conduct the researches on the kinematics theory, collision theory, and on moving strategy algorithm for bionic robot fish, but also more researchers and students can take part in the platform development and in the contest, thus to make their further contributions for the technology of Chinese underwater robot.

Obstacle avoidance is an important content for robot self-protection [3], formation controlling and task fulfillment [4, 5]. In robot simulation, obstackes can be modeled dynamically or statically. By now, in URWPGSim2D, the obstacles are only static ones, that is to say, the color, size and location of the obstacles are fixed. In order to highlight the stereo effect of obstacles, the color of the obstacles is set to linearly gradual change from the obstacle center to its top, where at the start the color is white, at the end that is grey. This is shown in Figure 1. The modeling of static obstacle is relatively simple, which only need to set the coordinate and color of the given vertices. In collision process, the mass of static obstacle is infinite by default, thus if collide with a robot fish or the ball, the location of obstacle will not change [2, 6]. Static obstacles mainly provide regular material, and act as the goal or the bounds, so are indispensability. In order to improve the alterability and innovativeness, and at the same time increase the interest of the contest, this paper designs and implements dynamic obstacles on the original URWPGSim2D.

Received June 24, 2013; Revised August 18, 2013; Accepted September 17, 2013

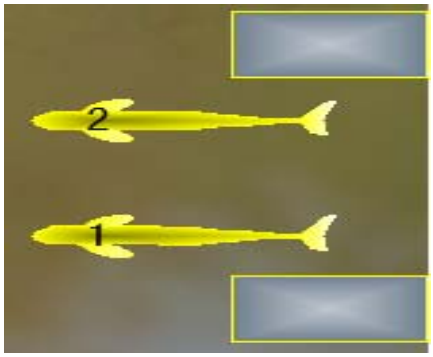

Figure 1. Static obstacle

2. Design Mentality of Dynamic Obstacle

Dynamic obstacles can move around, and their color, size, location, and velocity can be changed. According to their velocity and move track, the obstacles can be divided into random ones and fixed ones. To random dynamic obstacle, we can set its velocity and track, thus obstacle will lack of predict, and thus induce more uncontrollability into contest strategy programming, which will finally make contest more depend on luck. With the fixed dynamic obstacle, the player can correctly control the move parameters of obstacle in order to avoid obstacle, and even take advantage of the obstacle to start an attack, thus can further extend the programming space of contest strategy. Both the size and velocity of dynamic obstacles can be set by the player, thus to increase the variability and portability of obstacles in different contests. In the view of programmers, the dynamic obstacles can be set as evadable ones or forcibly dribbling ones, thus these obstacles can holdback certain behaviors at one hand, and can protect certain target at the other hand. Therefore, this paper studies the designing dynamic obstacles of fixed track and velocity, and designs two kinds of dynamics, which are evadable ones and strong transport ones respectively. We will introduce the relative design as follow.

2.1. Design of Evadable Dynamic Obstacle

Evadible dynamic obstacle is obstacle that robot fish need to avoid from. The distinct characteristic of the evadable dynamic obstacle is that, both the offense and the defense side should avoid the obstacle. Taking the 2v2 ball fight (shown in Figure 2) as the example, in front of the gate, we can set one rectangular obstacle, thus if the offense side want to goal, they must fully study the moving rule of the obstacle, and take advantage of a good chance, thus lest the going ball be push away by the long margin of the obstacle. And similarly, once the goal success, the defense side must fish out the ball and at same time avoid the rectangle obstacles, otherwise, the ball will be push back by the short margin of the obstacle.

Relatively, the defense side has more advantage, since it only need to avoid the short side of margin, and it can push the ball in the scope of obstacle's long margin, then wait the obstacle to push away the ball, thus to fulfill the defense successfully.

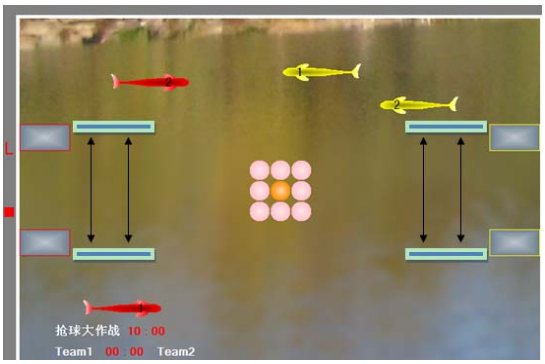

Figure 2. Dynamic obstacle in the ball fight

2.2. Design of Forcibly Dribbling Obstacle

Forcibly dribbling obstacle are those that can be used in the dribbling, and its distinct characteristic is that, between dynamic obstacles there exist some angle, thus it is easy to dribble the ball away, and the direction of dribbling is favor to one side of the contest. As shown in Figure 5, in the contest field of 5vs5 polo, near the top and bottom bounds, some dynamic obstacles with V-like shape are added in. these obstacles periodically and horizontally move from the back to the former field. And taking the V-like obstacles near the top bound, when the obstacle move from the left to the right, the shape of V make that in case the ball move into its moving scope, the ball will get into the corner of V, and be pushed from the back of the left player to the former field of the right player, thus give some convenience for the left player to start a further attack. Similarly, the obstacles near the bottom bound are favor for the right player to start its attack. Besides this, it is worth to mention that of the obstacles, the attack characteristic is higher than the defense. Because of the V-like corner, whenever one ball is clipped, it is difficult to fish it off the inner angle of the obstacle, so it is easy to hold the ball. On the other hand, contrarily, at the back of the V-like obstacle, it is very difficult to hold the ball, and at most only can pop the ball to the direction of defense. So, the left player must do the best to make use of the inner angle of the upper obstacles, thus to start an attack, but not to make use the back of the bottom obstacles for defense.

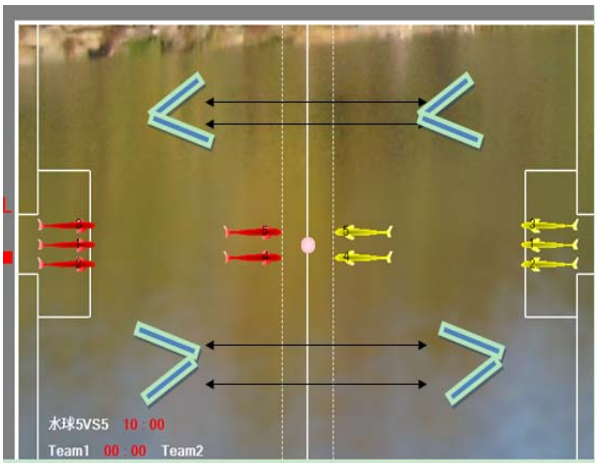

Figure 3. Dynamic obstacles in the 5vs5 Polo

3. Implementation of Dynamic Obstacle

Adding dynamic obstacles will affect many platform modules, such as the collision process module, the game designing module, and the environment painting module.

3.1. Modeling Ideas and Implementation of Dynamic Obstacle

Compared with static obstacle, besides color, quality, coordinate and Vertex, for supporting dynamic movement, in the model of dynamic obstacle there should add in some new properties, for example, indication of moving status, angular velocity, line velocity, circulation period. And at the same time, for the dynamic obstacle we should define constructor method and the dynamic Assignment method related to properties such as velocity, quality, thus for code reusing, object of dynamic obstacle can inherit its farther class, the static dynamic.

We can take two kinds of method to implement the circular movement of dynamic obstacles. The first kind of method is according to the critical values of time, such as simulation period, the angular velocity of switching, and the direction of line velocity. The other kind of method is according to the critical values of coordinate, when obstacle enter into the scope of certain coordinates, the line velocity and angular velocity will be switched at once. When the acceleration is 0, the first kind of method is more convenient and accuracy for motion controlling.

Using the object-orientation technology, we encapsulate dynamic obstacle as class

Rectangular Dynamic, in which we implement two interfaces, which are IC loneable and Idss Serializable, thus to support the serialized communication between objects. In Rectangular Dynamic, properties include the Name, the Indication byte which shows if allow to delete, the Vertex, and Color of one obstacle object. The constructor of Rectangular Dynamic is given below, and for all the parameters, we give detail introduction.

In the implementation of the circular movement, we use one period count function, and use variable CircleTimes to store the period number of one cycle, use TimesCouter as the counter, which is added 1 after one cycle. When it is equal to the predefined CircleTimes, the moving direction of dynamic obstacle will be changed, and Times Couter will be reset to zero, and the next move cycle will be started.

```
/// <summary>
/// the base class of rectangle obstacle in the simulated field
/// </summary>
public class RectangularDynamic : ICloneable, IDssSerializable
{
    /// the name of obstacle
    public string Name;
    /// if the object allow to be deleted
    /// in the initiation, value of IsDeletionAllowed  is assigned as false
    public bool IsDeletionAllowed;
    /// the non-parameter constructor of obstacle
    public RectangularDynamic()
    {
        SetRectangularDynamic();
        for (int i = 0; i < 4; i++)
        {// for initializing the four vertex of the rectangle
            PolygonVertices.Add(new xna.Vector3(0, 0, 0));
        }
    }
    /// the constructor with parameter
    /// </summary>
    /// <param name="strName">name of the rectangle obstacle </param>
    /// <param name="positionMm">location of rectangle obstacle（in the field coordinate, the unit is mm）</param>
    /// <param name="colorBorder">color of rectangle obstacle </param>
    /// <param name="colorFilled">color used to fill the rectangle</param>
    /// <param name="borderWithPix">width of rectangle obstacle(pixel) </param>
    /// <param name="lengthMm">lengthe rectangle obstacle（mm）</param>
    /// <param name="widthMm"> width of rectangle obstacle(mm)</param>
    /// <param name="directionDeg">direction of width of rectangle obstacle（unit is rad）</param>
    /// <param name="velocityMmPs">current velocity (mm/s)</param>
    /// <param name="velocityDirectionRad"> current backup direction(rad∈[−π,π] </param>
    /// <param name="angularVelocityRadPs">current angular velocity( rad/s)</param>
    /// <param name="circleTimes">period number in one cycle</param>
    public RectangularDynamic(string strName, xna.Vector3 positionMm, Color colorBorder,
        Color colorFilled, int borderWithPix, int lengthMm, int widthMm, float directionDeg, float velocityMmPs, float velocityDirectionRad, float angularVelocityRadPs,int circleTimes)
```

```
{
    Name = strName;
    PositionMm = positionMm;
    ColorBorder = colorBorder;
    ColorFilled = colorFilled;
    BorderWidthPix = borderWithPix;
    LengthMm = lengthMm;
    WidthMm = widthMm;
    DirectionRad = xna.MathHelper.ToRadians(directionDeg);
    IsDeletionAllowed = true;   // allowing delete by default
    VelocityMmPs = velocityMmPs;
    VelocityDirectionRad = velocityDirectionRad;
    AngularVelocityRadPs = angularVelocityRadPs;
    CircleTimes = circleTimes;
    TimesCouter = CircleTimes;//initialized value of the counter
}
```

3.2. Realization of Collision Process Module

Collision process is more common in 2-dimensional and 2-dimensional games, nearly all the current popular network game can not do without collision detect. In the virtual world, when reach at a wall or obstacle, the person must stop, not pass through. When two light objects collide with each other, both of the two will be popped away, not pass through each other, and when Deformable objects collide with each other, the result will be different with the collision strength.

For example, the efforts of collision with iron sheet and with sponge sheet are different. And collision at different angle the effect will also be different two. From this we can see that, only after adding collision process, the game can be more really.

The key task in Collision detect is to prevent the pass through between object. Based on computer graphics, combing reasonable time and special detecting algorithm, we can prevent the collision and passing through.

There are many kinds of collision modeling methods we can adopt, of which the main parameters are the shape, material, and accuracy of the modeled object. By now, the always used collision modeling methods are that of spatial decomposition [7], of layered Bounding Volume [8], of bintree [9], and that of quality-spring [10]. In the early year of collision detect, the most usually used method is of spatial decomposition, which represent the object by many cell of equivalent volume [11], which is valid only when the object is equably distributed. When there are more than one object in the field and distances between the object are short, there need to iteratively partition the field into smaller cells, thus will induce more consume of computation time and storage. Lately, along with the development of computer graphics, collision modeling method of layered Bounding Volume obtained broad application, since by this method, no matter the objects are rigid, or with complicated boundaries, using method of layered Bounding Volume both can well enwrap the detected object, and according to requirement of the detection accuracy, it can modify the model compactness to meet platform requirement. Compared with method of spatial decomposition, method of layered Bounding Volume can be implemented more validly, with more less consume of computation resources. Since on the current 2-dimensional robot fish simulation platforms, the environmental information and simulated object include border environment, obstacles, simulation polo, simulated robot fish, and et al. In which, all the first three are regular and ametabolic object, so, using the simple Bounding Volume can not only simulate the collision completely, but improve platform efficiency also.

Bounding Volume takes certain space in the 2-dimensional plane, and can enwrap one or more than one object of irregular shape. The main idea of Bounding Volume is to describe the more complicated object by Bounding Volumes, which of some simple geometrical shape, so, we only need to detect Bounding Volumes. The basic kinds of Bounding Volumes are Sphere(S), AABB (Axis-aligned Bounding Box), OBB (Oriented Bounding Box), K-Dop (K-

direction Discrete Orientation Polytope), Convex Shell (CS) and et al., which are shown in Figure 4.

According to different requirements of modeled object, and by synthetically comparing the expected characteristics of Bounding Volume, we can select one from different Bounding Volume. Here the expected characteristics of Bounding Volume include the following five.

(1) Resources consumption in intersectant test
(2) Simulation compactness
(3) Computational complex
(4) Complicated degree of circumrotation and shape transformation
(5) Occupancy rate of EMS memory

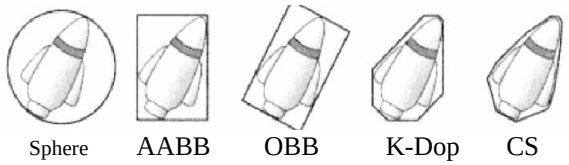

Figure 4. Sketch map of Bounding Volumes [12]

Among Bounding Volumes in the above figure, the complicated degree increases from the left to the right, and there is the same of modeling accuracy. In the five Bounding Volumes, the simplest is Sphere, of which, the stored information is least, the detection and condition is simplest, and thus for fast collision detection. Except Sphere, all the others have certain complication and construction condition, below give further analysis as to the expectation characteristics of other 4 Bounding Volumes.

(1) AABB

In research history of collision detection, usage of AABB is the longest and widest. The definition of AABB is the smallest rectangle, which encloses one given object, and parallel the coordinate axes. It is very easy to construct AABB, in which we only need to record the corner coordinates, thus, to an AABB Bounding Volume, only 6 float data need to be stored, so the computation and diction of this kind of Bounding Volume is easier. Of course, its demerits are clearer as well as, since the compactness of AABB is relatively worse, especially when AABB enclosing one long and narrow object which is located along with the anti-diagonal direction of AABB, at this case, the redundant space is very large, and will lead to a mass of redundant Bounding Volume detection.

(2) OBB

OBB is proposed based on AABB. The main difference between OBB and AABB is whether the axis direction of Bounding Volume can be changed. OBB is the smallest rectangle enclosing object whose relative coordinate axis is at arbitrary direction. Construction of OBB is relative difficult, in which it needs to find out the optimal direction of OBB axis, besides needs to store coordinate extremum, it also needs to record the floor coordinates of axis direction, thus both storage and detection complication are higher. The merit of OBB is that model compactness is improved by sacrificing storage, especially, when model rigid object whose axis is transformable, we only need to modify the axis direction.

(3) K-Dop

K-Dop of one object is a convex wrap which enclose the object, and normal vectors of K-Dop's surfaces are all from one fixed direction. This make the construction difficulty and detection complication is medial, and modeling compactness is better, and the storage increase with K. When object take place rotation, by method of K-Dop it need to update the corner coordinates, just this will induce some computations, but if using some kinds of optimization, the efficiency of K-Dop can be improved largely.

For summary, Table 1 shows the validation of all the basic models by comparing their five expectation characteristics.

Table 1. Comparison of basic Bounding Volumes

Expectation characteristics	Comparison result
Resources consumption in intersect test	K-Dop>OBB> AABB>Sphere
Simulation compactness	K-Dop>OBB> AABB>Sphere
Computational complex	OBB>K-Dop> AABB>Sphere
Complicated degree of rotation and shape transformation	OBB>K-Dop> AABB>Sphere
Occupancy rate of EMS memory	K-Dop>OBB> AABB>Sphere

Abiding by the above analysis, we can model all the objects on the URWPGSim2D, the relative modeling modes are listed in Table 2.

Table 2. Modeling types for object on URWPGSim2D (not including robot fish)

	Polo	Pool border	gate	Circle obstacle	Rectangle obstacle
model characteristic	Sphere dynamic	Line static	AABB static	Sphere static/dynamic	OBB static/dynamic

Since the border of URWPGSim2D is a closed rectangle, and all the object are included in the rectangle, so the border is modeled as 4 lines, which parallel the x and y coordinate axes respectively, thus to make convenience for the later collision detection. By now, on URWPGSim2D, obstacles have shape of circle of rectangle, and are divided into dynamic ones and static ones. To circle obstacles, whether dynamic or static, by Sphere model already can meeting the requirement of collision detection. To dynamic rectangle obstacles, since their axes direction and shape will not change, so by OBB model can perform more accurate collision detection.

On URWPGSim2D, the simulation period is set as 100ms(10-1s), in each period, all the collisions of all the objects must be detected. Since even under ideal condition, there are at least 7 to 8 objects, and besides this, the algorithm is always complicated, thus will induce more loads in URWPGSim2D running.

On the 2-dimensional simulation platform of robotic fish, main objects are included in. Although in actual scene, main collisions can take place at the same time. But in computer processing, the collisions are processed serially, that is to say, at each time slice, only one collision is detected, so, when more than one objects have taken place collision, there need to given the priority order to each collision according to visional effect and platform requirement [13-14]. On URWPGSim2D, in the increasing order of collision priority, the first one is the collision of obstacle with robotic fish of the polo, then the collision between robotic fishes, the one between robot fish and the polo, and at last the one between field boundary and robotic fish and the polo. Concretely, from the view of vision effect, all the objects should locate within the pool, thus the collision between the dynamic object and the pool boundary has the highest priority, so we need to detect it firstly, and contrarily, collision between the dynamic obstacle and other dynamic object has the lowest priority, so we only need to detect it at last. Since adding dynamic obstacle will induce some influence on the collision process module, so the dynamic obstacle should provide some essential parameters to the process module, which describe the current collision status, the circumcircle model (which is the smallest circle that can enwrap the dynamic obstacle), and all the corner of dynamic obstacle.

We define all the parameters as follow.

```
/// the current collision status public CollisionType Collision;
/// the backup of current collision status public CollisionType PreCollision;
/// the parameters of collision detect
/// radius of the circumcircle( the unit is mm) public int CircumcircleRadiusMm;
```

```
/// corner list of obstacle rectangle, in which the order is the top-left, top-right,left-bottom, right-bottom, and all the unit is mm
public List<xna.Vector3> PolygonVertices = new List<xna.Vector3>(4);
```

the length, width, direction, and border thickness are defaultly given out by method Set Rectangular Dynamic (), as an example, the following Set Rectangular Dynamic () constructs a dynamic obstacle of 100 mm length and width, at horizon direction, and of 0 border thickness.

```
/// set default parameters to the dynamic parameter
    public void SetRectangularDynamic()
    {
        LengthMm = 100;         // default lengthe
        WidthMm = 100;          // default width
        DirectionRad = 0;       // default direciton
        BorderWidthPix = 0;     // default border thickness
    }
```

Afer setting the parameters as above, method Calculate Collision Detection Paras () can be called to compute the current circumcircle radius and 4 corners of the dynamic obstacle's rectangle. The detailed code of Calculate Collision Detection Paras () is listed below, in which variable PositionMmis the center of obstacle.

```
/// can compute the collision parameter and is called by constructor of derived class.
    public void CalculateCollisionDetectionParas()
    {
///to calculate circumcircle radius CircumcircleRadiusMm = (int)Math.Sqrt(LengthMm * LengthMm + WidthMm * WidthMm) / 2;
        float sine = (float)Math.Sin(DirectionRad);
        float cosine = (float)Math.Cos(DirectionRad);
/// to calculate all the corner of obstacle's rectangle
        PolygonVertices[0] = new xna.Vector3(PositionMm.X - LengthMm * cosine / 2 + WidthMm * sine / 2, 0, PositionMm.Z - LengthMm * sine / 2 - WidthMm * cosine / 2);
        PolygonVertices[1] = new xna.Vector3(PositionMm.X + LengthMm * cosine / 2 + WidthMm * sine / 2, 0, PositionMm.Z + LengthMm * sine / 2 - WidthMm * cosine / 2);
        PolygonVertices[2] = new xna.Vector3(PositionMm.X + LengthMm * cosine / 2 - WidthMm * sine / 2, 0, PositionMm.Z + LengthMm * sine / 2 + WidthMm * cosine / 2);
        PolygonVertices[3] = new xna.Vector3(PositionMm.X - LengthMm * cosine / 2 - WidthMm * sine / 2, 0, PositionMm.Z - LengthMm * sine / 2 + WidthMm * cosine / 2);
    }
```

3.3. Realization of Obstacle Painting Module

Since dynamic obstacle is always in movement, its location and shape should be updated in each period. The conditions under which all the obstacles related to current simulation mission should be reset to the default location are listed as below:

The current program is restarted;

The simulation mission is changed（the program is switched）; And

The button to reset the contest environment is clicked.

The total painting flow is show in Figure 5.

Where Draw（）is the painting method for dynamic obstacle. In the definition of class dynamic obstacle, different instance can override this method, using different parameter. Draw() is defined as.

```
public void Draw(ref Graphics g)
 {
  DrawHelper.DrawRectangle(ref g, PositionMm, DirectionRad, LengthMm, WidthMm, BorderWidthPix, ColorBorder, ColorFilled);
 }
```

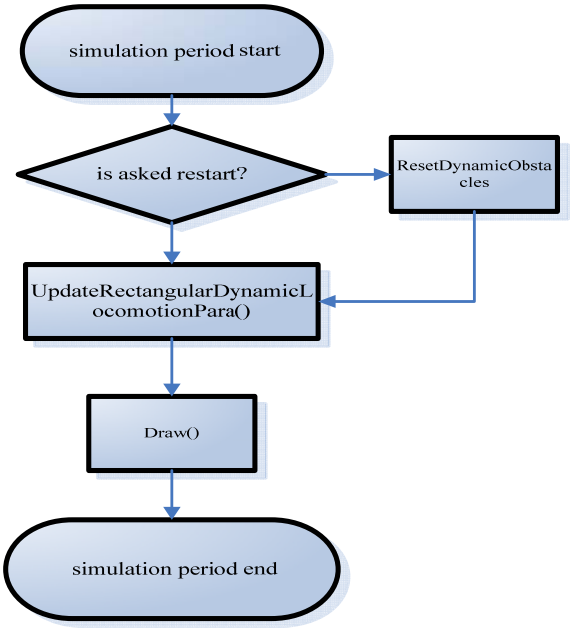

Figure 5. Painting flow of one simulation period

According to the number of simulation period, method Update Rectangular Dynamic Locomotion Para () update the movement parameters of one dynamic obstacle, which include the coordinate of center, the velocity, direction of the velocity, the angular velocity.

The move track screenshot sequence of the dynamic obstacle before the left goat in the ball fight is shown in Figure 6.

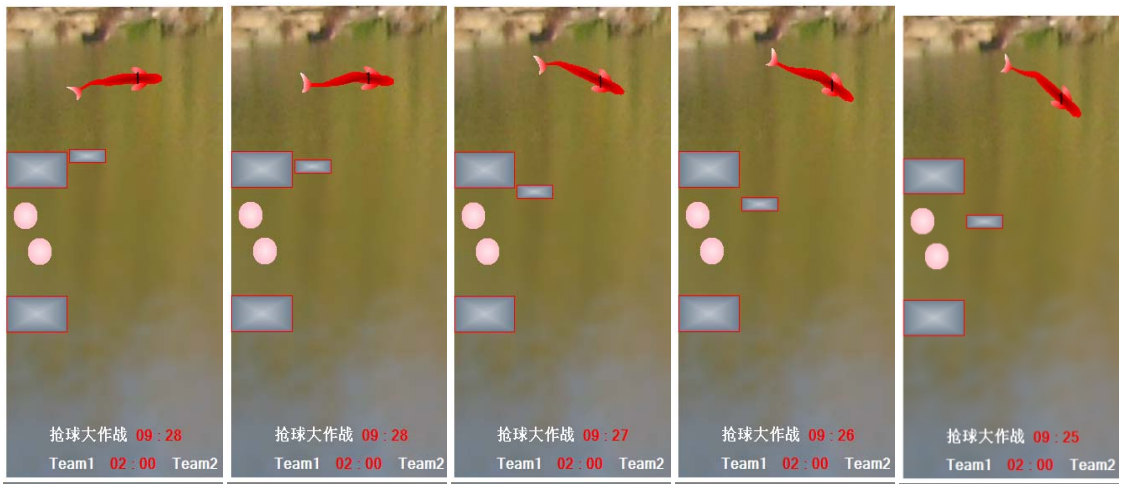

Figure 6. Movement track Screenshot of dynamic obstacle

4. Conlusion

Simulation program of robot fish is one platform for study and validate control theory. In order to increase the programmable space, we need to improve the program and make some innovation. This paper firstly discussed the design ideas for avoid dynamic obstacle and for forcibly dribbling obstacle. Then by the object-oriented ideas, described the modeling process of dynamic obstacles, including the inheritance from static obstacle. Then give and analyze the key code. And at last the realization results are provided to show the validation of the described

design. In the platform test, static obstacle has provides the indispensable environmental materials, and dynamic obstacle has successively increase contest interest and variability.

In order to further increase the variability of obstacle, the future work is to set the shape of obstacle changeable, for example, to take the shape of acaleph, whose size can grow up and shrink. Whether the shape need be changed depends on the requirement, for example, in order to dribble, we can change the shape of obstacle from a 1-like to a C-like one. We can also assign some special effect to the obstacle, for example, when robot fish touch the obstacle, it will slow down, stun, be fouled out or et al. thus to impel robot fish voluntarily avoid the obstacle. And further, we can also set the velocity and track randomly.

Acknowledgment

This paper is jointly sponsored by the Project of Construction of Innovative Teams and Teacher Career Development for Universities and Colleges under Beijing Municipality (IDHT20130519), and by the Funding Project for Graduate Quality Courses Construction of Beijing Information & Science Technology University, and by the Funding Project for Graduate Science and Technology Innovation Projects of Beijing Information & Science Technology University.

References

[1] Youbing Li, Qiong Cai, Penghui Chen, and et al. Design and implementation of the simulation system for underwater bionic robots. *Journal of Central South University (Science and Technology)*. 2011; 42: 551-554.
[2] Youbing Li. Underwater Robot Water Polo Game introduction. Beijing: College of Engineering, Peking University. 2011.
[3] TQiuhong Gao. Application of Multi-Sensors Information Fusion for Self-protection System of Robot. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 11(3): 1619-1625.
[4] Li Yan-dong, Zhu Ling, Sun Ming. Adaptive RBFNN Formation Control of Multi-mobile Robots with Actuator Dynamics. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11(4): 1797-1806.
[5] Yuli Zhang, Xiaoping Ma, Yanzi Miao. Chemical Source Localization using Mobile Robots in Indoor Arena. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11(7): 3718-3727.
[6] Jing Ren, Guangming Xie. Collision Detection on Robot Fish 2D Simulation Platform. *Ordnance Industry Automation*. 2011; 12: 87-90.
[7] ZHOU Yun-bo, YAN Qing-dong, LI Hong-cai. Collision Detection Algorithms Analysis in Virtual Environment. 2006: 103-107.
[8] SUN Xiao-guang, WANG Ming-qiang. Research on collision detection algorithm based on bounding box. *Modern Manufacturing Engineering*. 2009; 4: 122-124.
[9] WC Thibault. Set Operations on Polyhedral Using Binary Space Partitioning Trees. *Computer Graphics*. 1987; (21): 153-162.
[10] X Provot. *Deformation Constraints in a Mass-spring Model to Describe Rigid Cloth Behavior*. Proceedings of Graphics Interface. 1995: 147-154.
[11] MA Deng-wu, YE Wen, LI Ying Survey of Box-based Algorithms for Collision Detection. *Journal of System Simulation*. 2006; (18): 1058-1064.
[12] C Ericson. Real-Time Collision Detection. Morgan Kaufmann Publishing. 2005.
[13] Hua Bao, Shu-qin Li, Qin-qin Guo. Design and realization of synchronized swimming of URWPGSim2D. *Journal of Beijing Information Science and Technology University*. 2011; (05): 84-88
[14] Shu-qin Li, Xia-hua Yuan, Hua Bao. Research on formation control of multi-robotic fish. *Journal of Software*. 2013; 5(3): 349-357