Neural Network와 Robocode를 이용한 동적 객체에 대한 Targeting 기법의 연구

김정훈°, 이지형
성균관대학교 전자전기컴퓨터공학과
yeoshim@gmail.com°, jhlee@ece.skku.ac.kr

A Research of Targeting Technique for Dynamic Objects with Neural Network and Robocode

Jung-hoon Kim°, Jee-hyong Lee
Dept. of Electrical and Computer Engineering, Sungkyunkwan University

요 약

우수한 능력의 인공지능 개체로 구성된 게임은 그렇지 못한 게임에 비해 더 나은 흥미를 사용자에게 제공할 수 있다. 미국 Valve사의 Half-Life, Counter-Strike 및 한국 Dragonfly사의 Special-Force와 같은 실시간 FPS 전투게임에서 상대편에 대한 검색 및 목표 화하는(Targeting) 기법은 인공개체의 전투력에 중요한 하나의 요소이다. 하지만 이 같은 경우의 Targeting은 정적인 대상에 대한 것이 아니라 동적인 대상에 대한 것이므로 단순한 산술 계산으로는 실용적인 효과를 내기 힘들다. 본 논문에서는 Neural Network를 이용한 학습기법을 사용하여 동적인 개체에 대한 효과적인 Targeting기법을 제안한다. 제안한 기법은 매 순간 변화하는 상황정보와 Virtual bullet이라는 가상 미사일 개념을 활용하여 학습 Data를 모델링한 후 Neural Network로 학습시켜 효과적인 Targeting이 가능하도록 구현하였다. 제안한 기법은 Java기반의 탱크전투 시뮬레이션 Framework인 Robocode에 적용하여 그 성능을 평가하였다. 제안된 기법으로 제작된 Robot(Crystal 1.0)은 '2006 Robocode Korea Cup에서 우승을 차지하였다.

1. 서 론

게임 속에 등장하는 인공개체의 인공지능 우수성은 게임의 흥미적 관점에서 중요한 요소 중의 하나이다. Doom, Half-life, Counter-Strike와 같은 실시간 FPS 전투게임에서 등장하는 인공개체들은 여러 가지 특성을 가지고 있으나 FPS라는 게임의 특성상 목표물에 대한 Targeting 기법이 인공개체의 전투력에 중요한 역할을 차지한다.

정적인 개체의 Targeting은 목표 개체의 좌표를 이용한 단순한 산술 계산만으로도 충분히 만족할만한 결과를 얻을 수 있으나 동적인 개체에 대한 Targeting의 경우 매 순간 목적 개체의 상태 값이 변하므로 그 순간순간에 적응적인 결과 값을 낼 수 있는 기법이 있어야만 실용적인 성능을 기대할 수 있다.

본 논문은 이러한 동적인 개체의 효과적인 Targeting을 위해 Neural Network를 이용한 학습을 인공개체에 적용하여 더 나은 인공개체를 구현하기 위한 기술을 연구하는데 그 목적을 둔다.

2. Neural Network와 Robocode

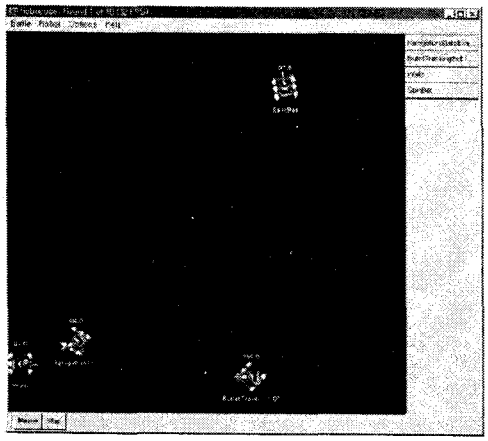
여기에서는 제안된 방법을 실험 및 검증하기 위해 이용한 Robocode Framework와 일반적인 Neural Network 기술에 대하여 설명한다.

2.1. Robocode

Robocode는 IBM Alphaworks의 Mat Nelson이 개발한 Java기반의 Robot(Tank) 전투 시뮬레이션 Framework이다[3]. 그림1에서 알 수 있듯이 Robocode는 그래픽적인 시뮬레이터이기 때문에 탱크들의 행동패턴을 관찰하기에 용이하다. On-line community가 활성화되어 있을 뿐만 아니라 매년 세계 각지에서 경연대회가 개최되므로 다른 프로그래머들과의 경쟁 및 정보공유가 용이하다.

또한, Robocode의 활발한 community 활동, 안정된 구조 그리고 사용의 편의성으로 인해 유전알고리즘(Genetic-Algorithm)을 적용하여 인공개체의 행동패턴을 자동으로 생성해주는 연구[1]에서 시뮬레이터로 활용되었고 Robocode내의 Robot 전투력 강화에 대한 연구[2]도 진행되었다.

Robocode Framework의 전투장에서는 프로그래머에 의해 설계된 Robot과 다른 Robot들은 경쟁하며 전투를 벌인다. Robot은 자신과 상대편의 다양한 정보를 모을 수 있고 이 정보를 기반 하여 대응 행동을 결정한다. 능동적이고 다양한 행동패턴을 생성하여 게임에서 승리하기 위해서는 적절한 행동전략을 수립해야 한다. 또한 상대편들에 대응하는 행동과 에너지의 효율적인 사용도 고려해야 한다.



<그림 1> Robocode 실행화면

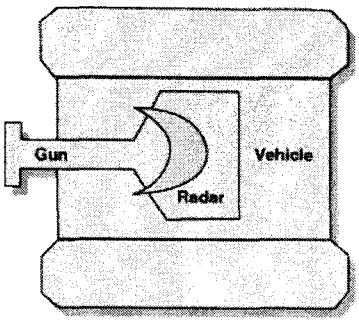
2.1.1. Robot 구성요소 및 특징

Robot은 제한된 에너지를 가지고 있으며 미사일 발사와 상대편과의 충돌발생시 일정량의 에너지가 감소한다. 만약 에너지가 0이 되면 Robot은 파괴되며 상대편 Robot를 맞추었을 때 보상으로 일정량의 에너지가 증가한다.

Robot은 그림 2과 같이 Gun, Radar, Vehicle(Body)로 구성된다.

- Gun
  미사일을 발사하는 장치
- Radar
  다른 Robot의 위치를 감지하는 장치
- Vehicle
  Robot을 이동하는 장치

각 구성요소는 통합적/독립적으로 회전이 가능하며, 기본적으로 Robot의 진행방향으로 정렬되어 있다.



<그림 2> Robot(Tank)의 구성[3]

2.1.2. Event 처리

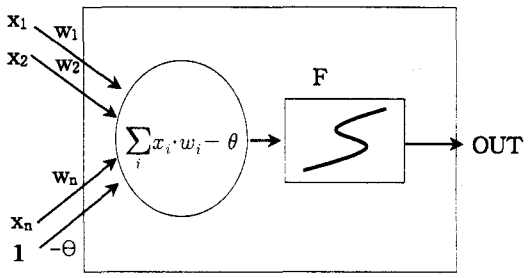
Robot은 Robot의 현재 상태에 따라 다양한 이벤트가 발생하며, 이러한 이벤트를 유용하게 활용함으로서 Robot의 전투 능력을 향상 시킬 수 있다. 다음은 Robot에서 발생하는 Event의 내용을 정리한 것이다.

<표 1> Robot에서 발생하는 Event들

이 벤 트	설 명
onBulletHit()	내가 발사한 총알이 다른 Robot에 맞았을 때 발생한다.
onBulletHitBullet()	내가 발사한 총알이 다른 Robot이 발사한 총알에 맞았을 때 발생한다.
onBulletMissed()	내가 발사한 총알이 빗나갔을 때 발생한다.
onDeath()	내가 죽었을 때 발생한다.
onHitByBullet()	내가 다른 Robot이 발사한 총알을 맞았을 때 발생한다.
onHitRobot()	다른 Robot가 충돌했을 때 발생한다.
onHitWall()	벽에 충돌했을 때 발생한다.
onRobotDeath()	다른 Robot이 죽었을 때 발생한다.
onScannedRobot()	다른 로봇이 탐지되었을 때 발생한다.
onWin()	게임에서 승리했을 때 발생한다.

2.2. Neural Network

신경과학자들의 연구에 의하면 복잡한 인지 작업을 수행하는 각 신경세포는 약 천 개 내지 십만 개의 다른 신경세포와 신경절을 통해 서로 연결되어 정보를 전달한다. 이런 인간 두뇌의 기능을 모방하고자 하는 인공 신경회로망은 노드(node) 혹은 처리요소(processing element : PE)라고 하는 많은 수의 뉴런들이 복잡하게 연결된 망으로 구성된다. 이러한 기능을 수행할 수 있는 인공 신경회로망의 뉴런 구조는 그림 3과 같이 표현할 수 있다.



<그림 3> OUT = $F(\sum_i x_i \cdot w_i - \theta)$

하나의 뉴런은 X_1, X_2, ... , X_n의 n개의 입력을 위한 연결선을 가지며 각 입력 성분은 각각의 연결선에 가해지는 가중치 W_i와 곱해진 후, 이들의 가중치 합이 어느 임계치(threshold) θ이상이면 뉴런은 활성화되어 출력이 된다. 이때 비선형 함수(nonlinear function) F를 수행한 활성화 값이 뉴런의 최종적인 출력 값이 된다. 각 뉴런의 출력은 생물학적 신경세포와 마찬가지로 다른 뉴런들

의 입력이 된다.

뉴런 1개는 단순한 기능을 갖지만 이들이 연결된 신경회로망은 강력한 기능을 수행할 수 있다. 가장 간단한 신경회로망은 입력뉴런들로 구성된 입력 층과 출력뉴런들로 구성된 출력 층으로 구성된 단층 신경회로망(single-layer neural network)으로서 일반적으로 입력층은 단순히 입력벡터들을 전송하는 역할만을 수행한다.

일반적으로 신경회로망이 커지고 복잡해질수록 더 나은 계산 기능을 수행할 수 있게 된다. 신경회로망의 입력층과 출력층사이에 새로운 층들을 추가하는 다층 신경회로망은 대체로 더 복잡한 기능을 수행할 수 있다. 다층 신경회로망에서 입력층과 출력층 사이의 층들은 은닉층(hidden layer) 혹은 중간층(internal layer)이라 부른다.

본 논문에서 제안한 기법에서 이용한 신경망은 3개의 은닉층을 가지는 다층신경망으로써 상대의 정보와 현재의 상황정보를 기반으로 한 6가지 정보를 입력 Data로 가진다.

3. Neural Network를 이용한 Targeting 기법

동적인 목적 개체를 효과적으로 검색 및 목표 화하기(Targeting) 위하여 Neural Network의 학습기능을 활용한다. 이를 위해 현제의 여러 가지 정보와 Virtual Bullet 개념을 이용하여 학습 Data를 모델링 한다. 여기서 이용되는 주요 정보들은 Robocode API에서 제공하는 Heading과 Bearing 개념에 의해 얻어진 값들이다.

Virtual Bullet이란 상대의 정보와 현재의 상황정보를 기반하여 미사일을 발사했다고 가정하는 생각에서 시작한다. 즉 현재의 상황에서 미사일을 발사하면 상대에게 입히는 효과를 기준으로 그 미사일이 어느 정도의 유효성을 가지는지를 판단하고 이 과정을 매 순간 반복하여 유효한 다수의 가상 미사일 군집을 유지 및 관리하는 기법이다. Virtual Bullet에서 유지하는 유효한 가상 미사일 군집, BulletQueue는 매 순간 유효성 검증을 실행하여 유지되는 정보를 갱신하며 이는 곧 학습 Data를 갱신하는 효과를 낸다.

학습되어 나온 결과는 적을 Targeting하는 각도의 보정 값이며 이 값을 단순계산에 의한 결과 값과 유효성 비교를 하여 보다 더 유효하다 판단되는 값을 선택한 후 이전에 Targeting되어 있던 각에 선택한 보정 값을 적용하여 미사일을 발사하게 된다.

3.1. Neural Network

학습을 위한 Neural Network는 3개의 은닉층을 가지는 다층신경망으로써 6개의 입력 값을 가지며 출력 값은 목적 개체를 향하는 각도이다. 각각의 입력 값 들은 다음절에서 설명한다. 전달함수는 아래의 수식으로 보인 비선형 로지스틱 함수를 사용하였으며 학습률 μ값은 0.5로 설정하여 학습시켰다.

$$f(x) = \frac{1}{1+\exp^{-2x}}$$

3.2. 학습 Data Model

Robot은 매 순간 Virtual Bullet을 발사하며 그 순간 감지된 적의 정보를 이용하여 다수의 유효한 Virtual Bullet들을 Queue에 유지한다. 이렇게 유지된 Virtual Bullet과 현재 감지된 여러 가지 상황정보를 Neural Network의 학습 data로 활용하여 적을 Targeting하기 위한 값을 얻어낸다. 그림 4는 학습을 위해 이용하는 Neural Network의 입력 값들로서 Virtual Bullet의 속성 값 들이다.

```
input[0] = vb.distance;
input[1] = vb.hX;
input[2] = vb.vX;
input[3] = vb.aX;
input[4] = vb.bulletVelocity;
input[5] = vb.getBearing(enemyX, enemyY) /
           vb.factor;
```

<그림 4> Neural Network의 입력 값

(1) input[0]: bullet 발사 시 감지한 적까지의 거리
(2) input[1]: 감지된 적과 Robot의 상대 각, 그림 5
(3) input[2]: bullet 발사 시 감지한 적의 속도
(4) input[3]: bullet 발사 시 감지한 적의 가속도
(5) input[4]: 발사된 bullet의 가속도, bullet의 강도에 의해 결정, 그림 6
(6) input[5]: 감지된 적과 자신의 Robot 포신과의 상대 각, factor는 거리에 따른 등급 값, 그림 7

```
double hX = Math.asin(
   Math.sin(sre.getHeadingRadians() -
            sre.getBearingRadians() +
              advRobot.getHeadingRadians()) );

public double getHeadingRadians()
{
   //   Robot이 향하고 있는 각도
   //   0 <= heading < 2*PI
   return heading;
}

public double getBearingRadians()
{
   //   감지된 Robot에 대한 각도
   //   자신 Robot의 heading값과 상대적인 값
   //   -PI < bearing <= PI
   return bearing;
}
```

<그림 5> 감지된 적과 Robot의 상대 각

```
public void setBulletPower(double bulletPower)
{
        BulletPower = bulletPower;
        BulletVelocity = 20 - bulletPower * 3;
}
```

<그림 6> bullet의 가속도

```
public double getBearing(double eX, double eY)
{
   return Utility.getRelativeBearing(
      EnemyHeading, RobotX, RobotY, eX, eY);
}

public static double getRelativeBearing(
            double itemHeading, double originX,
            double originY, double targetX,
            double targetY)
{
   double a = ((Math.atan2(targetX - originX,
    targetY - originY) + Utility.FOUR_QUARTERS)
     % Utility.FOUR_QUARTERS) - itemHeading;
   if(Math.abs(a) > Math.PI){
      a += (a < 0 ? Utility.FOUR_QUARTERS
                  : -Utility.FOUR_QUARTERS);
   }
   return a;
}
```

<그림 7> bullet의 getBearing()

```
double enemyHeading =
 Utility.getHeadingPoints(x, y, enemyX, enemyY);

public static double getHeadingPoints(
   double originX, double originY,
   double targetX, double targetY)
{
  return ( Math.atan2(
            targetX-originX, targetY-originY) +
            Utility.FOUR_QUARTERS) %
         Utility.FOUR_QUARTERS;
}
```

<그림 8> EnemyHeading(적이 향하고 있는 각)

3.3. Heading & Bearing

상대편의 좌표를 예측하여 조준하기 위해 getHeading() method와 getBearing() method를 이용한다. getHeading() method는 현재 Robot(자기 자신)이 향하고 있는 방향을 각도로 알려준다. 그리고 getBearing() method는 레이더에 포착된 상대 Robot에 대한 각도를 자신의 Robot을 기준으로 해서 알려주는 method이다. getHeading() method는 화면 위쪽을 0°, 아래쪽을 180°로 잡은 상태에서 자신의 Robot이 향하고 있는 각도 값을 돌려준다. 만약 자신의 Robot의 앞에 상대 Robot이 있고 자신의 Robot이 현재 60° 방향으로 있다면 getHeading() method의 값은 60이고 getBearing() method의 값은 0으로 enemyBearing이라는 변수는 이 두 가지 method에서 받은 값을 더해서 상대 Robot의 위치를 저장해 두는 변수다. 다시 말해 getHeading() method의 값은 60이고, getBearing() method의 값은 0이므로, 둘을 더한 enemyBearing의 값은 60이 된다. 이 값을 이용하여 다음의 수식을 이용하면 상대편의 좌표를 알아낼 수 있다.

```
double enemyX = getX() + e.getDistance() *
            Math.sin(Math.toRadians(enemyBearing));
double enemyY = getY() + e.getDistance() *
            Math.cos(Math.toRadians(enemyBearing));
```

3.4. 학습 및 보정

매 순간 변화하는 상황에 적응하기 위하여 학습 Data로 이용되는 BulletQueue의 미사일들은 매 순간 그 유효성을 검증받는다. 유효성 검사 시 거리에 따라 설정해 둔 등급 값을 가중치로 활용한다. 유효하다는 판단은 현제 미사일의 위치가 적의 위치와 일정 거리 내에 존재하는지의 유무로 판단한다. 그림9는 유효성 검증의 주요 구현부이다.

학습 후 출력되는 결과 값은 적을 향한 Targeting각도이다. 결과 값을 적용하여 그 값이 유효하다 판단하면 학습 Targeting기법에 일정 값의 점수를 준다. 이때 단순 좌표 계산을 이용한 Targeting기법의 유효성도 검사하여 그 값이 유효하면 역시 일정량의 점수를 준다. 이후 학습과 비 학습 Targeting기법의 점수를 비교하여 높은 점수의 Targeting기법을 적용하여 Targeting 각도를 설정한 후 발사한다.

```
public boolean isNNHit(
   double bearing, double eX, double eY, long t)
{
   double adjustedHeading =
                        EnemyHeading + bearing;
   return   Point2D.distance( eX, eY,
      RobotX + getBulletDistance(t) *
         Math.sin(adjustedHeading),
      RobotY + getBulletDistance(t)  *
         Math.cos(adjustedHeading) ) <= 20;
}
```

<그림 9> 학습된 결과 값의 유효성 판단

4. 실 험

제안한 방법을 증명하기 위해 Robocode의 Sample Robot 중 이동패턴이 판이하게 다르며 다른 Sample Robot들에 비해 승률이 좋은 wall-robot, corner-robot, spinBot 그리고 crazy-robot과 대전을 시켜 다양

한 이동패턴들에 대한 성과를 실험하였고 마지막에는 외국의 사용자가 제작한 오픈소스 Robot과 대전시켜 전체적인 전투력을 측정하였다.

(1) Wall-robot
전투장의 벽을 따라가는 이동패턴을 가진 Robot
(2) Corner-robot
전투장의 각 모서리 부근에서 국지적으로 불규칙적인 이동을 하는 패턴을 가진 Robot
(3) Crazy-robot
어떠한 규칙 없이 마구잡이로 이동하는 패턴을 가진 Robot
(4) SpinBot
작은 원을 그리며 이동하는 패턴을 가진 Robot
(5) BlackPearl
외국의 사용자가 제작한 오픈소스 Robot

4.1. 실험 결과

표 2는 각 Robot들과의 대전결과이다. 각 로봇마다 10Round 전투를 총 10회에 걸쳐 대전시켰으며 그 결과 점수의 평균값을 비교하였다.

<표 2> 실험 결과

상대편 Robot	결과점수(My Robot/상대편)
Corner	1644/26
Wall	1581/30
Crazy	1469/22
SpinBot	1678/16
BlackPearl	899/557

5. 결 론

정적인 개체에 대한 Targeting은 좌표정보를 기반으로 한 단순한 산술계산으로 충분히 실용적인 결과를 기대할 수 있다. 하지만 매 순간 개체의 상황이 변화하는 동적개체에 대한 Targeting은 이러한 단순 산술계산만으로는 실용적인 결과를 기대하기 힘들다.

본 연구에서는 이와 같은 동적개체의 Targeting을 위해 Neural Network를 이용한 학습을 활용하여 더욱 효과적으로 동적개체를 Targeting 할 수 있는 기법을 제안하였다. 제안한 기법은 매 순간 변화하는 동적개체의 상황정보와 Virtual Bullet이라는 개념을 이용하여 학습 Data를 모델링한 후 Neural Network로 학습시키는 방법을 사용하였다. 학습한 결과를 기반으로 한 Targeting 기법은 효과적으로 대상 개체를 Targeting 할 수 있었으며 이를 활용한 인공개체는 더욱 향상된 인공지능을 가질 수 있을 것으로 기대된다.

제안한 기법으로 구현한 Robot(Crystal 1.0)은 '2006 Robocode Korea Cup('06. 07. 12, 한국 IBM본사)에서 우승을 차지하였다.

6. 참 고 문 헌

[1] Jin-Hyuk Hong, Sung-Bae Cho, "Evolution of Emergent Behaviors for Shooting Game Characters in Robocode," IEEE, 2004.

[2] Kazuyuki Kobayashi, Yu Uchida, Kajiro Watanabe, "A study of battle strategy for the Robocode," SICE, August 4-6, 2003.

[3] Sing Li, "Rock'em, sock'em Robocode," 2003. http://www-128.ibm.com/developerworks/library/j-robocode/index.html

[4] Dana Triplett, "AlphaBot: An interview with Robo-code creator Mat Nelson," May, 2002. http://www-128.ibm.com/developerworks/library/j-nelson/index.html