Robocode達人たちが明かす秘訣: 円形方式の標的合わせ

円弧を描いて移動するロボットに完ぺきな精度で命中させる

Alisdair Owens
Student
Student

2002年 5月 日

円形方式の標的合わせは、直線方式の標的合わせを習得した後の次の段階です。少しだけ高度な数学を利用することにより、このシステムでは、円弧を描いて移動するロボットに完ぺきな精度で弾丸を命中させることができます。しかも、直線上を移動するロボットに対する攻撃と同じ効率の良さを保っています。Alisdair Owensは、この技法を実装する方法を説明し、試運転のためのサンプル・ロボットを提供します。

このヒントを読めば、円形方式の標的合わせのしくみを十分に理解することができます。まず、基本的な技法のしくみを説明し、次に、簡単な反復によって精度を劇的に向上させる方法について説明します。さらに、ソース・コードも提供します。このコードは、読者が自分のロボットに容易に組み込むことができます。

標的合わせのしくみ

円弧を描いて移動しているロボットのx方向の変化とy方向の変化を表す疑似コードはかなり簡単で、ラジアンで計算するとすれば、次のようになります。

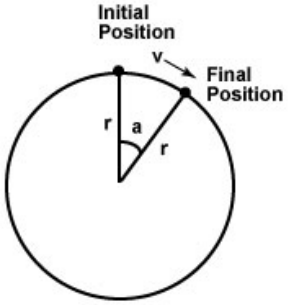
```
x##### = cos(initialheading) * radius - cos(initialheading + changeinheading) * radius
y##### = sin(initialheading + changeinheading) * radius - sin(initialheading) * radius
```

この式で、 initialheading は、最初の位置における敵ロボットの方向、 changeinheading は、我々の弾丸が飛んでいる間に方向が変化する量、 radius は、その上でロボットが移動していると想定する円弧の半径です。

必要なデータを計算する

図1は、私たちが必要とするデータの多くを示しています。 r は、ロボットが移動している円弧の半径です。 a は、方向の変化です。 v は、敵ロボットが移動している速度です。

図1. 円弧を描く移動



敵に正しく標的を合わせるには、いくつかのデータが必要です。それは、ロボットの現在の方向、一動作 (turn) あたりの方向の変化、現在の速度、そして弾丸が命中する時刻です。これらのデータを利用すると、敵ロボットが移動している円弧の半径と、敵ロボットの最終的な方向 (弾丸が敵ロボットに到達するはずの時刻における方向) を計算できます。弾丸の命中位置は、次のようにして計算します。

- **一動作あたりの方向の変化**: この値は、 `headingchangeperturn = (heading2 - heading1)/time` という式で計算できます。この式で、time は、2つの測定値の間の時間です。この下のコードに示されているとおり、結果は正規化する必要もあります。
- **着弾時間**: このシステムの目的の達成のためには、単純な `time = getTime()+(range/(20-(3*firepower)))` という式を使用します。この式で、range は、弾丸を発射した時点での敵ロボットまでの距離、 firepower は、使用するつもりの攻撃パワーです。弾丸が命中する時点で自分のロボットから標的までの距離が変わらないと仮定するのは、かなり無理のある前提ですが、この記事の後半で反復処理を開発するまでの間はそれで我慢します。
- **半径**: この値は、 `radius = velocity/headingchangeperturn` の式から計算します。

コード

リスト1は、円形方式の経路予測に必要なすべてのコードです。ここで注目してほしいのは、標的の方向の変化がごくわずかである場合は、直線方式の標的合わせを使用しなければならないということです。この方法を強制するのは、半径が非常に大きくなった場合にそれを格納するのに使用している `double` 変数がオーバーフローする危険性を軽減するためです。いずれにせよ、方向の変化がごくわずかであれば、実際にそのことを考慮に入れる必要はありません。

リスト1. 円形方式の標的合わせのコード

```
public Point2D.Double guessPosition(long when) {
    /**time is when our scan data was produced.  when is the time
    that we think the bullet will reach the target.  diff is the
    difference between the two **/
    double diff = when - time;
    double newX, newY;
    /**if there is a significant change in heading, use circular
    path prediction**/
    if (Math.abs(changehead) > 0.00001) {
        double radius = speed/changehead;
        double tothead = diff * changehead;
        newY = y + (Math.sin(heading + tothead) * radius) -
                    (Math.sin(heading) * radius);
        newX = x + (Math.cos(heading) * radius) -
                    (Math.cos(heading + tothead) * radius);
    }
    /**if the change in heading is insignificant, use linear
```

```
    path prediction**/
    else {
        newY = y + Math.cos(heading) * speed * diff;
        newX = x + Math.sin(heading) * speed * diff;
    }
    return new Point2D.Double(newX, newY);
}
```

結果を改善する

リスト1のコードを試してみると、スピン・ロボット (円を描いて移動するサンプル・ロボット) に対する攻撃が顕著に改善されることがわかります。しかし、おそらく、発射する弾丸のうち、かなりの数が命中しないことにも気付くでしょう。これは、単に狙いがまずいわけではありません。弾丸が標的に到達するまでの時間の推定値がまずいのです。この技法を改善するには、ごく単純な反復を使用して時間の推定値を計算し、この推定値を標的合わせシステムに渡します。そうすれば、弾丸が標的に到達する時点での、標的から自分までの距離の改善された推定値を取得できます。これを何回か繰り返せば、ほとんど完ぺきな推定値が得られるというわけです。

リスト2. 反復コード

```
/**This function predicts the time of the intersection between the
bullet and the target based on a simple iteration.  It then moves
the gun to the correct angle to fire on the target.**/
void doGun() {
    long time;
    long nextTime;
    Point2D.Double p;
    p = new Point2D.Double(target.x, target.y);
    for (int i = 0; i < 10; i++){
        nextTime =
    (intMath.round((getRange(getX(),getY(),p.x,p.y)/(20-(3*firePower))));
        time = getTime() + nextTime;
        p = target.guessPosition(time);
    }
    /**Turn the gun to the correct angle**/
    double gunOffset = getGunHeadingRadians() -
                (Math.PI/2 - Math.atan2(p.y - getY(), p.x - getX()));
    setTurnGunLeftRadians(normaliseBearing(gunOffset));
}

double normaliseBearing(double ang) {
    if (ang > Math.PI)
        ang -= 2*PI;
    if (ang < -Math.PI)
        ang += 2*Math.PI;
    return ang;
}

public double getrange(double x1,double y1, double x2,double y2) {
    double x = x2-x1;
    double y = y2-y1;
    double h = Math.sqrt( x*x + y*y );
    return h;
}
```

円形方式の標的合わせの性能を改善する

円形方式の標的合わせシステムは、標的合わせシステムとしてほとんど固まっています。つまり、これを改善するための余地は多くありません。しかし、性能を改善できるかもしれない小さな改善方法がいくつかあります。

- **速度の平均をとる**: 計算の根拠として標的の最終速度をとる代わりに、速度の移動平均を出して加速の効果を補正したいと考えるかもしれません。
- **一動作あたりの方向の変化の平均をとる**: 一動作あたりの絶対的な方向の変化の平均を計測すれば、たとえばスピン・ロボットほど軌道が一定でない標的に対して有益な効果を期待できます。

ダウンロード

内容	ファイル名	サイズ
Source code	j-circular.zip	6KB

著者について

Alisdair Owens

Alisdair Owens は、英国サウサンプトン大学のコンピューター・サイエンスの学生です。Java 言語によるプログラミングに2年間携わっており、Robocode に特に関心を寄せています。