Appendix 1 Contents

Appendix 1

Robocode の高等技術　- 円形予測 -

このドキュメントでは、Robocode の高度なテクニックについて解説しています。
インターネット上ではたくさんの高性能なロボットが公開されています。ここで取り上げるのは、そのようなロボットが採用している標準的なテクニックです。
ここでは、円形予測について取り上げています。これは回転運動するロボットを攻撃するためのターゲッテングです。
このサンプルロボットは、STAGE 5 で解説した AdvancedRobot クラスの Droideka に組み込みます。

円形予測

SpinBot や Crazy などのような曲線的に動くロボットにエネルギー弾を命中させるには、円形に動く敵ロボットの予測位置を求める必要があります。そのための計算方法は、それほど難しくありません。

SnippetBot を公開している Alisdair Owens が、IBM developer Works の解説記事でそのためのテクニックを解説しています。ここではこの円形予測機能を Droideka に組み込みます。

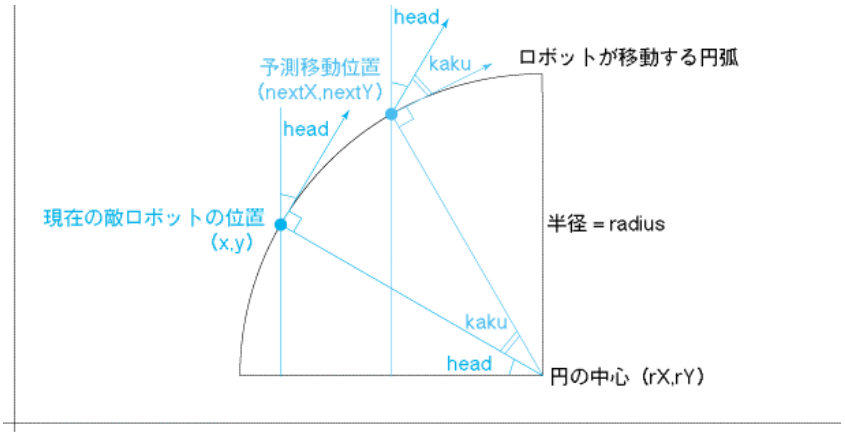
- **「Robocode 達人たちが明かす秘訣：円形方式の標的合わせ」**
  http://www-6.ibm.com/jp/developerWorks/java/020930/j_j-circular.html

■円形に動く敵ロボットの位置計算

敵ロボットが円形に動く場合の位置を計算する方法を考えてみましょう。

まず、敵ロボットが円弧上を移動していると想定します。円弧は、半径＝radius・中心座標（rX, rY）と想定します。ロボットの移動方向を検出すると、どの瞬間でも円弧の接線方向になります。つまり円弧の上を移動しながら、方向を変えていくのです。そのため、ロボットの進行方向と円の中心方向の間の角度は必ず直角になります。

●半径＝radius・中心座標（rX，rY）の円弧上を移動する敵ロボット



現在の敵ロボットの位置を（x,y）、移動角度を head とします。この角度は、バトルフィールドの上方向を基準としたラジアンです。三角形の内角の和は 180 度になるので、円の中心を通る水平線と現在の位置を結ぶ線が作る角度も head になります。

ここから、現在の敵ロボットの位置を求める式は次のようになります。

```
x = rX - radius * Math.cos(head)
y = rY - radius * Math.sin(head)
```

続いて、敵ロボットの予測地点の式をたてます。敵ロボットが円弧上を移動して、（nextX, nextY）の位置まで来たとします。進行方向は、kaku 分だけ変化しています。三角形の内角の和が 180 度なので、円の中心を通る水平線と現在の位置を結ぶ線が作る角度は head + kaku になります。

敵ロボットの予測位置を求める式は、次のようになります。

```
nextX = rX - radius * Math.cos(head + kaku)
nextY = rY - radius * Math.sin(head + kaku)
```

この４つの式を使って、現在位置（x，y）から予想位置（nextX，nextY）を求めると次のようになります。後半の式にある円の中心座標（rX，rY）のところに、前半の式を代入するのです。

```
nextX = x + radius＊Math.cos(head)
      -radius＊Math.cos(head + kaku)
nextY = y + radius＊Math.sin(head)
      -radius＊Math.sin(head + kaku)
```

円の半径（radius）と進行方向の変化分（kaku）がわかれば、この式を使って予測地点を求められます。

■計算に必要な情報

円形予測を行うには、敵ロボットの進行方向がどのくらい変化しているかを調べます。ここから角度変化のスピードと半径を求めます。

進行方向の変化は onScannedRobot()イベントで getHeadingRadians()メソッドを使って調べられます。これを前回の角度と比べるのです。さらに、前回と今回のイベントで、どれだけの時間が経過したかを求めます。そして、角度の変化をその時間間隔で割り算すると、角度変化のスピードが求められます。

進行方向の変化 --------- `e.getHeadingRadians() - target.head`
計測時間の間隔 --------- `getTime() - target.checkTime`
角度変化のスピード ------ **（進行方向の変化）／（計測時間の間隔）**

ロボットの情報を管理するクラスに、角度変化のスピードを格納する chahgehead フィールドを用意しておきましょう。半径を計算するにはラジアンの性質を利用します。ラジアンでは、角度と半径から、次のようにして円弧の長さがわかります。

円弧の長さ ＝ 角度 ＊ 半径

先ほど、角度変化のスピードを求めました。敵ロボットが speed という速度で移動していると、次の式が成り立ちます。

speed ＝ 角度変化のスピード ＊ 半径

この式から、円弧の半径を求められます。

■エネルギー弾の到達予想時間

敵ロボットの進行方向の予測変化角度（kaku）は、次の式で求めます。

kaku ＝ 角度変化のスピード ＊ エネルギー弾の到達時間

角度変化のスピードについては、エネルギー弾の到達予想時間は次の式で求めます。ここでは、敵ロボットとの距離が変化しないと想定しています。これはかなりの単純化になりますが、とりあえずどのくらい効果があるか試してみましょう。

エネルギー弾の速度 ＝ 20 － 3 ＊ power
エネルギー弾の速度 ＝ 17　（power=1 の場合）
到着予想時間 ＝ 敵ロボットまでの距離 / エネルギー弾の速度

■円形予測と直線予測を切り替える

これで、円形予測に必要な情報は揃いました。しかし、このままでは Walls のような直線運動をしているロボットにも円形予測を使ってしまいます。円形予測と直線予測を切り替えるにはどうすればいいでしょうか。

そのためには角度変化のスピードに注目します。角度変化がほとんどない場合は、敵ロボットが直線運動しているのです。次のコードは、予測計算メソッドです。

```
public void setNextXY(long when, double x0, double y0,
                      double balletSpeed) {
    double diff = when - checkTime;
    //double newY, newX;
    //直線予測と円形予測を切り替える
    if (Math.abs(changehead) > 0.00001) {
    //角度の変化が大きいときは、円形予測
        circular(diff);
    }
```

```
   //角度の変化が小さいときは、直線予測
    else {
        liner(diff, x0, y0, balletSpeed);
    }
    //return new Point2D.Double(newX, newY);
}
```

角度変化のスピードである changehead フィールドがほぼゼロに等しい場合には、直線予測を行うため liner()メソッドを呼び出します。ゼロよりも十分に大きいときは、円形予測を行うため circular()メソッドを呼び出します。

■プログラムのソースコード

円形予測を追加したロボットは、Droideka とほぼ同じです。違いは、円形予測を行う点です。Appendix 1 で登場したロボットのサンプルコードは、付録 CD-ROM の次のフォルダに収録してあります。

・付録 CD-ROM の「sample_code」→「Appendix1」フォルダ

・ファイルの内容

Droideka_Circular.java ---- 円形予測を実装した Droideka
Enemy.java ---------- 敵情報を管理するクラス
CircularEnemyRad.java ----- 円形予測を実装した Enemy クラス
Direction.java ---------- 方向管理クラス
Lib.java ---------- 共通処理クラス

■ロボットを評価する

Droideka に円形予測を組み込むと、SpinBot との対戦結果がかなり改善されます。直線予測だけでは 7 勝 3 敗だったのが、10 戦全勝になりました。ただし、SpinBot に対する攻撃がすべて命中しているわけではありません。これは、円形予測の計算で、敵ロボットとの距離が変化しないと仮定しているためです。これを解消するには、求めた予測地点から、到達予想時間を再度計算し、その時間からまた予測時間を計算するというステップを繰り返します。この解決方法は、Alisdair Owens によるオリジナル記事で解説されています。そちらを参考にしてください。