ロボカップ４足サッカーロボットのための ファジィ論理を用いた意思決定エンジン
Decision Engine Based on Fuzzy Logic Rules for Autonomous Robots in Four Legged League

奥山 哲郎　Massayoshi Sugimoto　大崎 嗣豊
Tetsuro Okuyama　Tsugutoyo Osaki
石野　明　篠原　歩
Akira Ishino　Ayumi Shinohara

東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University

Abstract: In this paper, we introduce a decision engine for autonomous robots in RoboCup four-legged robot league. The whole system is a platform which can be used for all programs which need a decision process between numerical entries and a set of actions. Robots are particularly appropriate to use a decision process because they have to choose a motion accordingly to their environment. We have implemented the decision engine in agent players. The rules for the decision engine are quite simple and rapid to construct compared to our previous if-then-rule-based system. In this way, each developer can create and improve behaviors solely.

1　はじめに

近年の科学技術の進展により，様々なタイプのロボットの開発が進められている．特に，自ら状況判断をしながら行動を決定する自律型ロボットは，人間に代わって仕事をする存在として産業分野のみならず，危険作業，災害救助，医療，家事など様々な場面での活躍が期待されている．RoboCup サッカーは人工知能とロボティクスの研究促進を目指し発足されたものである．ロボカップサッカーでは自律型ロボットにサッカーをさせる．そのためにはロボットの設計と制御，物体認識と戦術が重要になってくる．ロボカップでは複雑な戦略をつくり多様な動きをするように各チームで開発を行っている．複雑な戦略をより簡潔に書くために，簡単な行動を実行するためのビヘイビアを作りそれらを組み合わせることで複雑な戦略を作っている [1, 2, 3]．また，Saffiotti ら [4] は階層構造のビヘイビアとファジィ論理を組み合わせた手法を提案しており，ルールを宣言的に書くことができ，さらに追加することも容易なシステムを用いて最適なビヘイビアの選択を行っていた．ロボカップでは短時間に開発を行わなければならないが，C++でコンパイルして実験を行うのは時間がかかる．我々は戦略を Lua 言語を用いて記述することで開発がスムーズに行える．ファジィ関数やファジィルールも Lua で記述し，意思決定エンジンのシステム全体を Lua で記述した．そうすることによって AIBO の中でこのシステムを動かすことができる．本稿では，Lua 言語によるファジィ論理を用いた意志決定エンジンと，ロボカップサッカーへのその適用について述べる．

奥山 哲郎　東北大学情報科学研究科
〒 982-0826 仙台市青葉区荒巻字青葉 6-6-05
Tel:022-795-7132, Fax:022-263-9233,
E-mail:okuyama@shino.ecei.tohoku.ac.jp

2　ファジィ論理

ファジィ論理 [5] は我々の意志決定エンジンの基礎となっている．以下の節ではファジィ論理の概要について述べる．

2.1　ファジィ真理値とファジィ関数

ファジィ真理値とは [0,1] の実数区間で定義される値である．ファジィ論理とブール論理を比べた場合，ファジィ真理値の 0,1 はそれぞれブール論理の *false*,*true* に対応しており，0 と 1 の間のファジィ真理値は真である度合いを表している．ファジィ論理はブール論理の拡張になっているため，2 つは同時に使用することができる．

ファジィ関数について説明する．ファジィ関数は実世界からの入力をファジィ真理値として返す関数である．例えば，関数の入力がロボットからボールまでの距離だった場合，関数が出力するファジィ真理値は実世界上でのボールの近さの度合いを表している．

2.2　ファジィ演算子とファジィルール

ファジィ論理の演算子はブール論理で用いられる演算子と同じであるが，ファジィ論理の場合それらの演算子についての解釈は様々である．我々は and 演算子 (,) と implication 演算子 (<=) のみを用いた．and 演算子は 2 つのファジィ真理値を引数とし，結果として引数の最小値を返す．implication 演算子はファジィ真理値と and 演算子の計算結果の継承を表し，それらの計算結果をアクションの引数として結びつける演算子である．implication 演算の結果，アクションの引数となった値はあるルール下での有効性を表す．つまり引数となった値が大きければ大きいほど，その値を持つアクションは適切なアクションであるといえる．

3　意思決定エンジン

意思決定エンジンとはロボットのアクションを外界の環境に応じて決定するプログラムであり，我々はファジィ論理を利用してこれを作成した．このエンジンはファジィ化エンジン，推論エンジン，実行エンジン，状態遷移機械から成り，図 1 のように表される．また，これらは全てスクリプト言語 Lua を用いて実装した．以下ではこれら意思決定エンジンを構成する要素について述べる．

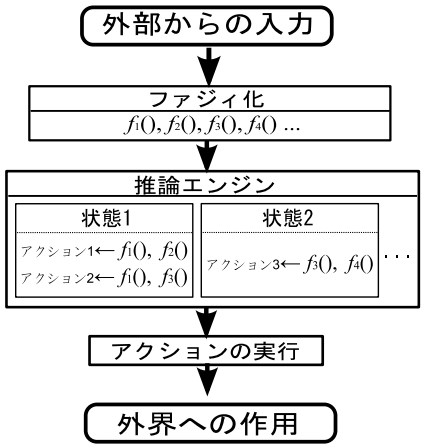

図 1: 意思決定エンジンの概要

```
catchTheBall = Fuzzyficator:new()
catchTheBall:setBase(inCatchRange)
catchTheBall:insertFP("inCatchRange", same)
catchTheBall:insertFP("inNotCatchRange",invert)
```

図 2: 新しいファジィ関数の作成

3.1　ファジィ化エンジン

意思決定エンジンを用いるためには，初めに外部からの様々な入力をファジィ真理値に変換する必要がある．このファジィ化を行うためのファジィ関数を作成するのがファジィ化エンジンである．ファジィ化エンジンにファジィ化したい現実世界の入力とファジィ関数を入力することで，エンジンがそれらを結びつける作業を行う．具体的なファジィ化の操作は，プログラム中で図 2 のように記述する．

3.2　推論エンジン

外界から新しい情報が入力されファジィ真理値が更新された場合，各ルールのスコアは推論エンジンによって更新される．ここでいうスコアとはルールやアクションが適切であるかどうかの指標のことであり，エンジンにより選ばれたファジィ真理値がそのルールやアクションのスコアとなる．この時，適切なアクションを選択するために各ルールが持つスコアの比較を行わなければならない．各ルールのスコアの計算した結果からアクションを決定する処理は推論エンジンで行われる．推論エンジンは全てのルールのスコアを計算し，その中でスコアの一番高いルールが保持しているアクションが選択される．

意思決定エンジンは各パラメータが更新された後，周期的に呼び出され，直前のアクションが終了して

```
Active :
approach <= seeTheBall, inNotCatchRange
search   <= dontSeeTheBall
shoot    <= inCatchRange

Passive :
search   <= dontSeeTheBall
support  <= seeTheBall

Choice :
setActive  <= NearestPlayer
setPassive <= notNearestPlayer
```

図 3: ルールと状態の作成

いる場合のみエンジンによって選ばれたアクションが実行される．アクションはビヘイビアあるいはモーションといったロボットの動作を表す．

3.3 ルール集合と状態遷移機械

意志決定エンジンは状態遷移機械と結びついて動作するよう設計されている．状態遷移機械の各状態はそれぞれ個別のルールの集合を持っている．ロボットの動作中は，意思決定エンジンは必ずどれか一つの状態にいる．意思決定エンジンのファジィ真理値が更新された場合，現在の状態に属するルールに関してのみスコアの更新が行われる．これは計算時間の短縮と，状況に不適切なアクションが実行されない事につながり，またルールを階層的に構築することが可能となる．また，各状態の遷移命令は推論エンジンにおいては 1 つのアクションとして処理される．推論エンジンにより状態遷移を行うアクションが選ばれた場合は，外界に対するアクションは実行されず状態遷移のみが行われ，次のエンジンの更新時は遷移先の状態のルールのみに対しスコアの更新が行われることになる．図 3 は 3 つの状態とそれに属するルール集合の例である．

3.4 ルールコンパイラ

戦略部分の記述を容易にするために，ルールを簡略に記述したスクリプトを Lua プログラムに変換するコンパイラを開発した．コンパイラにより出力された Lua プログラムは AIBO 内で動作する．コンパイラにスクリプトを入力することにより，アクション，ルール，状態の作成と各状態に属するルール集合の作成，初期状態の設定が記述された Lua プログラムが作成される．

```
 -- create action
tempAction = {name="shoot", f=shoot}
a0= Action:new(tempAction)
tempAction = {name="approach", f=approach}
a1= Action:new(tempAction)
tempAction = {name="support", f=support}
a2= Action:new(tempAction)
tempAction = {name="setPassive", f=setPassive}
a3= Action:new(tempAction)
tempAction = {name="setActive", f=setActive}
a4= Action:new(tempAction)
tempAction = {name="search", f=search}
a5= Action:new(tempAction)

 -- create rules
r1=createRule("approach", "seeTheBall",
                "inNotCatchRange")
r2=createRule("search", "dontSeeTheBall")
r3=createRule("shoot", "inCatchRange")
r4=createRule("search", "dontSeeTheBall")
r5=createRule("support", "seeTheBall")
r6=createRule("setActive", "NearestPlayer")
r7=createRule("setPassive", "notNearestPlayer")

 -- create states
Passive = DEState:new()
Passive:insertRule(r4, r5)
Choice = DEState:new()
Choice:insertRule(r6, r7)
Active = DEState:new()
Active:insertRule(r1, r2, r3)
changeState(Active)
```

図 4: 図 3 のスクリプトから作成されるプログラム

このコンパイラの特徴としてコンパイラ自体が Lua で記述されているため，ロボットの内部でスクリプトをコンパイル可能であることが挙げられる．これにより，簡易スクリプトをメモリースティックにコピー，もしくはネットワーク経由で転送するだけでロボットを動作させることができるようになった．この結果プログラム上のミスが減り開発時間が短縮されたため，より戦略の向上に集中する事が可能となった．

3.5 実行エンジン

推論エンジンによりアクションが選ばれた後，最後に実行エンジンによりアクションが実行される．

すべてのアクションは実行用の内部関数を持っており，その関数を用いることで実行エンジンにアクションが渡され，アクションが実行される．

この節で述べたように，意思決定エンジンを用い

ることでロボットの振る舞いを容易に変更することが可能となった．そして，意思決定エンジンに自分の情報だけでなく他のロボットの情報も取り入れることで，試合中の協調動作を可能とした．次は協調動作について説明する．

3.6 マルチエージェント

我々のチームではそれぞれのロボットが独立した意思決定エンジンを持ち，状況に応じた動作を行っている．この意思決定エンジンの動作決定プロセスにおいて，各ロボットに試合中，互いに通信を行わせ，自己位置や自機からボールへの相対距離などの情報を共有することにした．このことにより，推論エンジンが他のロボットからの情報を取り入れて各ルールのスコアを更新できるようになり，より状況に即した行動ができるようになった．

いくつかのファジィ関数は他のロボットからの情報を入力として取るように作成されており，それらが出力するファジィ真理値を基にして協調動作を行う．例えば，各ロボットのボールまでの距離を共有すれば，ボールに一番近いロボットがボールに近づき，他のロボットはそのロボットをサポートすることが可能となる．また，各ロボットの状態の切り替えはそれぞれの意思決定エンジンがオートマトンの状態を切り替えることにより行っている．図 5 にボールが見えている状態での 2 機のロボットのパラメータを示す．この図から，前述したボールまでの距離による各ロボットの状態の切り替えが行われていることがわかる．

```
Ball:
dist_far = 0.10
dist_close = 0.8
pan_straight = 0.74
pan_left = 0.38

seeTheBall = 1

nearest = 1

FP result :
search  = 0
support = 0
turn    = 1

Best result :
turn    = 1

setActive
```

(a) ボールから近い時

```
Ball:
dist_far = 1
pan_straight = 0.09
pan_left     = 0.93

seeTheBall = 1

nearest = 0

FP result :
search  = 0
support = 1
turn    = 0

Best result :
support = 1

setPassive
```

(b) ボールから遠い時

図 5: 協調動作時の状態選択

```
Input(ruleSet, FuzzyValueSet) FuzzyValue.update()
for rule ∈ ruleSetdo
    ruleścore = min(rule.fuzzyValues)
    If rule.score >= action.score[rule.action] then
        score[rule.action] = rule.score
    end
end
bestAction = max(score.action)
execute(bestAction) | action ∈ Action
```

図 6: 意志決定エンジンのアルゴリズム

3.7 アルゴリズム

意思決定エンジンのアルゴリズムを図 6 に示す．意思決定エンジンは推論を行うためのルールと，ルールが計算に使っているファジィ真理値を入力とする．ファジィ真理値は外界からの情報を基に作られるものであり，ルールとアクションはそれぞれスコアをもっている．これから意思決定エンジンのアルゴリズムについて説明していく．まずすべてのファジィ真理値を外界の情報に応じて更新し，その後，各ルールのスコアを計算する．スコアはそのルールに含まれるファジィ真理値の最小値とする．また各アクションのスコアは，各アクションが含まれているルールのスコア中で，最大のものをとる．そして最後に全アクションの中でスコアが最大であるアクションを最適なアクションとして意志決定エンジンが選択し，実行する．

図 7 にアルゴリズムの動作例を示す．左の表は，各ルールにおいてどのファジィ関数のスコアが選ばれたかを示している．特に approach はファジィ関数を 2 つ持っているが，この場合はより低いスコアを持つ関数が選ばれる．また，右の表はどのアクションが選ばれたかを示しており，この場合は一番スコアの高いアクションである approach が選ばれている．そしてこれらの結果から，意志決定エンジンによりアクション approach が実行される．

ルール	ファジィ関数	スコア
approach	seeTheBall	1
	inNotCatchRange	0.82704
search	dontSeeTheBall	0
shoot	inCatchRange	0.17296

(a) ルールスコアの選択

アクション	スコア
approach	0.82704
search	0
shoot	0.17296

(b) アクションの選択

図 7: 意志決定エンジンの実行例

4 実装の比較

この節では簡単なアタッカーのプログラムを用いてスクリプトの例を示す．アタッカーの役割は点を取ることである．アタッカーはボールに近づき相手ゴールにボールを運ぶ．もしボールを見失ったら，その場で回転してボールを捜す．また複数のプレーヤーがいる場合にはお互いに情報交換をすることで，もし自分よりボールに近いプレーヤーがいる場合は，ボールに近づかずにサポートを行う．

これら一連の動作を行うために我々のチームで実際に使用している試合用のスクリプトソースの一部を掲載する．ソースは 2 種類掲載しており，図 8 は if-then ルールで記述された従来のソースであり，もう一方の図 9 は同じソースを意思決定エンジンを用いて書き直したソースである．これら 2 つのソースで実際にロボットを動かした場合，ロボットは全く同じ動作をする．

以前のシステムでは図 8 のようにプレーヤーは状況に合わせて適切な処理を行っていた．自分が一番ボールに近いときは，*playing.active* という関数を使ってボールに近づく．他のプレーヤーが自分よりボールに近いときは，自分はサポートに回るため *playing.passive* が選ばれる．これらの関数は違うアクションをするように実装されているが，似たような if-then ルールで書かれている．そのためプレーヤーに状況に応じた適切なアクションをさせるためには，関数内に if-then ルールを増やしていくことになる．

新システム（図 9）では実装がより簡単になり，かつ早く作れるようになった．以前書いていた大量の if-then ルールを書く代わりに，いくつかファジィ関数を定義するだけである．またファジィ化，ファジィ真理値，アクション，ルールと状態は最初に定義しておく．これらは簡単に作ることができ，また記述も少ない．

ボールの距離だけではなく，そのほかの外界からの情報も状態の区別に有用である．以前まで状態を作ることは実装の量が増え大変だったが，ルールベースで実装することにより簡単になった．例えば以前は状態を複数もつことが困難だったため，他のロボットとのボールの距離を比較することによって状態を切り分けていた．しかし状態を増やすことにより他のアイボの位置情報を使って状態を切り分けることができる．あるプレーヤーが攻めているときには，他のプレーヤーは守っているという実装も考えられる．

意思決定エンジンを使用することの利点は他にもある．まずファジィ論理を使うことにまた，if-then ルールの条件文の代わりにファジィ関数を定義することにより実装が簡単になり，可読性の高い記述が可能となった．スクリプトを作る際に状態を増やすことが簡単な点も新しいシステムの利点である．以前のシステムでは状態を増やすためには同じような関数をつくらなければならなかった．その結果，状態数が膨れ上がり，制御構造が管理しづらかった．一方，新システムでは新しいルールを追加するだけである．また，ファジィ関数とそれらを制御する部分ははっきりと分かれていて，スクリプトの管理が簡単である．

5 おわりに

本論文では，意志決定エンジンを組み込むことによりロボットサッカーの戦略部分における，記述が容易でありかつ人為的なミスが起こりにくい開発環境を構築した．この意志決定エンジンは最小もしくは最大をとる演算子しか使用していないため，処理にかかる時間も少ない．この事からハードウェア資源の制限されている RoboCup サッカー四足リーグにおいて，意志決定エンジンはサッカープレイヤーを作成するのに十分実用的であるといえる．

今後の課題として，非ファジィ化の実装が挙げられる．非ファジィ化とは，アクションを選択した際のファジィ真理値をアクションの入力として与えることである．現在の意志決定エンジンはアクションを実行する際，定義されたアクションをただ実行することしかできない．だが，非ファジィ化を用いることで，アクションを実行する際に，たとえば速度や方向といったアクションを実行する際に必要な値を指定することができる．

```
function playing.run()
   if nearestPlayer > 0.5 then
      current_role = "active"
   else
      current_role = "passive"
   end

   if current_role == "active" then
      playing.active()
   else
      playing.passive()
   end
end

function playing.active()
   visionLib:detectObjects()
   if seeTheBall then
      local ydist = ball_dist *
                    sin(ball_angle)
      local xdist = ball_dist *
                    cos(ball_angle)
      if xdist < 210 and
         abs(ydist) < 30 then
         shoot
      else
         approach
      end
   else
      search
   end
 end

function playing.passive()
   visionLib:detectObjects()
   if seeTheBall then
      support
   else
      search
   end
end
```

図 8: 旧システム上でのアタッカースクリプト

```
–create a fuzzyficator object

Active :
approach  <= seeTheBall, inCatchRange,
             NearestPlayer
search    <= dontSeeTheBal, NearestPlayer
setChoice <= notNearestPlayer

Passive :
support   <= seeTheBall, notNearestPlayer
search    <= dontSeeTheBall, notNearestPlayer
setChoice <= NearestPlayer

Choice:
setActive  <= NearestPlayer
setPassive <= notNearestPlayer

function playing.run()
   visionLib:detectObjects()
   Role:update()
   catchTheBall:update()
   seeTheBall:update()
   DecisionEngineUpdate()
end

function inCatchRange()
   local ydist = ball_dist *
                 math.sin(ur2r(ball_angle))
   local xdist = ball_dist *
                 math.cos(ur2r(ball_angle))

   local score = 1 - math.min((xdist/7 +
                 math.abs(ydist))/120 , 1.0)
   return score
end
```

図 9: 新システム上でのアタッカースクリプト

参考文献

[1] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[2] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[3] M. Loetzsch, M. Risler, and M. Jungel. Xabsl - a pragmatic approach to behavior engineering. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*, pp. 5124–5129, October 2006.

[4] A. Saffiotti, K. Konolige, and E. H. Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 76(1-2):481–526, 1995.

[5] R. R. Yager and L. A. Zadeh. *An Introduction to fuzzy logic applications in intelligent systems*. Kluwer Academic Publishers, 1992.