プロジェクト実習 レポート

天井カメラとロボットアームによる「スーパーボールすくい」

浮ヶ谷 敦生，奥 健太，徳田 浩平

実習期間 2004 年 6 月 7 日 〜 2004 年 9 月 13 日

2004 年 9 月 14 日 提出

目次

1 **課題** 3

2 **システム構成** 3
2.1 システムの概要 . . . 3
2.2 使用装置 . . . 3

3 **ビジョン部** 6
3.1 処理の流れ . . . 6
3.2 機能 . . . 7
3.3 アルゴリズム . . . 8

4 **戦略部** 12
4.1 処理の流れ . . . 13
4.2 アルゴリズム . . . 14
4.3 機能 . . . 19

5 **ユーザインターフェイス部** 20
5.1 処理の流れ . . . 21
5.2 機能 . . . 22

6 **アーム部** 22
6.1 座標系 . . . 22
6.2 処理の流れ . . . 22
6.3 機能 . . . 23
6.4 初期設定 . . . 23
6.5 ソケット開始 . . . 23
6.6 可動範囲チェック . . . 24
6.7 アーム動作 . . . 24
6.8 データ送受信 . . . 30
6.9 終了処理 . . . 30

7 **実験** 30
7.1 実験方法 . . . 30
7.2 実験結果 . . . 30

8 **考察** 31

付録 A 35
A.1 スレッド間同期について . . . 35

A.2 スレッド間での情報のやり取りについて 35
A.3 マルチスレッドプログラミングをする上でのコツ 36
A.4 3 人でチームプログラミングをしたことによる教訓 36
A.5 ロボットの安全対策 37

付録 B アーム操作マニュアル 39
B.1 前準備 39
B.2 MELFA-BASIC IV プログラムの転送 40
B.3 操作手順 41
B.4 動作確認 42

1　課題

天井カメラを有するロボットアームシステムを使用し，Windows を OS とする PC でロボットアームの制御処理とカメラの画像処理を実装する．それらを統合していわゆる「スーパーボールすくい」を実現する．

2　システム構成

2.1　システムの概要

図 1 に示すとおりシステムを構成する．実験装置のハードウェアブロック図を図 2 に示す．コントローラがアームの動作を制御し，そのコントローラを PC でリアルタイムに外部制御する．ティーチングボックスは手動でアームを操作することができる．また，実験装置の概観を図 3 に示す．

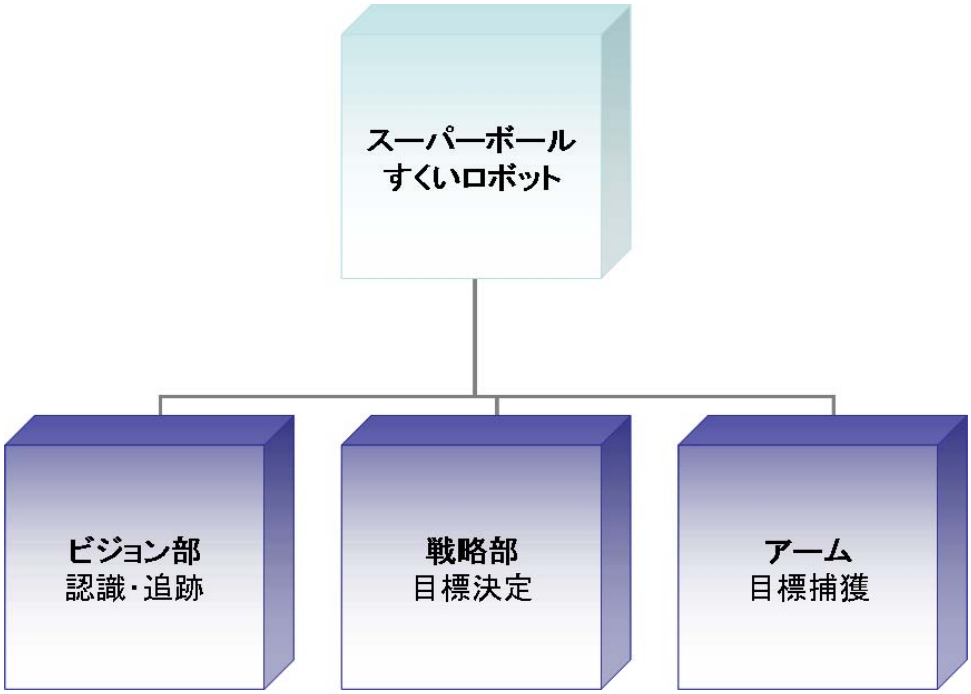

図 1　システム構成

2.2　使用装置

アームは三菱電機 MELFA RV-3AL （図 4）を使用した．ポイはアームの先端に図のようにテープで固定した．

カメラは SONY 社製のデジタル出力カラーカメラモジュール DFW-VL500 （図 5）を使用する．DFW-VL500 は，IEEE1394-1995 を採用したフルデジタルカメラである．色再現性を考え，CCD には『原色フィルター』『正方格子』『全画素読み出し』という Wfine CCD を採用している．データ転送の物理層には，400Mbps

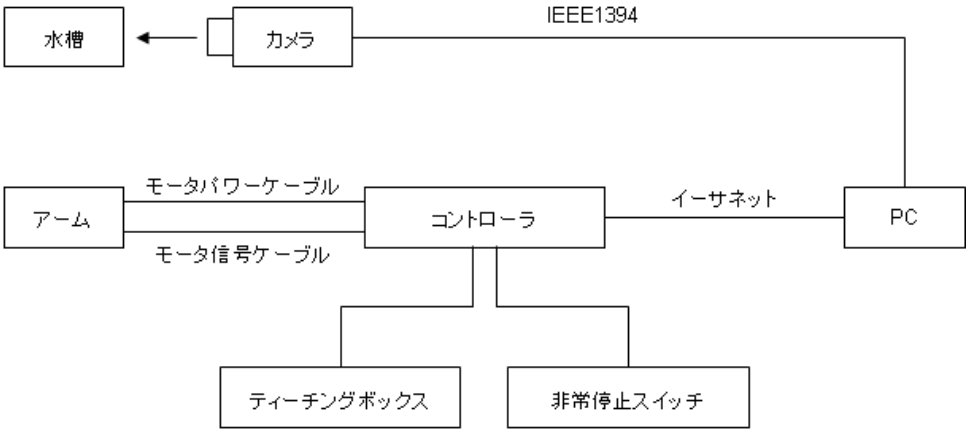

図 2　ハードウェアブロック図

図 3　実験装置の概観

を採用し VGA(640x480),YUV(4:2:2)30fps の出力が可能である．また，DFW-VL500 は，12 倍ズームレンズを採用している．

水槽は縦 69cm（上部），58cm（下部），横 114cm（上部），103cm（下部），深さ 20cm のものを使用し，水は 10cm の深さまで入れた．

図 4　MELFA RV-3AL

図 5　SONY DFW-VL500（文献 [1] より）

3　ビジョン部

ビジョン部では，DirectShow [2, 3] を用いてカメラから水槽画像をキャプチャする．その水槽画像から，候補ボールを探索し，ターゲットボールに対してトラッキング処理を行う．本章では，ビジョン部における全体の処理の流れを解説し，つづいてクラス構成および各機能を解説する．さらに，その各機能を実現するためのアルゴリズムについて解説する．

3.1　処理の流れ

ビジョン部の処理の流れを図 6 に示す．キャプチャスレッドにおいて，10ms ごとに繰り返しカメラキャプチャを行い，キャプチャ画像をメイン処理へ送る．メイン処理において，キャプチャされた水槽画像を取得すると，それより 2 値画像を作成する．つづいて，候補ボールを探索し，そのうちターゲットボールをトラッキングする．また，各処理において，2 値画像，候補ボール座標およびターゲットボール座標を戦略部へ渡す．

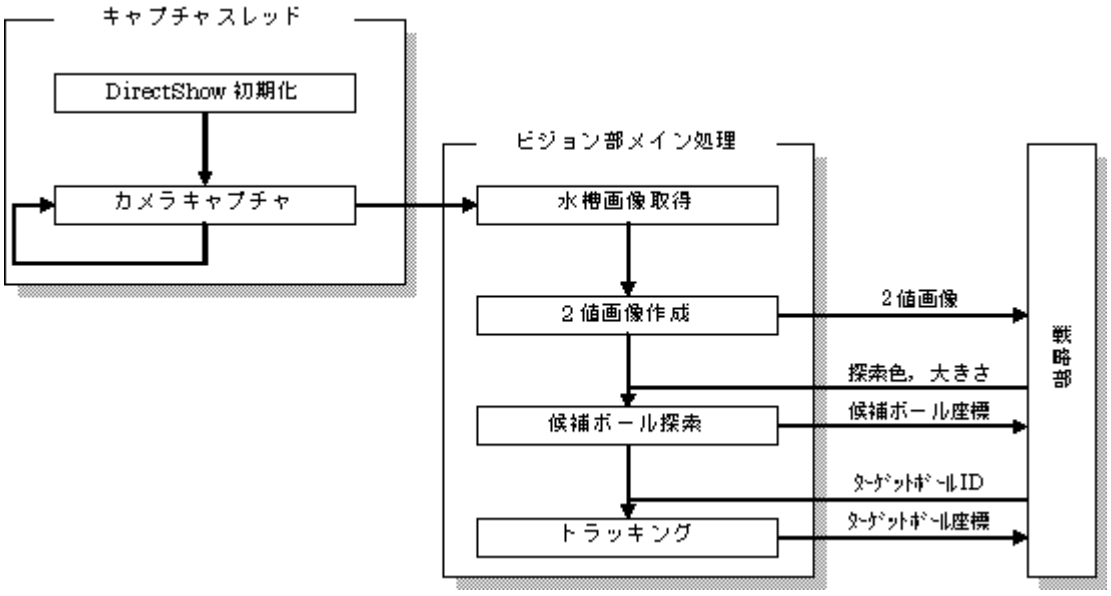

図 6　ビジョン部の処理の流れ

3.1.1　水槽画像取得

キャプチャスレッドからキャプチャ画像を取得する．そのキャプチャ画像より水槽部以外の余分な部分を除去する．さらに歪み補正を加え，水槽部の領域のみを矩形領域として抽出する．

3.1.2　2 値画像作成

候補ボールの位置探索および戦略部におけるポイ沈め可能領域探索を行うために，キャプチャされた水槽画像の背景差分を取り，さらにそれを 2 値化する．2 値画像は 3.1.1 項で取得された水槽画像およびあらかじめ

取得しておいた背景画像により作成する．

3.1.3 候補ボール探索

候補ボール探索のために，背景が黒でボールの色および位置情報のみを抽出したボール探索用画像を作成する．探索用画像は，3.1.1 項で取得した水槽画像および 3.1.2 項で作成した 2 値画像より作成する．この探索用画像を用いてラベリング処理を行い，戦略部より指定された色および大きさを持つボールを探索する．抽出された複数のボールを候補ボールとして登録しておく．

3.1.4 トラッキング処理

戦略部により指定されたターゲットボールに対してトラッキング処理を行う．トラッキング処理を行うことによりターゲットボールの位置を取得し続けることができる．さらに，ターゲットボールの位置を中心としたトラッキング領域を定義することにより，処理をその領域内にのみ絞ることができるため，全体としての処理速度が向上する．

3.2 機能

ビジョン部は図 7 のように 4 つのクラスから構成される．

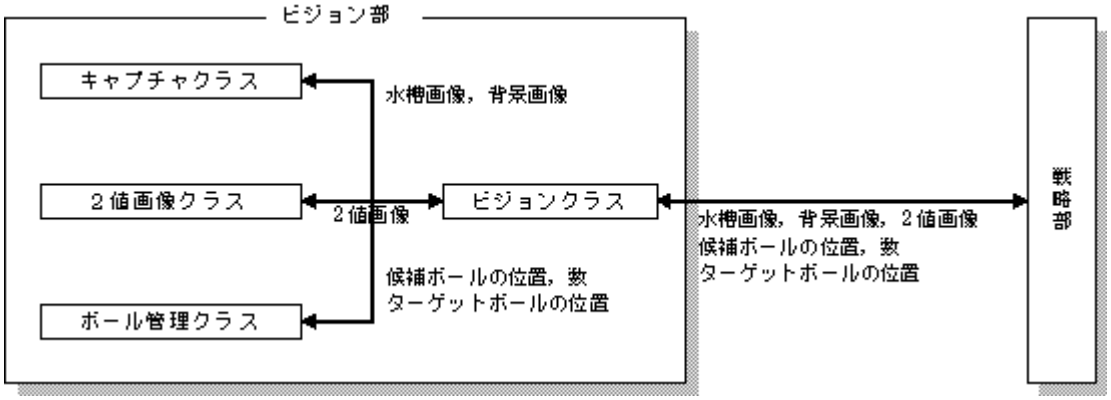

図 7　ビジョン部のクラス構成

1. ビジョンクラス
   ビジョン部の統合クラスであり，このクラスの処理を代表するメンバ関数はビジョンスレッドとして実行される．また，戦略部と以下の 3 つのクラスとのインタフェース部となる．
2. キャプチャクラス
   カメラからのキャプチャを行う．水槽画像および背景画像を管理する．
3. 2 値画像クラス
   2 値画像の作成および管理を行う．
4. ボール管理クラス
   ボール探索用画像の作成および管理，候補ボールの探索および登録，ターゲットボールのトラッキング

処理を行う．

3.2.1 ビジョンクラス

ビジョンクラスはビジョン部の統合クラスであり，前述のキャプチャクラス，2 値画像クラス，ボール管理クラスの 3 つのクラスへの参照を持ち，それぞれのメンバ関数を必要に応じて呼び出す．それぞれのクラスと戦略部とのインタフェースとなり，3 つのクラスより取得した情報を戦略部へ渡す．また，このクラスの main 関数はビジョンスレッドとしてプライマリスレッドから呼び出され，その後戦略部と連携しながらビジョン部の処理を順に実行する．

3.2.2 キャプチャクラス

キャプチャクラスは，キャプチャスレッドにより水槽画像をキャプチャする．また，そのキャプチャした水槽画像および背景画像を管理する．ビジョンクラスへ水槽画像および背景画像を渡す．

3.2.3 2 値画像クラス

2 値画像クラスは，キャプチャした水槽画像および背景画像を用いて，背景差分 2 値画像を作成する．また，その作成した 2 値画像を管理する．ビジョンクラスへ 2 値画像を渡す．

3.2.4 ボール管理クラス

ボール管理クラスは，キャプチャした水槽画像および 2 値画像を用いて，ボール探索用画像を作成する．また，その探索用画像を管理する．さらに，その探索用画像を用いて，候補ボールをラベリングにより探索し，さらにターゲットボールに対するトラッキング処理を行う．ビジョンクラスへ候補ボールの位置および候補数，ターゲットボールの位置情報を渡す．

3.3 アルゴリズム

ここでは，以下の各処理におけるアルゴリズムについて解説する．

1. 背景差分 2 値画像作成
2. ボール探索用画像作成
3. ラベリング処理
4. ボールの色の一致判定
5. ボールの識別
6. トラッキング処理

なお，本プログラムで用いた水槽画像，背景画像，2 値画像および探索用画像はすべて $320 * 192$ の大きさを持つ．

3.3.1 背景差分 2 値画像作成

図 8 のように, キャプチャされた (a) 水槽画像およびあらかじめ取得しておいた (b) 背景画像より，(c) 背景差分 2 値画像を作成する．その作成手順を以下に示す．

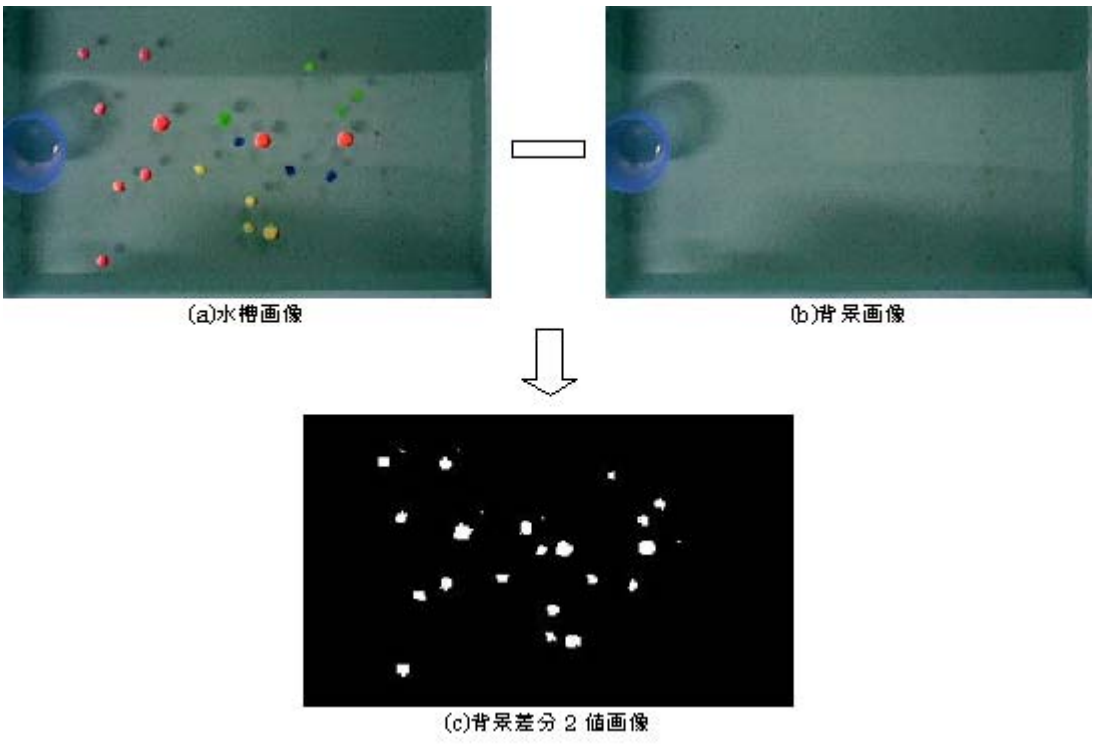

図 8　背景差分 2 値画像

step 1　水槽画像および背景画像の各画素における RGB 値を取得する．
step 2　水槽画像および背景画像の対応画素における RGB 値の差分値を取る．
step 3　差分値がある閾値以下の画素を黒 (0) 画素，それ以外を白 (1) 画素として，2 値画像を作成する．

まず，水槽画像の各画素における RGB 値を取得する．同様に，背景画像の各画素における RGB 値を取得する．各画像の対応画素における RGB 値の差分値を取得する．水槽画像上の画素 (x, y) における RGB 値の i 成分を $a_{x,y,i}$，背景画像上の画素 (x, y) における RGB 値の i 成分を $b_{x,y,i}$ とすると，画素 (x, y) における RGB 値の i 成分の差分値 $c'_{x,y,i}$ は，

$$c'_{x,y,i} = |a_{x,y,i} - b_{x,y,i}| \tag{1}$$

となる．この差分値より，背景部分と非背景部分との 2 値化処理を行う．2 値化処理には閾値を用いる．閾値を t とすると，背景差分 2 値画像上の画素 (x, y) における RGB 値の i 成分 $c_{x,y,i}$ は，

$$c_{x,y,i} = \begin{cases} 0 & (c'_{x,y,i} \leq t) \\ 1 & (c'_{x,y,i} > t) \end{cases} \tag{2}$$

となる．この閾値を小さくするとノイズが多くなる．逆に大きくするとノイズが少なくなるが，ボールを見落とす可能性が生じる．したがって，プログラム実行環境に応じて，ボールを正確に識別でき，かつ，ノイズ

が入らないように適切に閾値を設定しなければならない．なお，今回のデモ実行環境（NAIST 情報科学研究科棟内ロボット実験室）においては閾値 30 が適正値となった．

3.3.2 ボール探索用画像作成

図 9 のように，(a) 水槽画像および (b) 背景差分 2 値画像より，(c) ボール探索用画像を作成する．単純に各画像の対応画素の AND を取ることにより，2 値画像における黒 (0) 画素部分にはそのまま黒画素，白 (1) 画素部分にはスーパーボールの色情報が設定される．

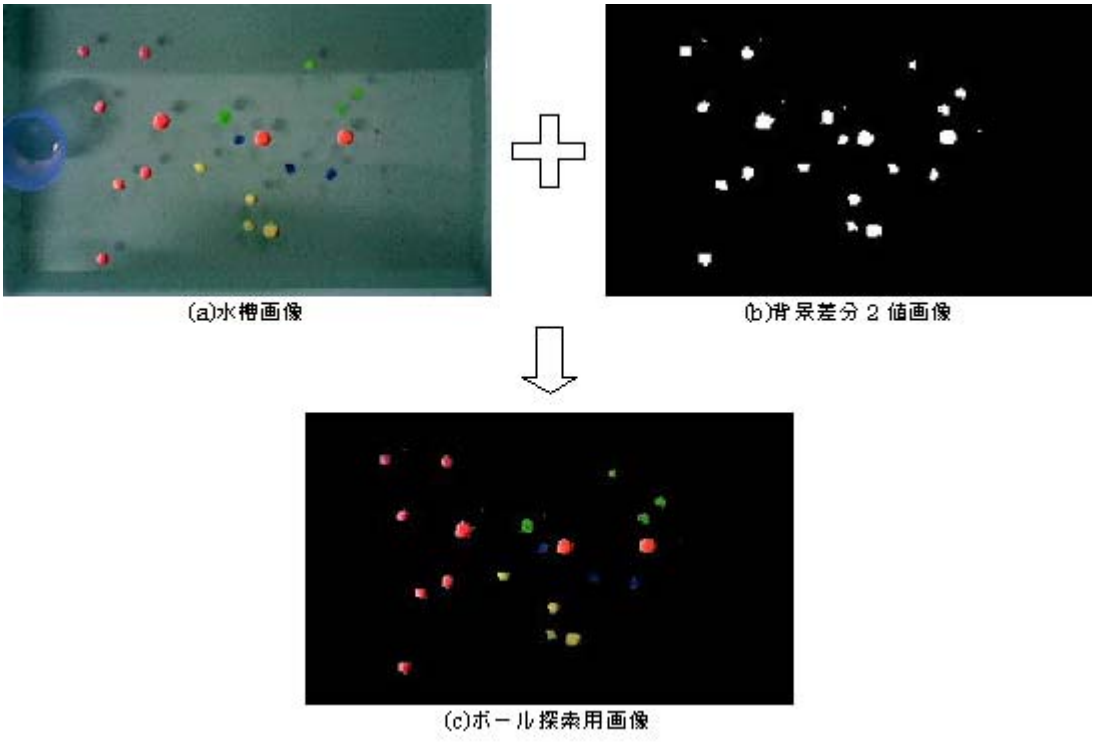

図 9　ボール探索用画像

3.3.3 ラベリング処理

ボール探索用画像より，指定された色を持つボールを探索するためにラベリング処理 [4] を行う．その処理手順を以下に示す．

step 1　ラベリング用 2 元配列 $l[320][192]$ を用意し，全要素を-1 で初期化する．

step 2　探索用画像の左上画素から x 方向に走査する．

- step 2.1　画素 (x, y) のラベル $l[x][y]$ が設定されていないか確認する．
- step 2.2　探索色と画素 (x, y) の色とを比較する．
- step 2.3　両色が一致すると判定されれば，ラベリング用配列 $l[x][y]$ に現在のラベル i を設定する．
- step 2.4　その (8 方向) 隣接画素に対して step 2.1〜2.4 を行う．

step 3　現在のラベル i をインクリメントする．

step 4　右下画素に到達するまで step 2,3 を繰り返す.

ラベリング処理を行う前に，探索用画像と同じ大きさ $(320 * 192)$ を持つラベリング用 2 元配列を用意する．探索用画像の左上画素から x 方向に走査する．探索色と画素 (x, y) の色とを比較し，これらが一致するかを判定する．(この色の一致判定の方法については 3.3.4 項ボールの色の一致判定で解説する．) 一致すると判定されれば，ラベリング用配列に現在のラベルを設定する．その画素の 8 方向の隣接画素に対しても色の一致判定を行い，一致すればラベリング用配列に同ラベルを設定する．ここで，ラベルは 0 から始まり，抽出されたオブジェクト順にラベルを付けるものとする．すべての隣接画素に対する探索が終われば，次のオブジェクトを探索するために走査をつづける．右下の画素まで走査が終われば，ラベリング処理は終了となる．
(なお，ここで抽出したオブジェクトの面積および重心座標を求めておく．)
面積は同値でラベル付けされた画素数，重心座標は各画素の座標の平均を取ることにより求めることができる．ここで求めた面積は 3.3.5 項ボールの識別において用いる．

3.3.4　ボールの色の一致判定

ボールの色の一致判定を行うために，あらかじめ実在するボールの色を定義しておく必要がある．そのために，図 10 のようなボールの色定義画像を用いる．これは，キャプチャした水槽画像から対象とする色を持つボールの部分のみを抽出したビットマップデータである．このデータから，各画素の HSV（H:色相，S:彩度，V:明度）値を取得し，うち色相の最大値 h および最小値 l を取得する．また，色相の平均値を求め，それを基準値 b とする．さらに，基準値，最大値および最小値より，色の一致判定の際の色相の許容範囲 a を求める．ここで，許容範囲は単純に最大値と最小値との差の 1/2 とする．色の一致判定の際には，対象の画素の色相がこの定義域 $(b \pm a)$ に含まれるかを調べることによって判定する．

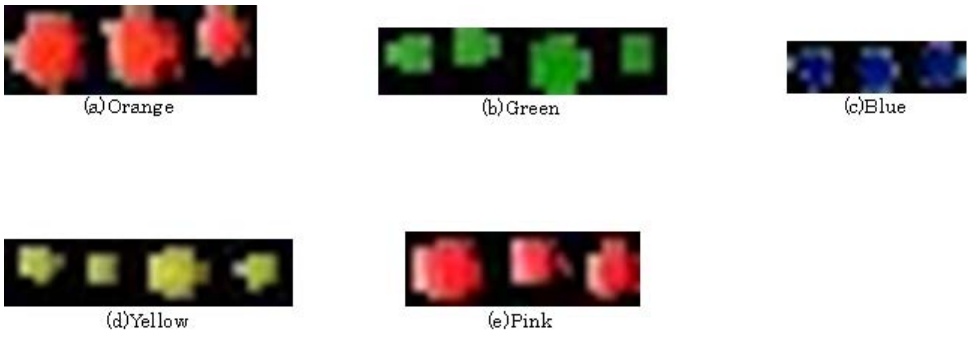

図 10　ボールの色定義画像

3.3.5　ボールの識別

3.3.3 項ラベリング処理および 3.3.4 項ボールの色の一致判定により，指定された色を持つオブジェクトは抽出できた．しかし，抽出したオブジェクトすべてがボールであるとは限らない．つまり，ノイズ部分を誤って抽出している可能性もある．したがって，ボールの識別を行う必要がある．本プログラムではこの識別を面積によって行う．あらかじめボールの面積の最大値および最小値を設定しておく．抽出されたオブジェクトの面積がこの設定範囲内にあればボールとして識別する．これによって，ノイズ部分の除去が可能となる．さら

に，大小ボールの面積の境界値を設定しておくことで，大小を識別することが可能となる．なお，本プログラムでは最大値を 100，最小値を 10，境界値を 40 と設定した．図 11 は，探索候補として小さな青いボール (blue,small) を設定した際の探索結果である．

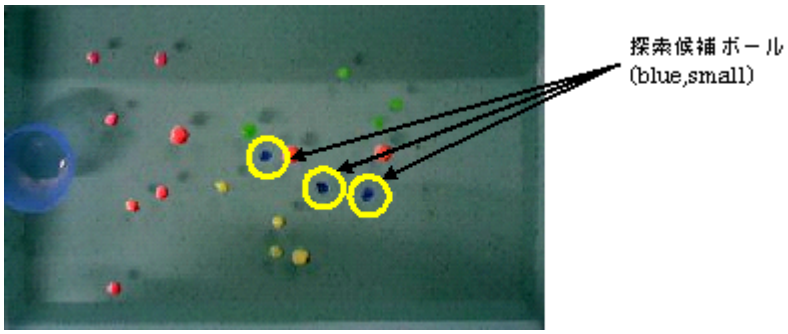

図 11　探索候補ボールの抽出（blue,small の例）

3.3.6　トラッキング処理

戦略部からターゲットボールが指定されると，そのターゲットに対するトラッキングを開始する．そのトラッキング処理の手順を以下に示す．

step 1　指定されたターゲットの位置を取得する．
step 2　トラッキング領域を，その位置を中心とした一定の大きさに限定する．
step 3　トラッキング領域内でもっとも中心に近い探索スーパーボールをターゲットとし，その位置を更新する．
step 4　step 2〜4 を繰り返す．

まず，指定されたターゲットの位置を取得する．その位置を中心とした，一定の大きさを持つトラッキング矩形領域を作成する．この領域内においてもっとも中心に近い探索ボールをターゲットとし，その位置を更新する．これにより，フレームごとにターゲットボールをトラッキングすることが可能となる．トラッキング領域の大きさが大きいほどボールの速い移動に対応できる．

しかし，領域内にターゲットボールと同タイプのボール（図 12 の例では，(blue,small)）が入ると，ターゲットが入れ替わる可能性が生じる．また，探索領域を広げることになり，全体の処理パフォーマンスが低下する．できるだけ正確にターゲットをトラッキングしつつ，ある程度の処理速度を保つように，トラッキング領域の大きさを適切に設定する必要がある．なお，本プログラムではこの領域の大きさは 50pixel 四方の正方形とした．

4　戦略部

戦略部は他の全てのモジュールと連携して，このシステム全体を統括する．ポイを沈められる場所を探し，どのボールを実際に掬うのか決定し，更にゲームの開始，終了の同期をとる．まず戦略部全体の処理の流れを解説し，次に細かいアルゴリズムを解説し，最後にそれぞれの機能を持つオブジェクト毎の解説を行う．

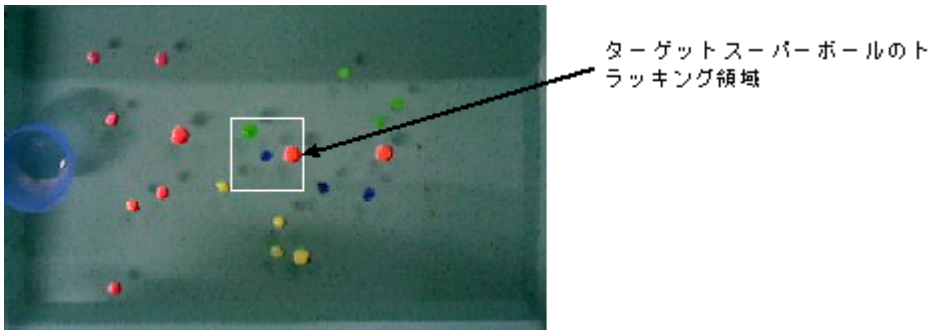

図 12　ターゲットボールのトラッキング

4.1　処理の流れ

戦略部はシステム全体の流れを統括すると共に，カメラ部から得た情報を元にどのボールを掬うか判断し，アーム部へボールの座標，ポイを沈める位置を指示する．戦略部が行う処理を大まかに表すと図 13 のようになる．

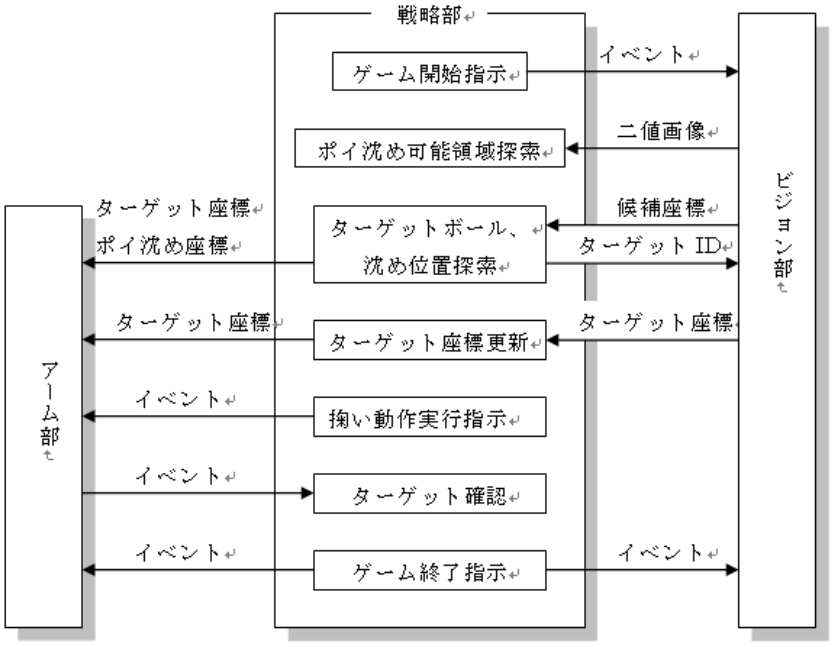

図 13　戦略部処理手順

ゲーム開始直後から順に

I インターフェイス部からターゲットボールの種類を指定されると，ゲームの開始をビジョン部，アー

ム部に対して知らせる．

II ビジョン部から背景差分の 2 値画像を受取り，ポイ沈め可能領域マップを生成する．

III ビジョン部からターゲット候補ボールの座標列を受け取り，それと障害物マップを元に，複数ある候補ボールの内からターゲットボールを選ぶ．

IV 初期位置としてビジョン部，アーム部にターゲットボールの座標を知らせる．

V アーム部がターゲット掬い可能位置へ移動するまでの間，ビジョン部から逐次送られてくるターゲット座標をアーム部へ知らせる．

VI アーム部が掬い上げ可能状態を知らせて来たら掬い上げ指示を出す．

VII アームがボールをお椀へ落とすのを待って，1 ゲームの終了をビジョン部，アーム部へ知らせる．

という処理を行っている．

4.2 アルゴリズム

上記 II，III，IV のアルゴリズムをフローチャートなどを交えて解説する．
(IV は座標変換の計算について解説する)

4.2.1 II ポイ沈め可能領域マップ生成アルゴリズム

ビジョン部から背景差分の 2 値画像を受け取る．(図 14) この 2 値画像は障害物があるところを 1，ないところを 0 としてビジョン部との間で取り決めてある．
(2 値画像は各座標の値が 0 と 1 だけで構成される画像である．ただし視覚化したとき見やすいように 0 の場所は (R,G,B) 値を (0,0,0)，1 の場所は (R,G,B) 値を (255,255,255) にしてある．)

これを元に障害物にぶつからずポイを沈められる場所を探し，マップを生成する．(図 15) この図は実際に中でマップとして利用されている物を，視覚化したものである．青い部分がポイを沈められる中心点の集合となる．白い障害物の部分を避けるように形作られている．

障害物マップの作成アルゴリズムは図 16 のようになる．

図 14　元になる 2 値画像

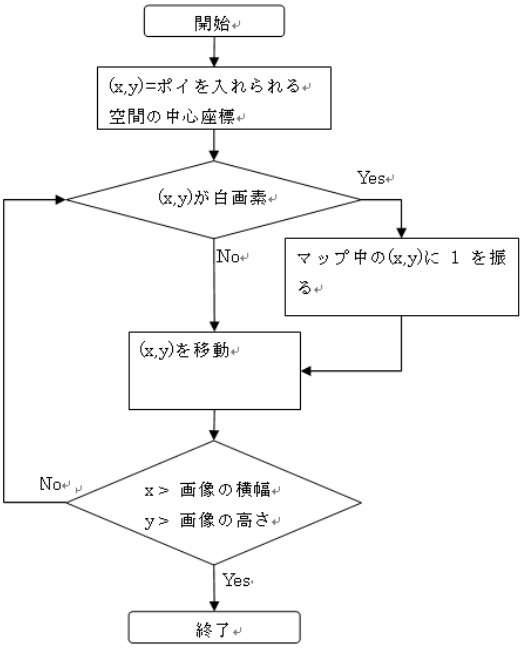

図 15　生成されたポイ沈め可能領域マップ

4.2.2　III ターゲットボール決定アルゴリズム

ビジョン部から取得したターゲット候補ボールの座標列の中からターゲットボールを選ぶ．(ターゲット候補ボールが一つしかないときは選択の余地はないが，複数あった場合には次に示す基準でその中から一つを選ぶ．)

その為に候補ボールそれぞれについて，最も近い距離にあるポイ沈め可能位置を探索しそのボールのポイ沈め位置とする．(水槽中に障害物に当たらずにポイを沈められる場所が 1 箇所も無かった場合，単純に水槽の中央を沈め位置としている．)

次にその結果と候補ボール座標とポイ沈め可能領域マップを用いて，複数ある候補ボールの内からターゲットボールを選ぶ.

ターゲット選択の基準は,

- ボールからポイ沈め位置までの距離が短いこと
- アームの動作開始位置からポイ沈め位置，ボール位置を経由して，お椀位置までの総経路長が短いこと

である.

沈め位置までの距離だけを見てアームの動作経路長が長くなってしまったり，逆にアームの動作経路長が最短になる様にボールを選んで，沈め位置とボールが離れていたらスマートなやり方ではないと判断し，このような式にした．この 2 条件がバランスよく満たされているものを選ぶのが最適であると思われたので，次の式 (3) で候補ボールそれぞれについて”掬う為の手間”の値を計算し，最も小さいものをターゲットに決定している.

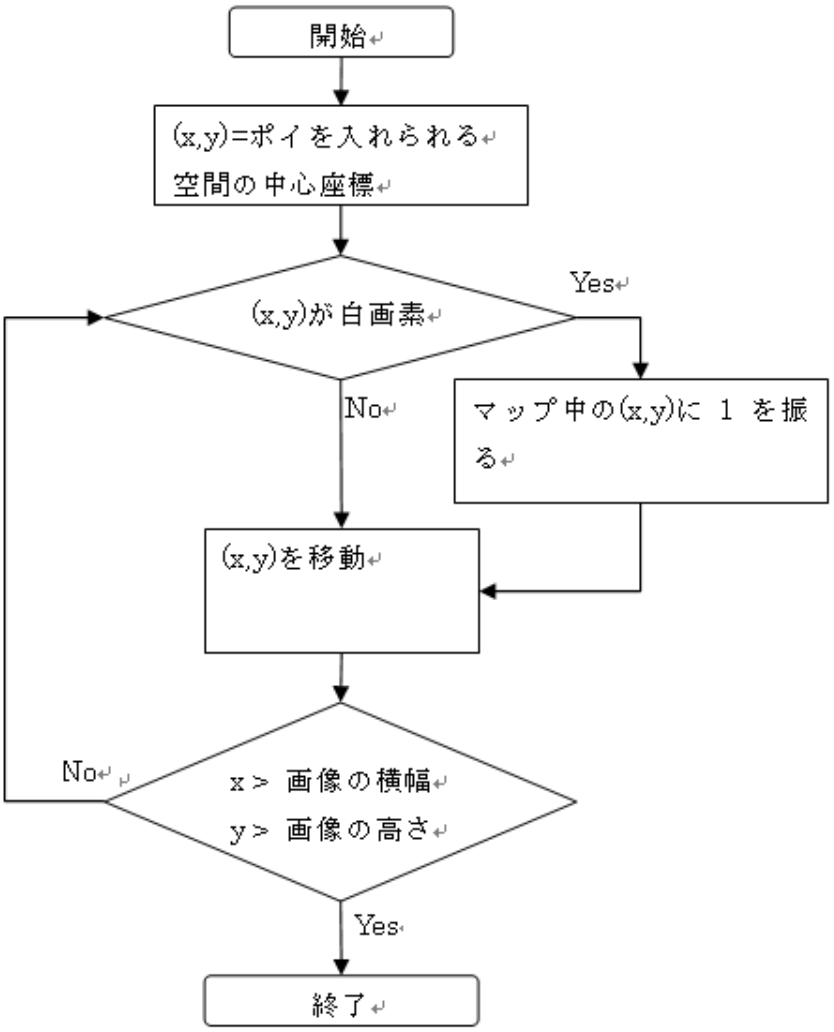

図 16　ポイ沈め可能マップ作成アルゴリズム

$$p_i = d_i * \frac{l_i}{s} \tag{3}$$

ただし，

p_i：ボールの得点
d_i：最短沈め位置までの直線距離
s：動作開始位置からお椀までの直線距離
l_i：動作開始位置から，ポイ沈め位置，ボール，お椀までの経路長
(i は候補ボール座標値の配列のインデックス))

である．

従って，沈め位置までの距離が近いほど，最短経路長が短いほど高得点になる．
実際にはプログラムは次の図 17 に示すようにこの処理を行っている．

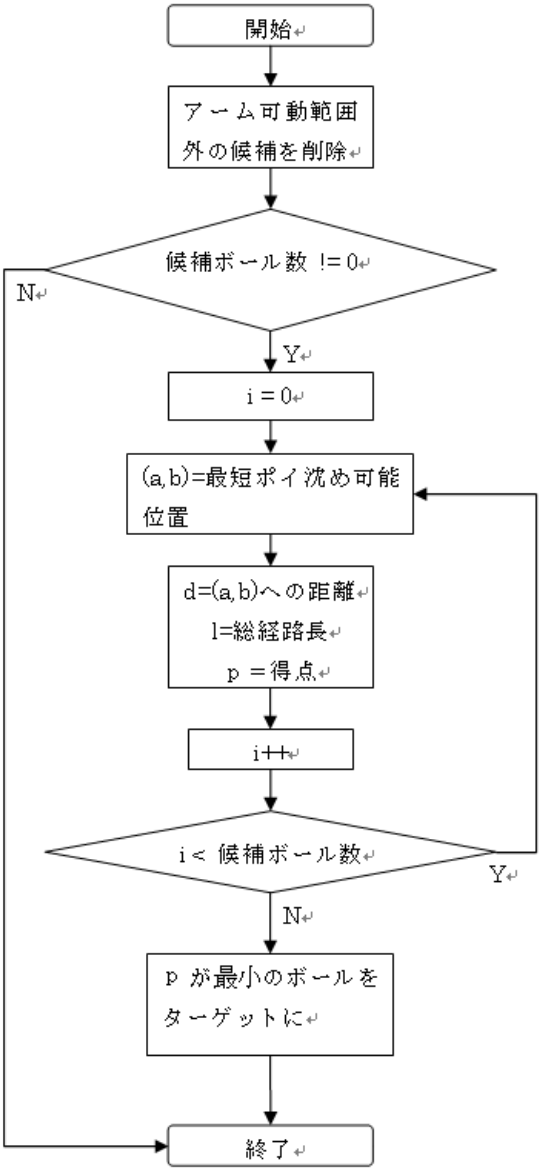

図 17　SelectTarget アルゴリズム

このプログラムで II のマップ (図 15) を用いた結果を図 18 に示す．

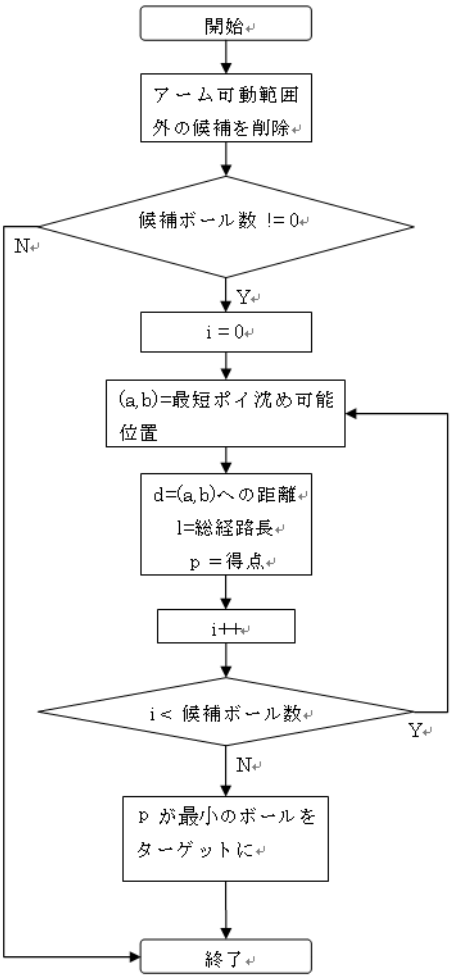

図 18　ターゲット選択の様子

4.2.3　IV 座標系の変換

戦略部はターゲットの位置としてビジョン部，アーム部にターゲットボールの座標を知らせる．戦略部は 2 値画像を扱う点から，基本的に画像座標系 (WindowsDIB 座標系) で処理している．従ってアーム部へターゲット座標や沈め座標を知らせたり，アーム部からアームの有効動作範囲を取得したりする場合にはそれぞれの座標値を相互に変換しなければならない．

実際の水槽の大きさを測り，水槽をビジョン部のカメラでキャプチャした静止画と比較した結果，変換の式は次の様になる事がわかった．

$$\begin{cases} x_{arm} = 3.3 * y_{cam} + 290 \\ y_{arm} = (\frac{imWidth}{2} - x_{cam}) * 3.3 \end{cases} \tag{4}$$

ただし，x_{arm}:アーム座標系の x 座標
y_{arm}:アーム座標系の y 座標
x_{cam}:カメラ座標系の x 座標
y_{cam}:カメラ座標系の y 座標
$imWidth$:画像の横幅（現在 320)

アーム座標系は実際の物理的距離に基づいていて，1mm=1 単位である．290 という定数は，アーム座標系であるアームの基部中心から水槽下端中央までの距離 (290mm) である．y_{arm} の係数 3.3 は，今のカメラの位置/設定だと 1pixel に約 3.3mm 収まる計算になる為である．

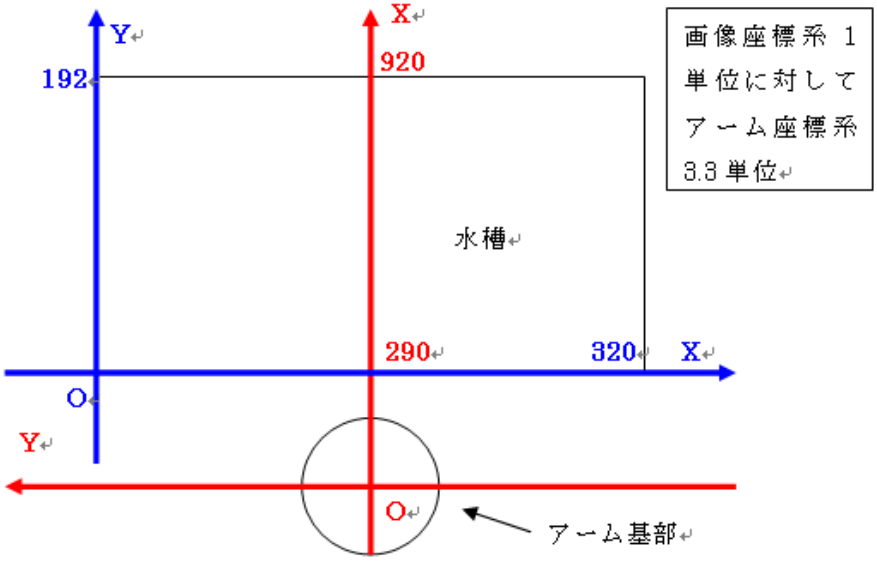

図 19　アーム座標系とカメラ (画像) 座標系の関係

4.3　機能

戦略部を設計するに当たって，必要な機能は主に

1. アーム部やビジョン部と情報をやり取りし，ポイ沈め可能領域作成処理やターゲット決定処理を呼び出す
2. ２値画像からポイを沈められる場所を探す
3. 候補座標配列とポイ沈め可能領域マップからターゲットボールと沈め位置を決定する

と考えられたので，戦略部は主に 3 つの代表的なクラスを中心に形成されている．上記 3 つの役割を受け持つクラスがそれぞれ戦略クラス，ポイ沈め可能領域探索クラス，ターゲット選択クラスである．

この他にもう一つ重要なクラスとしては，スレッド管理クラスを作成した．このクラスは，マルチスレッドプログラム特有の同期処理やスレッド間通信に関する処理をまとめたクラスとなっている．

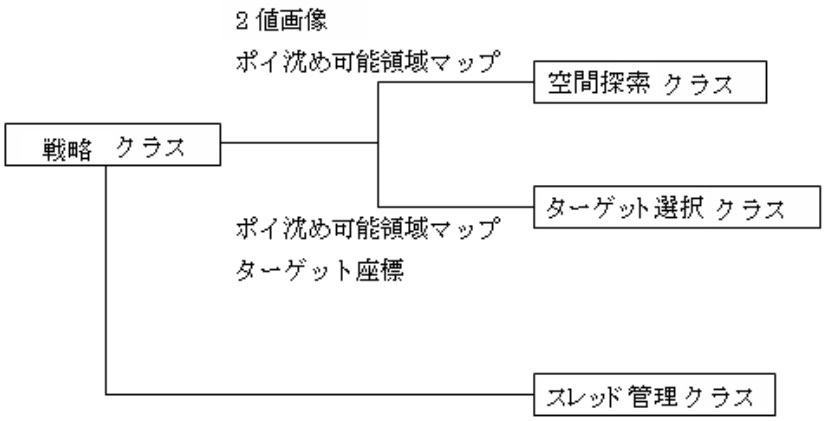

図 20　戦略部クラス構成図

1. 戦略クラス
   戦略クラスは戦略部の中心となるクラスである．戦略クラスは戦略部とその他の部分とのインターフェイスとしての機能と，その為に必要な簡単な機能 (カメラ部とアーム部との座標変換など) を持つ．
2. 空間探索クラス
   空間探索クラスはポイ沈め可能領域探索を行う．探索に必要な変数と関数を集約している．生成したポイ沈め可能領域マップを保持し，それを戦略クラスへ渡す．またポイ沈め可能領域マップを画像化し，要求に応じてウィンドウへ表示する．
3. ターゲット選択クラス
   ターゲット選択クラスはターゲットを選択するために必要な変数と関数を集約している．候補ボール座標の配列とポイ沈め可能領域マップを受け取り，ターゲットボール座標と沈め位置座標を決定する．メンバーであるターゲット決定関数は，沈め位置座標とターゲットボールの ID(配列のインデックス) を返す．
4. スレッド管理クラス
   スレッド管理クラスはスレッド間で同期を取ったり，共有メモリを使って通信するときに便利な関数をメンバーに持つ．イベント名を指定してそのイベントがシグナル状態になるのを待つ関数や，いずれかのスレッドが終了していることを知る変数を持っている．

5　ユーザインターフェイス部

ユーザインターフェイス部は本システムのユーザインターフェイスを表示，管理する．
実際のウィンドウの様子は図 21 の様なものである．

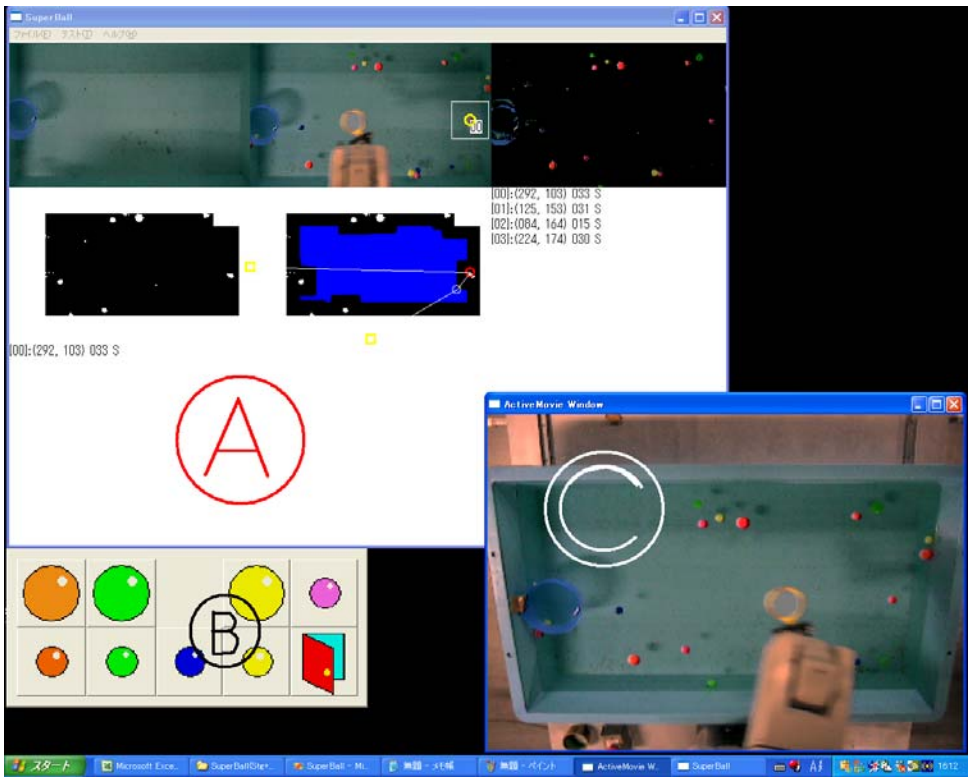

図 21　本システムのユーザインターフェイス

Aのウィンドウが処理途中の様子を表示し，Bのパネルはユーザがボールの種類を選択するときに使用する．Cのウィンドウは現在の水面の様子をカメラがキャプチャした画像である．

5.1　処理の流れ

ユーザインターフェイス部はプライマリスレッドで動作する．ユーザインターフェイス部は本システムが立ち上がると途中経過ウィンドウ (図 21) Aとボタンパネル (図 21 B)，キャプチャウィンドウ (図 21 C) を表示する．その後はシステムの終了が指示されるまでメッセージループをし続け，ユーザーの入力に対する処理を呼び出す．ユーザはウィンドウCで掬いたいボールのタイプを選び，ボタンパネルBの対応するボタンを押す．

そしてユーザから指定されたターゲットボールのタイプをビジョン部へ伝えるのもユーザインターフェイス部の役割である．

ターゲットボールのタイプは色と大きさを保持する構造体を定義しており，予め実際に使用するボールにあわせて値を設定してある．

5.2 機能

ユーザインターフェイスはユーザからの操作を待ち受ける main 関数と，受けた操作によって処理の種類を変えるプロシージャ関数，ボタンパネルに関する表示，操作を管理するボタンパネルクラスからなる．

main 関数は通常の Windows プログラムのエントリーポイントとなる WinMain 関数をそのまま利用している．

プロシージャ関数も同様に通常の Windows プログラムで記述するコールバック関数そのものである．ボタンパネルクラスはダイアログボックスを使いボタンパネルを表示し，ボタンパネルに対する入力イベントを受けて，プロシージャ関数へ入力の種類を伝えるメッセージを投げている．

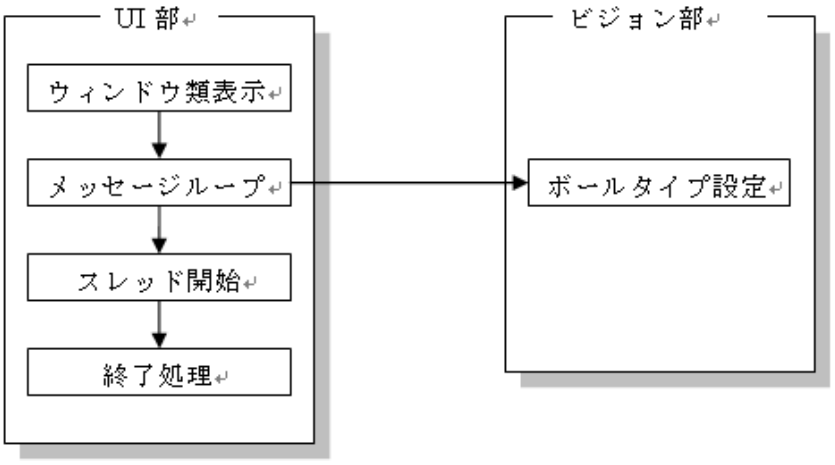

図 22　ユーザインターフェイス処理手順

6 アーム部

6.1 座標系

使用するアームの座標系を図 23 に示す．図のツール長とは先に取り付けたハンドの長さのことである．X, Y, Z 軸は図のとおりアームの台座を原点とし，ハンドの先端を制御点とし mm 単位で表される．A, B, C 軸は角度を表し，単位は度（degree）である．

6.2 処理の流れ

1. 初期設定を行う．
2. ソケットを開始.
3. ポイ入れ位置とターゲットの決定を待つ．

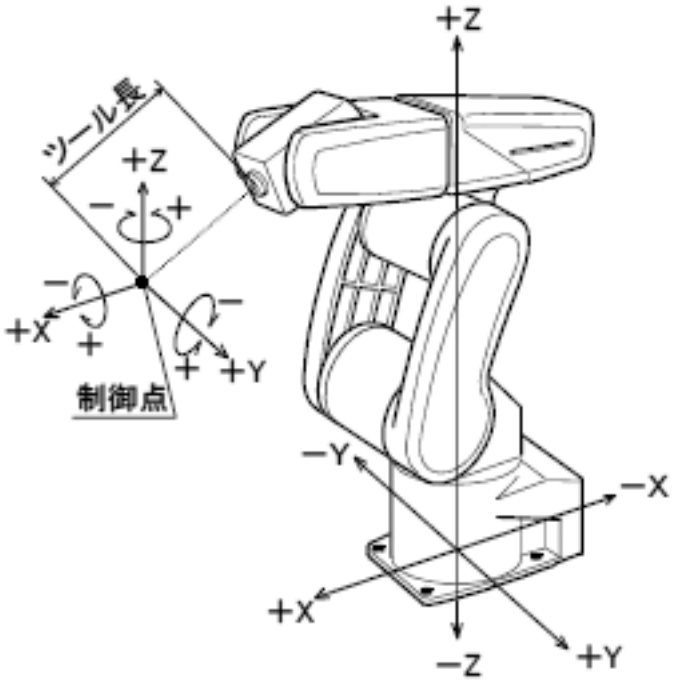

図 23　アームの座標系

4. 戦略部よりポイ入れ位置とターゲットの $X-Y$ 座標を受け取る．ポイの長さを考慮してアームの座標に変換し，これが動作可能範囲内かチェックする．
5. ポイを水の中へ入れる．
6. ターゲットに到達したと戦略部が判断するまでターゲットをトラッキングする．到達していれば 7 へ．
7. ターゲットをすくい上げる．
8. ターゲットを器に入れる．
9. 初期位置へ戻る．
10. ゲームが終了されるときに終了処理を行う．

6.3　機能

6.4　初期設定

送信データに組み込む PC の IP アドレス，接続先のポート番号，指令するデータ型の設定を行う．これらの値は固定値である．

6.5　ソケット開始

- Windows Socket DLL の初期化
- IP アドレス・ポートの設定

- ソケット作成

6.6 可動範囲チェック

アームの可動範囲を図 24 に示す．図の全領域が水槽の水面全体の範囲である（以下同様）．X, Y, Z 軸の原点（アームの台座）は図の下側にある．ティーチングボックスを用いて手動でエラーが起きない，かつ，ポイが水槽の縁ギリギリの場所を探すよりこの範囲を求めた．

また，実装では簡単のため図 25 に示すとおり，円弧の部分を直線と近似した．

図 24　アームの可動範囲

戦略部からの動作指令位置がこの範囲外であればエラーを返す．

また，この可動範囲でターゲットがすくえる範囲は図 26 のとおりである．

6.7 アーム動作

戦略部から動作座標（X, Y）と動作種類（ポイを水中に入れる，ターゲットを追跡する，すくい上げる，の 3 種類）を受ける．動作種類にあわせて適切な姿勢をとって動作する．アームの動作は次のように行っている．

1. コントローラへの送信データの作成をする．
2. データを送信する．
3. データを受信する．
4. 戦略部にアームの現在位置を渡す．
5. 指定位置に到達したかを判定する．到達していなければ，2 へ戻る．到達していればループを抜ける．

図 25　アームの可動範囲（実装用）

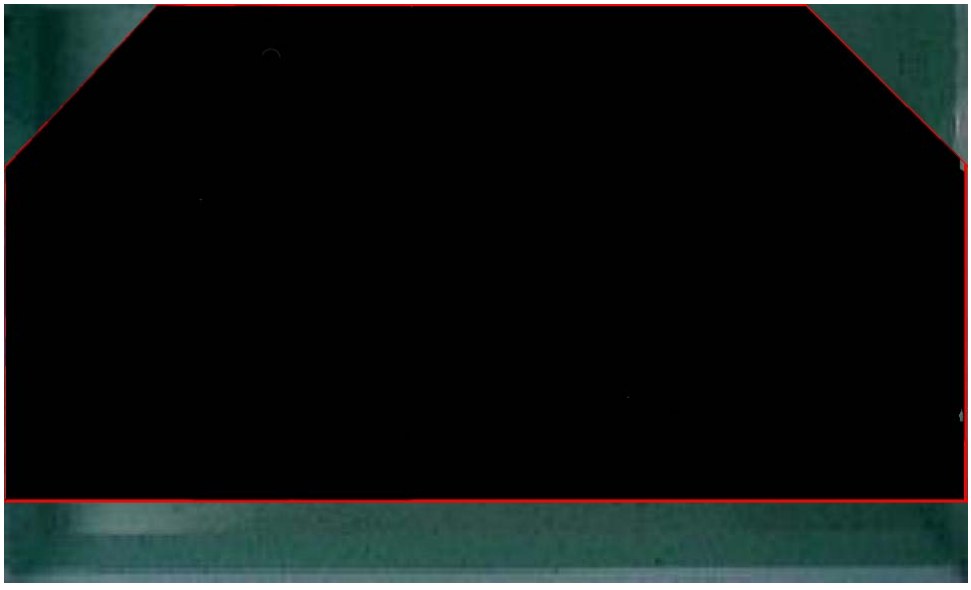

図 26　ターゲットをすくえる範囲

6.7.1 **ポイ入水**

ポイを入水点上に移動　初期位置から水平に入水点の真上へ移動する（図 27）.

ポイを水中に入れる　ポイを 20 度[*1]傾けながら入水する[*2].（図 28）

ポイを水平にする　ターゲットを追跡するときの水の抵抗が少なくなるようにポイを水平にする（図 29）.

図 27　ポイを入水点上に移動

6.7.2 **ターゲット追跡**

戦略部からのすくい上げ命令が来るまで，ターゲットの下へ水中移動し続ける（図）30.

6.7.3 **すくい上げ**

ポイを真上へ移動　ポイを 20 度傾けながらポイを水中から出す[*3]（図 31）.

ターゲットを器に入れる　（図 32）

初期位置へ戻る　オクルージョンをなくすために初期位置へ戻る（図 33）.

[*1] スーパーボールが落ちないぎりぎりの角度を経験的に求めた

[*2] 水の抵抗を減らすために傾けている．また，アームの可動範囲を広げるため，Y 軸の値が正のときと負のときとで傾ける方向を変えている

[*3] ポイ入水のときと同様の理由である.

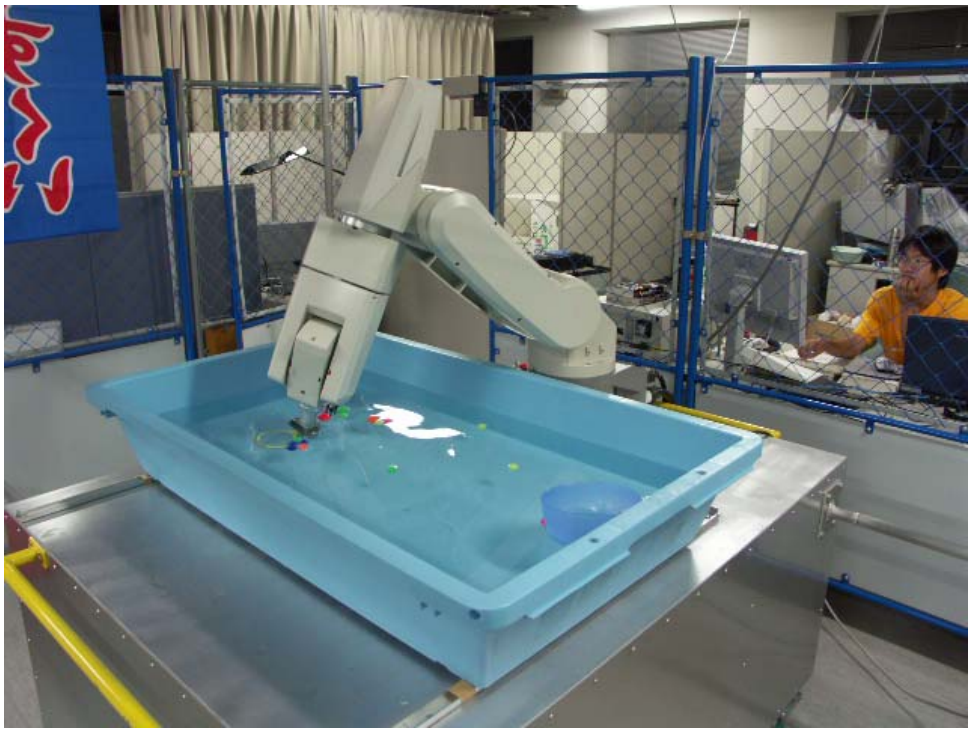

図 28　ポイを水中に入れる

図 29　ポイを水平にする

図 30　ターゲットの下へ水中移動

図 31　ポイを真上へ移動

図 32　ターゲットを器に入れる

図 33　初期位置へ戻る

表 1　実験に用いたスーパーボールの色およびその個数

色	大	小	計
Orange	2	1	3
Green	1	3	4
Blue	0	3	3
Yellow	1	3	4
Pink	1	5	6
計	5	15	20

6.7.4　送信データ作成

アームコントローラの仕様で決められた型で上記の座標を組み込んだデータを作成する．

6.8　データ送受信

移動先の位置を送信し，アームの現在位置を受信する．データを送信しないと受信できないので注意が必要である．

6.9　終了処理

アームコントローラへ終了命令を送り，アームを折りたたんで完全に終了する方法と，終了命令を送らないで，プログラムのみを終了する方法がある．前者はアームコントローラのサーボが OFF になるのでそれ以降プログラムの命令を受け付けないが，後者はサーボが ON のままなので，続けて操作できる．

7　実験

7.1　実験方法

本システムを用いてスーパーボール掬いの実験を行った．本実験によるスーパーボール掬いの成功率によりシステムを評価する．本実験において用いたスーパーボールの色およびその個数（大小の内訳）は，表 1 の通りである．この計 20 個のスーパーボールを水槽上に自由に浮かばせた状態で実験を行った．試行回数を表 2 に示す．各色 10 回ずつの計 50 回とする．なお，ボールサイズの大小の試行回数の内訳は表 2 の通りである．

7.2　実験結果

本実験の結果を表 3,4 に示す．表 3 は，コース別の成功率である．表中の外・中・内は，それぞれアームから見て，外側，真中，内側を表し，左・中・右は，それぞれアームから見て，左側，真中，右側を表す．表 4 は，色・サイズ別の成功率である．

表 2　スーパーボール掬い実験の試行回数

色	大	小	計
Orange	10	0	10
Green	3	7	10
Blue	0	10	10
Yellow	2	8	10
Pink	0	10	10
Total	15	35	50

表 3　実験結果 1：コース別成功率

	左	中	右	計
外	.571	.556	.750	.600
中	.500	.889	.750	.800
内	.750	.714	.500	.667
計	.615	.720	.667	.680

表 4　実験結果 2：色・サイズ別成功率

色	大	小	計
Orange	.600	-	.600
Green	1.000	.714	.800
Blue	-	.800	.800
Yellow	1.000	.375	.500
Pink	-	.700	.700
計	.733	.657	.680

8　考察

コース別に見ると，予想通り中央部における成功率が高かった．これは，中央部はポイ沈めのスペースにも余裕があり，アームにとって得意な姿勢（手前から奥に掬う）で臨めるためであるといえる．アームから見て，外側および内側の部分は，苦手なコースであり，成功率も低かった．ポイを沈めてから，トラッキングによるスーパーボールを追跡するための移動に伴い，アームによる水流が発生する．外側のボールを追跡する際には，この水流によって，ターゲットのスーパーボールがアームの移動方向と同じ方向，つまりさらに外側に流される．このためアームの可動範囲外になり，失敗してしまう．これは，流れやすいサイズの小さいスーパーボールに対してよく見られた．また，特に右内側における成功率が最も低かった．このコースはアームが

最も苦手なコースであり，トラッキングにより追跡ができたとしても，アーム自身によりスーパーボールが隠れたり，移動させたりしてしまい，トラッキング処理が追いつかなくなってしまい，結果失敗してしまう．

色別に見ると，グリーンおよびブルーが最も成功率が高く，イエローが最も低かった．オレンジおよびピンクはその中間程度といえる．まず，イエローはポイの色と同じであるため，トラッキングの際，スーパーボールとポイが惑わされるという現象によるものであった．この問題は，ポイを黒などのスーパーボールの色とは異なる色で塗りつぶすことにより，解決できる．オレンジおよびピンクは色相だけでは両者の見分けが難しく，それが成功率に影響した．これより，色を識別するためには色相以外の要素（RGB，HSV など）も見なければならないということがいえる．これに対して，グリーンおよびブルーは他とはっきりと識別でき，成功率が高くなった．

サイズ別に見ると，大きい方が成功率が高かった．これは，先述したとおり，小さい方が流れやすいためである．さらに，大きい方は面積が大きいことからトラッキングにおいて見失う危険性も少なかったことがいえる．総合的に見て，小さいイエローの成功率が最も低かった．これは，「小さい」，「イエロー」という苦手な条件を合わせ持つためである．

今回，トラッキングが開始された段階において，全体の処理をトラッキング領域のみに絞るようにした．全体に対して処理を行った場合，トラッキングの処理速度が遅く，スーパーボールの移動に対応できなかった．処理をトラッキング領域に絞ることによって，処理速度が向上し，正確にトラッキングを行うことができた．

ただし，アームの移動や衝突により，スーパーボールが瞬時に移動した場合，トラッキングが失敗する．そこで，今後の課題として，スーパーボールが瞬時に移動しても，正確にトラッキングするアルゴリズムを考えなければならない．また，これが可能となれば，より高速に移動する金魚に対してもトラッキング処理ができるのではないかと考える．

謝辞

本プロジェクトを進めるにあたり，課題の提示およびその解決のための適切な技術的指導をいただいた，木戸出正継教授ならびに河村竜幸先生に，心より感謝の意を表します.

参考文献

[1] 「SONY Japan」http://www.sony.co.jp/Products/ISP/products/interface/DFWV500-j.html

[2] 浦野収司著,「サンプルで学ぶ DirectX 9 プログラミングテクニック」(株式会社ディー・アート, 2003 年)

[3] 大澤文孝他著,「DirectX 9 実践プログラミング」(株式会社工学社, 2003 年)

[4] 酒井幸市著,「Visual Basic & Visual C++ による デジタル画像処理入門」(CQ 出版株式会社, 2002 年)

[5] 住吉乱著,「Visual C++ 逆引き大全 500 の極意」(株式会社秀和システム, 2002 年)

[6] Aaron Cohen, Mike Woodring 共著, 金森玲子訳,「Win32/C++ マルチスレッドプログラミング詳説」(O'REILLY ジャパン, 1999 年)

[7] 北山洋幸著,「Win32 API システムプログラミング with Visual C++ 6.0」(CUTT システム, 2000 年)

[8] 「MELFA BFP-A5929-M」

[9] 「MELFA BFP-A5949-K」

[10] 「MELFA BFP-A5946-M」

[11] 「MELFA BFP-A8080-D」

[12] Lewis Napper 著, トップスタジオ 訳編, 江村豊 監修 「Winsock 2.0 プログラミング」(ソフトバンクパブリッシング, 1998 年)

[13] 吉田弘一郎 著 「極める Visual C++」(技術評論社, 1998 年)

付録 A

A.1 スレッド間同期について

一般的にマルチスレッドプログラムでは，スレッド毎に全く異なる進度で処理が進む．あるスレッド A で作成したデータを別のスレッド B が利用したいとしたら，スレッド B はスレッド A のデータ作成を待って，完成すると同時に自分の処理を始めるといった調整が必要になる．このような調整を行うことを”同期を取る”と呼ぶ．同期を取るにはいくつかの方法があるが，本システムではイベントという仕組みを利用している．イベントは OS が用意するオブジェクトの一つで，信号発信状態 (シグナル状態) と信号非発信状態 (非シグナル状態) とを切り替えることができる．また，あるイベントオブジェクトが信号発信状態になる (シグナル化) のを待つという API が用意されている．これを組み合わせて同期を取ることができる．

具体的に言うと，

- スレッド A，スレッド B の同期のためにイベントオブジェクト AB を用意する．
- スレッド B はイベント AB のシグナル化を待つ状態に入る．
- スレッド A がデータを作成した時にイベント AB をシグナル化する API を呼ぶ.
- スレッド B はイベント待ち状態から解放されて，引き続き処理を続けることができる．

本システムではこの仕組みを使って同期を取っている．例えば最初に戦略部がゲームの開始をビジョン部，アーム部に知らせると書いたが，あれは直接メッセージなどを送っているわけではない．戦略部がゲーム開始イベントをシグナル化すると，それを待っているビジョン部，アーム部がそれぞれ処理を開始するということである．

A.2 スレッド間での情報のやり取りについて

本システムでは，スレッド間での情報のやり取りにメモリマップドファイル (ファイルマッピングオブジェクト) を利用している．メモリマップドファイルは，複数のスレッドで共有できるメモリ領域を作成する．

マッピングオブジェクトは OS が用意するオブジェクト (カーネルオブジェクト) の一種で，本来はファイルをメモリ上にマップする為のものである．通常はメモリ上にマップするファイルを指定してファイルマッピングオブジェクトを作成するが，ファイルを指定せずにサイズだけを指定して空のファイルマッピングオブジェクトを作成することもできる．そしてそのメモリ領域の先頭ポインタを取得し，そこへスレッド間で共有したいデータをコピーするのである．この機能とイベントを利用して，スレッド間でデータを受け渡している．

具体的には次のようにする．

- データの送信側でファイルマッピングオブジェクトを作成し，先頭ポインタを取得する．
- 送信側はデータを共有メモリへコピーする．
- 送信側でデータマッピング完了のイベントをシグナル化する．
- 受信側はイベント待機から開放され，ファイルマッピングオブジェクトへのポインタを取得する．
- 受信側でローカルへデータをコピーする．

A.3 マルチスレッドプログラミングをする上でのコツ

A.3.1 デバッガについて

シングルスレッドプログラミングでは，実行時エラーの追跡には開発環境などに付属しているデバッガが役に立つ．しかしマルチスレッドプログラミングでは，スレッドごとにスタック領域が独立しているため，デバッガだけでは状態を追いきれない．こういったデバッグモードが当てにならない場合は，ところどころテキストファイルに途中経過を吐かせると状況が分かりやすい．

例：”[戦略スレッド]:at main() ターゲットボール座標 (x, y)”とテキストに吐き出す等.

A.3.2 カーネルオブジェクトの生成数

排他処理を行う為のクリティカルセクション構造体やイベントオブジェクトはいくつでも用意できる．わかりやすいプログラムは同期処理や排他処理のトラブルを生み出しにくくするので，どういうルールでいくつこういったカーネルオブジェクトを用意するのかは重要である．

今回の実装を通して感じたのは，スレッドの数などは関係なく，その処理の種類だけ用意するのがわかりやすいということである．例えば排他処理のためのクリティカルセクション構造体なら，複数のスレッドからアクセスされる可能性のある変数の数だけ用意する．同期をとるためのイベントオブジェクトなら，イベントの種類だけオブジェクトを用意した方が役目が分かりやすい．こうすれば将来的に，他にスレッドができて同じ変数にアクセスしようとしても問題が起きないし，そもそもどのスレッドがその変数にアクセスする可能性があるのかを考える必要がなくなる．

A.3.3 不揮発性変数

通常 CPU がある計算を行うとき，メモリから読み出した情報を CPU 内のレジスタにキャッシュしておくことが良くある．マルチスレッドプログラムは，スレッドの切り替え時にこの CPU 内レジスタを含む CPU の状態などを丸ごと置き換える．従って同一プロセス内のスレッドが，同じメモリアドレス空間を使用していても，短い時間で見れば同一の変数がスレッドごとに異なる値を持つ可能性がある．

これを避けるためには，変数を宣言する時に volatile キーワードを頭につけて記述する．（例：volatile int value;) volatile キーワードはそれを宣言した変数が，CPU 内のレジスタにキャッシュされないことを保証する．つまり volatile 宣言された変数を計算に使う時は，外部メモリから値を読み出し計算を行い，結果を必ず外部メモリに書き戻す．これによって同一の変数がスレッド毎に値が異なることを防ぐことができる．

A.4 3 人でチームプログラミングをしたことによる教訓

それぞれの実装技術の得手/不得手を最初によく認識しておくべきである．最も知識，経験が多いものが他のメンバーをサポートすることになるので，その分レポートの記述や実験，実装環境の固有値調査など他の面で負担をうまく割り振ると良い．これは単に心理的なものだけではなく，実装期間にも影響する．

A.5 ロボットの安全対策

本デモで使用するロボットアームにはマニュアルモードとオートモードがある．マニュアルモードはティーチングボックスというコントローラを人間が操作することでアームが動作する状態．オートモードは内部にコマンド解釈機能を持つコントローラを通して，専用 BASIC 言語か一般の PC 上で動作するプログラムでアームをコントロールする状態.

マニュアルモードでは，ティーチングボックスの背面にあるデッドマンスイッチというスイッチを押し続けていないとアームは動作しない．従って誤動作の心配などは極わずかである．

それに対しオートモードでは，コントロールしているプログラムに誤りがあると，プログラマの想定していない位置へアームが動作することがある．アームの誤動作までシミュレートできるシミュレータがあれば大分安全性は高まるが，残念ながら存在しない．

従って誤動作による事故を防ぐには，オートモードの時はロボットに近づかない，プログラムは入念に作成する事などが重要である．

しかしプログラミングはともかく，ロボットには近づかなければならない機会が多い．その際にオートモードを解除し忘れる可能性がある．或いはポイの交換などで余りに頻繁に近づくため，面倒になってオートモードのままで放置する様になる危険性もある．更に一般に本ロボットアームを紹介する時の事を考えると，人間側の注意力のみに頼った安全対策には限界があると思われる．

そこで人がロボットの可動範囲に近づく際には，必ずオートモードを解除しなければならない仕組みを考案した.

ロボットの周りをフェンスで封鎖し，唯一の入り口にはセーフティドアスイッチ (OMRON 製 D4NS 型) を付けた．(図 34,35)

セーフティドアスイッチはロボットのコントローラに接続されており，スイッチが切れるとコントローラはアームの動作電源を切って警告ブザーを鳴らす．スイッチにはキーがあり，キーが刺さっていると ON，抜かれていると OFF となる．従ってオートモード中にドアスイッチからキーが抜かれた状態になると，ロボットは停止する．(図 36)

ドアスイッチのキーはドアの開閉を妨げる器具に固定されており，ドアを開ける際にはその器具を引き抜かなければならない．従ってドアを開けて中に入る際にはドアスイッチが必ず切られる為，ロボットが誤動作する事はスイッチやコントローラ自体が故障した場合を除いてあり得なくなる．

物品：

- フェンス (扉付き/なし数枚)
- ドアスイッチ (2 接点タイプ OMRON 株式会社製 D4NS)
- 配線材 (キャブタイヤケーブル 4 芯タイプ，防水型ケーブルクランプ OA-W1609)

参考 web サイト:

- フェンスの製造・購入先
  「株式会社　ニチホ」http://www.nichiho-net.com/
- 安全装置の製造元

「オムロン株式会社」http://www.fa.omron.co.jp/

- ケーブルコネクタ製造元

  「オーム電機」http://www.ohm.co.jp/

図 34　ドアスイッチ

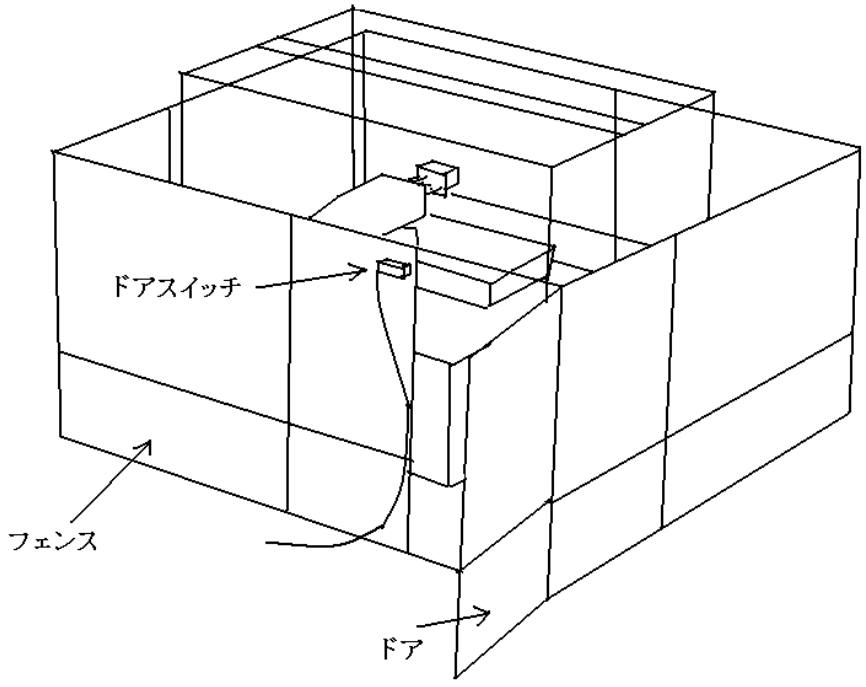

図 35　フェンス配置図

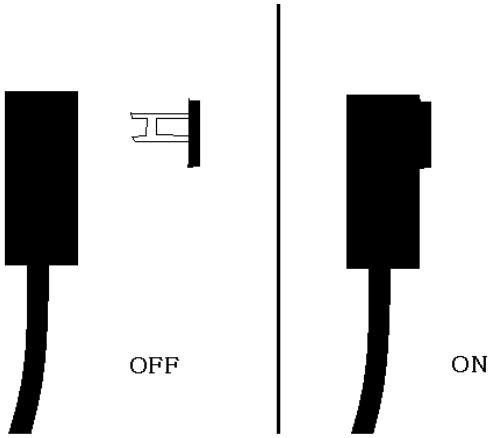

図 36　ドアスイッチの ON/OFF

付録 B　アーム操作マニュアル

アームを外部制御機器（PC 等，以下 PC と記述する）で操作する場合のマニュアルを書く．

B.1　前準備

ロボットに組み込む BASIC プログラムを PC で作成し，それをロボットコントローラに転送する．

B.1.1　MELFA-BASIC IV プログラムの作成

外部制御機器からの命令を受信するために，ロボット専用言語 MELFA-BASIC IV を用いたプログラムを作成しなければならない．プログラムの作成はテキストエディタで編集する方法と，パソコンサポートウェアのプログラム編集ツールを使用する方法の 2 通りがある．どちらの場合でもファイルは”*.prg”（*は 1, 2, . . . , 9）という名前で保存するプログラム編集ツールを用いる場合，行の編集を有効にするには編集した行で Enter をタイプしなければならないので注意．

下にプログラムソースの例を示す．

```
10␣OPEN␣"ENET:192.168.0.2"␣AS␣#1␣'␣ファイル番号１にイーサネットとしてパソコン側の IP アドレスを指定
20␣MOV␣P1␣'␣初期位置 P1␣へ移動␣（任意の位置を␣P1␣としてティーチング）
30␣MXT␣1,0,1200␣'␣ファイル番号 1␣から与えられた指令値の通り動作．直交座標指定．フィルタ時定数．
40␣MOV␣J1␣'␣終了位置 J1␣へ移動
50␣HLT␣'␣中断
60␣END␣'␣終了
J1=(0.000,-90.000,168.000,0.000,0.000,0.000,0.000,0.000)␣'␣間接座標指定
P1=(607.150,0.000,348.380,-180.000,0.000,-180.000)(7,0)␣'␣直交座標指定
```

B.1.2 MXT 命令について

- MXT 命令を実行すると，ネットワーク接続された PC から動作制御のための位置指令を取り込むことができる．
- 動作制御時間（約 7.1msec）で 1 つ位置指令を取り込み，動作できる。
- 指令値をサーボに送った後，コントローラから PC に現在位置などのコントローラの情報を送信する．
- PC からコントローラに指令値を送信した場合のみ，コントローラから PC に返信する．

MXT 命令の引数は順に，「ファイル番号」,「指令位置データ型」,「フィルタ時定数」である．このうち変更するのはフィルタ時定数のみで良い．フィルタ時定数が小さすぎるとアームがエラーで停止するなど，動作が安定しないので 800(msec) 以上が望ましいが，逆に大きすぎるとアームの動作が遅くなってしまうので適当に調節する必要がある．本実験では 1400(msec) を使用した．

B.2 MELFA-BASIC IV プログラムの転送

パソコンサポートウェアのプログラム編集ツールの転送機能を使用してロボットコントローラに転送する（図 37）．

図 37　プログラム管理

B.3 操作手順

開始手順

1. PC の IP アドレスを 192.168.0.2 にする（ネットワーク設定を切り替えるツールを用いるのが便利）.
2. ロボット室の電源を ON にする（図 38）.
3. ロボットコントローラの電源を ON にする（図 39，1）.
4. コントローラのモード切替スイッチを AUTO（Ext.) にする（図 39，14）.
5. 表示切替ボタンを数回押してプログラム番号を表示する（図 39，7）.
6. UP/DOWN ボタンを数回押して実行する BASIC プログラムの番号を選択する（図 39，15）. BASIC プログラムについては B.1.1 を参照.
7. スタートボタンを押してプログラムを実行し，PC からの命令待機状態にする（図 39，2）.
8. PC からアームを操作する.

終了手順

1. サーボが OFF になっているか確認する．なっていなければ OFF にする（図 39，10）.
2. ロボットコントローラの電源を OFF にする（図 39，1）.
3. ロボット室のアーム用の電源を OFF にする（図 38）.

図 38　ロボット室電源

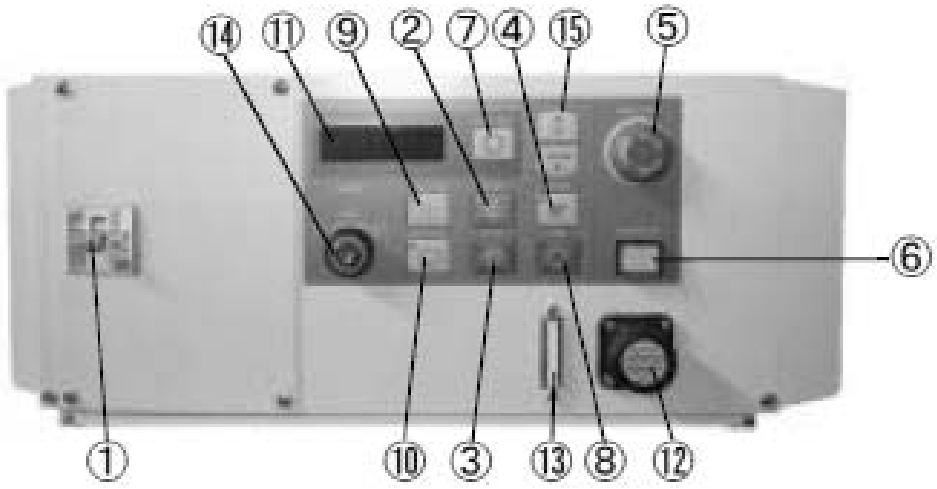

図 39　ロボットコントローラ

B.4　動作確認

パソコンサポートウェアのモニタリングツールを使用し，座標位置をリアルタイムに確認する（図 40）.


動作状態 [1 :]
閉じる(X)
メカ1
RV-3AL
ABSリミット
- [deg,mm] +
-180.00 J1 180.00
-110.00 J2 160.00
-5.00 J3 189.00
-180.00 J4 180.00
-140.00 J5 140.00
-220.00 J6 220.00
-80030.00 J7 80030.00
-80030.00 J8 80030.00
関節リミット
- [deg,mm] +
-160.00 J1 160.00
-90.00 J2 140.00
15.00 J3 169.00
-160.00 J4 160.00
-120.00 J5 120.00
-200.00 J6 200.00
-80000.00 J7 80000.00
-80000.00 J8 80000.00
直交リミット
- [mm] +
-10000.00 X 10000.00
-10000.00 Y 10000.00
-10000.00 Z 10000.00
現在位置
関節系座標
[deg,mm]
J1: 60.30
J2: 13.43
J3: 121.46
J4: 0.00
J5: 45.11
J6: -2.32
J7: 0.00
J8: 0.00
直交系座標
[deg,mm]
X: 266.91
Y: 467.91
Z: 434.97
A: -180.00
B: 0.00
C: -117.38
L1: 0.00
L2: 0.00
ハンド状態
ﾊﾝﾄﾞ1 : 開
ﾊﾝﾄﾞ2 : 開
ﾊﾝﾄﾞ3 : 開
ﾊﾝﾄﾞ4 : 開
ﾊﾝﾄﾞ5 : --
ﾊﾝﾄﾞ6 : --
ﾊﾝﾄﾞ7 : --
ﾊﾝﾄﾞ8 : --
ﾏｼﾝﾛｯｸ状態 : OFF
ｻｰﾎﾞ ON/OFF: ON
[mm/sec]
先端速度 : 0.00

図 40　動作状態