放電加工機遠隔操作

ワークの自動測定方式

指導教員　小林和彦教授

知能機械システム工学科

１０４１００３

白倉　友也

目次

第１章　緒言

放電加工は、金型加工を中心に普及し、難加工材料、難加工形状、あるいは放電加工の特性を生かし多くの用途を生み出している。また、数値制御技術も進み、加工速度、精度も向上していて、現在では精密加工分野ではなくてはならない加工法になっている。

他方では、パソコンの普及に伴い、インターネット技術は急速に発展し続けている。本研究は、これからのネットワーク社会で、必要とされるシステムとして、会社などから NC データを作成後、通信回線によりインターネットを通して遠隔地の機械に NC データを送信し、必要に応じて操作を行い、加工状態を確認する。このような放電加工の遠隔操作技術についての研究を行う。

また、従来のようなワーク基準面を決め、機械テーブルに平行にワークをセットアップする作業は非常に手間がかかり、熟練者でないと時間がかかってしまう。そこで今回、機械テーブルに平行にセットアップしなくても、ワーク座標及び傾きをカメラで検出して、画像処理を行い、そのセットアップのズレの補正をおこなって、誰もが簡単にセットアップできるようにするシステムの研究を行う。

1.1　放電加工について

ここで今回の研究に用いられる放電加工について説明する。

放電加工とは、銅などの柔らかい導電性の材料を電極として、焼きの入った金属や超硬合金など非常に硬い材料（工作物）を精度よく加工することができる。この場合、工具と材料は直接接触して加工するのではなく、数ミクロンから数十ミクロンの隙間で繰り返し火花放電を発生させることによって、ワークを溶融除去しながら加工する。放電加工は、一つ一つの放電が連続的におこなわれるが、それは 1 秒間に数千ないし数万の放電を発生させ、それによって生ずる多数の放電痕の累積によって加工がおこなわれる。一般に放電加工には、今回の研究でも使用する形彫放電加工とワイヤ放電加工がある。基本的にこれらの仕組みは同じであるが、用途に応じて使い分けられる。

形彫放電加工機は、特定形状の電極を用いて、その反対形状に対応するように投影加工をおこなうものである。加工された形状は、電極と放電クリアランス（数μm程度から数十μm）を隔てた現物合わせ形状となっている。このクリアランス精度は、主として電気条件と加工液の清濁の度合いによって決まる。また、加工形状が大きくてもクリアランスは均一に加工されるなどの特徴がある。

そのため、加工機を運転制御するためには、加工進度に応じて電極を加工方向に進行させる主サーボ機構のほかに、電極の見かけの寸法を拡大する方向（横方向）に相対運動を与える機構（揺動機構）などが必要となる。これらは、数値制御との組み合わせで動作制御される。

また、ワイヤ放電加工機の場合は、通常φ0.03~0.3mm のワイヤを糸のこのように巻き取りながら、加工物を固定した加工テーブルを数値制御装置によって駆動し、あらかじめプログラムした形状に加工を行う。加工される形状はAPT（Automatic Programming Tools）によってプログラム加工されるため、特定形状の電極を必要としない。一筆書きのできる形状であれば、どんな複雑な形状であっても、加工が可能である。
また、歯車のような特定形状曲線をもつ加工物は、専用のソフトウェアを使用してプログラムを作成できる。また、プログラムした形状は、数値制御のオフセットを与えることにより、抜き型のクリアランスを任意に変たり、はめあいの隙間を変えるような、寸法変化を与えることができる。

ワイヤ放電加工機の加工寸法は次のようにして定まる。

直径 0.2mmのワイヤで加工すれば、0.3mm 程度の加工溝幅が加工されるが、このワイヤ直径からの拡大量に再現性があるので、加工に先立って特定の電気条件における試し加工を行い、寸法拡大量とワイヤ直径に対する数値制御オフセットを行って加工すれば、プログラムに対応した加工寸法となる。

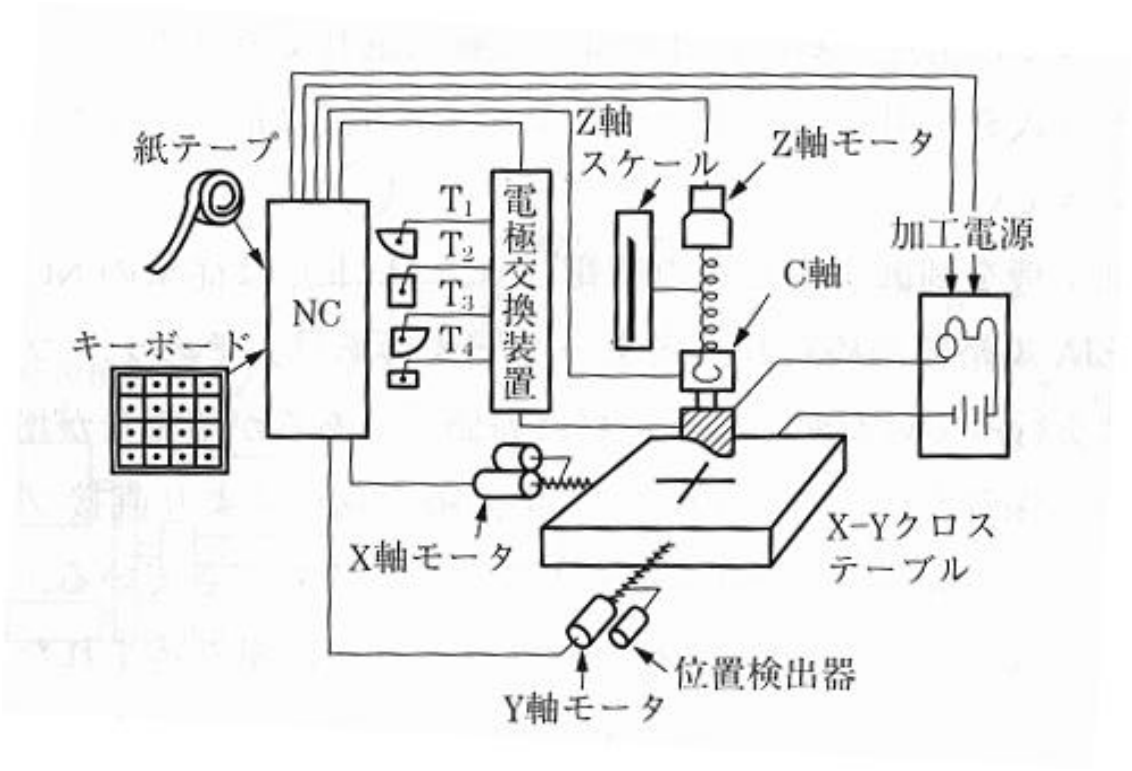

図 1.1 形彫放電加工機の原理

1.2 ワークの自動測定方式について

放電加工を行うためには、機械テーブルとワーク（加工物）を平行に取り付けなければならないため、非常に熟練性や時間を必要とする。そこで今回この作業を改善するため、機械テーブルとワークを平行に取り付けなくても精度良く加工が行えるワークの自動測定方式について研究を行った。

まず、機械テーブルのどこにワークが置かれているかカメラを用いて検出させ、その近くまで測定子を移動させれば、放電加工機の端面位置決め機能により自動的に測定子をワークの側面に接触させた後、図 1.2 のように測定子を X 方向に規定の距離だけ移動

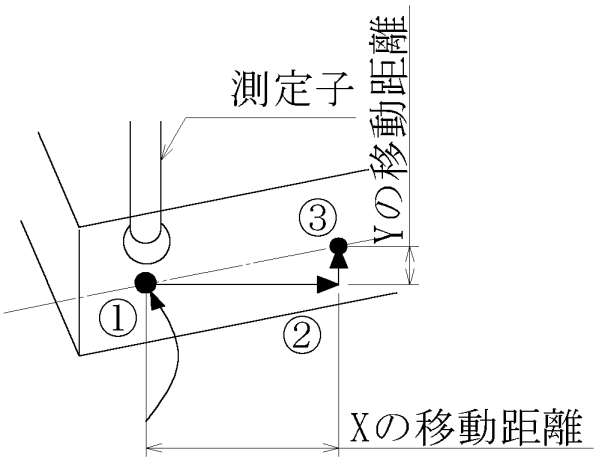

図 1.2 測定子を用いたワークの傾き角検出

させて、再び端面位置決めによりワーク側面に測定子を接触させて Y の移動距離を検出する。以上の X、Y の移動距離から、次の式が成立する。

$$\mathrm{Tan}\,\theta = \text{Y の移動距離} / \text{X 移動距離}$$

そこで、θが小さいときは tanθ　θであるから、ワークの傾き角θが求まる。

以上のようなカメラによるワーク座標認識から測定子を用いたワークの傾き角検出について研究を行った。

1.3 放電加工機遠隔制御について

現段階においては、三菱電機製の放電加工機遠隔操作ソフト DNC（ワイヤ放電加工機）図 1.1 により、インターネット通じて変化する加工状態をモニタリングするとともに，必要に応じて直接機械を操作することができるシステムはすでに確立している。

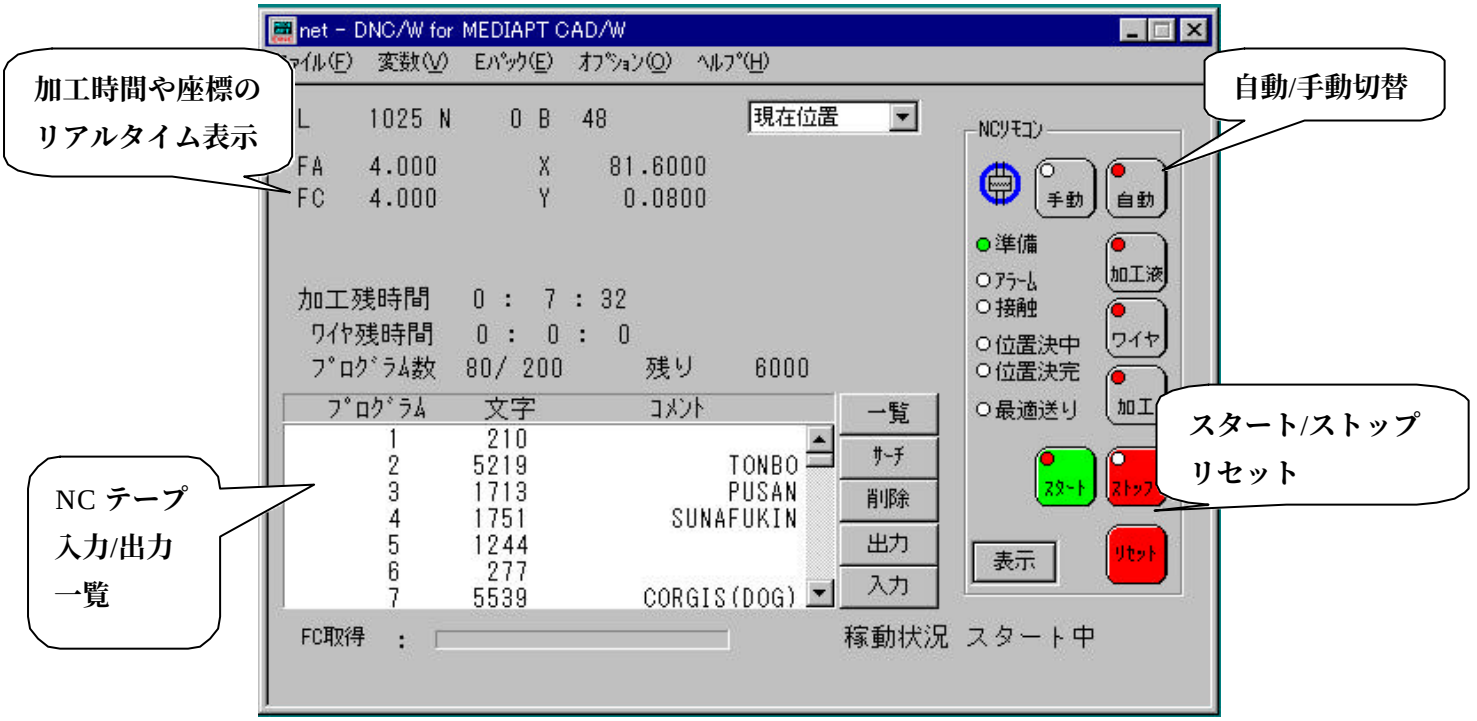

図 1.3 DNC によるワイヤ放電加工の遠隔操作画面

今回はワークの自動認識を中心として、NC ソケット通信プログラムによるコンピュータから放電加工機へのデータ転送について研究を行った。

そこで、まず今回遠隔操作に用いたソケット通信について簡単に説明を行う。

ソケットとはアプリケーションをネットワークに接続するため端点であり、アプリケーションをネットワークに接続させるものである。その通信方法はクライアント/サーバー方式で行われる。それは、一方のアプリケーションを常に使用可能な状態に置き（サーバー）、もう一方のアプリケーションから必要に応じてサービスを要求する（クライアント）となっている。

サーバーはソケットを作成し、それに名前（IP アドレス、ポート番号）を付けて、クライアントが識別できるようにする。その後サービスの要求を待機する。クライアントはソケット作成し、サーバーソケットの名前を識別してから、「プラグイン」して対話を開始する。その後はデータを双方に送信することが可能となる。

第２章　従来技術と本研究との関連

2.1　放電加工制御用 NC 技術について

本研究は、NC 制御技術の発展により、現在技術である本研究へとつながるので、まず放電加工機における NC 制御技術の変遷について述べていくことにする。

NC 装置は軌跡を数値で表現し、この数値に従って駆動装置であるモータなどを動かす位置制御が主な役割であるが、1960 年代に普及しはじめて以来、制御技術、センシング技術が進化するとともに加工制御装置に成長してきた。

図 2.1 に NC 制御技術の進化の過程を示す。

入力された形状情報にモータなどの駆動装置を忠実に位置制御することを主体とした NC 装置を第 1 世代としている。しかしながら、ワイヤ放電加工機はプログラムされた図面形状どおりに位置制御を行うだけでなく、放電加工機特有の短絡現象などを回避する最適制御を当初より取り入れていた。また、金型加工用途が多く、比較的複雑な加工形状のプログラミングが一品一様で要求されたため、ミニコン NC 上に自動プログラミングシステムを搭載するなど、すでにこの当時、一般工作機械の進化の方向を示す機能を備えていた。ワイヤ放電加工法は NC 制御技術なしでは工作機械としての活用が困難な加工法であり、そのため NC 制御技術の先駆的役割を果たせたのだと考えられる。

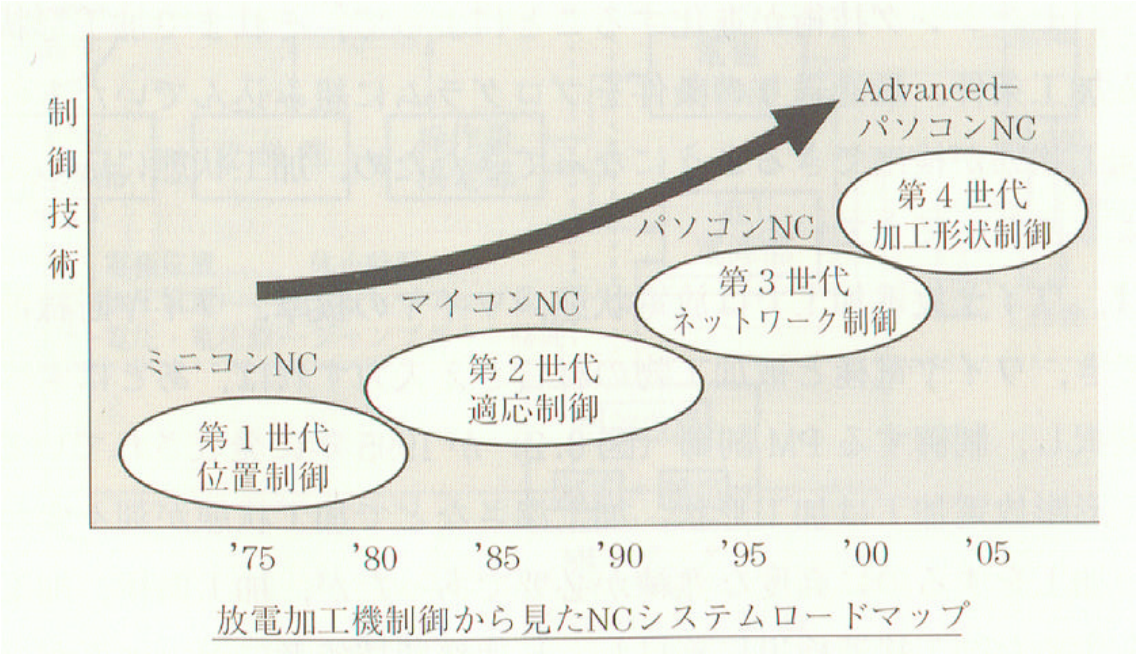

図 2.1　NC 制御技術の進化

次に放電加工における NC 制御技術について説明を行っていく。

図 2.2 には NC 制御装置のブロック図を示す。NC 制御部は入力情報を解析し、シーケンス処理あるいは演算処理する演算制御部と機械の動きを制御するサーボ系に大別できる。

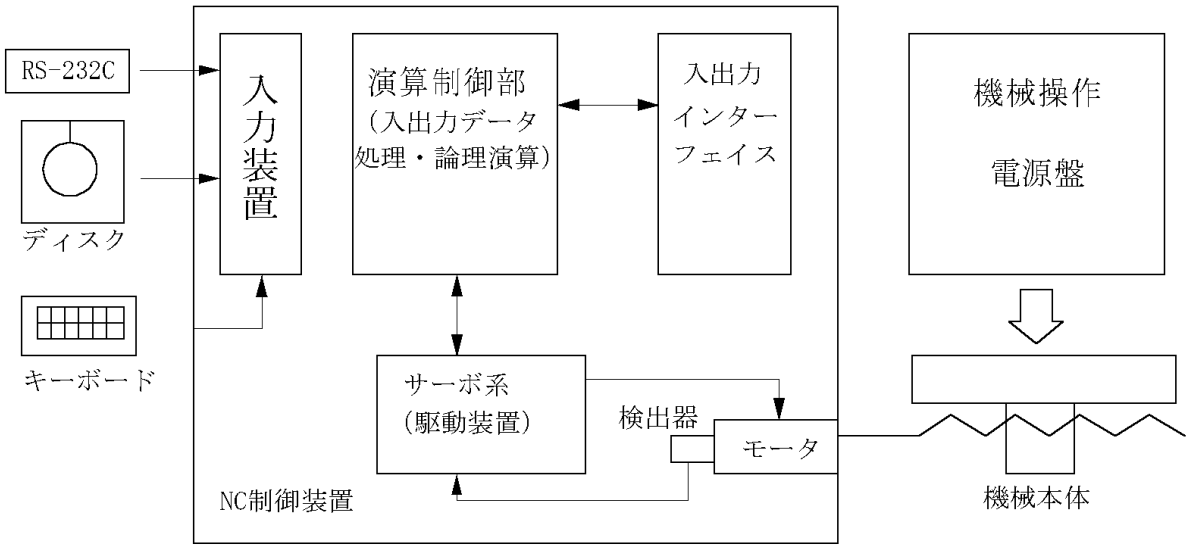

図 2.2　NC 制御装置ブロック図

2.2 遠隔操作について

加工機を制御するための情報（NC コード）は、紙テープに EIA 企画で決められた穴を一文字ずつあけ、そのテープ情報を読み取って制御を行っていた。しかし、紙テープは安価ではあるが情報量が増加するに従い、かさばり保管が大変なためディスク、RS232C により直接プロミング装置から転送するなどの方法がとられた。

現在では、遠隔地から NC データを作成後、電話回線を通して加工機に NC データを送信し、加工を行い、加工の状態を確認するといった方向に進み、加工機の遠隔操作機能のソフトが確立している。その様子を図 2.3 に示す。

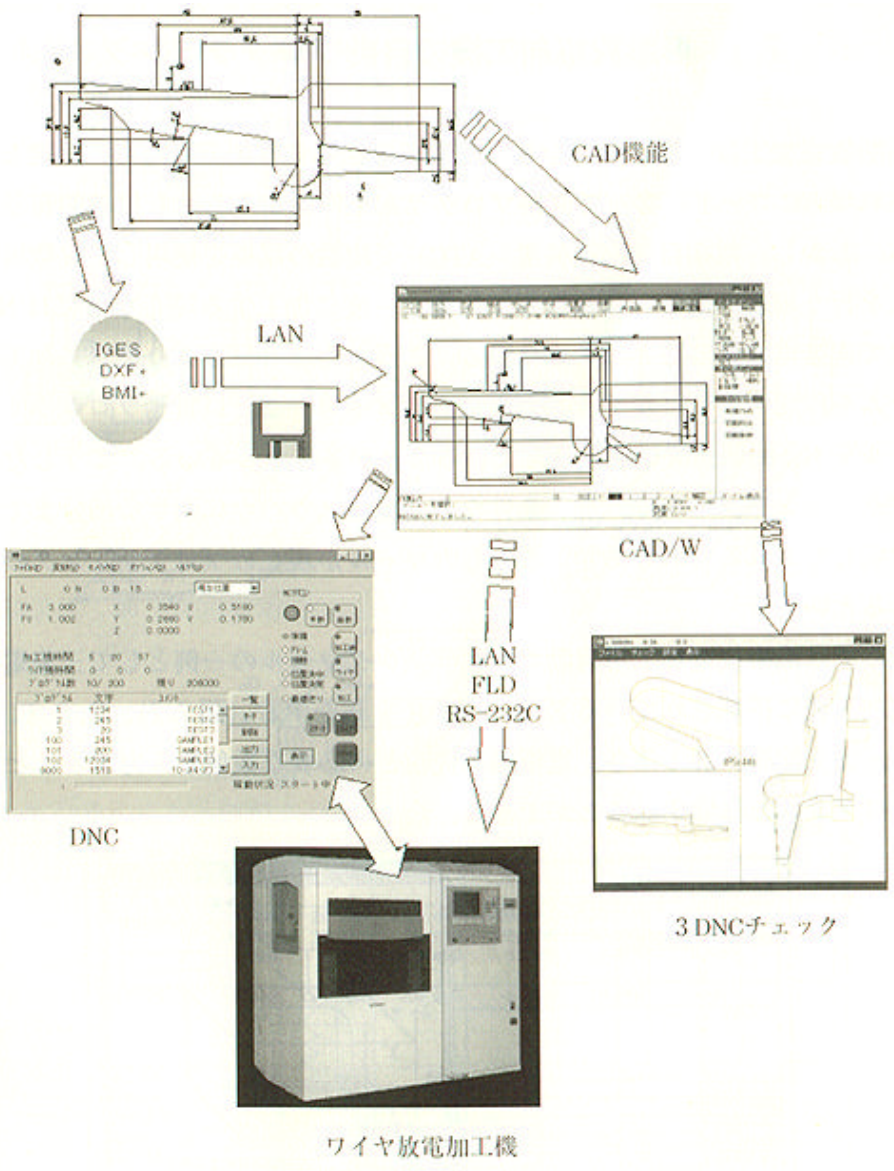

図 2.3 従来の遠隔操作技術の流れ

2.3 ワークの位置決めにおける従来技術

放電加工機は、電極とワークを精度よくセットアップすれば、その後の加工は自動運転により、加工精度の高いものが可能となる。従来のワークの取付け方法は、他の工作機械と同じように、機械テーブル上にセットアップするワークの X、Y 座標基準面を、テーブルの移動方向座標に完全に平行にセットアップすることが要求される。この作業はかなりの時間と熟練を要する場合が多い。

一方、金型加工は、毎回形状の異なった一品加工が多いため、セットアップを容易にするためのパレットや特殊治具を使うことが少なく、ワークのセットアップにはかなりの時間と熟練を必要としているのが現状である。

本研究で行う技術は、上記の作業を自動化することを目指しており、これが実現すれば自動段取りが可能となり、ワークのセットアップと基準だし作業において、大きな技術的進歩が期待できる。

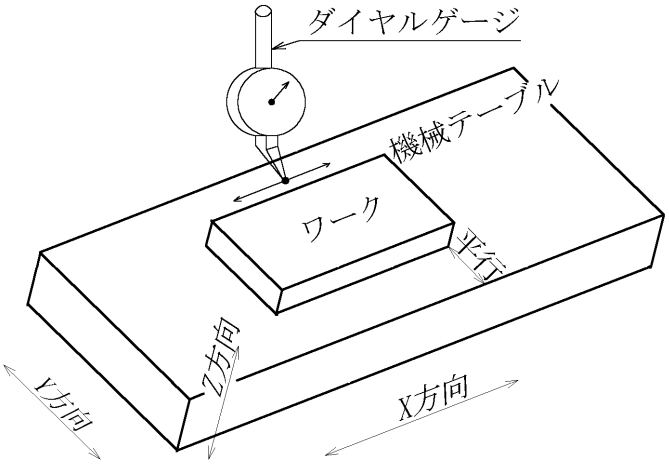

図 2.5 従来のワークの位置決め

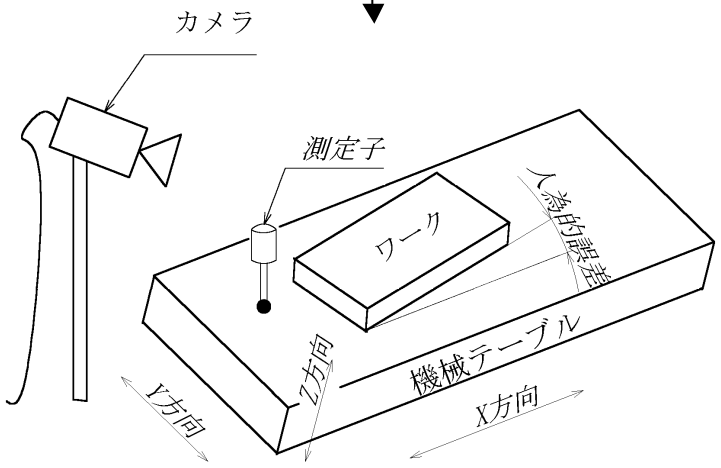

図 2.6 カメラを用いた人為的誤差の補正

第３章　ＮＣ 装置によるワークの傾き角自動測定

3.1　研究目的

放電加工機での段取り作業においてワークのセットアップは、まず、ワークとテーブルを平行に取り付ける必要がある。現在ではダイヤルゲージを使用してワークのセットアップを行っているが、この作業は、非常に手間がかかり、また熟練と経験を要するかなり難しい作業である。

そこで誰もが簡単にワークの取付けを行うようにするには、カメラによってワークの座標を取得し、その付近まで測定子をもっていき、ワークの傾き角を測定（1.2 参照）する。そして、座標回転機能で加工したい図形をワークの傾き角度分だけ回転させればよい。

本研究は、まずワークが機械テーブル上のどこに置かれているか、すなわちワークの位置と大まかな形状をカメラ用いて認識する。次に、ワークの傾きを自動的に測定し、（図 3.1）その測定データに基づいて補正を行うということである。ワークの位置と傾きを求めるために、本章では、Sherlock32 という画像処理シュミレーションソフト 2 種類を使用して、解析アルゴリズムの研究開発を行った。

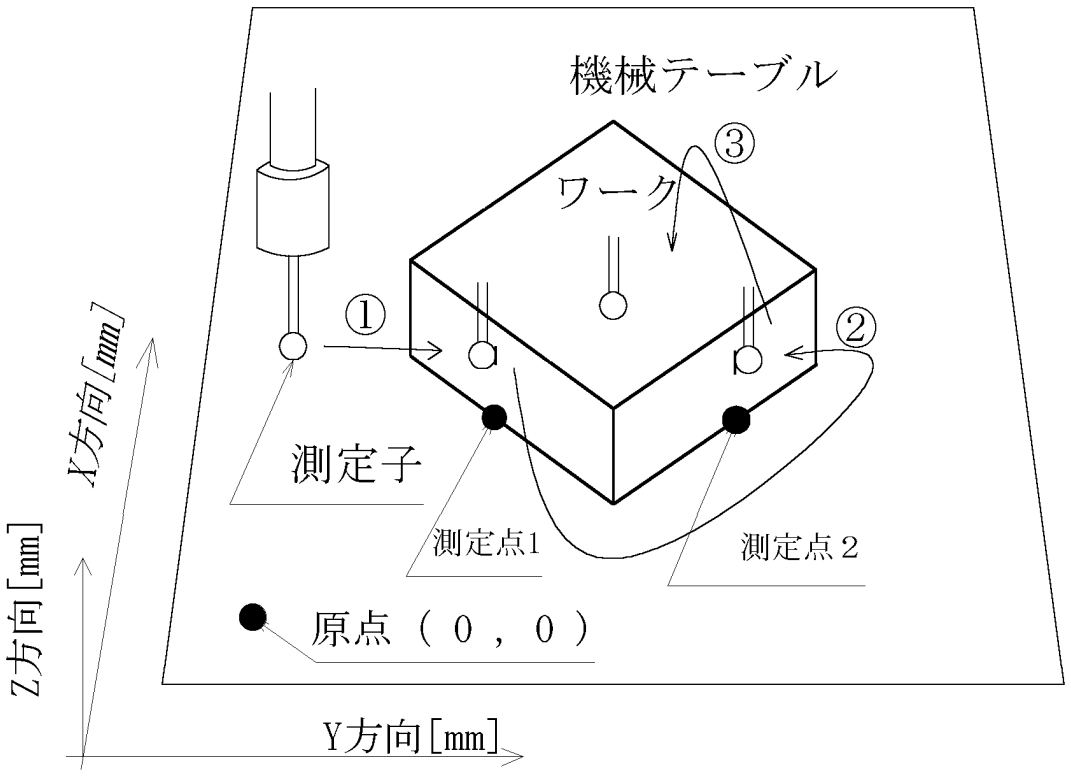

図 3.1 測定子を用いたワーク座標の測定方法

3.2 検出システム

今回は、放電加工機の機械テーブルの代わりに方眼紙を使用し、その上にワークを置いて測定を行った。そして、ワーク全体をカメラ(図 3.3 参照)で写し、画像取り込みボード（ITI 社製　IC-2COMP）で画像をパソコン(図 3.4 参照)に取り込んだ後、パソコン上の画像処理シュミレーションソフト Sherlock32 により、ワークの特徴点座標の検出を行った。

そのイメージを図 3.2 に示す。

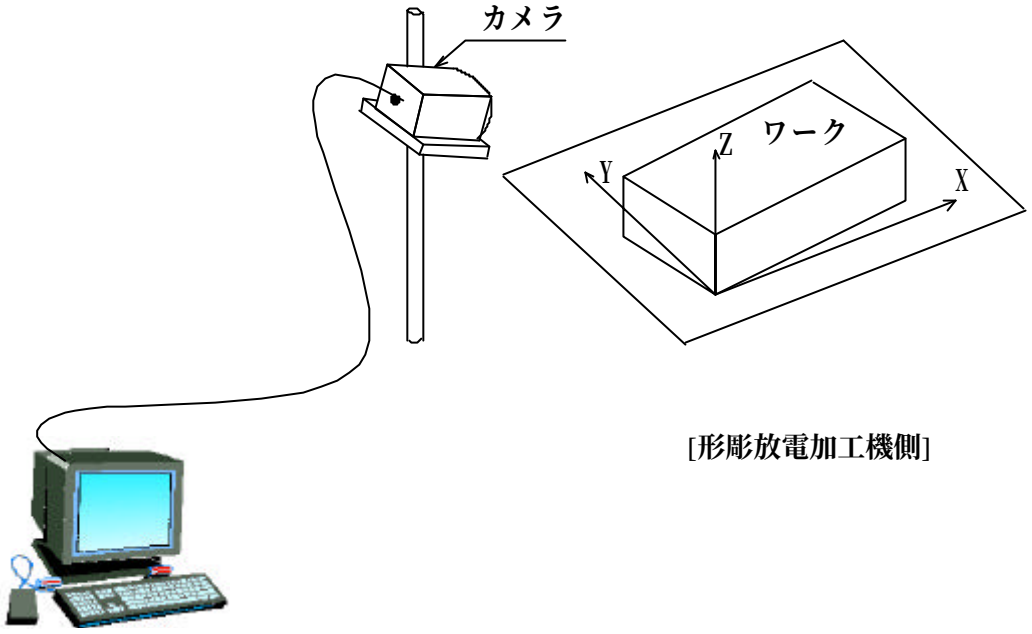

図 3.2 ワーク座標検出システム

図 3.3 カメラ

カメラ　SONY 製 EVI-G20(NTSC)

有効画素数[768(H)×494(V)]

図 3.4 OptiPlex GX1

パソコン　Dell コンピュータ製

CPU：PentiumⅢ600[MHz]

メモリ：128[MB]

3.3 解析アルゴリズムの開発

座標検出を行うための座標検出シミュレーションを、画像処理ソフト Sherlock32(図 3.5 参照)を用いて行い、そこで有効と思われる 2 つの解析アルゴリズムについての評価を行った。

今回座標検出精度の目標値は、座標１、座標２、ワーク高さ(Z 値)について共通に誤差 3[mm]以下を目標とした。

a)“SmART Search”による解析アルゴリズム

この解析アルゴリズムは一種のパターンサーチ処理なので、ワークに特徴をもたせて、検出精度を上げる必要がある。そこで、今回はワークに特徴線を付けて検出を行った。

b)“Line”による解析アルゴリズム

この解析アルゴリズムは緑の線上でワークの角(エッジ)の暗と明の変化を見つけて、その座標を検出させる解析方法である。そのためワーク側面の上下の角に黒い線を引くことで暗と明の変化を強調させて検出精度を向上させた。

図 3.5　Sherlock32&MVTools

Imaging Technology 社製　画像処理ソフト Sherlock32＆MVTools

3.4 解析アルゴリズム

今回ワークの自動測定用に 2 種類の解析アルゴリズムについて開発を行った。その座標検出の精度については誤差 3[mm]以下を目標とした。

① SmART Search による座標検出

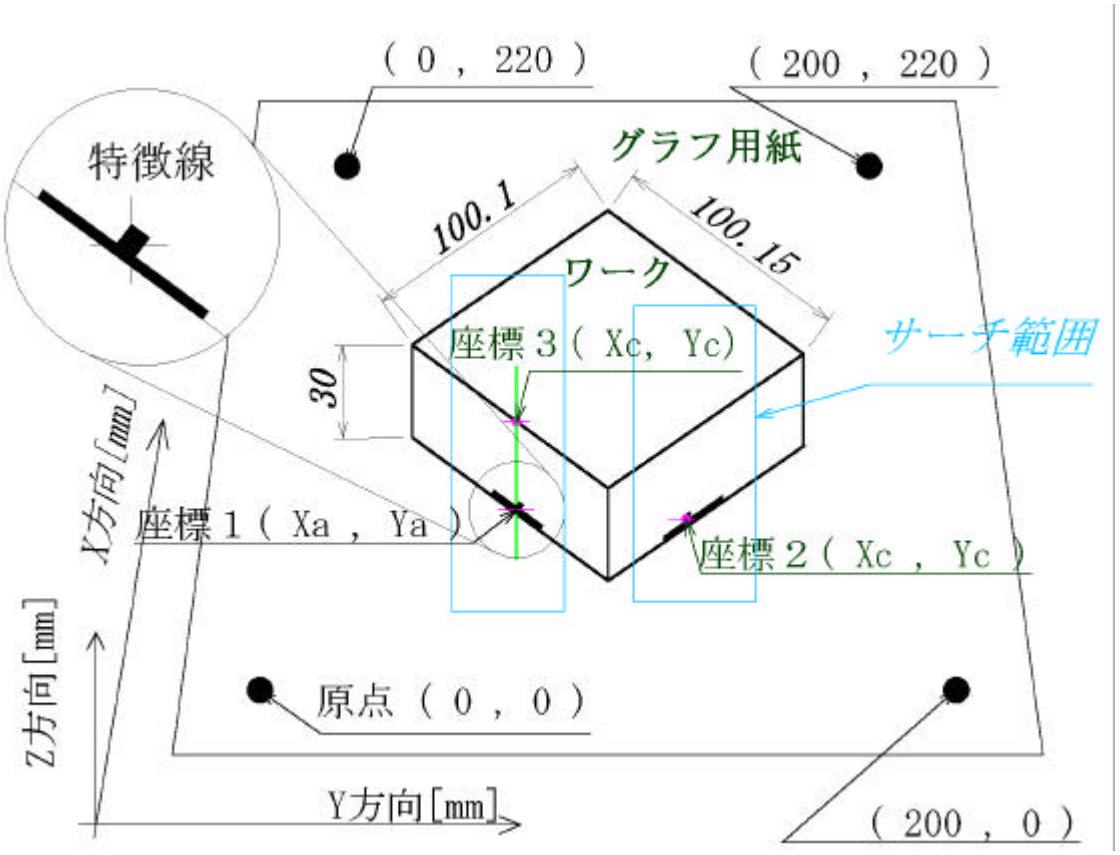

図 3.6 SmART Search に座標検出

検出方法：

この解析アルゴリズムは一種のパターン・サーチ処理を行っているので、ワークに特徴を持たせることが重要となる。そこで、図 3.6 にあるような特徴線をワークに付けることにより特徴線を強調させた。

1)まず、画面上にある 4 つの基準点により、カメラ座標から実寸座標に変換処理を行う。

2)そしてワーク側面下にある特徴線をスマート・サーチに登録を行い、青いサーチ範囲内で特徴線を探す。

3)特徴線が見つかるとサーチが終了して、その位置の**座標 1** が表示される。

4)**座標 2** に対しても同じように処理を行い、座標の検出を行う。

5)ワークの高さ(Z 値)の検出方法は、緑の線上の始点から終点に向かって、暗と明の変化を探す（つまり、ワーク側面の角を見つける）ことにより座標 3 が検出する。

6)座標 3 と座標 1 の距離を関数によって求め、それを**ワーク高さ（Z 値）**とする。

② ワーク側面に着色を加えて特徴点をつくり、サーチ処理による検出

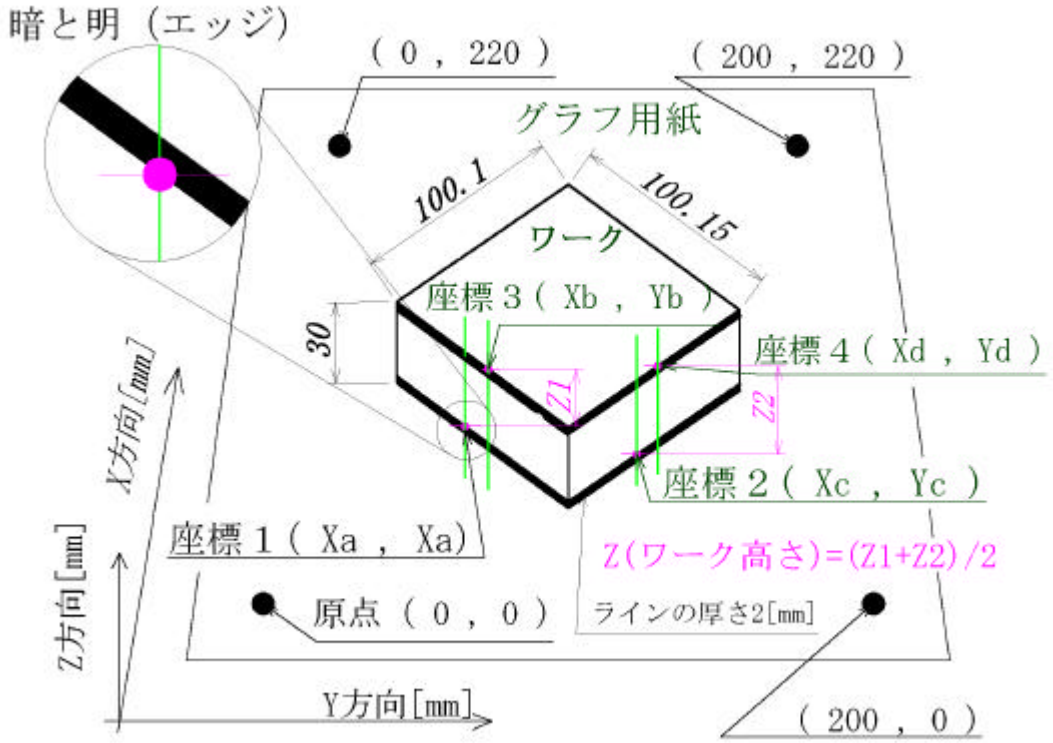

図 3.7 Line による座標検出

検出方法：

まず、解析処理を行う前に、ワーク側面上下の角に黒線を引くことにより、暗と明の変化（エッジ）をより強調させる。

1)前者の SmART Search 解析同様に 4 つの基準点より、カメラ座標から実寸座標に変換を行う。

2)次に緑線の終点（下）から始点（上）に向かって、暗と明の変化を探し座標１を検出する。

3)逆に始点（上）から終点（下）に向かって、暗と明の変化を探し座標 3 を検出する。

4)同じように処理を座標 2、4 にも行って、座標を検出する。

5)検出された座標３、座標 1 の距離を関数によって求めた値を Z１とする。

6)同じように座標 4、座標 2 の距離を求め、その値を Z2 とする。

7)以上で求まった Z1 と Z2 の平均値が、ワーク高さ（Z 値）となる。

3.5 前処理

Sherlock32 は、最大 5 つの段階の画像プリプロセッサ演算子（Invert, Threshold, 1×1, Dilate など）をサポートしている。プリプロセッサのダイヤログ・ボックスを使って演算子を設定する。プリプロセッサとは、あらかじめ決められた方法に従ってピクセル値を変更することである。

今回使用する、前処理のプリプロセッサ演算子は Threshold、Sharpen、Smooth、Median などである。次にこれらのプリプロセッサ演算子について説明を行う。

１）Threshold(2 値化処理)

しきい値より大きいピクセル 255 に設定する。他のすべてのピクセルは 0 に設定される。

パラメータ：

Threshold value 0-255　0～255 のしきい値

実行しますか？（1=はい、0＝いいえ）

２）(3×3）Sharpen（3×3 画像鮮明処理フィルタ）

Sharpen フィルタはエッジを鮮明にする。一定の輝度の領域は変わらない。Sharpenは､カーネルを使って 3×3 コンボリューションを適用することと同じである。

パラメータ：

実行しますか？（1=はい、0＝いいえ）

３）Smooth（3×3 ローパス・フィルタ）

Smooth は、画像からノイズや不要な値を除去する単純な移動平均型（非再帰的）ローパス・フィルタである。各ピクセルは、そのピクセル自身とその 3×3 の隣接ピクセル内にある 8 個のピクセルの平均値に置き換えられる。この演算子にはエッジをぼかす効果もある。この演算を何回も行うと、Area Peek 内の各ピクセルが領域のグレイ・レベルの平均値に設定される効果ある。

パラメータ：

Number of Times(maximum 10)　回数（最大 10）

４）Median（フィルタ処理）

Median フィルター演算子は、各ピクセルを、そのピクセルとそのピクセルに隣接する８個のピクセルの中央値に置き換える。中央値は、そのソートされた 9 個の隣接ピクセルの 5 番目の値である。

パラメータ：

実行しますか？（1=はい、0=いいえ）

3.6 解析演算子

Sherlock32 には、解析演算子を多数持っており、その処理対象に応じて最適な解析アルゴリズムに組み込まなければならない。本研究では、3.3 章で考えた 2 つの解析アルゴリズムについて、有効と思われる解析演算子をいくつか用いた。

解析演算子は、解析エリアのとりかたによって使用できるものが異なる。解析エリアは、四角領域内のピクセルを調べるもの（Area Peek）と線上の１ピクセル幅の領域を調べるもの(Line Peek)、その他にもいくつかの演算子があるが、今回は **Area Peek** と **Line Peek** を使用し、主な解析演算子は **Calibrate 演算子**、**Smart Search 演算子**、**First Edge 演算子**、**First Back Edge 演算子**を利用した。

次に今回使用した解析演算子３つについて処理の処理内容を説明していく。

１) Calibrate（座標変換演算子）

カメラ・キャリブレーションとは、カメラのピクセル座標系と実寸座標系の変換を行う。それは、４点座標系機能を使って、１つの画面に対して４箇所の位置を設定する。カメラ内の４箇所の位置と実世界座標が関連づけられると、Sherlock32 は内部の数値変換機能により、スケール、オフセット、回転、視覚的な歪みを補正するための座標変換係数を確立します。キャリブレーションが確立されると Stakeout Options から Calibrate same as オプションを選択することにより、他の画面にもその係数を利用することが可能である。つまり、１つのキャリブレーションの結果を多くのの画面で利用することができる。

以下に座標系編集ダイアログを表しているが、ここに実際の座標を入力する。

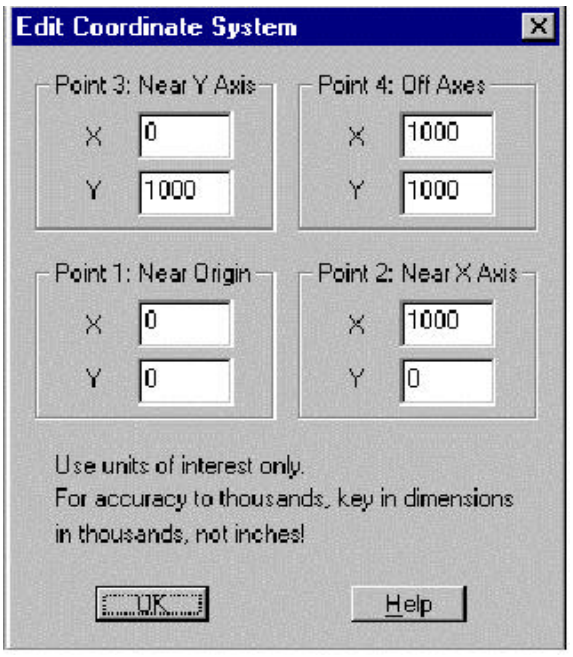

図 3.8　キャリブレーションのエディトダイアログ画面

２) Smart Search（Smart Alignment and Registration Tool）

スマートサーチは、サブピクセルの精度と他界信頼性で、登録パターンを高速で検出ツールである。SmART Search **は、パターンの回転/スケール変化/遮蔽/欠落や、非線形反射、反転、酸化、成膜、さまざまな輪郭タイプ画像などの実世界の問題を処理するよう設計されている。**

SmART Search **は、２つのサーチ・エンジンと２つのサブピクセル・サーチ・カーネルよって動作する。これらのサーチ・エンジンは「コリレーション・サーチ」と「幾何学サーチ」である。サブピクセル・サーチ・カーネルは、「標準サブピクセル・カーネル」と「**SmART **サブピクセル・カーネル」である。良好な（クリーンな）パターンの場合、標準サブピクセル・カーネルは、約** 1/40 **ピクセルの精度でパターンの位置をサーチすることができる。**

SmART **サブピクセル・カーネルは、劣化したパターンでも、**1/60 **ピクセル以上の精度でパターンの位置をサーチすることができる。**

「コリレーション・サーチ・エンジン」

コリレーション・サーチ・エンジンは、「通常の」テクスチャをベースにしサーチ・テクノロジーである。パターンは、そのクレースケール強度によって定義される。そのサーチプロセスは、登録パターン内の強度とサーチ画像内の考えられるインスタンスの比較である。サーチストラテジーを「スパイラル」と呼ぶ。パターンの考えられる位置は、登録パターンすなわちモデルの数ピクセルを使って再びチェックされ、得点の低い位置は排除される。このプロセスは、さらに多くのピクセルを利用し、パターンのインスタンスが見つかるまで１ピクセル単位の精度で繰り返される。その後、いずれかのサブピクセル・サーチ・カーネルが、サブピクセル制度でパターンの位置をサーチする。パターンが劣化していない場合、コリレーション・ベース・サーチ法の精度は、通常、1/40 **ピクセルである。図** 3.8 **は、登録パターンとその一致候補を示している。**

各一致候補は、最良の一致候補が見つかるまでさらに細かなステップで再評価され、サンプリングが繰り返される。

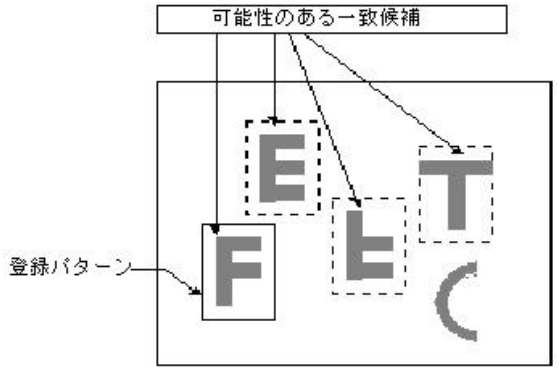

図 3.9 **登録パターンとその一致候補**

「幾何学サーチ・エンジン」

Geo Search(幾何学サーチエンジン)は、パターン検出の新しい概念である。Geo Search は、パターンの輪郭情報（エッジ・ピクセル）を抽出し、輪郭ピクセル間に幾何学的関係を形成する。例えば、円は円のあらゆる点が円の中心点から等距離上に位置した幾何学形状である。Geo Search は、実世界の画像に発生する問題（パターンの欠落や遮蔽、輪郭形状パターン、非線形反射）を確実に処理する。Goe Search は１ピクセル・サーチ・カーネルが、高いサブピクセル精度でパターンの位置をサーチする。

パターンのばらつきや劣化、照明の変化に左右されない幾何学ベース・サーチ法の精度は、通常 1/60 ピクセルである。図 3.9 は、円形の登録パターンと、その輪郭点間の関係を示している。輪郭の各点は、円の中心から等距離上にある。

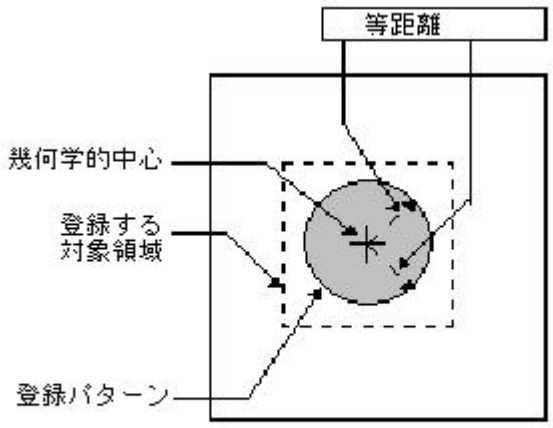

図 3.10 円形の登録番号と輪郭点間の関係

「標準サブピクセル・カーネル」

標準サブピクセル・カーネルは、近傍演算を行ってサブピクセル精度でパターン位置を計算する。標準サブピクセル・カーネルは、パターンが良好であれば 1/40 ピクセルの精度がある。精度はパターンの劣化に比例して低下する。

「SmART サブピクセル・カーネル」

SmART サブピクセル・カーネルは、近傍演算を行い、ニューラル・ネットワーク技術を利用してサブピクセル精度でパターン位置を計算する。SmART サブピクセル・カーネルは、実世界の画像の場合、1/60 ピクセル以上の精度がある。欠落、遮蔽、残骸、非線形反射といった劣化でも精度は変わらない。

３） Connectivity(ラベリング)

指定された値のピクセルの連結されたすべての「ブロブ」を探す。２値化後、白いブロブを検出するには、色値 255 のブロブを探ようにする。黒いブロブを検出するには、色値 0 のブロブを探す。

Reading：

Number of blobs found(N[]) **ブロブの面積**

Centroid of blob 0(P) **最大ブロブの重心**

Centroids of blobs(P[]) **ブロブの重心(配列)**

Top-left corner of bounding box for blob 0 (P)

最大ブロブの外接四角形の左上の座標

Top-left corner of bounding box for each blob (P[])

各ブロブの外接四角形の左上の座標

Heights of blobs (N[]) **ブロブの高さ**

Widths of blobs(N[]) **ブロブの幅**

Perimeters of blobs(N[]) **ブロブの周囲長**

Occupation Ratios of blobs(N[]) **ブロブの占有率**

Circularities of blobs(N[]) **ブロブの真円度**

NE/SW of blobs(N[]) **ブロブの** NE/SW

NW/SE of blobs(N[]) **ブロブの** NW/SE

NN of blobs(N[]) **ブロブの** NN

N of blobs(N[]) **ブロブの** N

S of blobs(N[]) **ブロブの** S

SS of blobs(N[]) **ブロブの** SS

C of blobs(N[]) **ブロブの** C

WW of blobs(N[]) **ブロブの** WW

W of blobs(N[]) **ブロブの** W

E of blobs(N[]) **ブロブの** E

EE of blobs(N[]) **ブロブの** EE

NE/SW2 of blobs(N[]) **ブロブの** NE/SW2

NW/SE2 of blobs(N[]) **ブロブの** NW/SE2

パラメータ：

Blob quantity – number of blobs to find **ブロブの数量：検出するブロブの数**

Blob color：255=white, 0=black **ブロブの色：**255=**白、**0=**黒**

Blob selection constraints **ブロブの選択条件**

Largest(high)blob area (pixels) **最大のブロブ領域（高）（ピクセル）**

Smallest(low)blob area(pixels) **最小のブロブ領域（低）（ピクセル）**

Tallest(high)blob height(pixels)　最も高いブロブの高さ（高）（ピクセル）
Shortest(low)blob height(pixels)　最も低いブロブの高さ（低）（ピクセル）
Widest(high) blob width(pixels)　最も広いブロブの幅（高）(ピクセル)
Narrowest(low) blob width(pixels)　最も狭いブロブの幅（低）（ピクセル）
Allow blobs touching edge(yes/no)
ブロブが画像端に触れてもかまわない　yes/no

Readings to return　（返す Reading 上記参照）

Sort reading by(Reading のソート基準)：-1=no sort(ソートしない)、0=size（サイズ）、1=X, 2=Y, 3=Column(列)

4=Row(行)、 5=X+Y 、 6=X-Y 、 7=H 、 8=W、 9=Perimeter(周囲長)、10=Occupation Ratio（占有率）、11=Circularity（真円度）、12=NE/SW、13=NW/SE

Sort direction(ソート方向)：0=lowest to highest(低→高)、1=highest to lowest(高→低)

Connectivity は Smallest と Largest の間の面積(ピクセル・カウント)、Shortest と Tallest の間の高さ、Narrowest と Widest の間の幅を持つブロブだけを考慮する。この制約条件を無視するには、これらのすべての値を“－１”に設定する。

Connectivity を使うと、Area Peek のエッジに接触するブロブをフィルタ処理することもできる。常に、制約条件を満たすブロブの総数が最初に返される。Connectivity は、必要に応じて、面積、重心、囲み枠の高さ/幅、周囲長、占有率、真円度、および一連の構造解析パラメータ（オプションの戻り値）も返すことができる。

周囲長は、同じ色の他のピクセルに完全には囲まれていない、ブロブ内のピクセルの数として定義される。

占有率は、連結ピクセルが含まれている囲み枠のパーセンテージである。たとえば、正方形のブロブは占有率が 100 になるが、細い線のブロブは、占有率が 10 より小さくなる場合もある。

真円度は、ブロブが円形であるとして仮定して、面積から計算される半径の比率に対する、周囲長から計算される半径の比率として定義される。この値は、完全な円の真円度が 100 になるように 100 倍される。面積の測定値は周囲長の測定値よりかなり再現度が高いので、実際の円の真円度が 100 に近くなることはほとんどない。しかし、さほど円形に近くないブロブは常に、より円形に近いブロブより真円度が低くなる。

Connectivity 解析は、オブジェクトの検出と分類に使う。互いに 8 方向連結する（つまり、3×3 のリージョン内で 8 個のどの隣接ピクセルも接触する）ことによってすべてのピクセルが関連した、１つまたは複数の明るいピクセルの島を探すことによって、オブジェクトの中心、面積、その他のパラメータを速やかに検出することができる。

４）**First Edge**

線の始点から終点に向かって、指定された強度より大きい最初の暗→明のエッジ変化を探す。線に沿って最初の暗→明のエッジ変化を探すには、正のエッジ強度を使用する。明→暗のエッジの変化を探すには、負のエッジ強度を使用する。

Reading：

Edge point(P)　エッジ点

Strength of edge(N)　エッジの強度

Distance from start of line to edge(N)　線の始点と終点の間の距離

パラメータ：

Minimum edge strength　最小エッジ強度

５）**First Back Edge**

線の終点から始点に向かって（後ろむきに）最も鮮明な暗→明のエッジ変化を探す。指定された強度より小さいエッジは無視する。

Reading：

Edge point(P)　エッジ点

Strength of edge(N)　エッジの強度

Distance from start of line to edge (N)

パラメータ：

Minimum edge strength　最小エッジ強度

3.7 測定方法

測定方法は、カメラをワークの斜め上方から取り込む方法で行った。
図 3.11 と図 3.12 はカメラとワークの距離関係を記載する。

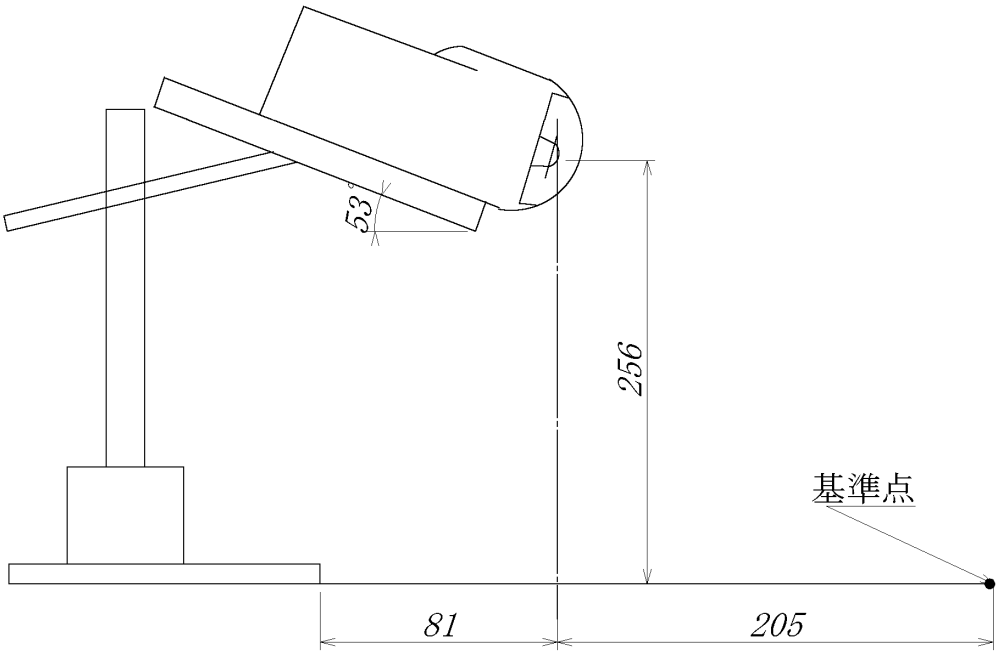

図 3.11　カメラの取り込み角度と基準点までの距離[mm]

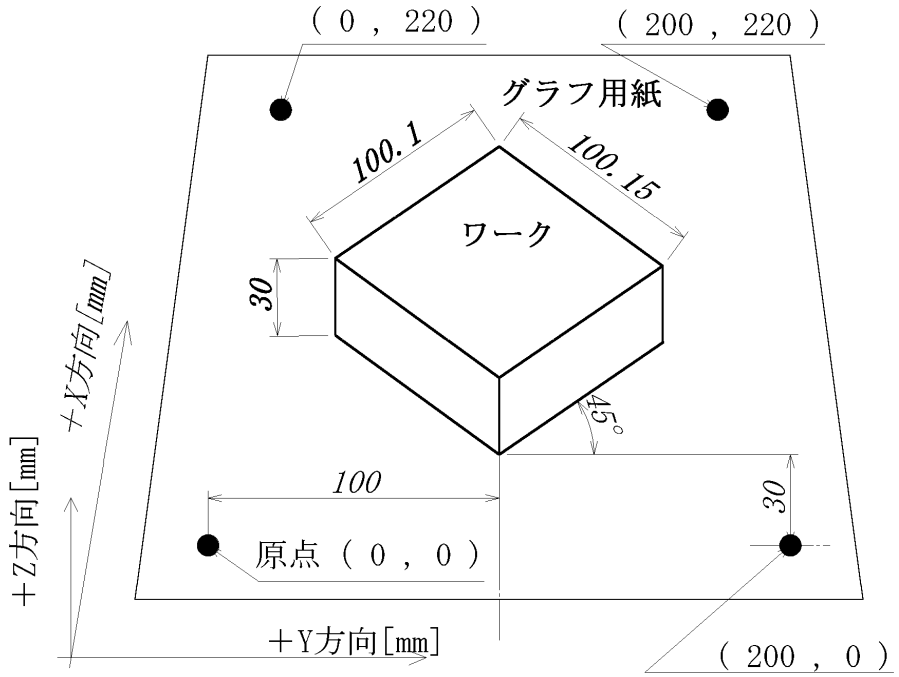

図 3.12　基準点からワークまでの距離[mm]

3.8 SmART Search による解析アルゴリズム１

（１）解析方法

まず、ワークに特徴をつけるために、ワーク側面に図 5.13 のように黒の油性ペンで着色を行った。

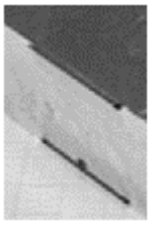

図 3.13　特徴線

次にピクセル座標から実寸座標へ変換するために、各 4 点にエリアを作り、そのエリアの中から、Connectivity 解析演算子を用いて、黒いブロブを見つける。
その後、Calibration 座標変換演算子を使い、ダイアログから 4 点の実寸座標入力を行った。図 3.14 がその様子を示している。

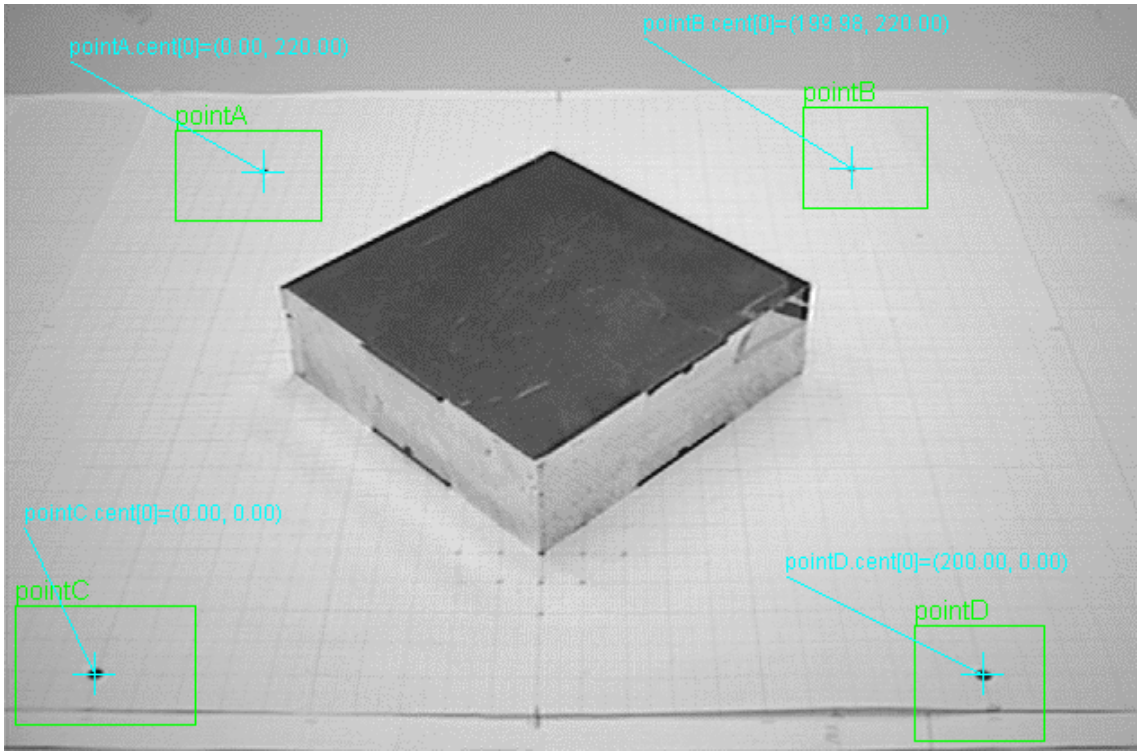

図 3.14　Calibration による実寸座標変換の様子

Calibration による座標変換後、解析演算子 SmART Search による処理を行った。この処理は 5.6（２）で説明したように、あらかじめ指定した領域内の輪郭（特徴点）を登録して、エリア内で同じ形状の画像が現れたら、そこをサーチする、といったものである。今回は座標１、２に加えて、Z（ワークの高さ）を測定するため、SmART Search と line の両方を用いた。（Line については 5.9 で説明を行う。）

SmART Search での登録方法はGeo Search と Correlation Search があるが、登録したエッジが不鮮明であったので、Correlation Search（図 5.15）を使用した。

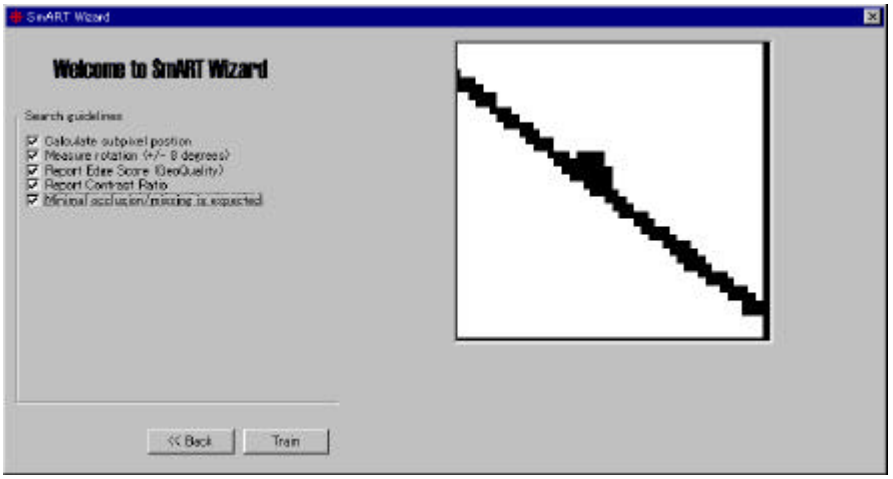

図 3.15　Correlation Search による特徴点の登録

登録後、エリアを広げて検索範囲を広くした。あとは、解析演算子が与えてくれた値を変数に入れ、モニターに表示した。これら一連の操作でワーク座標の検出が行えた。

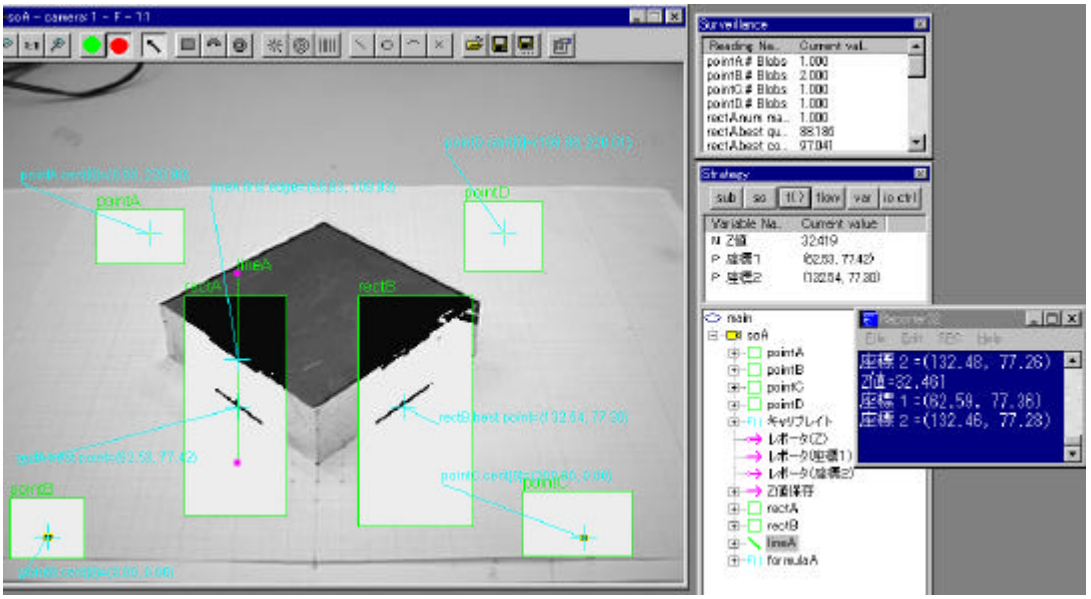

図 3.16　SmART Search による座標検出

レポーターを使用したワーク座標の検出結果表示（図 3.17）

図 3.17 ワーク座標の出力

（２）解析結果

このアルゴリズムを評価するために、実際に使用して解析を行った。

その結果を、グラフから読み取った実寸座標と解析によって計測された計測値とを比較したものを表 3.1 に示す。

表 3.1 測定結果

	測定値[mm]	実寸値[mm]	誤差[mm]	誤差[%]
座標１X[mm]	63.2	64.5	-1.3	-2.0%
座標１Y[mm]	66.64	68	-1.36	-2.0%
座標２X[mm]	123.38	121	2.38	2.0%
座標２Y[mm]	84.13	86	-1.87	-2.2%
ワークの高さ Z[mm]	31.54	30	1.54	5.1%

測定結果より、ワーク座標は誤差 3[mm]以内の目標値をクリアしており、ワークの高さについては誤差 2[mm]以内で

この SmART Search 解析にかかった時間は 11.567[msec]であった。

（３）考察

この解析アルゴリズムは目標値（測定誤差を 3[mm]以下）をクリアしているので、誤差の面から見れば十分実用可能であると思われる。

しかし、この処理はパターンマッチングなので、パターンの形状によっても左右される。今回のパターンはあまり特徴がないため、ワークが移動すると特徴点を見失い、サーチできなくなる恐れがある。

3.9 Line による解析アルゴリズム２

（１）解析方法

今回も黒いペンによりエッジ付近に特徴点を作り（図 3.18）、2 本の Line 上に”Line プリプロセッサ演算子”を組み合わせることにより、座標 1、2 及びワーク高さ（Z 値）座標の検出を行った。

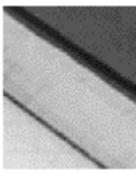

図 3.18 エッジの強調

次に、Calibration(座標変換)（3.8 参照）を行ったのち、Line をワークの側面それぞれに 2 本づつ描く。

そして、Line A、Line B には **Line Max Back Edge 演算子**を用いて、座標１、座標２を検出させ、Line B、Line D には **First Edge 演算子**により、ワーク上部のエッジを認識させることにより、その座標を得る。

Line Max Back Edge より得られた座標と First Edge 演算子で得られた座標を点変換関数の **Point to point distance** により、Z（ワーク高さ）が求められる。

また、Z の精度を高めるために、座標１、２の Z 値を、単項演算子関数 **Mean(avg)** を使って平均を出す方法にした。図 3.19 はこの処理を行っている様子である。

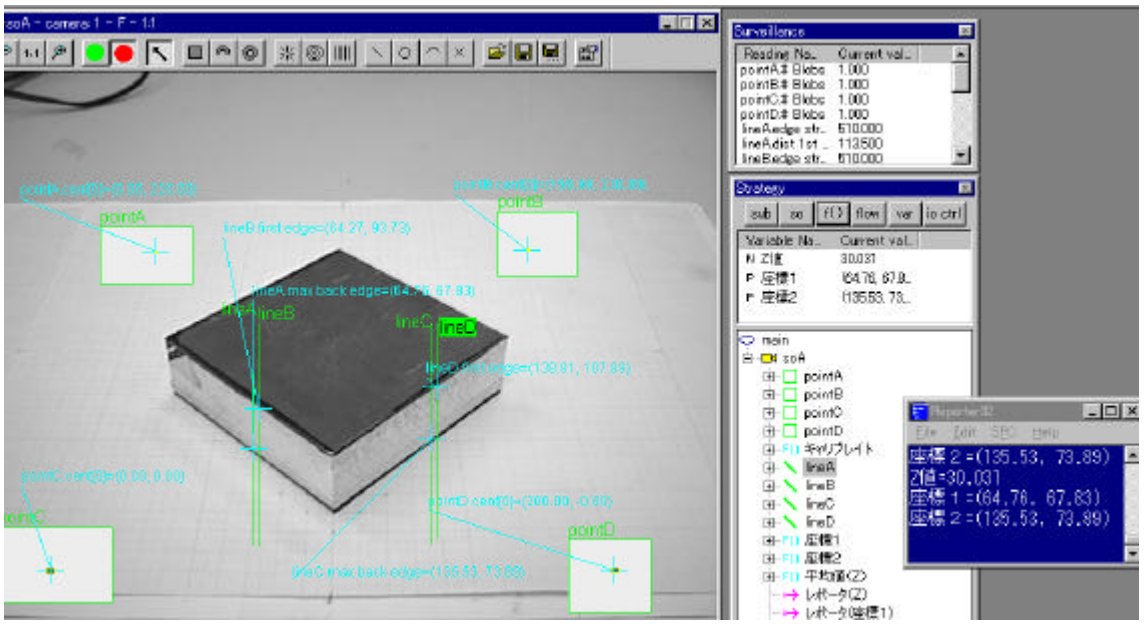

図 3.19　Line 解析による座標検出の様子

レポーターを使用したワーク座標の検出結果表示（図 3.20）

図 3.20　ワーク座標の出力

（２）解析結果

　このアルゴリズムを評価するために、実際に使用して解析を行った。
その結果を、グラフから読み取った実寸座標と解析によって計測された計測値とを比較したものを表 3.2 に示す。

表 3.2 測定結果

	測定値[mm]	実寸値[mm]	誤差[mm]	誤差[%]
座標１X[mm]	64.76	66.5	-1.74	-2.6%
座標１Y[mm]	67.83	65.5	2.33	3.6%
座標２X[mm]	135.53	138.5	-2.97	-2.1%
座標２Y[mm]	73.89	71.5	2.39	3.3%
ワークの高さ Z[mm]	30.03	30	0.03	0.1%

　測定結果より、ワーク座標は誤差 3[mm]以内の目標値をクリアしており、ワーク高さについては誤差 1[mm]以下に抑えることができた。
　この“Line”解析にかかった時間は 0.030[msec]であった。

（３）考察

　この解析アルゴリズムは SmART Search に比べて誤差が多くなっているが、目標の 3[mm]以下になっているので、実用可能な解析アルゴリズムであると思われる。
　この誤差については特徴線（点）の厚さ（2[mm]）によるものと考えられる。
　この解析の優れているところは、SmART Search に比べ、処理が単純であるためより安定した座標検出が可能である。

第４章　座標検出ソフトの製作

4.1 研究目的

第三章では、“ Sherlock32 ”を使用して、２つの座標検出解析アルゴリズムのシミュレーションを行った。そこで、“ **Line** ”解析は演算処理が単純で安定した検出が行えるため、座標検出に有効と判断した。

そこでこの章では、“ Sherlock32 ”における解析演算子を、関数として用意してくれている MVTools 関数ライブラリを使用し、Visual C++上で“ Line ”解析アルゴリズムを用いた実用的な自動ワーク座標検出ソフトの作成を行った。

4.2 実験方法及び装置

まず、Imaging Technology 社製　“ MVTools ”関数ライブラリで用意されている Line 解析に必要な関数を、プログラミング言語 Microsoft Visual C++ 6.0 上で組込んで画像検出ソフトを作成する。

作成方法としては、Visual C++の“ MFC(Microsoft Foundation Class) ”というクラスライブラリに“ MVTools ”の関数を組み込んでプログラムを作成する。

大まかには、カメラ（図 3.3 参照）から送られてきた画像をパソコンの画面に表示させるプログラムを作り、画像解析に必要な解析関数を組込んでいく。ソフト完成後は実際に座標検出を行って座標検出の精度を測定した。

4.3 MVTools ライブラリーについて

今回、MVTools の関数を利用して、プログラミングを行うため、簡単に MVTools について説明を行う。

（１）MVTools の構成

MVTools は、ITI 社の画像取込ボードを利用して画像解析を行うための関数ライブラリである。MVTools の主な部分は、３つの DLL から構成されている。

mvtools.dll	MVTools メインモジュール 画像前処理、画像解析、幾何演算の機能を実装
itimgutl.dll	Image Utilities モジュール 画像データ構造の基本的な操作を実装
itimgmgr.dll	Image Maneger モジュール 画像入力、画像表示等の入出力機能を実装

これら、DLL は Windows のシステムディレクトリにインストールされている。Visual C++で作成するプログラムから MVTools を利用するためには、DLL と同名の.lib ファイルをリンクし、やはり同名の.h ファイルをインクルードする必要がある。

（2）画像データ構造体－IDS

MVTools では、画像解析や画像入出力の関数とのインターフェースとして、IDS（Image Data Structure）と名づけられた構造体を使用する。この構造体には、サイズなどの画像情報とともに、画像バッファへのポインタが格納される。

IDS にはマスターIDS と RROI（Rectangle Region Of Interest）IDS の二種類があり、マスターIDS は独自の画像バッファを持つ IDS である。RROI IDS はすでに生成されたマスターIDS に関連付けられ、画像バッファの特定の領域を処理するための IDS なのである。

マスタ IDS を生成するには関数 im_create()を使用する。RROI IDS を生成するには関数 im_delete()をコールして破棄する必要がある。いずれも iteimgutl.dll に含まれる関数である。

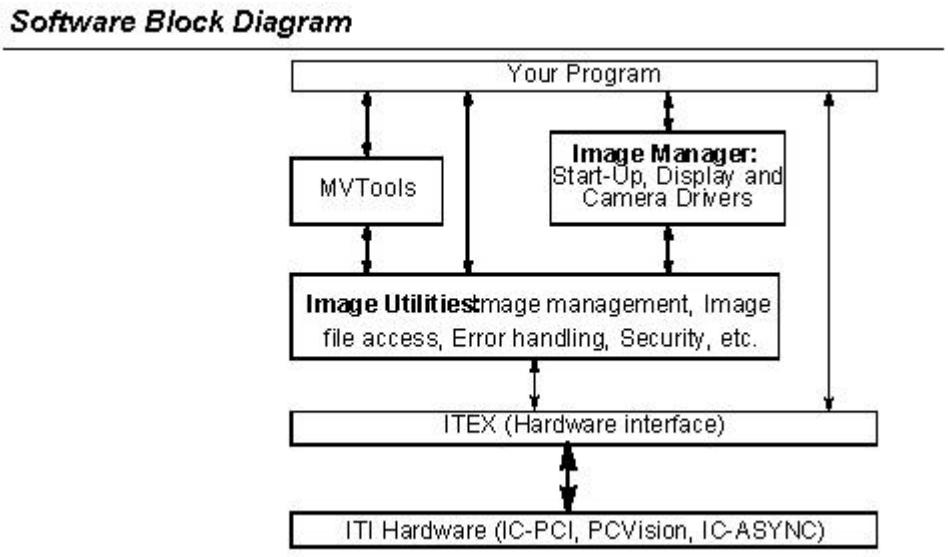

図 4.1MVTools 周辺のブロックダイアグラム（MVTools Software Reference より）

4.4 座標検出ソフト作成

今回座標検出ソフトを作成するために、MVTools が持っている、画像前処理、画像解析、幾何演算、画像ファイルの入出力などを、Visual C++に組み込むことにより、より柔軟性のあるソフトウェアを作成することができた。

座標検出の解析アルゴリズムは Sherlock32 で開発した line を使った解析アルゴリズムを採用した。また、Visual C++では MFC の“AppWizard”を使用した。

以下の図 4.2 に画像表示から保存まで、プログラムの流れを示す。

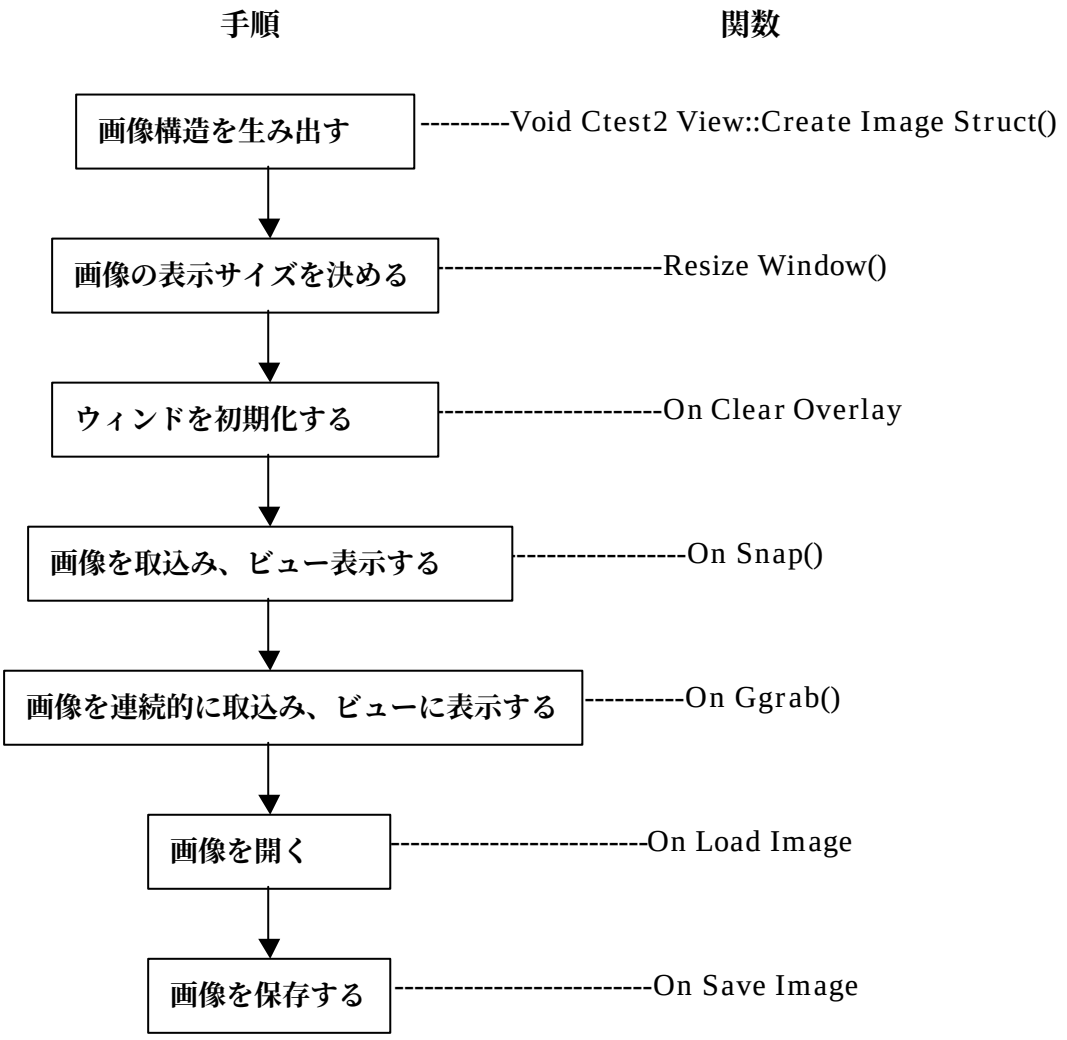

図 4.2 プログラムの流れ 1

次に座標検出までのプログラムの流れを図 4.3 に示す。

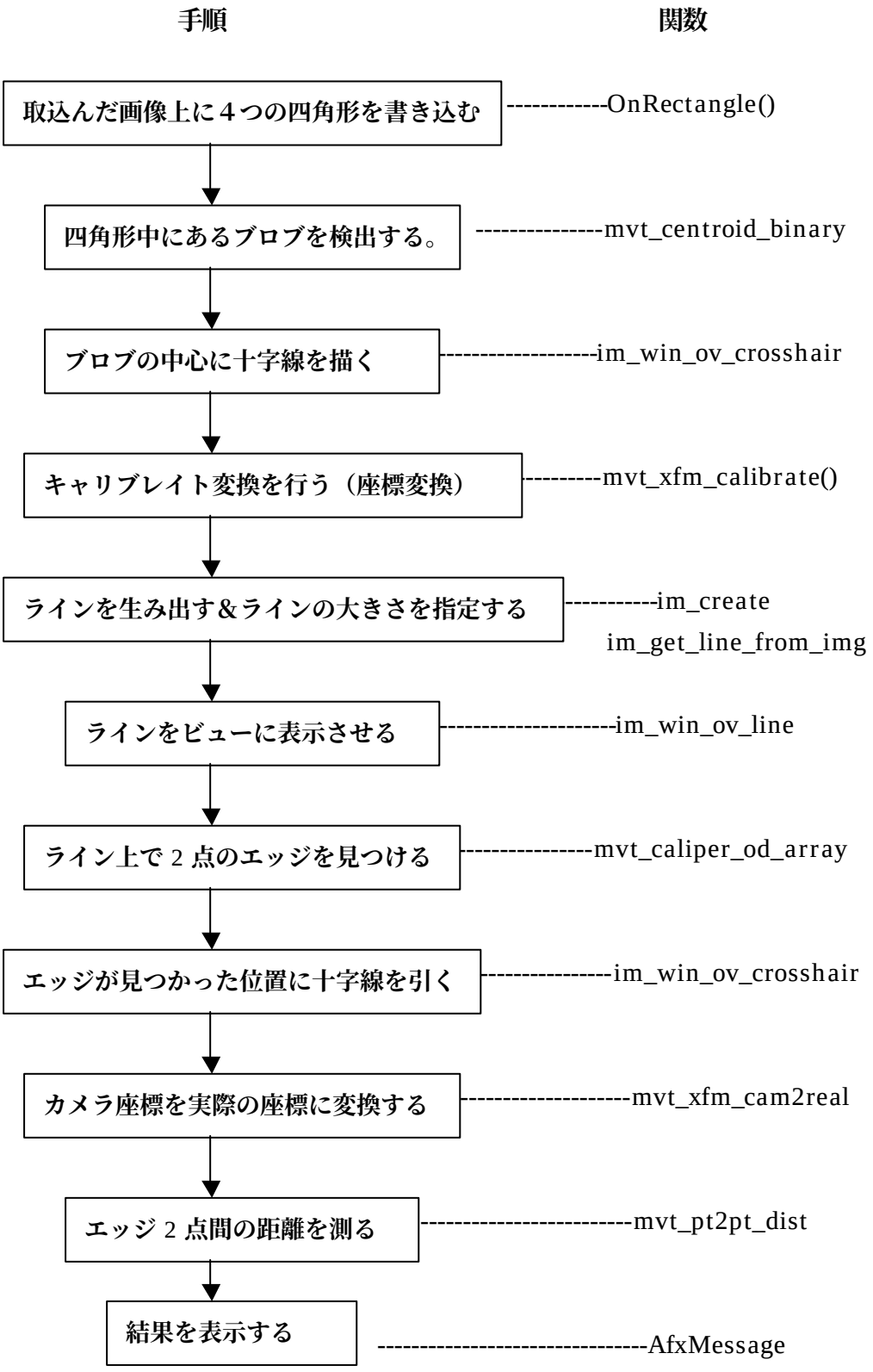

図 4.3　プログラムの流れ図 2

4.5 解析に用いられた関数

ここでは、画像前処理、画像解析、幾何演算、画像ファイル入出力の関数についての説明を行う。

１）画像前処理

1 枚または複数枚の画像を入力し、1 枚の画像を出力する関数である。これらは、mvtools.dll に含まれています。以下に今回使用した主な関数の説明を行う。

1-1, **mvt_threshold**(Image *im, BYTE thresh, BYTE below, BYTE above)

2 値化処理

パラメータ：

im　イメージの 2 値化

thresh　閾値

below　セットピクセル値、又は閾値以下にする

above　セットピクセル値を閾値より大きくする

画像の種類：

BYTE

戻り値：

IM_OK エラーなし

IM_BYTE_ONLY　MVTools は BYTE イメージ処理だけが可能である

1-2, **mvt_median**(Image *src, Image *dst)　メディアンフィルタ処理

パラメータ：

src　メディアンフィルタの画像ソース

dst　イメージ（結果）の行き先

画像の種類：

BYTE

戻り値：

IM_OK　エラーなし

IM_BYTE_ONLY　MVTools は BYTE イメージ処理だけが可能である。

IM_TOO_SMALL　イメージサイズが小さい。最低でも 3×3 ピクセルは必要

1-3, **mvt_invert**(Image *im)　　ピクセル値を反転する

パラメータ：

im　イメージデータ構造のポインタ

画像の種類：

BYTE、WORD

戻り値：

IM_OK　エラーなし

2）画像解析

1 枚または複数枚の画像を入力して、解析結果の数値を出力する関数である。mvtools.dll に含まれている。以下に今回使用した解析のための関数の説明を行う。

2-1, **mvt_centroid_binary**(Image *im, int thresh, double *xcent , double *ycent, DWORD *pCount ,DWORD coord_flag)　　閾値以上のピクセル点を見つける

パラメータ：

im　イメージデータ構造にピクセル処理を収納する

threst　閾値。

Xcent　X 点の位置（戻り値）

Ycent　Y 点の位置（戻り値）

pCount　2 値化の値やピクセルをカウントする

coord_flag　RROI やマスターイメージの相対座標

MVT_CF_PARENT_REF　マスターイメージの相対座標

MVT_CF_ROI_REF　ROI の相対座標

MVT_CF_Y_VIDEO　ビデオオーダの座標

MVT_CF_Y_CART　　直行座標のオーダ（Y を大きくする）

2-2, **mvt_edge_find_array**(Array *ar, int start, int dir, int thresh, double *pdist)

イメージ内で１つのエッジの配列変化をみる

パラメータ：

ar　**ピクセル配列の違いによりエッジの変化を見つける**

start　**サーチ開始の配列位置**

dir　**サーチ方向：**MVT_FORWARD **又は** MVT_BACKARD

thresh　**エッジの変化から考慮される閾値**

*pdist　**開始位置からエッジまでの距離**

画像の種類：

BYTE

戻り値

エッジ強度や位置（開始位置から相対）

2-3, **mvt_caliper_od_array**(Array *ar, int thresh, double *pod, double *pdist1, double *pdist2)　**外径のキャリパ測定を行う**

パラメータ：

ar　**キャリパ測定を行う配列**

thresh　**閾値。ただ、変化が同等や大きい場合にこの値を使う**

pod　**外径寸法(戻り値)**

pdist1　**エッジポイント配列の開始からの距離**

pdist2　**エッジポイント配列の終了からの距離**

画像の種類：

BYTE

戻り値：

物体のエッジに後ろから内側に向かっての外径や最大エッジ強度

2-4, **mvt_pt2pt_dist**(double x1, double y1, double x2, double y2);

２点間の距離を計算する

パラメータ：

x1　最初の場所での X 座標

y1　最初の場所での Y 座標

x2　２番目の場所での X 座標

y2　２番目の場所での Y 座標

画像の種類：

BYTE

戻り値：

位置（x 1、y1）と位置（x2、y2）間の距離

３）幾何学演算

数値を入力し、演算結果の数値を出力する関数。これも mvtools.dll に含まれる。

3-1, **mvt_xfm_caliblate** (MVT_COORD_XFORM *coord_xfm)

キャリブレイト座標変換構造

パラメータ：

Coord_xfm　MVT_COORD_XFORM データ構造のキャリブレイト

画像の種類：

BYTE

戻り値：

IM_OK

3-2, **mvt_xfm_cam2real**(MVT_COORD_XFORM *coord_xfm, MVT POINT *point)

カメラ座標から実際の位置に変換する

パラメータ：

Coord_xfm　MVT_COORD_XFORM データ構造

point　MVT_POINT　カメラ座標を実際の座標に変換する

cam_top_left　カメラキャリブレーションの左上ポイント

cam_bot_left　カメラキャリブレーションの左下ポイント

cam_top right　カメラキャリブレーションの右上ポイント

cam_bot_right　カメラキャリブレーションの右下ポイント

画像の種類：

BYTE

戻り値：

IM_OK

4）画像ファイル入出力

ファイルからの画像読み込み、及びファイルへの画像保存を行う。itimgutl.dll **に含まれる。**BMP、PCX、TIFF、COGNEX、**旧** ITI **の各形式をサポートしている。**

4-1, **im_win_paint**(Imgwin* iw, HDC hdc)

ビューにイメージを描く

パラメータ：

iw　**イメージのポインタ**

hdc　**デバイスコンテキストのハンドラ**

戻り値：

なし

4-2, **im_cam_live_start**(int cam, BOOL ext_trig, int timeout, ImgWin* iw, BOOL show_overlay)

カメラからイメージウィンドウへの取得を開始する

パラメータ：

cam　**ゼロベースカメラインデックス**

ext_trig　**外部トリガを有効にする**

timeout　**タイムアウトまでの時間**

show_overlay　**イメージウィンドウを上書きする**

戻り値：

IM_OK　**エラーなし**

IM_BAD_CAM　**無効なカメラ番号である**

IM_ACQ_TIMEOUT

4-3, **im_cam_live_stop** (int cam)

カメラからイメージウィンドウへの取得を停止する

パラメータ：

cam　ゼロベースカメラインデックス

ext_trig　外部トリガを有効にする

iw　イメージウィンドウ

show_overlay　イメージウィンドウを上書きする

戻り値：

IM_OK　エラーなし

IM_BAD_CAM　無効なカメラ番号である

IM_ACQ_TIMEOUT　タイムアウトが起った

4-4, **im_win_delete** (Imgwin* iw, BOOL close_win)

イメージウィンドウを消去する

パラメータ：

iw　イメージウィンドウのポインタ

close_win　イメージウィンドを閉じる前に構成を消去する

戻り値：

IM_OK　エラーなし

4-5, **im_file_read** (Image *im, char *filename, DWORD flags)

イメージファイルを読み込む

パラメータ：

im　イメージデータ構成にイメージファイルを読み込む

filename　読み込むファイル名

flags　イメージファイル型やバイト変換のフラグ見本

戻り値：

IM_OK　エラーなし

IM_NO_IM　イメージの im が見つからない

IM_BAD_TYPE　このファイルイメージ im データ型を読み込むことができない

IM_NO_MEMORY　パラメータブロックファイルに割り当てられない。
メモリがフリーでない

IM_BAD_FILE　ファイルをオープンできない、読込み失敗、イメージのビット深さが間違っている

4-6, **im_file_write** (Image *im, char *filename, int compress, DWORD flags)

イメージをファイルに書き込む

パラメータ：

im　**イメージファイル構成をファイルに書き込む**

filename　**書き込むファイル名**

comress　**圧縮を有効にする：**

IM_COMPRESS　**圧縮フォーマットで保存する。圧縮は 2 進数（2 値化）イメージだけである。グレースケールを圧縮するとまず、うまくいかない。**

flags　**イメージファイル形式又は圧縮形式のフラグを指名する**

戻り値：

IM_OK　**エラーなし**

IM_BAD_FILE　**自動でファイル形式を決定できない。ファイルが開けない、またはファイルを読めない**

IM_BAD_TYPE　**間違った形式、又はこの形式のイメージは書込みできない**

4-7, **im_win_update** (Imgwin* iw, BOOL show_overlay)

イメージウィンドウを更新する

パラメータ：

iw　**イメージウィンドのポインタ**

show_overlay　**先頭のイメージ上にイメージで上書きする**

戻り値：

IM_OK　**エラーなし**

4-8, **im_cam_snap** (int cam, BOOL ext_trig, int timeout, image* im)

イメージをイメージウィンドウに出力する

パラメータ：

cam　**ゼロベースのカメラインデックス**

ext_trig　**外部のトリガを有効にする**

timeout　**タイムアウトまでの時間（秒）**

im　**イメージを出力する**

戻り値：

IM_OK　エラーなし

IM_BAD_CAM　無効なカメラ番号

IM_ACQ_TIMEOUT　タイムアウトが生じて、待っている状態。

4-9, **im_win_ov_rect**(ImgWin *iw, BYTE color, int x, int y, int dx, int dy)

イメージウィンドウに短形を描く

パラメータ：

iw　イメージウィンドウのポインタ

color　ラインの色

x,y　短形の左上の座標

dx　短形の幅

dy　短形の高さ

戻り値：

IM_OK　エラーなし

4-10, **im_win_ov_crosshair**(ImgWin *iw, BYTE color, int x, int y, int size)

イメージウィンドウに十字線を描く

パラメータ：

Iw　イメージウィンドウのポインタ

color　十字線の色

x,y　十字線の中心の位置

size　十字線の垂直/水平方向の大きさ

戻り値：

IM_OK　エラーなし

4-11, im_win_ov_line(ImgWin *iw, BYTE color, int x1, int y1, int x2, int y2)

イメージウィンドウにラインを描く

パラメータ：

iw　イメージウィンドウのポインタ

color　ラインの色

x1,y1　ラインの最初の座標

x2,y2　ラインの最後の座標

戻り値：

IM_OK　エラーなし

4.6 実行画面

以下に示す図 4.3 が座標検出プログラム実行した様子である。

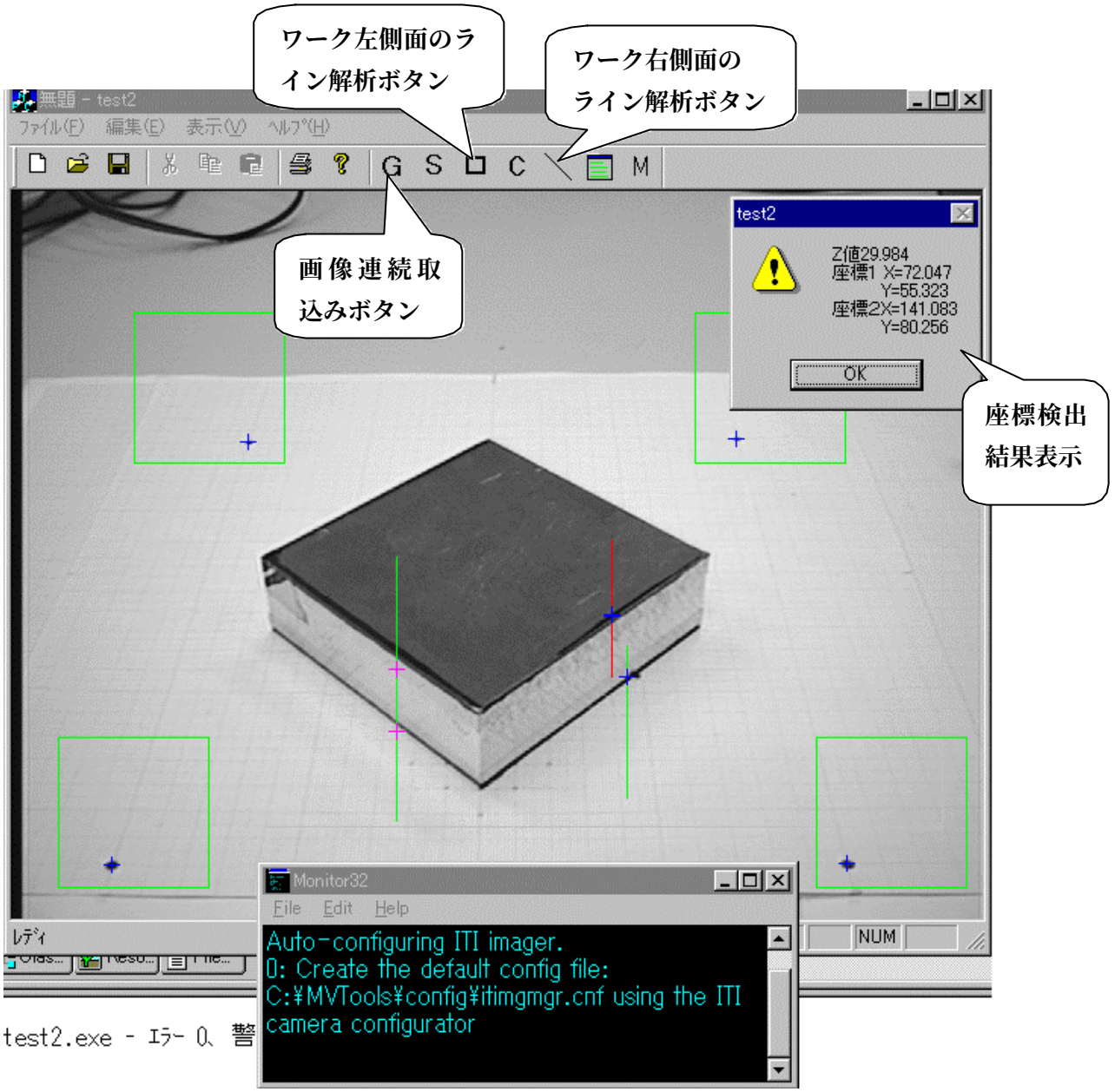

図 4.3　座標検出プログラムの実行結果

4.7 測定結果

今回の、アルゴリズムを評価するために実際に使って、解析を行った。
その結果、グラフから読み取った実寸法と解析による測定値とを比較し、そのまとめたものを表 4.1 に示す。

表 4.1 処理結果

	測定値[mm]	実寸値[mm]	誤差[mm]	誤差[%]
座標１ X[mm]	72.036	74.5	-2.464	-3.3%
座標１ Y[mm]	55.377	53	2.377	4.5%
座標２ X[mm]	141.094	144	-2.906	-2.0%
座標２ Y[mm]	80.186	77.5	2.686	3.5%
ワークの高さ Z[mm]	29.984	30	-0.016	-0.1%

この結果から、ワーク座標は 3[mm]以内の誤差で、ワークの高さについては 1[mm]以下に抑えることができた。

4.8 考察

以上の結果から、目標値（3[mm]以下）をクリアできたので、実用可能である。
しかし、Sherlock32 で行った 2 つの解析アルゴリズムより、誤差が大きくなった。
この原因は、前処理が足りなかったために、誤差が大きくなったと考えられる。

第５章　放電加工機の遠隔操作

5.1 研究目的

パソコンからインターネットを使って放電加工機と接続することで、放電加工機が遠隔地にある場合でも、作業者が遠隔地に行かなくても、パソコンから放電加工機を操作することができる。

そこで今回、ソケット通信（1.3 章参照）によりパソコンから放電加工機に NC プログラムの送信や加工機の操作などを行った。また、放電加工機からは、加工状態に関する情報がパソコンに送信される。

次に放電加工機遠隔操作の主な使用目的としては、

1）NC データの送受信
2）座標検出によって得られたデータの送信
3）加工状況の確認
4）放電加工機の操作

などが挙げられる。

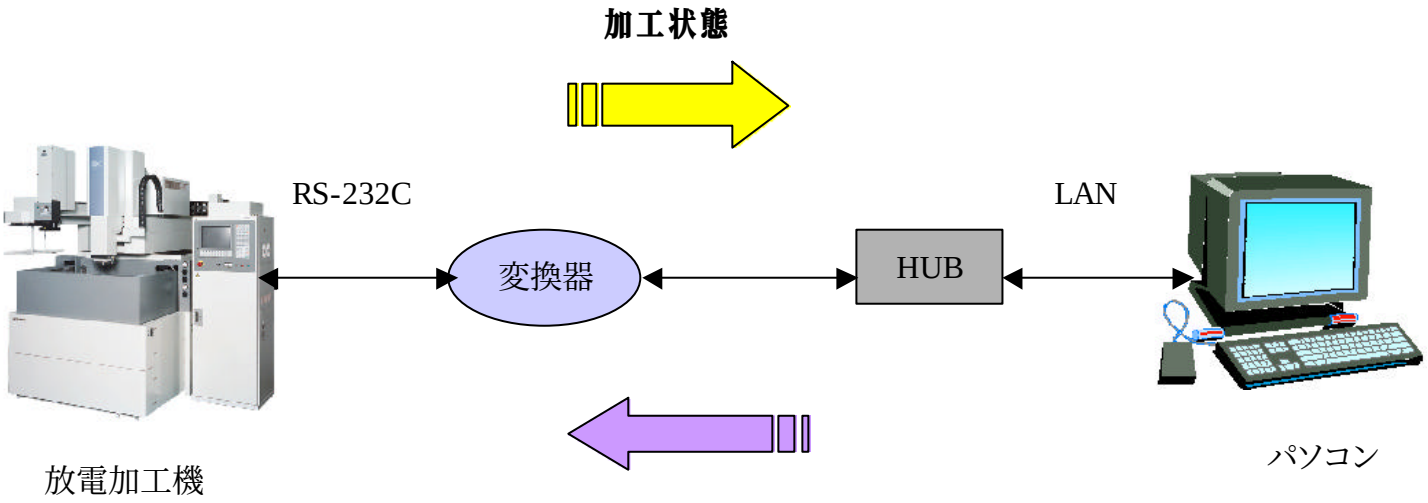

図 5.1 放電加工機の遠隔操作概要

5.2 実験方法と装置

今回行った放電加工機遠隔操作は、プログラミング言語 Visual C++を用いて、ソケット通信ソフトの作成を行った。パソコンは画像検出と同様の OptiPlexGX1 を用いた。

通信ソフトが完成後、パソコンから変換機（Comm Assist）まで LAN で接続し、変換機から RS232C で放電加工機と接続を行い、通信ソフトを実行させて、NC プログラムの送受信や放電加工機の操作などを行った。

5.3 通信プログラムの一例

次の図 5.2 はソケットプログラムによって放電加工機に接続し、放電加工機から NC プログをダウンロードしている状態の一例である。この研究はまだ途中であるが、今後本格的に遠隔操作について行っていくつもりである。

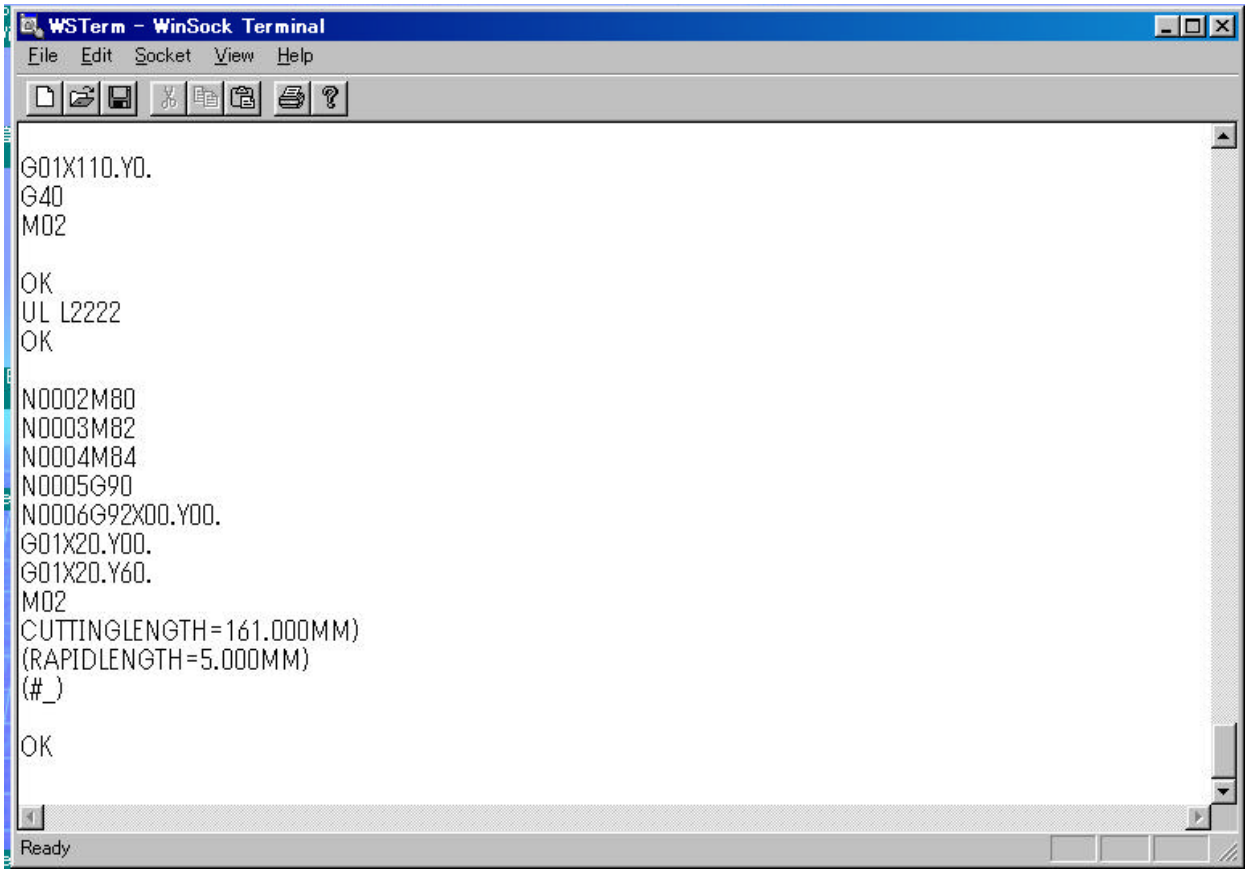

図 5.2 遠隔操作画面の一例

第 6 章　結言

今回は“ Sherlock32 ”による解析アルゴリズムの開発から始まり、その解析アルゴリズムが座標検出に有効であるかを評価するための座標検出シミュレーションを行った。

そこで、”SmART Search”と”Line”の 2 種類の解析アルゴリズムについて評価を行った結果、座標検出の安定性から”Line”解析アルゴリズムがより実用的だと判断した。

次に“ Sherlock ”によるシミュレーション結果から有効と思われる”Line”解析アルゴリズムを用いた、「座標検出のソフト開発」について行った。座標検出ソフトの検出結果は、Sherlock32 で行った場合に比べ少し誤差が大きくなったが、前処理などの補正を行えば、誤差を減らすことが可能だと思われる。

今後の課題はワークの座標検出の範囲を広く設定して、ワークがどこにあっても検出が可能で、ワークの特徴線を必要としない検出方法などが今後実用される上で重要となってくる。

次に Sock 通信を用いた放電加工機遠隔操作プログラムの作成し、パソコンから放電加工機に NC プログラム送受信や加工機の操作などを行った。

この実験でソケット通信が放電加工機の遠隔操作に有効であることが分ったので、今後は自動座標検出ソフトにこの通信プログラムを組込んで、遠隔地からカメラの制御を含めた放電加工機の遠隔制御について研究を行っていくつもりである。

最後に本研究にあたって、指導教員である小林教授をはじめ、実験に協力いただいた研究室の方々には心より感謝申し上げる。

参考文献

小林昭　監修　超精密生産技術大系第二巻　実用技術：株式会社フジ・テクノシステム
放電加工　　著者：富本　直一　製作：株式会社　日刊工業出版プロダクション
新 Visual C++6.0 入門　シニア編　著者：林　晴比古　発行所：ソフトバンク株式会社
WinSock2.0 プログラミング　江村豊　監修　　発行所：ソフトバンク株式会社

参考資料

座標検出プログラム：test2

test2View.h

```
class CTest2View : public CScrollView
{
public:
    int itsCam;
        void CreateImageStruct();
        int cam_dx,cam_dy;
        void ResizeWindow();

protected: // シリアライズ機能のみから作成します。
        CTest2View();
        DECLARE_DYNCREATE(CTest2View)

// アトリビュート
public:
        CTest2Doc* GetDocument();

// オペレーション
public:

// オーバーライド
        // ClassWizard は仮想関数のオーバーライドを生成します。
        //{{AFX_VIRTUAL(CTest2View)
        public:
        virtual void OnDraw(CDC* pDC);  // このビューを描画する際にオーバーライドされます。
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        virtual void OnInitialUpdate(); // 構築後の最初の１度だけ呼び出されます。
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        //}}AFX_VIRTUAL
```

```
// インプリメンテーション
public:


        virtual ~CTest2View();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// 生成されたメッセージ マップ関数
protected:

        //{{AFX_MSG(CTest2View)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnLoadImage();
        afx_msg void OnSaveImage();
        afx_msg void OnClearOverlay();
        afx_msg void OnGrab();
        afx_msg void OnSnap();
        afx_msg void OnRectangle();
    afx_msg void OnLine1();
        afx_msg void OnDialog();
        afx_msg void OnMean();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // test2View.cpp ファイルがデバッグ環境の時使用されます。
inline CTest2Doc* CTest2View::GetDocument()
   { return (CTest2Doc*)m_pDocument; }
#endif
```

```
////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ は前行の直前に追加の宣言を挿入します。

#endif
// !defined(AFX_TEST2VIEW_H__96D89F0C_E9BC_11D5_9F39_00C04F014AD1__
INCLUDED_)
```

---

test2View.cpp

```
// test2View.cpp : CTest2View クラスの動作の定義を行います。
//

#include "stdafx.h"
#include "test2.h"

#include "test2Doc.h"
#include "test2View.h"

#include "mvtools.h"
#include "itimgmgr.h"

#include "XYZDialog.h"


Erflag error;
DWORD i;

ImgWin* iw;
Image* im;
Image* ov;
Image* im2;
Image *tl_roi,*tr_roi ,*bl_roi, *br_roi;
```

```
Array* od_cal;  //アウトサイドキャリパ

MVT_BLOB_PARAMS* blob_params;
MVT_BLOB_RESULTS* blob_results;

double tl_x,tl_y,tr_x,tr_y,bl_x,bl_y,br_x,br_y;
double height,height2,d1,d2;

MVT_COORD_XFORM* xfm;
MVT_POINT rp1,rp2,rrp1,rrp2,rpp1,rpp2;

BOOL hasIO = FALSE;
double t1,t0;
BOOL first_time = TRUE;

CString InitialDir;
BOOL ViewProcesstime = TRUE;

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CTest2View

IMPLEMENT_DYNCREATE(CTest2View, CScrollView)

BEGIN_MESSAGE_MAP(CTest2View, CScrollView)
        //{{AFX_MSG_MAP(CTest2View)
                ON_WM_CREATE()
        ON_COMMAND(ID_LOAD_IMAGE, OnLoadImage)
        ON_COMMAND(ID_SAVE_IMAGE, OnSaveImage)
        ON_COMMAND(ID_CLEAR_OVERLAY, OnClearOverlay)
```

```
        ON_COMMAND(ID_Grab, OnGrab)
        ON_COMMAND(ID_Snap, OnSnap)
        ON_COMMAND(ID_Rectangle, OnRectangle)
        ON_COMMAND(ID_Line1, OnLine1)
        ON_COMMAND(ID_BUTTON6, OnDialog)
        ON_COMMAND(ID_Mean, OnMean)
        //}}AFX_MSG_MAP
        // 標準印刷コマンド
        ON_COMMAND(ID_FILE_PRINT, CScrollView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW,
CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTest2View クラスの構築/消滅

CTest2View::CTest2View()
{
        // TODO: この場所に構築用のコードを追加してください。
        itsCam=0;

}

CTest2View::~CTest2View()
{
        im_cam_live_stop(itsCam);
        im_win_delete(iw, FALSE);
        im_delete(im2);
}

BOOL CTest2View::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: この位置で CREATESTRUCT cs を修正して Window クラスまたはスタイルを
        //  修正してください。
```

```
	return CScrollView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CTest2View クラスの描画

void CTest2View::OnDraw(CDC* pDC)
{
	CPoint sp = GetScrollPosition();
	im_win_set_scroll(iw, sp.x, sp.y);
	im_win_paint(iw, pDC->m_hDC);
	// TODO: この場所にネイティブ データ用の描画コードを追加します。


}

void CTest2View::OnInitialUpdate()
{
	CScrollView::OnInitialUpdate();

	CSize sizeTotal;
	// TODO: このビューのサイズの合計を計算します。
	sizeTotal.cx = cam_dx;
	sizeTotal.cy = cam_dy;
	SetScrollSizes(MM_TEXT, sizeTotal);

	ResizeWindow();
}

/////////////////////////////////////////////////////////////////////////////
// CTest2View クラスの印刷

BOOL CTest2View::OnPreparePrinting(CPrintInfo* pInfo)
{
	// デフォルトの印刷準備
```

```
	return DoPreparePrinting(pInfo);
}

void CTest2View::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: 印刷前の特別な初期化処理を追加してください。
}

void CTest2View::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: 印刷後の後処理を追加してください。
}

/////////////////////////////////////////////////////////////////////////////
// CTest2View クラスの診断

#ifdef _DEBUG
void CTest2View::AssertValid() const
{
	CScrollView::AssertValid();
}

void CTest2View::Dump(CDumpContext& dc) const
{
	CScrollView::Dump(dc);
}

CTest2Doc* CTest2View::GetDocument() // 非デバッグ バージョンはインラインです。
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTest2Doc)));
	return (CTest2Doc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CTest2View クラスのメッセージ ハンドラ
```

```
int CTest2View::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CScrollView::OnCreate(lpCreateStruct) == -1)
		return -1;

	CreateImageStruct();

	hasIO = (im_input_get_qty() > 0);
	return 0;

}

void PrintExecutionTime(char* alg)	//ポイント作成時間
{
CString s;
	if (!ViewProcesstime) return;
	s.Format("Execution time for %s = %.2f mSec",alg,t1-t0);
	AfxMessageBox(s);
}


void CTest2View::OnLoadImage()	//画像を開く
{
	CString theFileName;
   CFileDialog dlg(TRUE,_T("BMP"),_T("*.BMP; *.TIF; *.PCX; *.ITI; *.COG"),
                                   OFN_HIDEREADONLY                    |
OFN_OVERWRITEPROMPT,                                  "Supported
Image Formats |*.BMP; *.TIF; *.PCX; *.ITI; *.COG|Bitmaps (*.BMP)|*.BMP|TIF
(*.TIF)|*.TIF|PCX     (*.PCX)|*.PCX|Imaging     Technology     Incorporated
(*.ITI)|*.ITI|Cognex (*.COG)|*.COG||");

	if (first_time)
		InitialDir     =     CString(im_get_install_dir(NULL,0))     +
CString("¥¥images");
	dlg.m_ofn.lpstrTitle = _T("Load Image");
```

```
	dlg.m_ofn.lpstrInitialDir = InitialDir.GetBuffer(0);
	if (dlg.DoModal() != IDOK)
		return;
	theFileName = dlg.GetPathName();
	first_time = FALSE;
	// initial dir persists
	InitialDir = theFileName.Left(theFileName.ReverseFind('¥¥'));
	im_cam_live_stop(itsCam);
	// clear current image
	mvt_constant(im,255);
	// do the file load
	im_file_read(im, theFileName.GetBuffer(0), IM_AUTO_FILE);
	im_win_update(iw, TRUE);

	//functions for ITEX escape
	MODCNF* pMod = im_get_capture_mod(0);

	MODCNF* pAMMod = im_get_am_mod(0);


}

void CTest2View::OnSaveImage()	//画像を保存する
{
	CString theFileName;
CFileDialog dlg(FALSE,_T("BMP"),_T("*.BMP"),
				OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
				"Bitmaps (*.BMP)|*.BMP|TIF (*.TIF)|*.TIF|PCX (*.PCX)|*.PCX|Imaging Technology Incorporated (*.ITI)|*.ITI|Cognex (*.COG)|*.COG||");

	if (first_time)
		InitialDir = CString(im_get_install_dir(NULL,0)) + CString("¥¥images");
	dlg.m_ofn.lpstrTitle = _T("Save Image");
```

```
	dlg.m_ofn.lpstrInitialDir = InitialDir.GetBuffer(0);
	if (dlg.DoModal() != IDOK)
		return;
	theFileName = dlg.GetPathName();
	first_time = FALSE;
	// initial dir persists
	InitialDir = theFileName.Left(theFileName.ReverseFind('¥¥'));
	im_cam_live_stop(itsCam);
	// do the file save
	im_file_write(im, theFileName.GetBuffer(0), IM_AUTO_FILE);

};

void  CTest2View::OnClearOverlay()		//イメージを消す
{
	// Clear overlay by setting it to zero
	mvt_constant(ov,0);
	im_win_update(iw, TRUE);

};


void CTest2View::CreateImageStruct()		//画像構造の生み出す
{
	if(iw)
		im_win_delete(iw, FALSE);
	im_cam_get_size(itsCam, &cam_dx,&cam_dy);
	iw = im_win_create(cam_dx, cam_dy, m_hWnd, TRUE);
	ov = im_win_overlay(iw);
	im = im_win_image(iw);

	// create image for computation
	if(im2)
		im_delete(im2);
	im2 = im_create(IM_BYTE, cam_dx, cam_dy);
```

```
	CSize sizeTotal;
	sizeTotal.cx = cam_dx;
	sizeTotal.cy = cam_dy;
	SetScrollSizes(MM_TEXT, sizeTotal);

	ResizeWindow();


};

void CTest2View::ResizeWindow()		//ウインドウサイズを決める
{
	//resize window to the size of image
	RECT SinkClientWindowRect;
	::GetClientRect(m_hWnd, &SinkClientWindowRect);
	HWND hParentWnd = ::GetParent(m_hWnd);
	RECT ClientWindowRect;
	::GetClientRect(hParentWnd, &ClientWindowRect);
	long VertAdjust = ClientWindowRect.bottom -
SinkClientWindowRect.bottom;
	long HorzAdjust = ClientWindowRect.right - SinkClientWindowRect.right;

	RECT WindowRect;
	::GetWindowRect(hParentWnd, &WindowRect);

	long lBorderHorz = (WindowRect.right - WindowRect.left) -
ClientWindowRect.right;
	long lBorderTop = (WindowRect.bottom - WindowRect.top) -
ClientWindowRect.bottom - lBorderHorz;

	::MoveWindow(hParentWnd,
			WindowRect.left, WindowRect.top,
			cam_dx + lBorderHorz +  HorzAdjust,
			cam_dy + lBorderHorz + lBorderTop + VertAdjust,
			TRUE);
```

```
        ::SendMessage(hParentWnd, WM_PAINT, 0, 0);
        ::SendMessage(m_hWnd, WM_PAINT, 0, 0);

};


void  CTest2View::OnSnap()              //画像と取込む
{
        // Snap a camera image
//      im_cam_live_stop(itsCam);
//      im_win_set_display_mode(iw, IW_DISP_DIB_IMAGE);
        im_cam_snap(itsCam, FALSE, 1, im);
        im_win_update(iw, TRUE);

}

void CTest2View::OnGrab()               //連続画像を取込む
{
        //      im_win_set_display_mode(iw, IW_DISP_DDRAW);
    //  im_win_set_display_mode(iw, IW_DISP_YCRCB);
        im_cam_live_start(itsCam, FALSE, 2, iw, TRUE);

}

void CTest2View::OnRectangle()
{

CString s;

        // TODO: この位置にコマンド ハンドラ用のコードを追加してください
                //rois を４つ描く（四角形）
        tl_roi = im_rroi(im,80,80,100,100);
        tr_roi = im_rroi(im,450,80,100,100);
        bl_roi = im_rroi(im,30,360,100,100);
        br_roi = im_rroi(im,530,360,100,100);
```

```
        //rois を４つ描く
        im_win_ov_rect(iw,IM_OV_GREEN,  tl_roi->x,  tl_roi->y,  tl_roi->dx,
tl_roi->dy);
        im_win_ov_rect(iw,IM_OV_GREEN,  tr_roi->x,  tr_roi->y,  tr_roi->dx,
tr_roi->dy);
        im_win_ov_rect(iw,IM_OV_GREEN,  bl_roi->x,  bl_roi->y,  bl_roi->dx,
bl_roi->dy);
        im_win_ov_rect(iw,IM_OV_GREEN,  br_roi->x,  bl_roi->y,  br_roi->dx,
br_roi->dy);

    //画像を反転させる
        mvt_invert(tl_roi);
        mvt_invert(tr_roi);
        mvt_invert(bl_roi);
        mvt_invert(br_roi);

        //４つの roi のそれぞれの点を見つける
        mvt_centroid_binary(tl_roi,120,&tl_x,&tl_y,NULL,0);
        mvt_centroid_binary(tr_roi,120,&tr_x,&tr_y,NULL,0);
        mvt_centroid_binary(bl_roi,120,&bl_x,&bl_y,NULL,0);
        mvt_centroid_binary(br_roi,130,&br_x,&br_y,NULL,0);


        //十字線を描く
        im_win_ov_crosshair(iw,(int)IM_OV_BLUE, (int)tl_x, (int)tl_y, 10);
        im_win_ov_crosshair(iw,(int)IM_OV_BLUE, (int)tr_x, (int)tr_y, 10);
        im_win_ov_crosshair(iw,(int)IM_OV_BLUE, (int)bl_x, (int)bl_y, 10);
        im_win_ov_crosshair(iw,(int)IM_OV_BLUE, (int)br_x, (int)br_y, 10);

        //
        xfm = mvt_xfm_create();
//

        xfm->cam_top_left.x = tl_x;      xfm->cam_top_left.y =tl_y;
        xfm->cam_top_right.x = tr_x;     xfm->cam_top_right.y =tr_y;
        xfm->cam_bot_left.x = bl_x;      xfm->cam_bot_left.y = bl_y;
```

```
        xfm->cam_bot_right.x = br_x;     xfm->cam_bot_right.y = br_y;


        xfm->real_bot_left.x = 0;        xfm->real_bot_left.y = 0;
        xfm->real_top_left.x = 0;        xfm->real_top_left.y = 220;
        xfm->real_top_right.x = 200;     xfm->real_top_right.y =220;
        xfm->real_bot_right.x = 200;     xfm->real_bot_right.y = 0;

        mvt_xfm_calibrate(xfm);

        //ラインを生み出す（左側）
    od_cal = im_create(IM_BYTE,500,1);

        im_get_line_from_img(im,263,241,263,415,TRUE, od_cal);

        //ラインを書く
        im_win_ov_line(iw, IM_OV_GREEN,253,241,253,415);

        //インサイドキャリパーで外側を測る
//      mvt_caliper_id_array(od_cal, 55,70, &height, &d1, &d2);
    mvt_caliper_od_array(od_cal, 70, &height, &d1, &d2);

        //ライン上に十字線を書く
        im_win_ov_crosshair(iw, IM_OV_MAGENTA, 253, 241+(int)d1, 10);
        im_win_ov_crosshair(iw, IM_OV_MAGENTA, 253, 241+(int)d2, 10);

        rp1.x = 253;
        rp2.x = 253;
        rp1.y = 241+d1;
        rp2.y = 241+d2;

  mvt_xfm_cam2real(xfm, &rp1);
  mvt_xfm_cam2real(xfm, &rp2);

  height=mvt_pt2pt_dist(rp1.x,rp1.y,rp2.x,rp2.y);
```

```
  if (ViewProcesstime)
        {
              s.Format("%.3f blobs were found,%.3f ", height,rp2.x);
              AfxMessageBox(s);
        }

}


void CTest2View::OnLine1()
{
CString s;
int lx,ly1,ly2;
int lxx,lyy1,lyy2;
Array* ar;
Array* ar2;
double edge;
double edge2;
int edge_x,edge_y;
int edge_xx,edge_yy;

    xfm = mvt_xfm_create();
//

        xfm->cam_top_left.x = tl_x;      xfm->cam_top_left.y =tl_y;
        xfm->cam_top_right.x = tr_x;     xfm->cam_top_right.y =tr_y;
        xfm->cam_bot_left.x = bl_x;      xfm->cam_bot_left.y = bl_y;
        xfm->cam_bot_right.x = br_x;     xfm->cam_bot_right.y = br_y;


        xfm->real_bot_left.x = 0;        xfm->real_bot_left.y = 0;
        xfm->real_top_left.x = 0;        xfm->real_top_left.y = 220;
        xfm->real_top_right.x = 200;     xfm->real_top_right.y =220;
        xfm->real_bot_right.x = 200;     xfm->real_bot_right.y = 0;

        mvt_xfm_calibrate(xfm);      //座標変換を行う
```

```
    ar = im_create(IM_BYTE,1000,1);
        ar2 = im_create(IM_BYTE,1000,1);

        lx = 405;
        ly1 = 400;
        ly2 = 300;

        lxx = 395;
    lyy1 = 230;
        lyy2 = 320;

        im_get_line_from_img(im,lx,ly1,lx,ly2,/*interpolate*/ TRUE, ar);
     im_get_line_from_img(im,lxx,lyy1,lx,lyy2,TRUE, ar2);


        //ラインを生み出す


//      mvt_invert(ar);
        //ラインを書く
        im_win_ov_line(iw, IM_OV_GREEN,lx,ly1,lx,ly2);
        im_win_ov_line(iw, IM_OV_RED,lxx,lyy1,lxx,lyy2);


        mvt_edge_find_array(ar,          MVT_START,          MVT_FORWARD,
MVT_MAX_EDGE, &edge);
        mvt_edge_find_array(ar2,MVT_START,                   MVT_FORWARD,
MVT_MAX_EDGE, &edge2);

        // draw edge point
        edge_x = lx;
        edge_y = ly1 - (int)edge; // edge is relative to (lx,ly1) start of line
        edge_xx = lxx;
        edge_yy = lyy1 + (int)edge2;
        im_win_ov_crosshair(iw, IM_OV_BLUE, edge_x, edge_y, 10);
```

```
        im_win_ov_crosshair(iw, IM_OV_BLUE, edge_xx, edge_yy,10);

        rrp1.x = edge_x;
        rrp2.x = edge_x;
        rrp1.y = edge_y;
        rrp2.y = edge_y;

        rpp1.x = edge_xx;
        rpp2.x = edge_xx;
        rpp1.y = edge_yy;
        rpp2.y = edge_yy;

        mvt_xfm_cam2real(xfm, &rrp1);          //実寸値に変換する
        mvt_xfm_cam2real(xfm, &rrp2);

        mvt_xfm_cam2real(xfm, &rpp1);
        mvt_xfm_cam2real(xfm, &rpp2);


    height2=mvt_pt2pt_dist(rrp1.x,rrp1.y,rpp2.x,rpp2.y);

if (ViewProcesstime)
        {
                s.Format("%.3f blobs were found,%.3f,%.3f ",rpp2.x,height2,rpp2.y);
                AfxMessageBox(s);
        }


}


void CTest2View::OnDialog()
{
CXYZDialog xyzdlg;

        xyzdlg.DoModal();
```

```
}

void CTest2View::OnMean()
{
CString s;
double a;


        a=(height+height2)/2;          //ワーク高さ(Z 値)の平均

if (ViewProcesstime)
        {
                s.Format("Z 値%.3f¥n 座標 1 X=%.3f¥n              Y=%.3f¥n 座標 2
X=%.3f¥n          Y=%.3f¥n",
                        a,
                        rp2.x,
                        rp2.y,
                        rrp1.x,
                        rrp1.y);
                AfxMessageBox(s);
        }

}
```
