

# 産業用 PC を利用した検査・計測ソフトウェアの開発技術

2024-02-01

## まえがき

本稿の内容は、ファクトリーオートメーション(FA)分野における PC ソフトウェア開発技術に関するものです。開発対象は、工場内で連続稼働するスタンドアロンタイプの画像検査機や計測装置で、既製の汎用検査機や汎用プログラムだけでは実現が難しく、いくつかの開発要素を含むシステムを想定しています。

現在の AI、IoT 分野で主流になっているインタプリタ言語(Python など)を利用して AI 研究の成果が得られ、工場内の検査・計測システムの製作に進展する場合、システムインテグレーターやソフト制作会社に開発を依頼するのが一般的ですが、後で述べる理由により、自社での内製を目指したい場合もあります。この際、ハードウェアメーカー提供の API ライブラリを参照(リンク)するために C 言語等のコンパイル型言語の使用が必要になることが多く、ソフトウェア開発の難易度は上がります。Python、ラズパイの利用で実現できたことでも、産業用 PC で再構築する際には意外と多くの検討事項が必要になり、例えば PC や周辺装置、開発言語の選定は担当者を悩ませるかもしれません。

そこで、筆者の製造業生産技術部署での経験をもとに、スタンドアロンで稼働し続ける工場内検査システムを想定して、開発課題に対する実践的なソフト開発技術を紹介したいと思います。インターネットに未接続の状態での開発や運用、数年に一度の工場定期点検期間内での保守作業(PC 更新など)は、現代においては少し特殊な業務形態(非定常作業)になるのではないのでしょうか。

IoT 関連技術の進歩は速く、本稿内容も一部は旧式化していますが、運用実績のある堅牢なシステムの構築に使用されています。2つの例題は、本稿のプログラミング技術の要点を凝縮しています。

本稿が本分野で働く読者に役立つことができれば幸いです。

---

<sup>1</sup> 本稿の”現在”は2023年1月である。

本稿の前提知識を以下に記します。

① 情報処理技術

次の用語に関する理解や経験があること。

要件定義、基本設計、詳細設計、実装、コンパイル、リンク(ビルド)、単体テスト、統合テスト、運用、保守

② C 言語、C++言語、統合開発環境 (IDE)

オブジェクト指向に深入りせずに C 言語のみの経験で問題はないのだが、STL を利用するための C++言語の浅い知識までは欲しい。本稿では、マイクロソフト社 Visual Studio と C++/CLI を用いた。<sup>2</sup>

③ 画像処理の基本知識

画像処理ライブラリ、ビットマップファイル、OpenCV、ルールベース、AI 検査など。

④ 光学系基礎知識

レンズ、ラインセンサーカメラ、エリアカメラ、照明装置の基本的な知識。適宜、メーカーの WEB サイトやカタログ等を参考にして知識を身につけることができる。汎用製品メーカーの WEB サイトの説明が参考になる。<sup>3</sup>

---

<sup>2</sup> 筆者は 2005 年から使用してきた。C++/WinRT が登場し、非推奨になりつつある。

<sup>3</sup> 例 キーエンス社 <https://www.keyence-soft.co.jp/group/products/>

## 内容

まえがき .....	1
1. 検査・計測システムの開発 .....	6
1-1. 画像検査の進展 .....	6
1-2. 検査・計測アプリの特徴 .....	6
1-3. 要件定義 .....	7
1-4. 開発方針の決定 .....	8
1-5. PC、周辺装置の選定 .....	9
1-6. OS の選定 .....	10
1-7. ソフト開発環境の選定 .....	11
2. 基本技法 .....	12
2-1. C 言語(含 C++,VC++)について .....	12
2-2. 開発環境のバージョン .....	12
2-3. 変数のスコープ、命名規則 .....	13
2-4. STL(C++標準ライブラリ) .....	16
2-5. 処理時間の計測 .....	16
2-6. CSV ファイル .....	17
2-7. ファイル内、GUI 内の日本語使用について .....	18
2-8. 領域確保とメモリーリーク .....	18
2-9. 各種 API の利用方法(コンパイル時、ビルド時の指定) .....	19
2-10. PLC との連携 .....	20
2-11. DIO ボード類 .....	21
2-12. アクチュエータ、ロボシリンダとの連携 .....	23

2-13. 複数台の PC の連携 .....	23
2-14. 光通信ボード .....	24
2-15. GUI 画面の基本形 .....	24
2-16. マルチスレッドプログラミング .....	25
2-17. スレッドセーフ(変数の排他制御) .....	26
2-18. スレッドセーフ(対 GUI コンポーネント) .....	26
2-19. カメラによる画像の取得 .....	27
2-20. 画像処理ライブラリ .....	27
2-21. 画像の描画 .....	28
2-22. 画像の保存 .....	30
2-23. 画像のピクセル値取得 .....	30
2-24. 簡易画像ビューワ .....	31
2-25. COM ポート、シリアル通信 .....	31
2-26. バッチファイルとパイプ .....	32
2-27. システムの自動起動 .....	33
2-28. ウィルス検知ソフト .....	33
2-29. さらなる展開 .....	33
3. テスト&デバッグ技法 .....	34
3-1. テスト環境 .....	34
3-2. ログファイル .....	34
3-3. デバッグ時のツールたち .....	35
3-4. デバッグ作業に関する補足 .....	35
4. 保守について .....	36

4-1. 予備 PC、HDD の準備.....	36
4-2. カメラ、照明調整.....	36
4-3. 故障対応.....	36
4-4. PC 更新.....	37
5. 例題.....	38
5-1. 素因数分解＋画像表示.....	38
5-1-1. 概要.....	38
5-1-2. 入出力データ.....	39
5-1-3. 画面構成.....	39
5-1-4. 画像ライブラリの利用.....	39
5-1-5. スレッド、メモリー、UI コンポーネントの相関図.....	40
5-1-6. マルチスレッド化の効果.....	41
5-2. ADS-B フライト監視システム.....	41
5-2-1. 概要.....	41
5-2-2. 入出力データ.....	42
5-2-3. 画面表示例.....	43
5-2-4. 座標系と幾何計算.....	44
5-2-5. 簡易地図とクリッピング処理.....	45
5-2-6. 連続稼働例.....	46
6. あとがき.....	47

## 1.検査・計測システムの開発

### 1-1.画像検査の進展

情報化社会の進展により、工場内 FA 分野においても、「見える化」の裾野が広がっています。ここ数十年で、製品の目視検査を脱却し、ルールベース検査の自動化、さらに AI 化、が進んでいます。検査の AI 化を例にとると、製品の画像を大量に撮像し、人による OK/NG 情報を付加して、脳の神経回路を模した深層のネットワーク構造で機械学習させることで、コンピュータは瞬時に人(この場合、工場内のベテラン判定員)と同等以上の判定をくだせるようになります。

典型的なルールベース検査の一つであるシート(パルプや不織布など)内の欠陥検出検査の例をあげます。[図 1]製品幅約1m、搬送速度10m/分、フィルムシート内の黒色異物(サイズ 100um 以上)を検出する画像検査機の導入を考えます。まずは、オフラインの環境で、この異物を検出できるかの撮像実験(静止状態)を行います。撮像実験のポイントの一つは照明の選定です。画像上、異物が認識できて、何らかの画像処理ソフトで検出が可能であれば、次の設計段階に進みます。

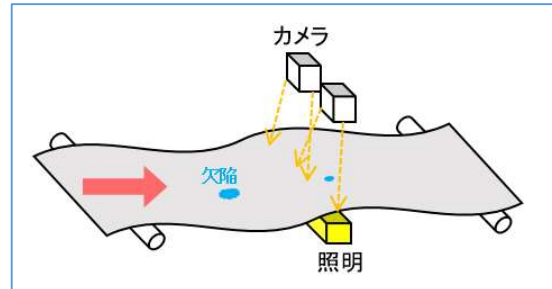


図 1 透過照明の画像検査

検査機設計段階のポイントの一つに、搬送時のブレや埃、遮光を考慮した撮像系の選定があげられます。ブレの有無は、光学系の被写界深度やシャッタースピードに影響を与えるので特に注意を要します。必要なカメラの台数もこの段階で決定し、要件定義書の骨格が作成されます。

### 1-2.検査・計測アプリの特徴

検査・計測アプリの代表的な特徴を次のタイプ A、B に分けします。

【タイプ A(主にインライン装置)】オペレーターは、起動&終了時と非定常時(異常時や定期的な保守作業時など)のみ、PC を操作する。平常運転時には、監視のみを行う。

【タイプ B(主に繰り返し計測装置)】操作者は、同一手順の作業(マウスやキーボード等を頻繁に利用)を繰り返し行う。

タイプ A アプリであれば、UI の見栄えやデザイン性の要求レベルは高くなく、テレビゲームで代表されるような3次元グラフィックスの処理は多くの場合不要です。そのため、標準の統合開発環境(IDE)を所有していれば(サードパーティーの特殊なツールを購入しなくても)、UI 操作を具現化できます。一方で、タクトタイムや連続運転の堅牢さが必要となります。

一方、タイプ B を対象にする場合には、単調作業による操作者の誤操作や疲労を避けるため、あるいは品質管理上の精緻な要求項目を実現するため、UI に工夫を凝らす必要がでてきます。マイクロスコープ型の検査装置が代表例です。

以降、**特徴タイプ A** の装置開発にウェイトを置き、タクトタイムの達成や保守を重要課題に据えて説明します。

### 1-3.要件定義

検査・計測のユーザー要望や課題を要求仕様書(URS)としてまとめる上流工程を要件定義といいます。以下は、URS の特徴です。

- (1) システム概要が、発注側(工場、研究所など)と受注側(システムハウス、検査機ベンダーなど)の両者の担当者およびその管理者に向けて、文章で記述された技術ドキュメントである。
- (2) 受注候補会社が概算見積を提出するのに必要な情報をすべて含む。
- (3) 運用・保守、設備更新段階においても利用されるうる基幹ドキュメントである。

URS 作成技法については割愛<sup>4</sup>、URS 完成後の次のステップ(実施初期段階で、システム構成の選定以降)からの作業について記します。

---

<sup>4</sup> 要件定義は、発注者、利用者、開発者間の調整(ヒアリング)が重要になります。

## 1-4.開発方針の決定

まず、検査・計測システムを汎用製品（既製の画像処理端末など）で具現化できるかの調査と評価をします。最近の画像処理端末は、照明の制御や画像処理アルゴリズムの GUI 上での開発機能が付加されており、かなり複雑な処理も可能になっています。

下図にシステム構成の3つの手法（A、B、C）を示します。<sup>5</sup>前術の”典型例”は、手法 B に相当し、汎用画像検査機で実現できます。更に、欠陥の発見毎にアクチュエータを利用して該当箇所にマーキングするような付加価値をつければ、手法 C 相当になります。

手法 A: PLC 内のシーケンス（ラダー図）のみでシステムを実現する。

手法 B: 汎用画像検査機の開発支援ツール等を用いて検査アルゴリズムを実現する。

手法 C: PC を導入して（汎用画像検査機だけでは実現困難な）アルゴリズムを開発する。

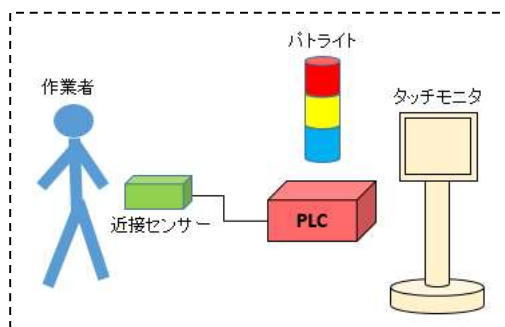


図 2 手法 A

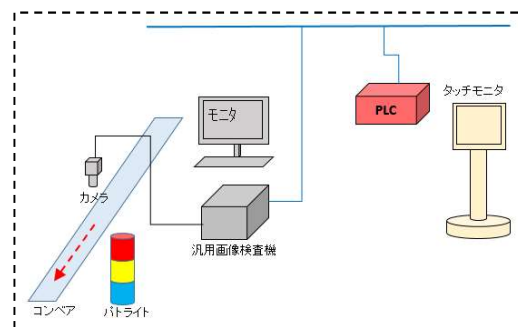


図 3 手法 B

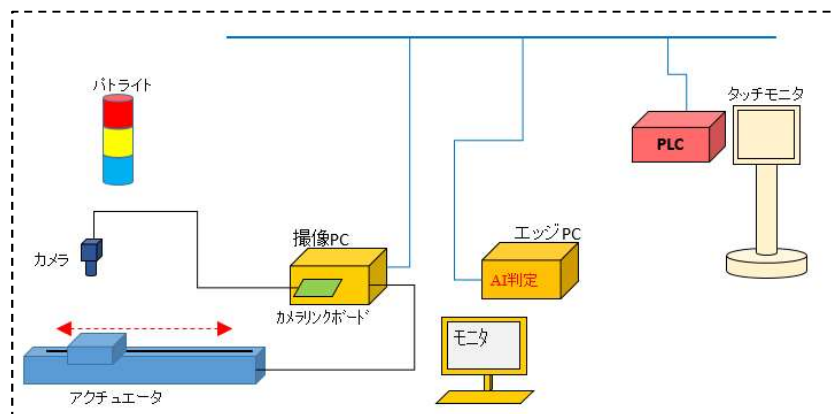


図 4 手法 C

<sup>5</sup> 産業用 PC と PLC については、<https://iotnews.jp/manufacturing/176285/>、<https://iotnews.jp/manufacturing/179310/> に説明がある。



システムの実装難易度、保守のし易さは、A→B→C の順です。PC の利用については、HDD や電源部の故障確率が PLC や汎用画像検査機と比較すると高いため、工場内では一般に手法 A,B が優位です。それでも C を採用する場合には、以下の理由が考えられます。

- (1) 特殊な周辺装置との接続、制御がある場合
- (2) 搭載するアルゴリズムが複雑であり、運用後の更新が想定されるとき
- (3) 検査・計測技術をコア技術として社内で長期に保持したいとき
- (4) コスト面(汎用検査機が割高になることもある)
- (5) AI 画像検査の学習画像撮像のため、全数画像保存を行いたいとき<sup>6</sup>

以下、手法 C を採用する場合の実践技術を説明します。

## 1-5.PC、周辺装置の選定

開発方針を決め、機器の選定に移ります。長期間の稼働を期待される工場では、装置や機器の選定は保守的です。必要機能を満たせば、新しい技術よりも、枯れた技術を選ぶことがあります。例えば、カメラリンクケーブルには根強い需要があります。この点は性能、価格重視の一般品(スマホやノート PC)の選定基準とは少し異なります。

産業用 PC は、CPU タイプ、タワー型/ラックマウント型、拡張ボード用のスロット種類、枚数、メモリーや HDD の容量等のカスタマイズが可能です。[図 5]出荷直後は OS(Windows)のみインストールされている場合が多く、他のソフトウェア(エクセルなど)は自分でインストールします。修理保障期間(無償/有償)も購入前に把握してください。<sup>7</sup>

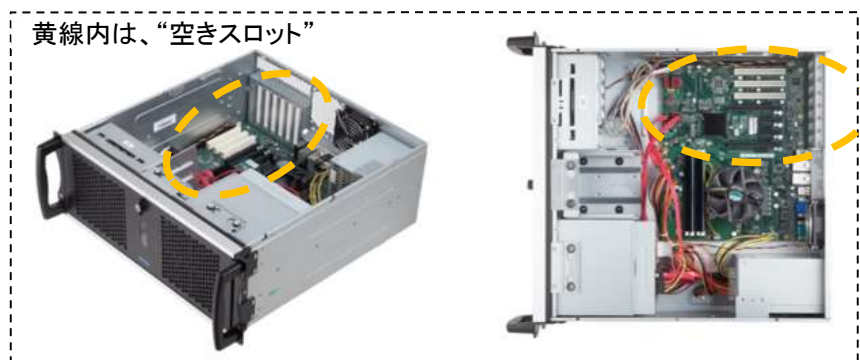


図 5 産業用 PC の内部(コンテック社ホームページより)

<sup>6</sup>従来の画像処理(ルールベース検査)は、判定の根拠を人が理論的に説明でき、判定アルゴリズム開発に必要な学習用 OK/NG 画像の枚数は、少なくてよかった。

<sup>7</sup>産業用 PC には販売終了後7年程度は修理が可能なようです。

PC に周辺装置を接続する場合の接続手段も重要です。有線接続には、①空きスロットに I/O ボードを装着 ②LAN ケーブル経由 ③USB ポート経由 ④COM ポート経由等を選択できます。例えばカメラ接続の場合なら、①、②、③から選ぶことができます。システムの要求仕様(画素数、転送速度、ケーブル長など)やコストから最適な形態を選定します。

周辺装置の代表である PLC は工場からメーカーや型式を指定される場合が多いです。PC⇄PLC 間の通信には、LAN 接続が一般的ですが、データ転送毎のアクセス時間が約 50msec 前後かかることがあり、タクトの厳しい工程への適応には十分な検討が必要です。

拡張カードには、カメラリンク対応ボード、DIO ボード、グラフィックボード、A/D 変換ボード等、多数の種類があります。選定の際の注意点として、最近の AI 用途のグラフィックスボードのパワーアップに伴い、PC 電源容量および隣接スロットの干渉(あるいは接触)問題を指摘しておきます。

無停電電源装置(UPS)も工場内稼働システムには常用品です。落雷などの停電時に一定時間電源を供給する本来の目的に加えて、複数台 LAN 接続された PC の自動シャットダウン機能等もあります。

2020 年頃から、産業 PC の HDD に変わって、SSD が搭載されるようになってきました。筆者は1~2T の外部記憶媒体として使用経験があります。HDD ではタクトタイム上不可能であった撮像画像の全数保存が可能になり、AI 向け学習データの蓄積に大変重宝しました。

ここ数年間のパソコンの性能、特に CPU の演算パワーについては、コア数の増加に重点が置かれており、SSD の搭載も相まって、総合パフォーマンスはかなり上がっていると言えます。

## 1-6.OS の選定

デスクトップ PC の約7割は、OS に Windows を搭載しており、現時点での第一選択肢です。<sup>8</sup>本稿では、PC、OS は汎用品(産業用 PC と Windows)を採用します。必要に応じて、メモリー、CPU 種選択、SSD 増設等のカスタマイズを行うことができます。

Windows は(狭義の)リアルタイム OS ではないので、演算処理の時間が一定ではありません。平均時間であればタクトが達成できるようなきわどい場合には、マルチスレッド化、マルチ PC 化などの技法を使えばシステム実現の可能性があります。これらの手法は後述します。

---

<sup>8</sup> Linux の選択する際は、周辺装置のドライバソフトの有無、運用後の保守作業などを考慮する必要がある。(一般に Windows OS の方が充実している。)

## 1-7. ソフト開発環境の選定

統合開発環境(IDE)として Visual Studio、開発言語として C++/CLI を選定して説明します。版は VS2022 です。VS2005 以降なら概ね上位互換性があります。

C++/CLI は VC2005 のリリースから15年以上たち、マイクロソフト社はこれを非推奨<sup>9</sup>とし、C#や C++/WinRT を推奨しています。(Visual Studio2022 ではデフォルトのセットアップ後に別途セットアップが必要になっています。)しかし、C++/CLI がサポートされる以前(つまり Visual Studio 2003 以前)の主力言語 MFC と比較すれば、ユーザーインターフェース(以下、UI と略)の作成やマルチスレッドの構築が容易であり、周辺装置や画像処理ライブラリの C 言語対応 API にリンクできることも考慮すると現時点でも有力言語です。<sup>10</sup>

なお、筆者は 2004 年頃 C++Builder(Borland 社→現在はエンバカデロ社)を採用して、画像検査機を開発した経験があります。ターゲット PC は Linux で、クロスコンパイル環境(Linux 上では Kylix)で実施しました。機能面では、両者(C++/CLI、C++Builder)とも検査・計測アプリを問題なく実現できました。

### C++のメリット

- (1) ハードウェアを扱うことが得意。メモリー領域の確保、削除が効率よくできる。
- (2) C++の標準機能で、STL、マルチスレッドを扱える。

### C++/CLI のメリット

- (3) Windows の GUI を (Win32programming や MFC と比べて) 容易に作成できる。
- (4) レガシー資産(C 言語、MFC ソースコード、Windows API)を扱うことができる。

他の言語(C++/WinRT や C++Builder)を選択する際の選定ポイントとしては以下の2点があります。

---

<sup>9</sup> (以前のバージョンで提供されていた)新規開発用のテンプレートが準備されなくなった。

<sup>10</sup> DIO ボード等のハードウェア提供メーカーの API(ライブラリ)のサンプルソースコードは、Visual Studio2013 頃を対象にテストされた物が多く残っている。一方で、C++/WinRT でのハードウェア制御の充実度が不明。マイクロソフト社には C++/CLI のサポート期間の延長を望みたい。

- (1) マルチスレッドを実現できるか。検査・計測システムでは、GUI 画面への高速画像表示をスレッドセーフで可能か。
- (2) メモリー領域の確保、開放を(ガベージコレクションでなく)制御できるか。

## 2.基本技法

### 2-1.C 言語(含 C++,VC++)について

C 言語は、元来 UNIX 開発言語であり、検査・計測システムの開発用言語としても、永く使用されてきました。クラスやオブジェクト指向の概念を付加した拡張が C++言語です。

VC++(Visual C++)は、Windows 向けに API やグラフィックス機能が付加された、複数種の C++の総称(Windows プログラミング, MFC, CLI 等)です。

C もしくは C++は、レガシー資産(ハードウェアドライバ、ライブラリー)を再利用しやすく(デバッグ時にソースコードを奥深く追って行ける)、現在でも検査、計測分野では、第一選択肢でしょう。もちろん UI 作成に C#を使用し、レガシーライブラリーを呼び出すことも可能ですが、すべてを C/C++言語で作成するよりも、技術的には難しくなります。

ところで C、C++言語の教科書知識(主に文法の説明)と Windows 上の UI 付きのプログラムを実現するための必要知識間にはギャップがあると思います。例えば、IDE により雛形として自動生成されたソースコード(MFC や CLI)は、初学者から見れば意味不明な部分です。また、C++/CLI のヘッダファイル(拡張子 h)内に関数実態に記述を行い、ソースファイル(\*.cpp)は空の状態で運用する状況は、C もしくは C++言語の教科書とかなり乖離しています。

### 2-2.開発環境のバージョン

以下の製品については、OS、IDE に対応した版(バージョン、リビジョン)を選択しないとベンダーのサポート対象外になるので、正確な把握が必要です。

- ・画像ライブラリ(OpenCV、Halcon 等)
- ・PLC と PC の通信用 API(例. 三菱電機製 MX Component)
- ・画像処理端末のドライバソフト
- ・カメラボード、DIO ボード等のインターフェースドライバ

上述のプログラムには、32bit or 64bit の選択<sup>11</sup>があります。

OS のサポート寿命は約10年、産業用 PC の保守期間は長くて7年程です。このため、運用期間10年を超えるシステムについては、新旧の技術内容を調査して、開発環境自体のバージョンアップも検討します。Visual Studio の C 言語 (MFC、CLI)<sup>12</sup>についていえば、バージョン間100%の上位互換性はなく、まれにですが、コンパイルが通らなかったり、実行時にエラーが発生したりします。

まあ、周辺装置のドライバが最新の OS や IDE に対応していない場合には、意図的に世代前の版を利用することもあります。

### 2-3.変数のスコープ、命名規則

変数のスコープは重要な概念ですが、C、C++言語の教科書知識と IDE 利用言語の実践的な使用方法の間には少しギャップがあります。自動生成されたソースコードのどの位置に変数(特に、CLI 独自のマネージ変数)を定義すれば、〈静的な〉グローバル変数として使用できるのかが分かりにくいかもしれません。

実用的なプログラムを書くためには、静的変数(プログラムが起動した後に唯一インスタンス発生し、終了まで存在する、グローバル変数)を使いこなす必要があります。カメラ、DIO ボード、PLC 通信ソフトなどは、アプリ起動時に初期化関数を呼び、その際に得たハンドル(API で利用するためのハードウェア用のポインタ)をグローバル変数に格納して、API 関数をいつでも利用することができます。

グローバル変数として、C++クラスの静的メンバ変数があり、STL のコンテナ (std::vector, std::map など)をグローバル変数として使用できます。2つの例題で、蓄積データや検査結果の格納領域に多用しています。

IDE やネット環境が未発達な環境下で複数人が共同してプログラミングした時代には、変数(特にグローバル変数)の命名規則やモジュールの結合強度の理解と管理(命名規則の順守)が重要でした。しかし、現在は1プログラム内で膨大なメモリー空間を使用できるので、モジュールを結合強度に注意して分割する必要性が減りました。また、コーディング中にカーソ

---

<sup>11</sup> 片方しかサポートしない製品を複数扱う場合には、特に注意が必要です。

<sup>12</sup> この20年間では、VS2003, VS2005, VS2008, VS2010, VS2013, VS2015, VS2019, VS2022

ルを置くだけで変数定義がリアルタイムで表示されるため、命名規則の緩さをカバーしてくれます。<sup>13</sup>変更が必要になった場合も、プロジェクト内の文字列の置換で瞬時に変更できます。

```
// Form1.h
//

//OpenCV 用
#include <opencv2/opencv.hpp>

#include <stdio.h>
#include <windows.h>

//STL 準備
#include <string>
#include <vector>
#include <queue>

//画像サイズ
#define IMG_W 2500
#define IMG_H 2000

//tmp フォルダ (画像、ログ) フォルダ
#define TMP_FOLDER "c:/tmp_PFD"

////////////////////
// 通常型のコモン変数はここ

//1 画像分のワーク
unsigned char plmgBuf_[IMG_W * IMG_H * 3];
unsigned char plmgBuf2[IMG_W * IMG_H * 3];

//撮像画領域 (リングバッファ)
unsigned char plmgLingBuf[IMG_W * IMG_H * 3 * IMG_NUM_MAX];

//キュー用の構造体
struct ImgInQue {
    int No;      // 0,1,2,...
    int pt;      // リングバッファへのポインタ
};

std::queue<ImgInQue> _queImg; //空

//画像カウンター
int gCnt = 0;

static CRITICAL_SECTION gCS;

int func_PFD_(long long N, std::vector<long long> & vecPFD) {
    ...省略...
    return(0);
}
```

<sup>13</sup> 筆者のソース内では、グローバル変数の先頭に“g”をつけています。

```

int func_PFD_(long long N, std::vector<long long> & vecPFD) {
    ...省略...
    return(0);
}

namespace CLR_Form {
    using namespace System;
    (略)
    using namespace System::IO;

    public ref class Form1 : public System::Windows::Forms::Form {

        //////////////////////////////////////
        // スコープ注意
        // グローバルに使用したいので、外側に置きます
        //////////////////////////////////////
        Bitmap^ bmpG1;
        Bitmap^ bmpG2;

        // gridView と連携するテーブル
        System::Data::DataTable^ _dataTable1;

    public:
        Form1(void) {
            InitializeComponent();
            //
            //TODO: ここにコンストラクタ コードを追加します
            //
        }
        //

    private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) {
        ...以下、省略...
    }
}

```

## 2-4.STL(C++標準ライブラリ)

C++標準ライブラリである STL は、汎用アルゴリズム群を提供します。検査、計測領域のプログラミングには必須です。初学者はまず、std::string の使い方を学んだ後、構造体やクラスを定義して、vector,map などと組み合わせた小プログラム群を書いてツールとして習得してください。検索アルゴリズムを自作していた時代もありましたが、今は map に登録して find すればよく、大変楽になりました。

後述の二つの例題中では、キュー(queue)、マップ(map,multimap)を排他制御(クリティカルセクション)して使用しています。

## 2-5.処理時間の計測

タクトタイムの実現性を早期から評価しておくのはシステム設計上の重要な課題です。そのためには、関数やアルゴリズムの実行時間を適切に計測、評価する必要があります。以下は、VC++ CLI での時間計測コードです。

```
DateTime start = DateTime::Now;
... (時間のかかる処理 必要ならN回繰り返す)
DateTime end = DateTime::Now;
TimeSpan t = end.Subtract(start);
MessageBox::Show((t.Seconds * 1000 + t.Milliseconds).ToString("#####"));
//ミリ秒単位で表示
```

一般に、時間計測関数の測定のはらつきは数十ミリ秒(以上)であるため、平均値や最大値で評価します。特に、アプリ起動直後の計測時間は、極端に遅くなることがあります。OS やプログラムの仕組み上、このブレの解消は容易ではありません。このような場合、実践的なテクニックとして、アプリ起動直後に空(から)検査を実行すればよいです。なお、実行時間は、デバッグモード or リリースモード、外付け記憶媒体(SSD or HDD)に依存するので、最短～最長の条件も認識しておきます。

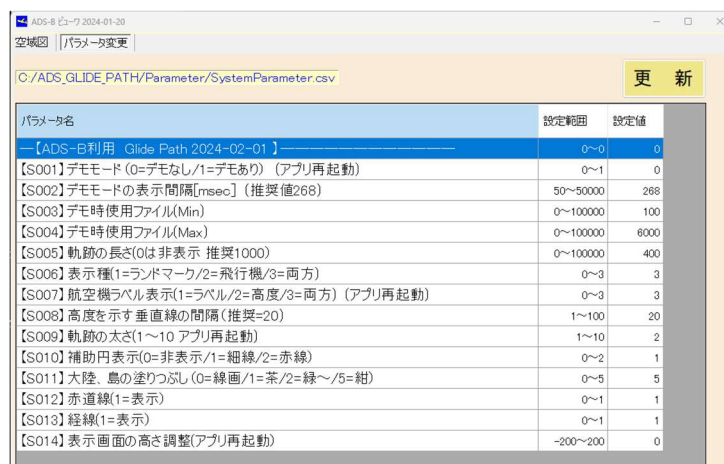


## 2-6.CSV ファイル

パラメータファイル、結果データファイル、各種ログファイルは、CSV 形式を使うのが便利です。初心者は、カンマ区切りファイルの読み書き用関数を（一度は）自作してみるとよいです。検査機では、銘柄（ブランド）毎のツリー構造ゆえ XML ファイルの選択もありますがプログラミングの難易度は上がります。

```
2024/01/29 9:01:46,Glide_Path 起動しました
2024/01/29 9:33:12,Glide_Path 起動しました
2024/01/29 9:33:36,交点が奇数個
2024/01/29 9:41:30,disappear:ANA642
2024/01/29 9:41:56,disappear:ANA293
2024/01/29 9:42:22,disappear:SKY706
...
```

検査機が運用されると、遠隔地から工場オペレーターに対して、“パラメータ”の変更を依頼することがあるかもしれません。CSV ファイルをテキストエディタで修正するのは避けたいところです。この場合、UI に VC++ の dataGridView が使用できます（実例2に実装例）。



パラメータ名	設定範囲	設定値
—【ADS-B利用 Glide Path 2024-02-01】—	0~0	0
【S001】デモモード (0=デモなし/1=デモあり) (アプリ再起動)	0~1	0
【S002】デモモードの表示間隔[msec] (推奨値268)	50~50000	268
【S003】デモ時使用ファイル(Min)	0~100000	100
【S004】デモ時使用ファイル(Max)	0~100000	6000
【S005】軌跡の長さ(0は非表示 推奨1000)	0~100000	400
【S006】表示種(1=ランドマーク/2=飛行機/3=両方)	0~3	3
【S007】航空機ラベル表示(1=ラベル/2=高度/3=両方) (アプリ再起動)	0~3	3
【S008】高度を示す垂直線の間隔(推奨=20)	1~100	20
【S009】軌跡の太さ(1~10 アプリ再起動)	1~10	2
【S010】補助円表示(0=非表示/1=細線/2=赤線)	0~2	1
【S011】大陸、島の塗りつぶし (0=線画/1=茶/2=緑~/5=紺)	0~5	5
【S012】赤道線(1=表示)	0~1	1
【S013】経線(1=表示)	0~1	1
【S014】表示画面の高さ調整(アプリ再起動)	-200~200	0

図 6 CSV ファイル修正用 UI

## 2-7. ファイル内、GUI 内の日本語使用について

ビジネス利用で Windows のフォルダ名やファイル名に日本語(漢字表記)を使用するのは普通のことですが、検査・計測システムでは、(ソートの順番不定や文字化けなどを避けるため)使用を避けたほうが無難です。<sup>14</sup>半角英数字の範囲内での使用を推奨します。

日本語(2バイト文字)の使用は、

- ①パラメータファイル内の説明文
  - ②GUI コンポーネントのテキスト(例. ボタンの名称)
  - ③ログファイル内の記述(頻度の少ないエラーメッセージに限定するのが無難)
  - ④ポップアップエラーメッセージ
- に限定するとよいです。

## 2-8. 領域確保とメモリーリーク

大量画像の連続領域メモリー確保する際、C++では malloc 関数を使いますが、開放(free)し忘れるとメモリーリークが起こります。最近の PC は大量に実メモリーを搭載しているため、短時間メモリーがリークしてもハングアップすることは少ないですが、連続運転下では絶対に避けるべきです。タスクマネージャーのパフォーマンスモニタのメモリー使用量を日単位で記録することでリークの有無のチェックができます。

“ガベージコレクション”を使う場合の注意として、メモリーの開放がプログラマーの意図した時点で発生しないことがあり、メモリーリークの確認テストがしにくかったことがあります。多量のメモリー確保と解放が必要な場合、malloc、free のペアで使うことも考慮します。

---

<sup>14</sup> ウォッチドッグのポーリング毎(数 100msec 間隔)、GUI 上のラベルにテキストを表示し続けるアプリを開発した際、テキストに日本語が混じると、まれにアプリがハングアップする現象に遭遇した。

## 2-9.各種 API の利用方法(コンパイル時、ビルド時の指定)

周辺装置(カメラなど)を制御する場合、装置メーカー提供の API を利用します。プログラマーがまず把握すべきは、対応 OS、対応言語(バージョン)、ビルド時ビット数(32/64bit)に対する対応状況(サポートの有無)です。

一般に、OS や IDE の更新頻度は数年に1度あり、装置の API(ドライバを含む)がそれらの更新に追いつかないことが多いです。この場合、上位バージョンでも動く可能性が高いので、機種選定時にテストする価値はあります。ただし、ビルド時ビット数は、同じでないと動きません。(例. あるメーカーの PLC は 32bit のみの対応であり、ある種の画像ライブラリは 64bit のみの対応である場合、同一プログラム内では、PLC と画像ライブラリを同時に制御できません。)

API 用の DLL を使うための設定場所は IDE 内のプロジェクトのプロパティ画面で、下記の①～③を設定します。

- ①コンパイル時のインクルードファイルの指定
- ②リンク時の LIB ファイルの指定
- ③実行時の DLL ファイルの場所の指定

Visual Studio では、ソリューションエクスプローラ内のプロパティで指定しますが、ここは設定の難所のひとつです。③については、EXE 形式と同一のフォルダに格納すれば特別な指定は不要です。

なお、特別な API として、Windows API (Windows 用のシステムコール関数群)があります。C、C++ のソースコードなら、数行のインクルード文の追加で使用できます。C++ CLI や C#の閉じた世界の関数だけでは実現が困難な場合にも、Windows API の利用で実現できることが多いです。(参照⇒付録の大サイズビューワ内の、拡大画像の保存部分)

## 2-10.PLC との連携

PC に PLC<sup>15</sup>を LAN 接続して、周辺機器との通信や制御を行います。

例1. エンコーダ信号を PLC 経由で受信し、PC 内で測長データとして利用できます。

例2. データの入力操作(生産する銘柄の選択、開始信号など)を行う場合に、マウスやキーボードからではなく、PLC に接続したタッチパネルを利用できます。

例3. 複数台の PC 間の通信手段に利用できます。<sup>16</sup>

LAN 接続された PLC と PC は、共有メモリー(デバイス)へのアクセスを通じて、開発を切り分けて(=分担して)行うことができます。システム開発の分担手順の例として、以下を紹介します。

- (1) 制御盤、PLC、端子台の設計、制作を電気・計装担当会社に依頼する。
- (2) PC 側製作担当者は、PC⇄PLC 間タイミングチャートを製作する。
- (3) 両者(電気・計装担当、PC 側製作担当)は、タイミングチャートに従って、プログラムを製作する。
- (4) 両者は、PLC⇄PC 間の接続テストを行う。

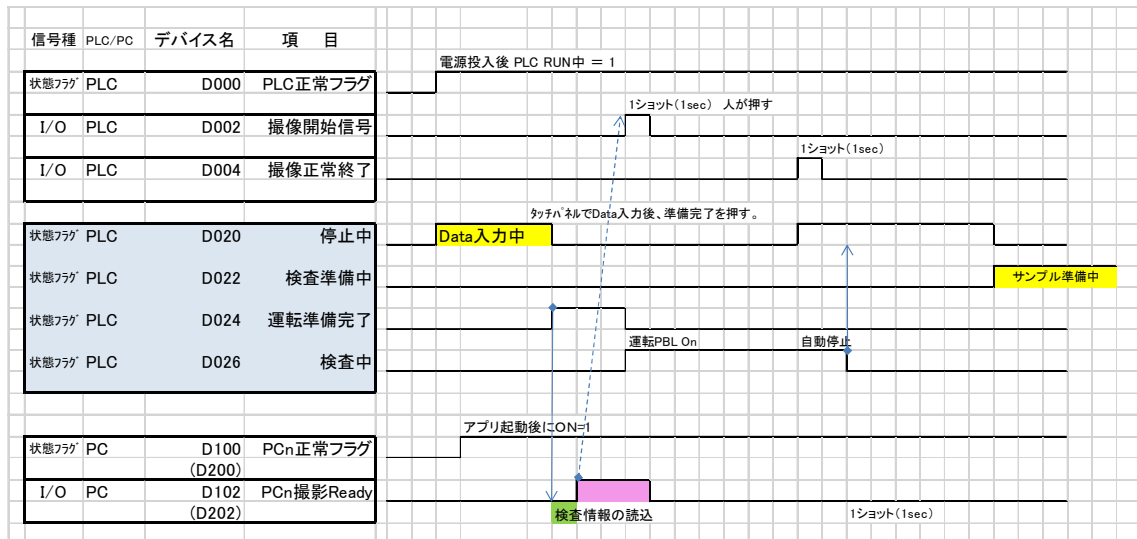


図 7 PLC⇄PC 間タイミングチャート例

<sup>15</sup> “シーケンサ”は、三菱電機の商品名で、一般名称は PLC。ソフト開発用 API は、会社毎に異なる。

<sup>16</sup> ソケット通信のコードを書くよりも扱いやすい。

## 2-1 1.DIO ボード類

PC 内 PCI バス用の空きスロットに DIO ボード<sup>17</sup>、A/D 変換ボード等を装着して周辺装置の制御やデータ I/O が可能です。DIO ボードの反応速度は LAN よりも速いので、タクトが厳しいときの PLC⇄PC 間の通信手段の補完選択肢にもなります。

PCI ボードはその大きさから、PCI、PCI Express × 1、PCI Express × 4、PCI Express × 8、PCI Express × 16 に分類されます。PCI Express × N の形式で“N”をレーン数と呼びます。N が大きいほど、転送速度は早くなります。

選定で注意すべきポイントは、

- ① ビット数 ボードのビット数に応じて、ケーブルや端子台の型式も異なります。
- ② レーン数 PC の空きスロット状況を把握します。
- ③ (必要時)供給電圧
- ④ 複数のボードを装着する場合には、総電源容量や物理的なサイズ(厚さ 隣のボードと干渉しないかを確認)も確認する必要があるります。
- ⑤ PCI ボード類は受注生産されることが多いため、納期には注意<sup>18</sup>を要します。

特に接続する周辺装置が多い場合には、システム構成図を正確に記載して誤選定(主にスロット数不足)を防ぎます。

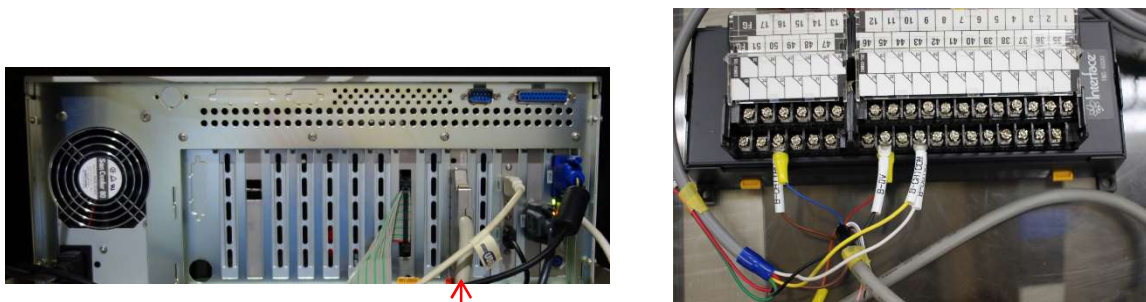


図 8 (左図)PC に装着された DIO ボード類 (右図)端子台

<sup>17</sup>正確には、DI ボード、DO ボード、その両方の機能を持った DIO ボードを示す。

<sup>18</sup>新型コロナや半導体不足の影響を受け、納期半年以上という時期があった。

以下は、DIO ボードの Read 関数の利用例のソースコードです。

```
HANDLE    hDIOHandle;    // DIO ボードのハンドル
.....

// DIO ボードのオープン
hDIOHandle = DioOpen("FBIDIO1",0); //インターフェース社の GPC2000 対応ボード
...

// 読み込み
int nBuffer[8];
int nRet = DioInputPoint(hDIOHandle, &nBuffer[0], 1, 2 ); //1 番目から、2 ビット分
In1=nBuffer[0];
In2=nBuffer[1];
...

// DIO ボードのクローズ
DioClose(DIO_Control::hDIOHandle);
```

図 9 DIO ボードの使用例(インターフェース社製品の例)

## 2-12.アクチュエータ、ロボシリンダとの連携

アクチュエータ、ロボシリンダの制御は、メーカー提供のコントローラやプログラムを用いて、予め可動範囲や可動コースを学習(“ティーチング”)して使うことが一般的です。

しかし、必用であれば PC と直接接続(USB ケーブル、LAN ケーブル、シリアルケーブル等)して、アルゴリズムを組み、サーボ ON/OFF、移動、停止等を制御することもできます。<sup>19</sup>

直接接続のための API 開発として、低レベルの COM ポート通信ソフト(後述)を作成する手法がありますが、<sup>20</sup>デバッグ時や調整時の物理的な干渉(衝突)には注意を払う必要があります。特に、出力の大きなモーターを操作するプログラムは、リスクアセス、安全対策(安全柵など)を検討し、機械、安全担当者などからの助言を得て開発を進める必用があります。本稿の中で一番開発難易度が高く、機械・電気系エンジニアの参画が必須となるでしょう。

## 2-13.複数台の PC の連携

(後述するマルチスレッド手法により)スレッドを分割してもタクト条件が満たされない場合や複数にシステムを分けた方が開発しやすい場合には、PC を複数台連携運用する手法があります。

LAN 上の共有 HDD に各々の PC からファイルにアクセスすることで実現できる簡単な手法があります。PC1 から、共有フォルダにファイル A を生成する。PC2 は、共有フォルダを監視し、ファイル A が生成されたら、アクション A を起こす、などです。原始的な方法ですが、タクトの制約が少ないシステムでは支障なく使えます。

もう少し反応速度が必要ならば、ソケット通信を各々の PC 内でプログラミングすることもできます。<sup>21</sup>両 PC に PLC を装備し、PLC を介して通信することもできます。

転送速度やアクセス時間をもっと速くしたければ、後述の光通信ボードの選択肢があります。この場合、PC 内には、光通信ボードを装着するスロットが必要です。

---

<sup>19</sup> 参考(IAI 社)[https://www.iai-robot.co.jp/download/tashakiki/pdf/SERIAL-COMMUNICATION\\_MODBUS\(MJ0162-3B\).pdf](https://www.iai-robot.co.jp/download/tashakiki/pdf/SERIAL-COMMUNICATION_MODBUS(MJ0162-3B).pdf)

<sup>20</sup>筆者は、過去に対象サンプル固定下でカメラを2～3軸ステージで操作するシステムを開発した経験を持つ。ステージが移動する間の待機処理や干渉回避が課題に加わった。

<sup>21</sup> 他の手段もあるが筆者未利用。例えば、MFC では Shared Memory(共有メモリー)機能が使える。

## 2-14.光通信ボード

光通信は、複数台の PC 間で高速な画像データ転送機能を提供します。各々の PC スロットに光通信ボードを装着し、光ファイバーケーブルで接続します。メーカーから提供される専用 API を使い、マスター、スレーブのソフト処理<sup>22</sup>を記述します。光ケーブルは、長さを 100m 程度まで延長でき、しかも、共有された光メモリーへアクセスが速いです。価格に見合った働きをするので、検討の選択肢に入れるべき技術です。特に、AI 画像検査システム（撮像 PC、AI 判定エッジ PC、学習データ保存 PC）の構築に適しています。

## 2-15.GUI 画面の基本形

PC 上のアイコンをダブルクリック（もしくは、PC 起動時に自動起動）して、GUI 画面を起動します。その画面には必要に応じて、下記項目を表示します。

- ① 銘柄選択
- ② 検査開始（／中断／停止）ボタン
- ③ パラメータ変更
- ④ 検査・計測中のデータ／判定結果等の表示
- ⑤ 各種メンテナンス機能

工場内で連続運転するシステムでは、①、②は、タッチパネルで入力されて PLC 経由で PC に伝えられることが多いようです。

③については、PC 作業に不慣れな操作者が使用する可能性がある場合には、専用の GUI 画面を設けるべきです。

④は、撮像画像や取得データを計測、判定処理して、グラフ、マップ等で可視化して表示します。ルールベース画像処理、AI 画像処理はここで行います。

⑤は、周辺装置の単体機能を確認できるアプリを設けておきます。例えば、DIO ボードなら、接点全部を ON/OFF する機能、カメラなら 1 画像撮像 & 表示機能、などです。

作成したアプリの EXE 形式や IDE 本体は、管理者モードでの実行を基本にします。ある PLC 通信アプリは、Windows7 以前の OS をベースに開発が停滞しており、最新 OS の管理

---

<sup>22</sup> アバールデータ社製品の API は、リング上の複数 PC を“ドアベル”で制御する。（中級のプログラミング技術が必要）



者権限の設定に追従していませんでした。このため、PC 自体のセキュリティレベルを下げたり、あるいは実行形式を管理者権限に上げたりしないと正常稼働しない場合があります。

## 2-16. マルチスレッドプログラミング

マルチスレッドの技法は、実運用に耐えうる検査・計測用のシステムを製作する際に必要なプログラミング技法です。

以下の利点があります。

- ① 時間のかかる処理を行っている最中に、中断ボタンを(遅延なく)受け付ける。
- ② 複数の周辺装置やデータ I/O に対する制御処理を分割して、非同期で稼働させることにより処理効率を上げる。

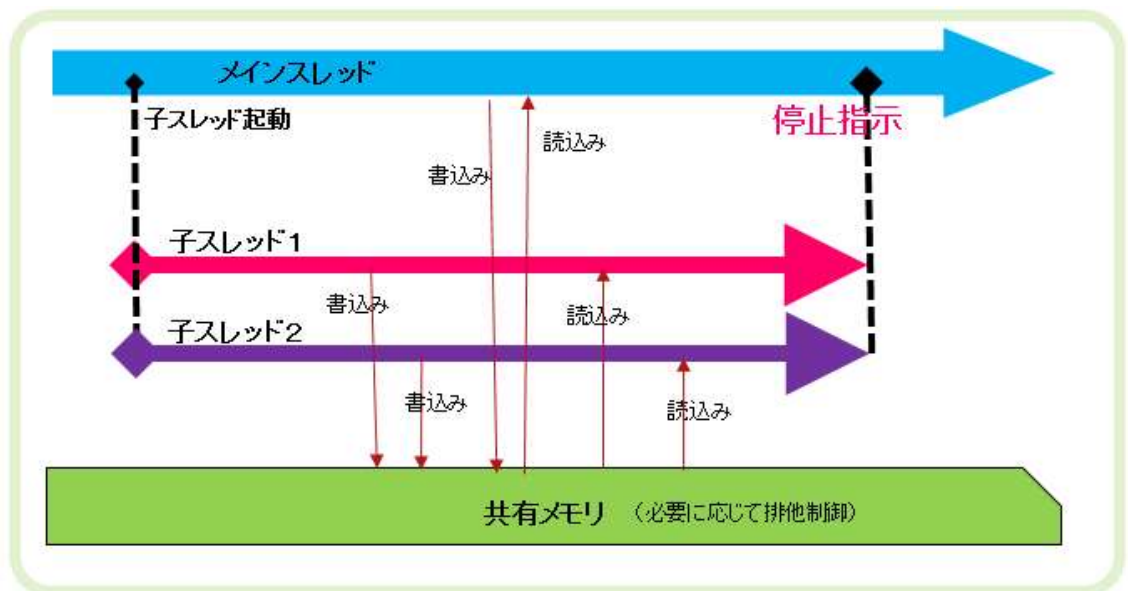


図 10 マルチスレッド概念図

スレッド数が増すごとにソフト開発の難易度は高くなります<sup>23</sup>。要件定義～設計～実装～デバッグ、更には相手側周辺装置（PLC 等）の挙動理解など多くの知識を総動員して、マルチスレッド対応の連続運転システムは開発されます。

<sup>23</sup> 例えば、<https://atmarkit.itmedia.co.jp/ait/articles/0503/12/news025.html> 筆者は、7 スレッドのアプリを作成したのが最多である。

後述の例題1の素因数分解アプリは、スレッド数を可変(計算部スレッド数1~5)にすることができます。スレッド数が増えると CPU 使用率がアップし、PC のパフォーマンスが向上します。

## 2-17.スレッドセーフ(変数の排他制御)

各スレッドは、同一変数に対する同時の読み／書きには競合が起きないように配慮する必要があります。配慮された状態を、“スレッドセーフ”と呼びます。

特に、検査データを格納したベクターやマップなどの STL コンテナに対して、追加と削除の競合は厳密に避ける必要があります。競合を避ける手法は複数ありますが、まずは、WindowsAPIのクリティカルセクションの利用を推奨します。コード内の EnterCriticalSection と LeaveCriticalSection の行間で対象変数を変更するようにコーディングすると、他のスレッド内の同様箇所(EnterCriticalSection と LeaveCriticalSection の行間)での競合を避けることができます。(片方は待ちの状態になる。)後述の2つの例題内では、共有の STL キュー(Queue)変数への追加、参照、削除の前後でクリティカルセクションを使用しています。

なお、排他制御のコードを入れ込まなくても、短時間のテストでは正常稼働するように見ることがあります。しかし、重大なバグ(リングカウンタのズレなど)を抱えることになるため、決して手を抜いてはいけません。

## 2-18.スレッドセーフ(対 GUI コンポーネント<sup>24</sup>)

(C#、VB でも同じ問題がありますが、)マルチスレッド時の競合問題は、共有変数だけでなく、GUI コンポーネントに対しても起こります。C++/CLI ワーカースレッドから、GUI コントロール(例.プログレスバー)へのスレッドセーフなアクセス(=操作)方法は、ネット上でも質問や手法が寄せられているテーマです。以下は、対策の1手法です。

C++/CLI では、子スレッド(ワーカースレッド)内から、“直接”的に GUI コンポーネント(例 テキストボックス、ピクチャーボックス)への書き換えを禁じています。<sup>25</sup>マイクロソフト社の解説では、子スレッド(ワーカースレッド)からプログレスバーを更新する際に、ReportProgress

<sup>24</sup> ピクチャーボックス、テキストボックスなどの GUI を構成する部品のこと

<sup>25</sup> コンパイル、ビルドはできるが、実行時にエラーを発生したり、意図しない挙動になったりする。VC++ のバージョンによって挙動が異なるかもしれない。

を使う例が提示されています。この関数はプログレスバー専用ではなく、非同期操作の進行状況をユーザーに報告する方法であるので、(プログレスバーを更新する代わりに)テキストボックスやピクチャーボックスでも使用できます。子スレッド(ワーカースレッド)内で ReportProgress 関数を発行し、親スレッド内の ProgressChanged 関数を誘発する際の引数 (e->ProgressPercentage) を利用し、これをポインタとして共有変数にアクセスし、文字情報や画像を更新表示できます。この際、ProgressChanged はすぐに呼ばれるとは限らないので、表示のリズムが少し乱れるように見えることはあります。<sup>26</sup>

## 2-19.カメラによる画像の取得

PC にカメラを接続して画像を取り込むアプリの撮像トリガ(=シャッター)には、外部トリガ信号(例 エンコーダ同期信号、在荷センサー信号など)、内部からのソフトウェア信号(例 マウスクリックで撮像)の2通りがあります。以下、外部トリガ信号をカメラリンク対応ボードで受け取る撮像系を想定します。

外部トリガ信号をカメラリンク対応ボードで受け取ると、画像取得完了時にコールバック関数が実行されます。プログラマーは、その関数内に画像処理アルゴリズムを記述することができます。コールバック関数を使用しないで、ウォッチドック方式で画像取得完了フラグを監視してもよいです。

カメラからの入力データは、バッファ(カメラボード上のメモリー)に転送されます。タクトが速いときは画像格納エリアにリングバッファ構造を採用して上書きによる欠落を避ける手法があります。(例題1参照)

エリアカメラ撮像でストロボ撮影する場合や、ラインセンサーカメラのエンコーダ同期撮像も、カメラリンク対応ボードのパラメータ設定により実現できます。<sup>27</sup>

## 2-20.画像処理ライブラリ

画像処理ライブラリには選択肢があり(例 Halcon、OpenCV(フリー))、単純な画像処理なら教科書を参考にして自作することも可能です。ライブラリを使う場合には、まずビルドビット数(32/64)を確認します。API 利用時の画像データ領域は、C 言語の BYTE 配列とは異なる特

---

<sup>26</sup> 同のような動きは、timer1\_Tick、fileSystemWatcher1\_Changed で見られる。この関数内は、GUI コンポーネント操作できるようだ。

<sup>27</sup> [https://www.avaldata.co.jp/solution\\_imaging/cameralink\\_tips/aiptool\\_encoder.html](https://www.avaldata.co.jp/solution_imaging/cameralink_tips/aiptool_encoder.html)

殊な形式(オブジェクト)になることがあります、BYTE 配列⇔特殊領域間の変換を習得する必用があります。<sup>28</sup>

例題 1 内の以下のコード部分が一例です。

```
cv::Mat img(H, W, CV_8UC3);    //OpenCV の画像領域確保  
std::memcpy(img.ptr(), &pImgLingBuf[ip], WH3); //C 言語の pImgLingBuf から img へコピー
```

有償ライブラリの場合、ランタイムライセンスを認識して管理することも重要です。(例 OS 更新時に、画像処理ライブラリが追隨できるか)

## 2-21.画像の描画

撮像システムで、画像をモニタ上に描画して、HDD や SSD に画像を保存することは基本的な機能です。“描画”は、GUI 画面内のピクチャーボックスに画像データをコピーすれば実現できます。ただし、C++/CLI の場合、大サイズの画像(例 ラインセンサ画像 4096x4096 以上)の Image::FromFile 関数利用では処理速度が劣るので、別手法を使います。下記ソースコードを参照。

---

<sup>28</sup> 予想外に情報が少なく、難儀した経験があります。

```

(1) 画像メモリ領域内のデータを画面 (pictureBox) に表示する手順
//bmp.LockBits(...)にて bmp をメモリにロックし、それを BitmapData に入れる。
//処理終了後、ロックを解除しないと描画できなくなる。

Bitmap^ bmpG2; //最外のスコープで定義
if (bmpG2) {
    delete bmpG2;
}
bmpG2 = gcnew Bitmap(W, H, System::Drawing::Imaging::PixelFormat::Format24bppRgb);
Drawing::Rectangle rect = Drawing::Rectangle(0, 0, W, H);
System::Drawing::Imaging::BitmapData^ bitmapData
    = bmpG2->LockBits(rect,
        System::Drawing::Imaging::ImageLockMode::ReadWrite,
        System::Drawing::Imaging::PixelFormat::Format24bppRgb
    );

// bitmapData ポインタ取得
Byte* pBuf = (Byte*)bitmapData->Scan0.ToPointer();
std::memcpy(pBuf, (ソース画像領域), W*H*3);

// bitmap の Unlock
bmpG2->UnlockBits(bitmapData);

pictureBox1->Image = bmpG2;
this->pictureBox1->Refresh(); //必須

```

```

(2) HDD 内の画像ファイルを>pictureBox1 に表示する手順
System::IO::FileStream^ fs1 = gcnew System::IO::FileStream(StrViewFile1,
    System::IO::FileMode::Open);
this->pictureBox1->Image = System::Drawing::Image::FromStream(fs1);
fs1->Close();

```

( c. f. Image->FromFile(S1) では、遅かったり、エラー停止したりする)

図 11 画像の描画例

## 2-22.画像の保存

画像検査システムにおける画像保存は、主にタクト律速で全数保存は実現が難しく、欠点箇所のみをクリッピングして保存するのが一般的でした。昨今、SSD を使って AI 画像用に多量の画像を全数保存することもできるようになってきたようです。

おおまかな経験値として、2500x2000 のカラーJPG 画像の HDD 保存には 100msec 程度の時間がかかります。タクトタイムが厳しい場合、SSD の利用(例外付け SSD を複数台交換しながら運用する)やクリッピングによる一部保存も考慮にいます。

画像を連続保存するシステムでは、1フォルダ内には多量(約5千個)以上のファイルを生成しないように注意が必要です。エクスプローラのパフォーマンスが急に落ちてきます。このような場合、時刻付きフォルダを自動生成して、振り分けるとよいです。

画像保存の関数は、複数の選択肢(IDE 提供、画像処理ライブラリ提供、カメラボードドライバ提供など)があります。非可逆型の保存(例 JPEG 画像)の場合には、圧縮率を指定できますが、同じ圧縮率でも使用した関数により微妙な差(ファイルサイズが少し異なる)が存在します。

## 2-23.画像のピクセル値取得

画像処理ソフトの開発では、画像データ(ふつう連続領域)から特定の座標のピクセル値を取得する計算が頻出します。C++/CLI の初心者資料には、Bitmap.GetPixel(設定時は、SetPixel)が掲載されています。しかし、この関数は、実行速度が遅く実用的ではありませんでした。回避策として、BITMAP のファイル構造を理解し(平凡な)バッファのアドレッシング技術を身につけてください。

(例)モノクロ画像のデータ(サイズ WxH 1画素8ビット)に対して、座標値(ix,iy)のピクセル値は以下で取得できます。

```
char cbuf[N]    N=WxH 分の画像バッファ
int ip = ix + W*iy;    (0 ≤ ix ≤ W-1, 0 ≤ iy ≤ H-1)
    (→ 0 ≤ ip ≤ (W-1)+W(H-1)=W-1+WH-W=WH-1)
int pixVal = cbuf[ip];
```

ROI(関心領域)のピクセル値取得の場合には、上式をループさせればよいです。

## 2-24.簡易画像ビューワ

検査、計測分野では、ラージサイズ画像(およそ、幅が4千画像を超えるサイズ)を扱うことがよくあります。(例 ラインセンサーカメラによる撮像 マイクロ스코プの連続撮像+画像連結機能) 画像の特定の部分を拡大して描画、画像保存(位置情報付き)したい要望は自然なものであり、自作プログラムの良い練習問題になります。<sup>29</sup>

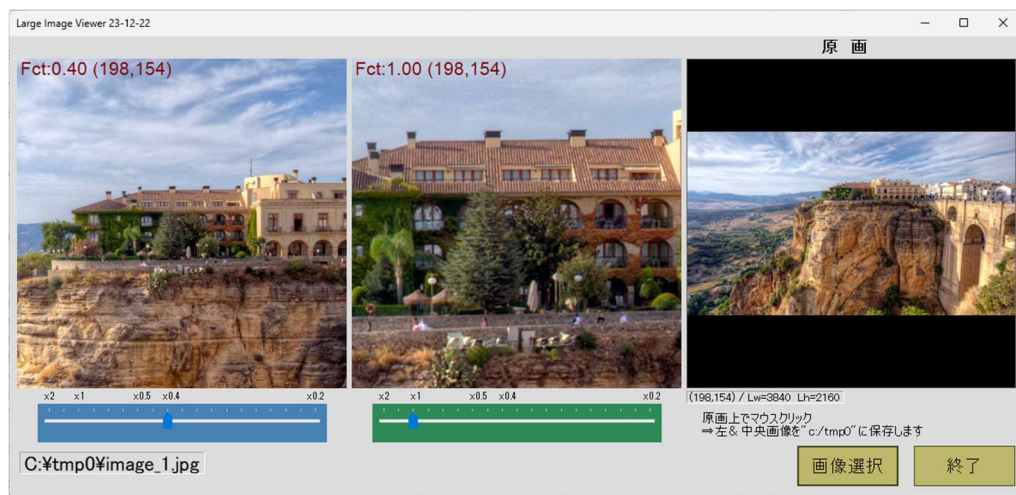


図 12 簡易画像ビューワ

## 2-25.COM ポート、シリアル通信

COM ポートは、MS-DOS 時代から存続する<sup>30</sup>シリアル通信用ポート(接続口)です。デスクトップ型 PC には今でも用意されています。シリアルポートのないノート PC でも、USB 口にシリアルポート変換コネクタを使うことができます。LAN 接続よりは低速で、近距離の通信に使用します。

カメラリンク対応カメラの場合の場合、シリアル通信ソフト(例 Tera Term)を用いてカメラ本体にコマンドを送ることにより設定(カラーバランス調整やゲイン調整)の変更・確認などを行うことができます。ポートの状況はデバイスマネージャのポート(COM と LPT)で確認できます。

<sup>29</sup> 付録の LIV\_Form.exe ソースコードを参照のこと。

<sup>30</sup> "COM1" ~ "COM9" 名は予約語で、この名のフォルダは作成できない。

更に、レガシー技術として、マスター&スレーブ方式の通信手順をコーディングすることで、“RS-232C 無手順通信”も利用できます。<sup>31</sup>



図 13 COM ポート(上部左側の4口)

## 2-26. バッチファイルとパイプ

バッチファイル(機能アップ版は PowerShell)は、実行したいコマンド列を記述した、拡張子”bat“のテキストファイルであり、MS-DOS 時代からの便利なツールです。

大雑把に言えば、UNIX のシェル(Shell)の簡易版であり、コマンドプロンプト上で使用します。パイプやリダイレクションも利用できます。検査・計測機システムでは、アプリの自動起動時に待機時間(秒単位)を設定したり、ping コマンドを発行して LAN 環境の正常接続を確認したりできます。

例題 Glide\_Path の補助アプリ(Stdin\_Apl1.exe ADS 電波受信プログラム)に、パイプを利用しています。dump1090.exe は、ADS アンテナメーカー提供のコンソールプログラムで、標準出力にひたすら航空機の座標を吐き出し続けるアプリケーションです Stdin\_Apl1.exe は筆者自作で、パイプでその座標情報を受けて、テキスト形式に保存する機能を有します。

dump1090\_with\_StdinAPL1.bat 内の記述

dump1090.exe --aggressive (中略) --mlat |Stdin\_Apl1.exe

<sup>31</sup> 筆者は、キーエンス社汎用画像処理端末や IAI 社アクチュエータと無手順通信した。



## 2-27.システムの自動起動

PC の起動後、アプリを自動起動するには、Windows の設定(“スタートアップ”フォルダにショートカットをコピーする)で行います。DB や PLC アプリの起動&準備を一定時間待ちたいときには、MS-DOS のバッチファイル(もしくは、PowerShell)の Sleep 関数を使うこともできます。

複数の PC を連携してシステムが構築されている場合には、PC やアプリの起動順番が重要になってきます。ある種の通信ソフトは、マスターPC とスレーブ PC の役割に分かれ、マスターPC は通信経路の開通確認をする必要があるため、マスターPC プログラム内ではポーリング待ちの必要があります。この場合には、マスターPC は後から起動するのがよいでしょう。

一方のシャットダウンですが、UPS を利用する場合、LAN 上の複数の PC を(一定時間の)停電をトリガとして、一斉にシャットダウンすることができます。また、Windows API の ExitWindowsEx 関数を利用すれば個別の PC のシャットダウンも可能です。

## 2-28.ウィルス検知ソフト

セキュリティ対策ソフトは、タクトタイムやパフォーマンスに影響を与えるアプリの一つです。工場内検査・計測システムでは、システム負荷を避けるために、インターネットへの接続を禁止して、オンラインのウィルスチェック機能を外したり、Windows Defender を無効化したりすることがあります。

その一方で、プログラムの更新時や蓄積されたデータの転送には外部記憶媒体(例 USB メモリーキー)を使用することもあり、ウィルス混入の危険が残ります。このような場合、オフライン環境でのウィルスチェックソフトを利用します。(Microsoft セーフティスキャナー)

## 2-29.さらなる展開

スタンドアロン検査機のテストが終了し、試験運用でよい評価ができれば、以下のような発展が期待できます。

- ・横展開(他の生産ラインへのリポート製作)
- ・生産管理システムとの連携
- ・AI 検査への拡張

本稿の内容を超えた新たな課題(光学系の再現性の確立、セキュリティ対策、機械学習など)が主役になります。

## 3.テスト&デバッグ技法

### 3-1.テスト環境

工場ライン内でテストやデバッグを行う場合、以下のような制約を受けることがあります。

- ・対象装置を利用できる期間、時間、マンパワーが限られる。
- ・ソフト開発には不向きな、安全衛生上の環境(危険エリア、電波状況、温度、騒音 etc.)
- ・情報セキュリティのため、ネットから分離されたスタンドアロン環境

このような状況下においては、ログファイル設計を充実して、限られたテスト期間を有効に利用する必要があります。

屋外、騒音環境下やクリーンルーム内における作業は、オフィス環境に比べて、作業効率が著しく落ちるため、(ハードウェアをダミー化してテストするなどして)結合テストまでに基本動作フローを通しておきます。騒音や電波状態によっては、共同作業者と電話連絡が取れない状況もありえます。

PLC や DIO ボードとの接続試験を行う場合、相手側(例 メカ側 PLC ラダー開発の設備担当者)の意思疎通をテスト前に行っておく必要があります。<sup>32</sup>それでも、実際に統合テストを行うと、お互いの想定の微妙な誤解などにより、想定外の状況に遭遇します。

### 3-2.ログファイル

現場でのテストは、あらかじめ準備した作業リストに従ってテストを実行し、ログファイルを生成するのが基本となります。結果の分析はできるだけ静寂な場所で行うのがよいです。

マルチスレッドのデバッグ時には、デバッガーは限定的にしか使えないので、ログファイルの出力機能を充実させておくことが重要です。ログファイルは、毎行に”時分秒”を記載し、CSV 書式にして、生成レベル(例 なし、最小、詳細)を設けておき、システム変数で切り替え

---

<sup>32</sup> 現場での開発を前提にした熟練者もいたが、相手側担当者が手持ち無沙汰になる。

可能にしておくといよいでしょう。複数スレッドからのタイムスタンプや共通のカウンター変数を追うことで、時系列の流れを確認することができます。

### 3-3.デバッグ時のツールたち

一般に、産業用 PC の初期状態は、最低限度のプログラムしかインストールされていないので、PDF リーダーや CSV 形式で出力されたログファイルを扱うための表計算ソフト(エクセル等)を準備<sup>33</sup>しておきます。また、ファイル内容比較ツール<sup>34</sup>(アプリ)を習得して、作業前後の相違点を正確に管理する必用があります。ソフトやパラメータ修正が不調に終わった場合、改造前の状態に戻す、というのが基本です。

開発期間中、特に連続運転のテスト中には、タスクマネージャー、リソースモニターによる監視(特にメモリーリークの検出)を実施します。

リモートデスクトップ作業の設定も習得しておくといよいです。作業現場の制約から、モニタやキーボードの設置する場所が確保できない場合には、LAN ケーブルをつなげて少し離れた場所に作業環境を確保できます。

### 3-4.デバッグ作業に関する補足

筆者の経験上、結合テスト～統合テストでのデバッグ作業は、普段の作業場所とは異なる環境(例 出張を伴う他県の工場内)で行うことが頻繁にありました。

出張準備品リスト、テスト計画書は現場作業の工程管理の重要な手段です。ケーブル一本忘れただけで、テスト中止になりえます。作業用の机、椅子など、工場で借用するものも手配、確認しておきます。

また、システムテスト時には、操作説明書や機器調整手順書用の写真撮影や、完成図書向けの基礎データを収集も並行して行うことが多いので、それらも計画に入れておきます。

---

<sup>33</sup> ネット接続されていない(オフライン)状態でも、ライセンス認識されて、動作することを確認のこと。

<sup>34</sup> 例えば、WinMerge など。

## 4.保守について

ハードウェア故障は確率的に避けられませんので、システムにトラブルが発生した際に、速やかに復旧するための保守技術が必要です。

### 4-1.予備 PC、HDD の準備

まず、予算段階から予備 PC, 予備 HDD を計画しておくことが大切です<sup>35</sup>。多くの産業用 PC では、購入時に予備 HDD を注文できます。HDD をまるごとコピーして保管しておきます。<sup>36</sup>

なお、現場で HDD を交換するために、制御盤の電源を止めて、マウントトラックのケーブルを外し、PC を取り出し、HDD を交換する一連の作業はかなり面倒なものです。選択の余地があるのなら、フロントパネルから HDD を交換できるタイプの PC をお勧めします。

### 4-2.カメラ、照明調整

産業用カメラの調整は手間がかかります。一般用途のデジタルカメラは AUTO 撮像モードで十分きれいな画像が写せます。一方、産業用カメラを故障交換する場合には、再現性を確保することを考えて、マニュアル調整が基本になります。マニュアル調整の際には、照明の明るさやホワイトバランス調整も同時に行う必要があるため、撮像系に適した”標準反射板”を決めて、調整を定量化する必要があります。<sup>37</sup>

なお、カメラリンク対応のカメラ調整には、①カメラリンクボードの設定 ②カメラ本体への設定 の2つがあり、混同しがちです。ホワイトバランスやカラーの色味調整は②で行います。普通、この調整は COM ポート経由で、“カメラメーカー”が準備したプログラム(カメラ内の ROM 更新用)を使います。

### 4-3.故障対応

トラブルが発生した際に、システムのハードウェアが原因なのか、別の要因なのかを的確に突き止めます。ハードウェアの故障に対しては、予備品との交換が基本になります。以下はシステム開発者が事前に行える項目です。

---

<sup>35</sup> 筆者は外付け SSD の実運用経験しかないため、システムディスクに SSD を使用した PC の保守については言及できない。

<sup>36</sup> 2 台の裸の HDD をつないで(PC を介さず)コピーできる器具もある。

<sup>37</sup> レンズのピント調整の定量化は決め手がなく、目視調整でした。

① 初期不良をつぶす。

産業用 PC と DIO ボード類間には、まれに、相性が悪く起動に不安があった経験があります。このような際は、HDD を外し、ボードは装着したままで、PC メーカーに動作テストを依頼しました。慎重を期すなら、設計段階で評価用に PC やボードを借用してテストする手もあります。<sup>38</sup>

② システム起動時に各種ハードウェアチェックを行い、正常/異常の稼働表示を行う。

例 モニタ画面上に、正常時は緑色、異常時は赤ランプを点灯します。ケーブルの緩みや抜けもこの時点で判断がつくようにします。

③ ハードウェアに対する命令(関数)を呼ぶ際に、戻り値がエラーの際は、ログファイルにその旨を記述する。

④ 保守情報をドキュメントに残す。

システムを納めたのちは、工場の保守担当者が一時対応を行います。UPS や PC のバッテリー交換は4年後以降に発生するので忘れがちです。

## 4-4.PC 更新

産業用 PC には7年程度の修理保証期間があります。その期間を超えた PC が故障した場合には、後継機 PC を用いて、システムを更新する必要がでてきます。

後継機 PC と最新 OS のもとでも、アプリを動作させるのは重要課題です。IDE(開発言語)や周辺装置のドライバ等も最新 OS に対応しているか、コンパイルやビルドが通るか、などの確認が必要となります。特に重要なのは、32bit/64bit のプラットフォームの選択です。機器 A の API は 64bit 未対応、機器 B の API は 64bit 対応のみ(32bit 対応を廃止)という状況が現実に存在しています。

長年運用されているシステムのソースコードは、年季の入った(機能拡張やデバッグの履歴などの履歴がある)ものです。筆者の経験では、ソースコードが残っていれば<sup>39</sup>、開発者以外の人でもアルゴリズム修正や改良は可能です。一方、ハード環境(ドライバソフトを含む)は適切な資料、現物品(CD など)がないと再構築できません。まっさらな PC に対して、各種周

---

<sup>38</sup> メーカーの営業担当者との定期的な情報交換をお勧めします。

<sup>39</sup> もちろん、仕様書や設計書があるに越したことはない。

辺装置、そのドライバ、IDE や画像処理ライブラリのインストール手順を記した作業マニュアルが重要となります。

## 5.例題

実稼働中のシステムを知ることは、よい勉強になりますが、企業では設計書、ソースコードは秘密文書であり、部外者は閲覧できません。たとえ自分が担当しているプログラムであっても、私用 PC にコピーするのはコンプライアンスに抵触する世の中になってきました。そこで、本稿の説明用に製作した2つの例題を提示し、画像や自作マップの高速表示技術を示すことにします。

例題1は、多数の基本技術を盛り込むことを優先にしたため、(画像処理と素因数分解が共存する)唐突感のあるアプリです。例題2は、ソフトウェアラジオ(SDR)を利用した航空機マップであり、まとまり感があります。いずれの例題も、数百 msec 毎以下でデータを取り込み、演算し、可視化表示する点は共通しています。ここでのテクニックを習得すれば、前述の図1程度の検査システムが構築できます。

### 5-1.素因数分解＋画像表示

#### 5-1-1.概要

一定間隔で撮像される画像(例題内では割込み撮像の代替として、画像 2500x2000 を定期的に生成する)を、①原画を描画 ②加工画像を表示 ③加工画像を JPG で保存する、アプリである。

1枚の処理時間は意図的にばらつきをもたせる。このため、“キュー”を利用して、データの取りこぼしを防ぐ。簡単な画像処理に OpenCV を用いる。

カウンタ値  $No$  (初期値=0)に対して、 $No \% 50$  (50で割った余り)を計算し、それに5を加えた数字に対して、2のべき乗を計算し、さらに  $No$  を加えた数値を  $NN$  とおく。

数式では  $NN = 2^{No \% 50 + 5} + No$

$NN$  を素因数分解する。数字の桁数の幅を大きくして、素因数分解にかかる時間にばらつきを持たせるための工夫であり、数式内の係数は適当に設定してある。

素因数分解は、初等的なアルゴリズムを利用する。

技法: マルチスレッド、リングバッファ、素因数分解、OpenCV、ビットマップ表示

### 5-1-2.入出力データ

(入力パラメータ)画像の撮像間隔(ここでは生成間隔)で、カウンターアップ間隔とほぼ等しい。最小値=10msec

(出力) 原画画像をカラー反転して、素因数分解結果を視覚的に表示する。最後の 100 枚のみが保存されるよう上書き保存する。

ログファイル(この中には、発見された素数リストが記載されている)

本アプリは、C:\tmp\_PFD 下に結果画像ファイル、ログファイルを生成する。

### 5-1-3.画面構成

下図に示す。



図 14 連続画像表示(素因数分解)

### 5-1-4.画像ライブラリの利用

VS2013 対応の opencv3\_1\_0(Cドライブ直下に C:\opencv3\_1\_0\build をコピー)を利用して  
いる。<sup>40</sup>実際に使用する API 関数は、ファイル保存、画像反転の2個である。コンパイル、リン  
クを通すために、プロジェクトファイルに数か所の追記が必要である。この設定は、  
CLR\_Form.sln を起動して確認できる。

<sup>40</sup> サイトはこちら <https://opencv.org/>

### 5-1-5.スレッド、メモリー、UI コンポーネントの相関図

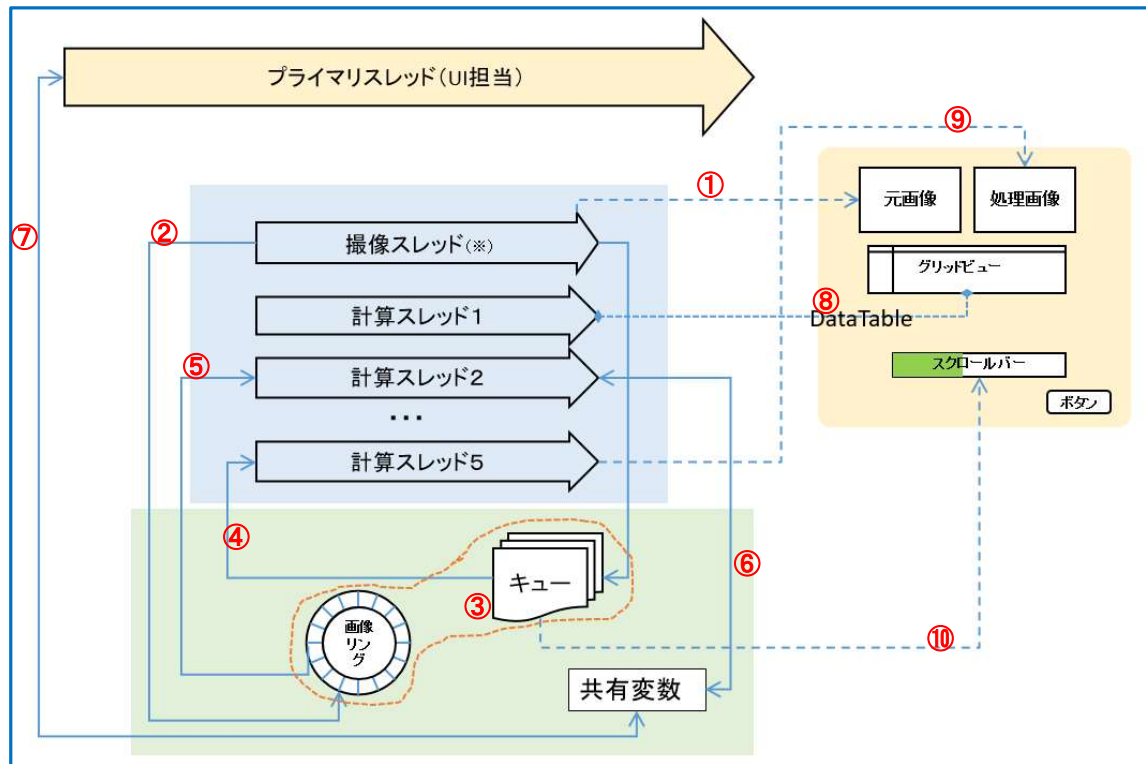


図 15 スレッドと共有変数の関連図

- ① ピクチャーボックスに画像を表示する (※)カメラ撮像の代替で、画像を生成
- ② 画像用のリングバッファに画像をコピー。同時にキューを追加。
- ③ キューに追加する際には、排他制御を行う (Critical Section)
- ④ 計算スレッド (最大5個) は、キューサイズを監視し、データが有れば受領する。
- ⑤ 上記④のデータ受領と同時に、画像データをコピーする。
- ⑥ 各計算スレッド、撮像スレッドは、共有変数にアクセスできる。
- ⑦ プライマリスレッド (メインスレッド) は、共有変数にアクセスできる。
- ⑧ グリッドビューとデータテーブルを関連しておけば、グリッドビューが自動更新される。
- ⑨ 素因数分解と並行して画像処理されたイメージを ReportProgress&ProgressChanged 経由で、ピクチャーボックスに表示する。
- ⑩ キューに蓄積されている個数をプログレスバーに表示する。



### 5-1-6.マルチスレッド化の効果

対象数値は、 $2^5 \sim 2^{54}$  の(32～約2京)の範囲内の自然数である。素因数分解の処理速度は、Core i7 ノート PC で、10京( $10^{17}$ )の自然数に対して約1秒かかる。

キューが増加しない最小の画像生成間隔に関して、スレッド数1個で 180msec、5個で 40msec 程度であった。また、スレッド数をあげると、CPU 使用率の上昇を確認できた。<sup>41</sup>

スレッド数	1	5	5	5
画像生成間隔 [msec]	200	200	100	50
CPU使用率	20	35	50	75

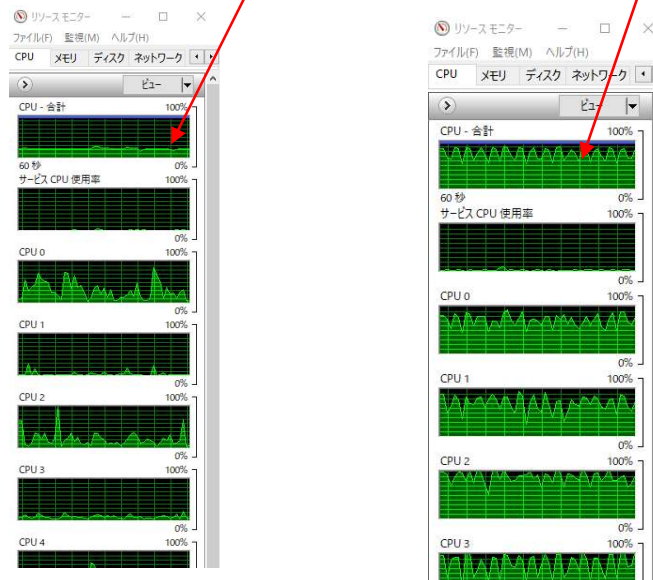


図 16 スレッド数と CPU 使用率

## 5-2.ADS-B フライト監視システム

### 5-2-1.概要

航空機の追跡アプリの簡易版である。ADS-B 用アンテナを PC に USB 接続して、リアルタイムで近傍の航空機の位置情報を取得し<sup>42</sup>、マップ上にプロットする。3次元的な軌跡 (Grid Path) を任意の視野方向で確認できる。パラメータ変更ファイル用の GUI を有する。

<sup>41</sup> 2023 年購入のコア数 16 個の PC なら間隔を 20msec まで下げられた。

<sup>42</sup> GPS、SDR(ソフトウェアラジオ)技術を利用している。

実行時モードが2つある。

モード1 リアルタイムで、ADS-B アンテナからの航空機位置座標を入力データとする。

モード2 デモモード。ケース1にて入手したデータセット(テキストファイル)を入力データとする。入力データと出力データは、同一ファイルであるが、フォルダの場所が異なる。

技法: マルチマップ(STL)、幾何計算用クラス、CSV ファイル読書、fileSystemWatcher

### 5-2-2.入出力データ

以下に ADS-B 信号をデコードしたデータ(1単位の例 tmp\_ADS\_B-00009012.txt)を示す。<sup>43</sup>

1秒間に約4ファイルの頻度で受信する。特に、下記の黄色のデータを利用する。(便名、高度、速さ、向き、緯度、経度)

Hex	Mode	Sqwk	Flight	Alt	Spd	Hdg	Lat	Long	Sig	Msgs	Ti -
86D2CC	S	3316	ANA18	6175	203	335	35.160	140.203	10	2336	1
841B6A	S	0442	SFJ43	27050	350	265	35.463	138.885	6	1473	0
8422F9	S	2212	JAL184	4275	181	300	35.268	140.026	34	3781	0
84B772	S	3361	ANA396	3125	186	330	35.417	139.892	15	8432	1
8511CA	S			13250	346	042			4	2	49

図 17 ADS-B デコードデータ例

<sup>43</sup> 筆者は、dump1090 を利用。参照サイト: RTL-SDR.COM また、Stdin\_Apl1(自作)により、ファイルを分割し、パイプ機能でファイル分割する。

### 5-2-3.画面表示例

筆者の受信環境<sup>44</sup>では、ADS-B 信号を4回/秒程度の頻度で受信できた。当該電波は直進性が高く、受信範囲はアンテナの設置高さに依存する。

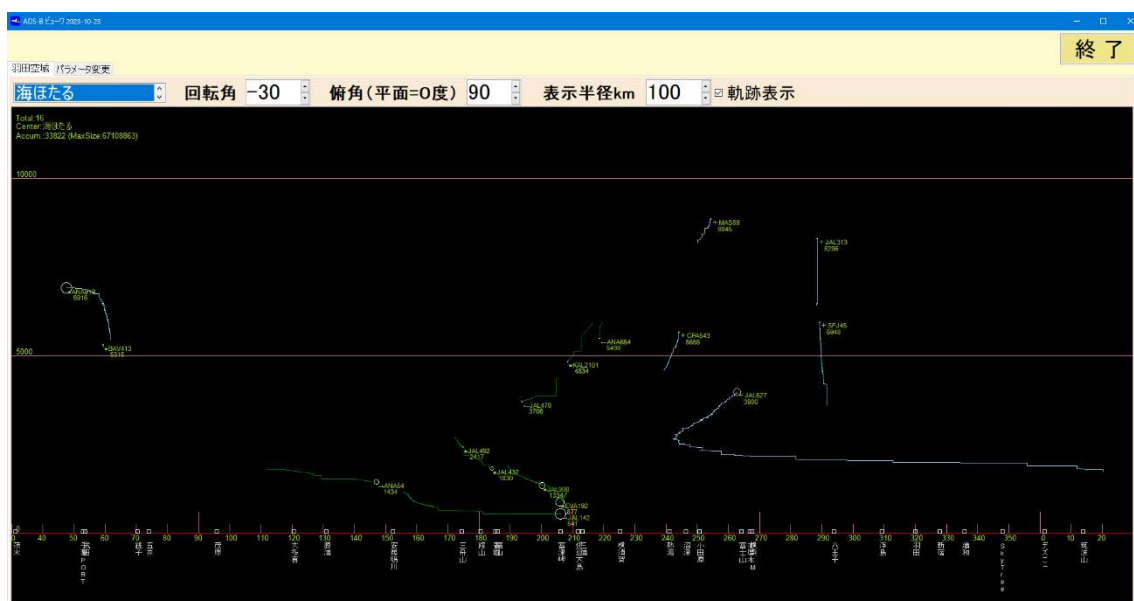
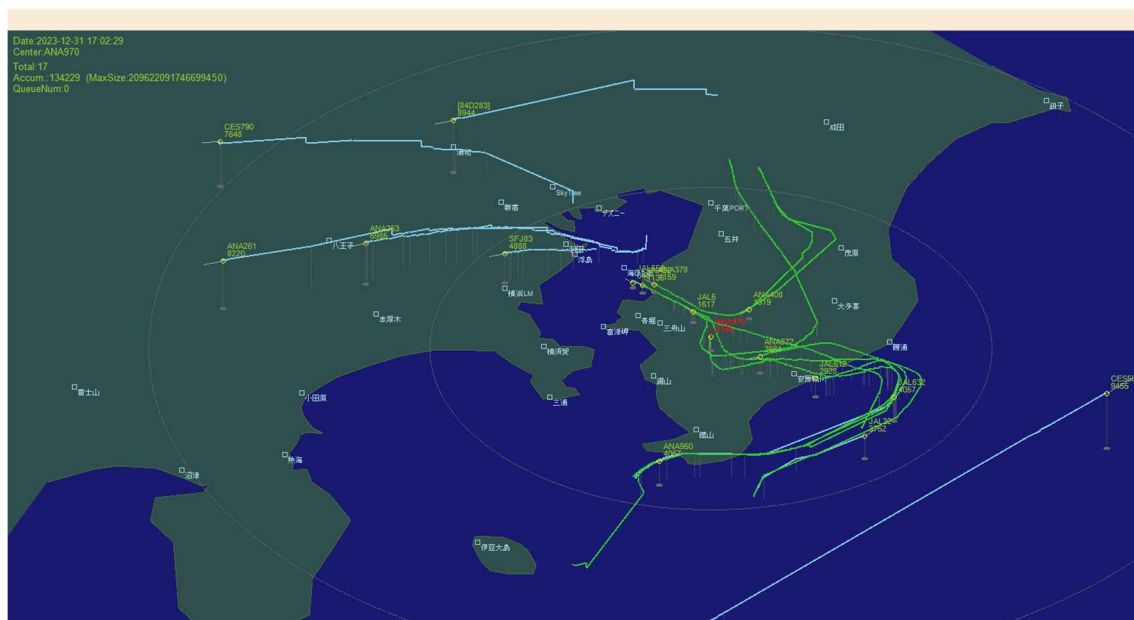
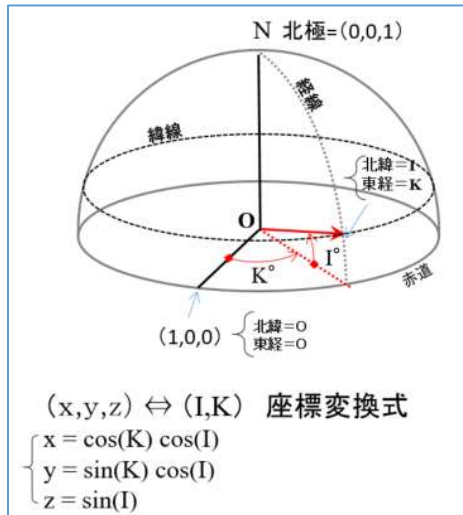


図 18 表示画面(上:斜視図 下:立面図)

<sup>44</sup> 標高 130m の三舟山(君津市)からは羽田空港内の駐機も識別できた。電波状況の良い太平洋上では、約 100km 先の航空機が確認できる。

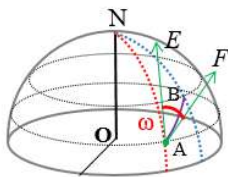
#### 5-2-4.座標系と幾何計算

2次元、3次元のベクトルや行列をクラス化して、ソースコードをわかりやすく記述できる。以下の図に球面上の位置座標(北緯、東経)の計算式を示す。



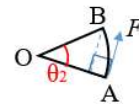
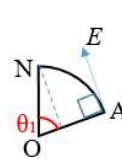
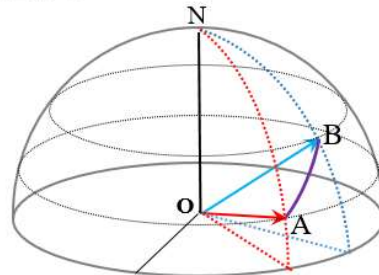
(N極基準の)Aから見たBの向き $\omega$ の定義  
弧OAN、弧OABに対して、点Aでの2つの接線AEとAFのなす角度とする。

( $\overrightarrow{AN}$ と $\overrightarrow{AB}$ の成す角度ではない)



弧ABの長さ=地球半径× $\angle AOB$   
= 6378.137 ×  $\theta_2$

A点( $A_i, A_k$ )、B点( $B_i, B_k$ )  
・AB間の距離=地球半径× $\angle AOB$   
・AからBを見たときの方位角 $\omega$ (北極を0度方向時計回りで0~360°)



$$\begin{aligned} \overrightarrow{AE} &= \overrightarrow{ON} - \cos \theta_1 \overrightarrow{OA} & \overrightarrow{AF} &= \overrightarrow{OB} - \cos \theta_2 \overrightarrow{OA} \\ \cos \theta_1 &= \langle \overrightarrow{OA}, \overrightarrow{ON} \rangle & \cos \theta_2 &= \langle \overrightarrow{OA}, \overrightarrow{OB} \rangle \end{aligned}$$

```
Vct3 vON(0, 0, 1);
Vct3 vOA, vOB; //3次元ベクトル D2Rは、deg⇔rad変換係数
vOA.x = cos(keidoA * D2R) * cos(idoA * D2R);
vOA.y = sin(keidoA * D2R) * cos(idoA * D2R);
vOA.z = sin(idoA * D2R);
vOB.x = cos(keidoB * D2R) * cos(idoB * D2R);
vOB.y = sin(keidoB * D2R) * cos(idoB * D2R);
vOB.z = sin(idoB * D2R);

double cosW = (vOA | vOB); //ベクトル内積
distABkm = Rkm * (acos(cosW)); //弧長距離

//Aは、N極点でないとき
Vct3 vAF = vOB - (vOA | vOB) * vOA;
Vct3 vAE = vON - (vOA | vON) * vOA;
double cosT = (vAF | vAE) / (vAF.length() * vAE.length()); //ベクトル内積
(この後、一工夫して、0~360°の間でωを決定できる)
```

図 19 幾何計算

### 5-2-5.簡易地図とクリッピング処理

簡易地図(海岸線とランドマーク地点)の実体は、SeaLine.csv と LocationList.csv である。前者は、大陸や島を反時計回りの閉曲線で定義し<sup>45</sup>、後者には、空港名、都市名などを記載する。大陸と海洋の塗り分けには<sup>46</sup>、大陸を閉曲線とみなして画像処理の穴埋め機能を使う手法がある。しかし、本件地図は、中心位置の変更毎に再計算が必要な正距方位図であるため、(画像処理でなく、)閉曲線の長方形クリッピング処理を行っている。この処理は、関数(sub\_Clipping)内で実現される。閉曲線のクリッピング処理は、「計算幾何学」の分野の入り口である。凸分解や隠点線処理は、3次元モデルを2次元に投影する際の重要なテクニックである。

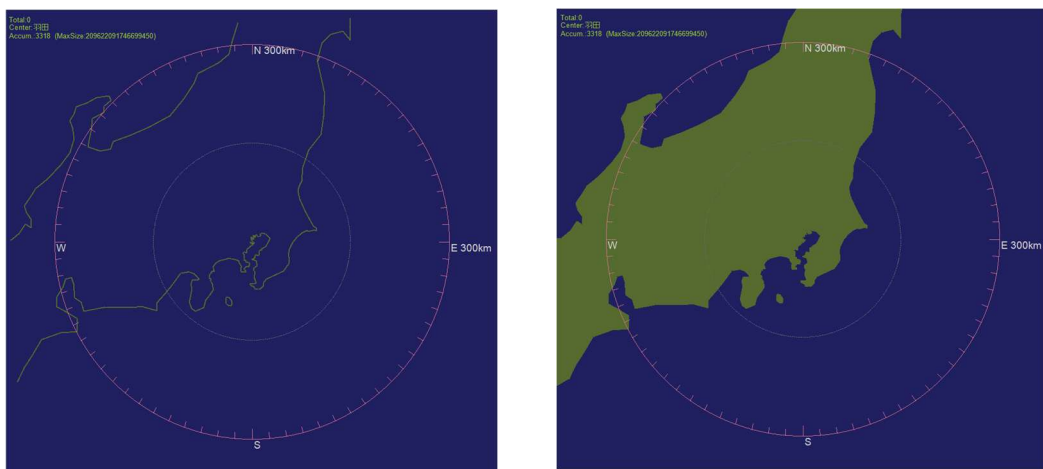


図 20 簡易地図の線画、塗りつぶし

以下の図にクリッピング処理の概念図を記す。<sup>47</sup>

ポイントとして、

- ① 閉領域(島と窓)の定義時に、向きを常に一定(例 時計回り)にすること。

<sup>45</sup> 全世界の海岸線に約 1700 点を使用した。羽田付近以外の海岸は粗い。

<sup>46</sup> Visual Studio の.NET ライブラリは、閉曲線の塗り潰し関数“FillPolygon”を提供しているが、クリッピング領域をはみ出る複雑な閉曲線(例. 本州島)には処理の限界がある。

<sup>47</sup> Known Bugs:①両極位置関係にある際の、遠方側大陸の塗り潰し ②閉曲線(大陸)がクリッピングウィンドウに(交差でなく)接する際の塗り潰しミス。

- ② 上述の向きを決めると、赤点線と青点線の接続は、自然に決定される。

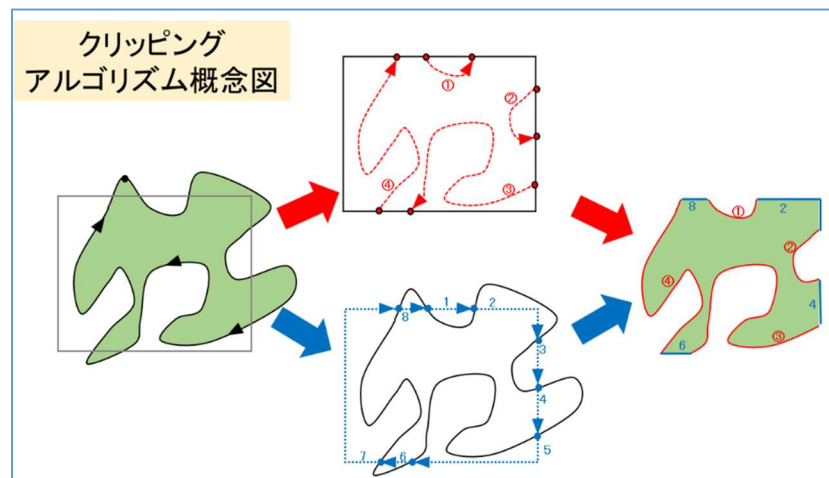


図 21 クリッピング処理概念図

#### 5-2-6.連続稼働例

以下は連続稼働の実績値です。

2024 年 1 月 16 日朝4時～夜23時(約19時間) この間、東京湾近辺で 1785 機を補足

このアプリは、航空機の軌跡の表示を ON/OFF する機能を有しますが、DB を持たないため multimap を利用して軌跡データを保持しています。このため、地図上から消えた機体のデータを捨てずにいるとパフォーマンスが落ちてきますので、適当なタイミングで不要なデータを削除しています。ソースコード内の「CGPCommon::mltMapTrail.erase(strKey.c\_str())」が該当。

連続運転する画像検査機でも同様な課題(例 欠点情報を vector に格納し続ける場合)が起こります。この場合、発生頻度と連続稼働時間を考慮して、対処方針を決めます。ある制限を超えたら、警告表示を画面やパトライトで表示するのも選択肢の一つです。

## 6.あしがき

筆者の開発経験をもとにした、小規模の検査・計測システムを内製する際のソフトウェア開発技術を紹介しました。

1980 年代 (Windows 以前) のミニコン～エンジニアリングワークステーションの全盛期に、多くの日本メーカー (自動車、電気、建築、鉄鋼・・・) が自社で CAD システムや解析プログラムを開発、販売していた時代があり、その頃に筆者はプログラミングを学び始めました。

「ソフトウェア生産技術」(電子情報通信学会 1985 年) は、要件定義、設計から保守管理までの内容をコンパクトに記述された教科書で、筆者の座右の書です。特に「第7章 例題 (ロケット発射地上支援システム)」の章は、タクトタイムを意識したペトリネット図の解説が際立っています。例題2の `Glide_Path` はこれを意識して製作したオリジナルのプログラムです。

インターネットが普及して以来、プログラミング、画像処理など様々な情報を国内外から無償で入手できるようになり、私も多くの情報を享受して仕事に活用してきました。これらの知識、経験の一部でも今後本分野にかかわる人たちに還元できれば幸いです。

以上

## 参考文献表(順不同)

- [1] 日向俊二、速効解決!逆引きハンドブック Visual C++ (テクニカル Tips シリーズ) ソシム社 2007
- [2] 藤野喜一、花田収悦 ソフトウェア生産技術、電子情報通信学会 1985
- [3] 北山洋幸 Win32/64 API システムプログラミング—32/64 ビットの共存 カットシステム 2011
- [4] 香取巻男、立田純一 すぐわかる!組込み技術教科書 エンベデッド技術と開発のポイントを基礎から理解 (組込み技術基礎シリーズ) CQ 出版社 2008
- [5] 画像処理の解説サイト イメージングソリューション <https://imaging-solution.net/>
- [6] ハーバート・シルト STL 標準講座—標準テンプレートライブラリを利用した C++プログラミング 多摩ソフトウェア 1999

本文内で参照したサイトは省略しています。

## 例題について

本稿オリジナルの例題は、EXE や ZIP ファイルのアップロードに制約のため、プロジェクト(ソースコードを含む)の公開は保留中です。