第3版

画像処理ユニット　NVP-Ax430シリーズ

ソフトウェア開発キット

NVP-Ax430SDK

ユーザーズマニュアル

maxell
マクセルフロンティア株式会社

はじめに

このたびは、NVP−Ax430シリーズの画像処理ユニット(NVP−Ax430CL/430ACL/435CL/435FCL)および ソフトウェア開発キット NVP−Ax430SDKをお買い上げいただきまして、誠にありがとうございます。

本マニュアルはNVP−Ax430シリーズの画像処理ユニットを使用したアプリケーション作成のための基本ソフトウェアであるNVP−Ax430SDKについての仕様、操作方法、使用上の注意事項などについて詳細を記載しています。また、別紙「コマンドリファレンス」は、NVP−Ax430SDKで提供するAPIについての仕様、注意事項などについて詳細を記載しています。正しくご使用いただくために、各章の熟読をお勧めいたします。
別紙「ハードウェアマニュアル」にNVP−Ax430シリーズのハードウェア仕様、設置方法、注意事項などを記載しています。取扱い、設置方法を誤ると重大事故の可能性がありますので、ご一読をお願い致します。

ご注意

- 本アプリケーションの操作を行う前に、本マニュアルの記載内容をよく読み、書かれている指示や注意を十分理解してください。誤った操作によりシステムの故障の原因となる場合がありますので十分ご注意ください。

- お客様の誤った操作に起因する、事故発生や損害につきましては、弊社は責任を負いかねますのでご了承ください。

- 弊社提供のハードウェアおよびソフトウェアを無断で改造しないでください。この場合の品質および安全につきましては、弊社は責任を負いかねますのでご了承ください。

- NVP-Ax430シリーズに搭載されているLinuxは、32ビットのLinuxシステムであり、管理できる日付と時刻の範囲は、UTC（協定世界時）時刻で1970年1月1日0時00分00秒～2038年1月19日3時14分7秒までとなります。それ以外の日付と時刻の管理が必要なアプリケーションには対応できませんのでご了承ください。

- NVP-Ax430シリーズに搭載されているLinuxは、リアルタイムOSではございません。お客様のアプリケーションで使用されるユースケースにて性能評価を実施頂き、実製品への適用可能性を十分検討頂くようお願い致します。

- 本マニュアルの内容について予告なく変更する場合があります。

略語および略称の説明

略語/略称	英語名	日本語名
NVP	NVP-Ax430CL/ACL , -Ax435CL/FCL	NVP-Ax430CL/ACL , -Ax435CL/FCL の略称
NVP-Linux	Linux installed on the NVP unit	NVPユニットに搭載されているLinux
PC	Personal Computer	パーソナルコンピュータ
IMP	Image Processor	画像処理プロセッサー
IMR	Image Render	描画処理プロセッサー
CIP	Civil Infrastructure Platform	社会インフラシステム向けの高信頼性、セキュリティ、リアルタイム性の向上した超長期サポートカーネル
SPI	Serial Peripheral Interface	シリアルフラッシュメモリのインターフェース
OHCI	Open Host Controller Interface	Microsoft社などが規定したUSB 1.0/1.1に対応したコントローラ
EHCI	Enhanced Host Controller Interface	USB接続の周辺機器とコンピュータ本体の通信を制御するUSBコントローラの種類の一つで、USB 2.0規格に対応したもの
xHCI	eXtensible Host Controller Interface	USB 3.0で標準的に用いられるEHCIの上位互換のコントローラ
UTC	Coordinated Universal Time	協定世界時。日本時間から9時間引いた時刻。
JST	Japan Standard Time	日本標準時
NAS	Network Attached Storage	ネットワーク接続型のストレージ
CIFS	Common Internet File System	Windowsのファイル共有の仕組みであるSMB（the Server Message Block）をWindows以外でも利用できるようにしたもの

すべての商標および登録商標は、それぞれの所有者に帰属します。

・Windows® の正式名称はMicrosoft® Windows® Operating Systemです。
・Microsoft、Windows、Visual Studioは、米国Microsoft Corporationの米国及びその他の国における商標または登録商標です。
・ARM、Cortexは、米国およびその他の国におけるARM Ltd.の登録商標または商標です。
・Linuxは、Linus Torvaldsの米国及びその他の国における登録商標あるいは商標です。
・本書では、商標または登録商標のマーク(® 、™)を省略して記載する場合がございます。

目次

1. 概要

1.1 NVP-Ax430SDKの概要

本SDKは画像処理ボードNVP-Ax430シリーズのソフトウェア開発キットです。NVP-Ax430シリーズのアプリケーションソフトウェア開発に必要なライブラリ、ドキュメントが含まれています。

1.1.1 特徴

- リモートコマンドにより、WindowsからNVP-Ax430/Ax435を制御するアプリケーション構築が可能です。
- NVP-Ax430/Ax435のLinux環境で動作するアプリケーション構築が可能です。
- Windows環境のみでLinux環境で動作するアプリケーション開発が可能です。

1.1.2 開発環境

本SDKを使用しWindows上のVisual C/C++ 環境でのアプリケーション開発とNVP-Linuxで動作するアプリケーションやインテリジェント処理モジュールなどのリモートモジュール開発を行うことができます。

(1) リモートコマンドを利用したアプリケーションの開発環境

リモートコマンドを利用したアプリケーションの開発には表1-1に示すWindows上のVisual C/C++ 環境が必要です。

表1-1 リモートコマンドを利用したアプリケーションの実行環境及び開発環境

項目		内容
ハードウェア	パソコン	Windows10が動作するOADG準拠のパソコン メモリ : 1Gバイト以上推奨 1000Base-T/100base-Tx ×1ポート以上
	NVP-Ax430CL NVP-Ax430ACL NVP-Ax435CL NVP-Ax435FCL	最大16ユニット
OS		Windows10 日本語 32bit/64bit (64bitの場合はWOW64で動作します)
コンパイラ		Microsoft Visual Studio 2013 Professional Update5 + MBCS(マルチバイト文字対応)MFCライブラリ C/C++ のみサポート または、それ以降のバージョン

(2) NVP-Linuxアプリケーションの開発環境

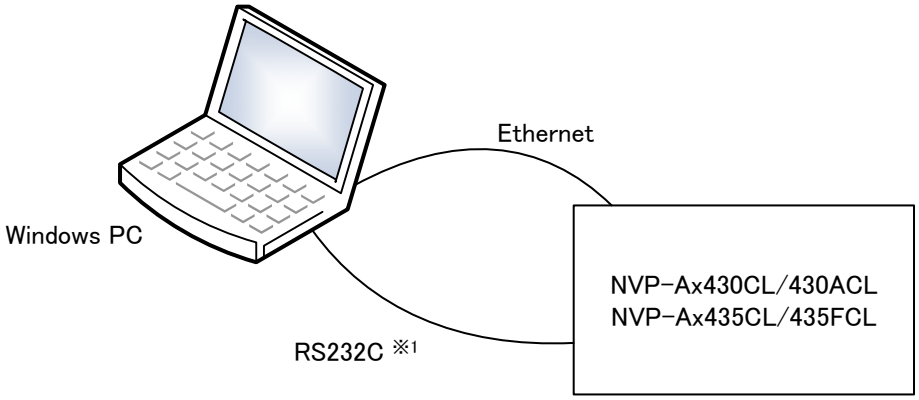
NVP-Linuxアプリケーション、リモートモジュール開発には本SDK以外に表1-2に示す環境が必要です。Windows環境のみでLinux環境で動作するアプリケーション開発が可能です。

表1-2 NVP-Linuxアプリケーションの開発環境

項目		内容
ツールの動作OS		Windows10
コンパイラ	gcc	gcc クロスコンパイラ・ツールチェイン : gcc version 5.5(gcc-linaro-5.5.0-2017.10)
ビルド支援ツール	make	ビルド自動化ツール
	CMake	makeファイル作成ツール
統合開発環境 (リモートデバッガ含む)		Eclipse version 4.5 (推奨)
ターミナルソフトウェア シリアル/SSHコンソール		TeraTerm

(3) アプリケーション開発環境のハード構成

図1-1にNVP-Ax430SDKの推奨ハード構成を示します。



※1 必須ではありません

図1-1 NVP-Ax430SDK推奨ハード構成

1.2 画像処理ユニットNVP-Ax430の概要

1.2.1 ハードウェア構成

NVP-Ax430CL/430ACL 及び NVP-Ax435CL/Ax435FCLのハードウェア構成一覧を表1-3に、NVP-Ax430CLのハードウェア構成を図1-2、NVP-Ax430ACLのハードウェア構成を図1-3、NVP-Ax435CLのハードウェア構成を図1-4、NVP-Ax435FCLのハードウェア構成を図1-5に示します。

表1-3 ハードウェア構成一覧

項目	NVP-Ax430CL	NVP-Ax430ACL	NVP-Ax435CL	NVP-Ax435FCL
CPU	ARM CortexA15 1.4GHz Quad Core			
画像認識プロセッサ	IMP , IMR			
システム/画像メモリ	システム・画像メモリ共有(1GB) + システムメモリ(1GB)			
ブート用メモリ	SPIフラッシュメモリ			
映像入力部 CameraLink	Base Configration 2CH		Base Configration 4CH	Base/Medium/Full Configuration 1CH 又は Base Configuration 2CH
映像表示部	HDMI 1CH			
アイソレーションI/O	入力8点、出力8点	入力16点、出力16点		
イーサネット	1000Base-T/100Base-TX/10Base-T 1CH			
SCI	RS232C 2CH + カメラリンクデータ通信用1CH			
RTC	搭載	搭載		
バッテリバックアップ	無し	有り		
USB3.0 Host	2CH			
SDカードスロット	1CH			
エンコーダー	搭載 2CH (IODIと共用)	搭載 2CH		
WDT	搭載			

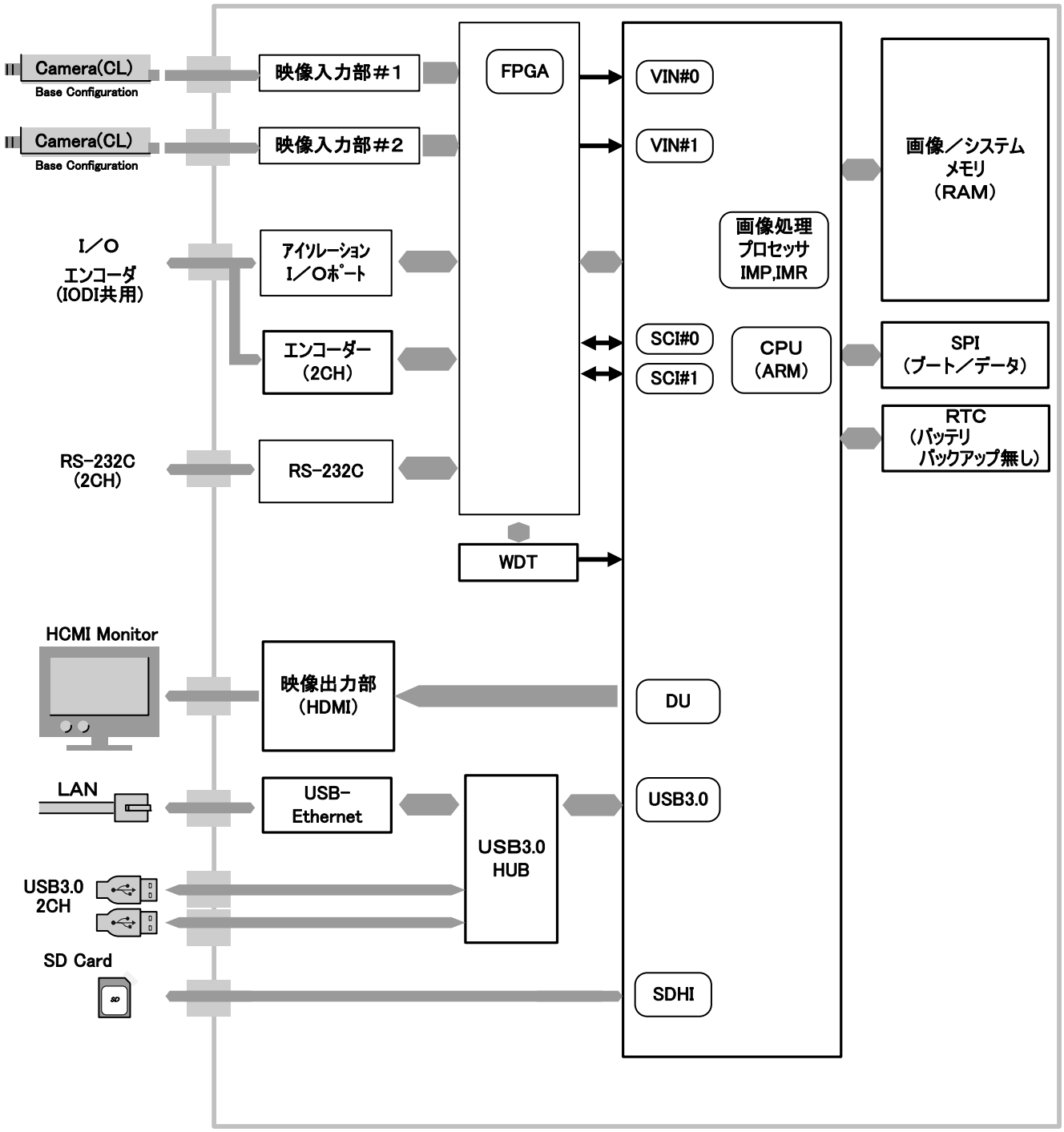


図1-2 NVP-Ax430CL/ハード構成

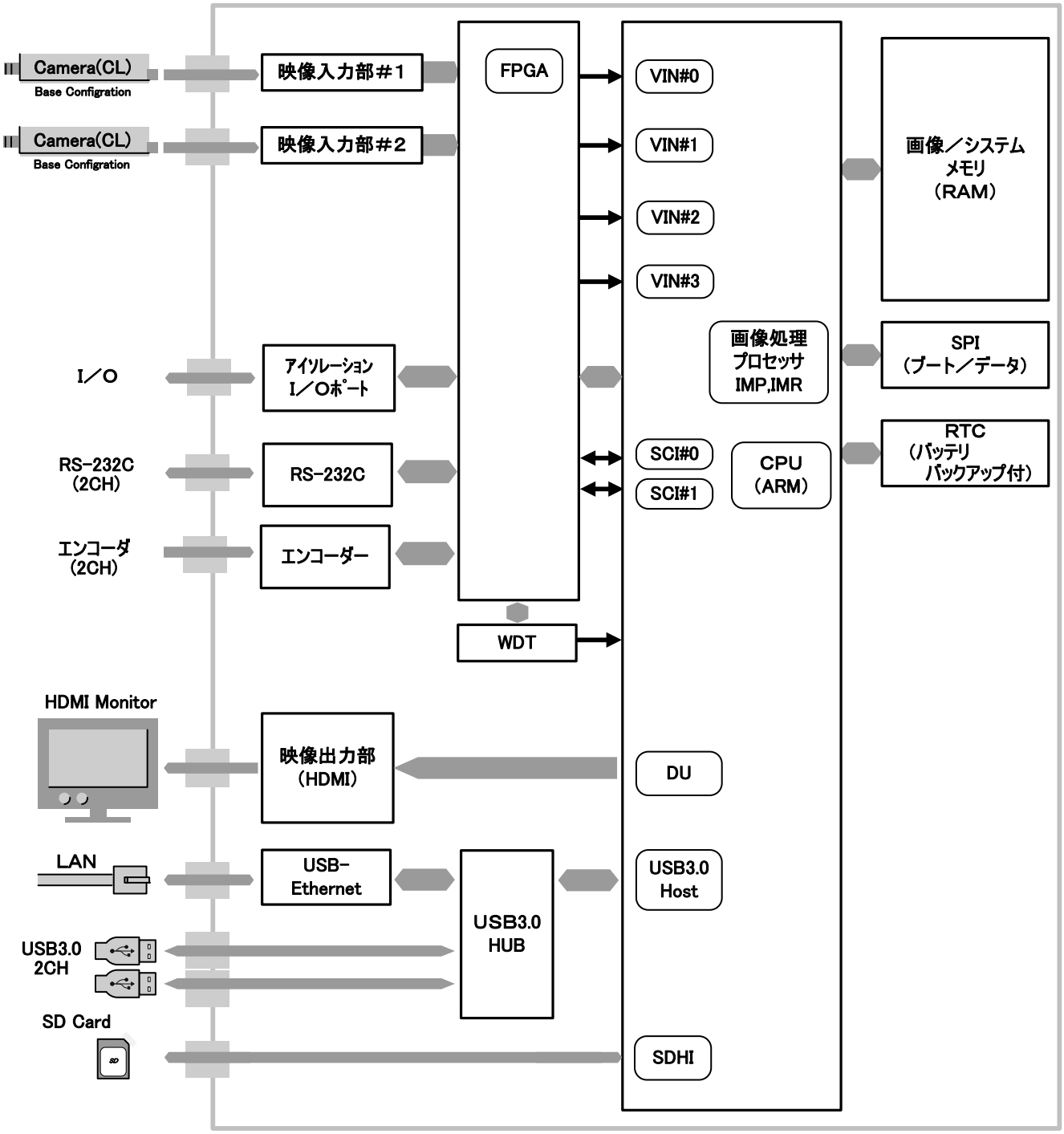


図1-3 NVP-Ax430ACLハード構成

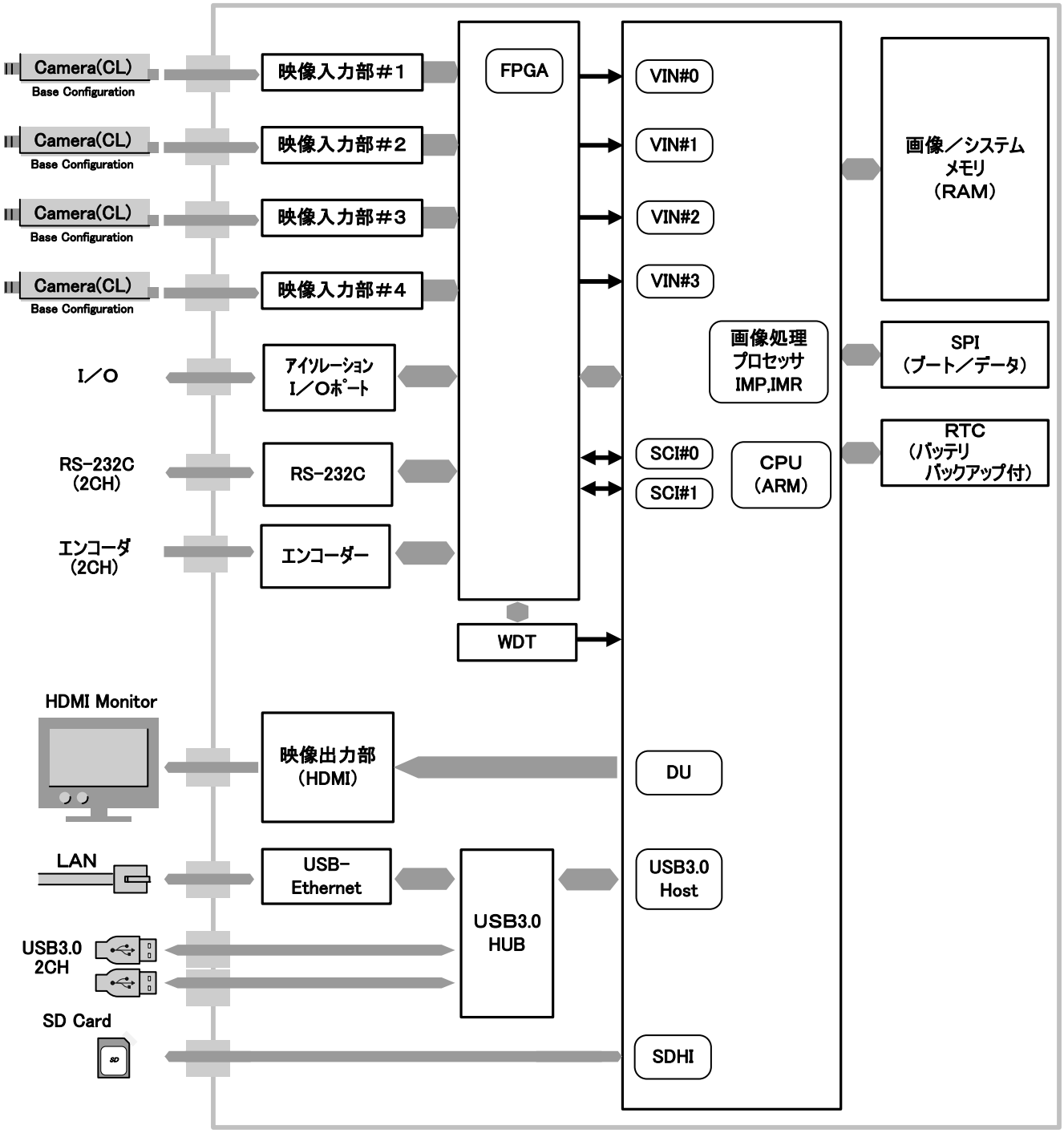
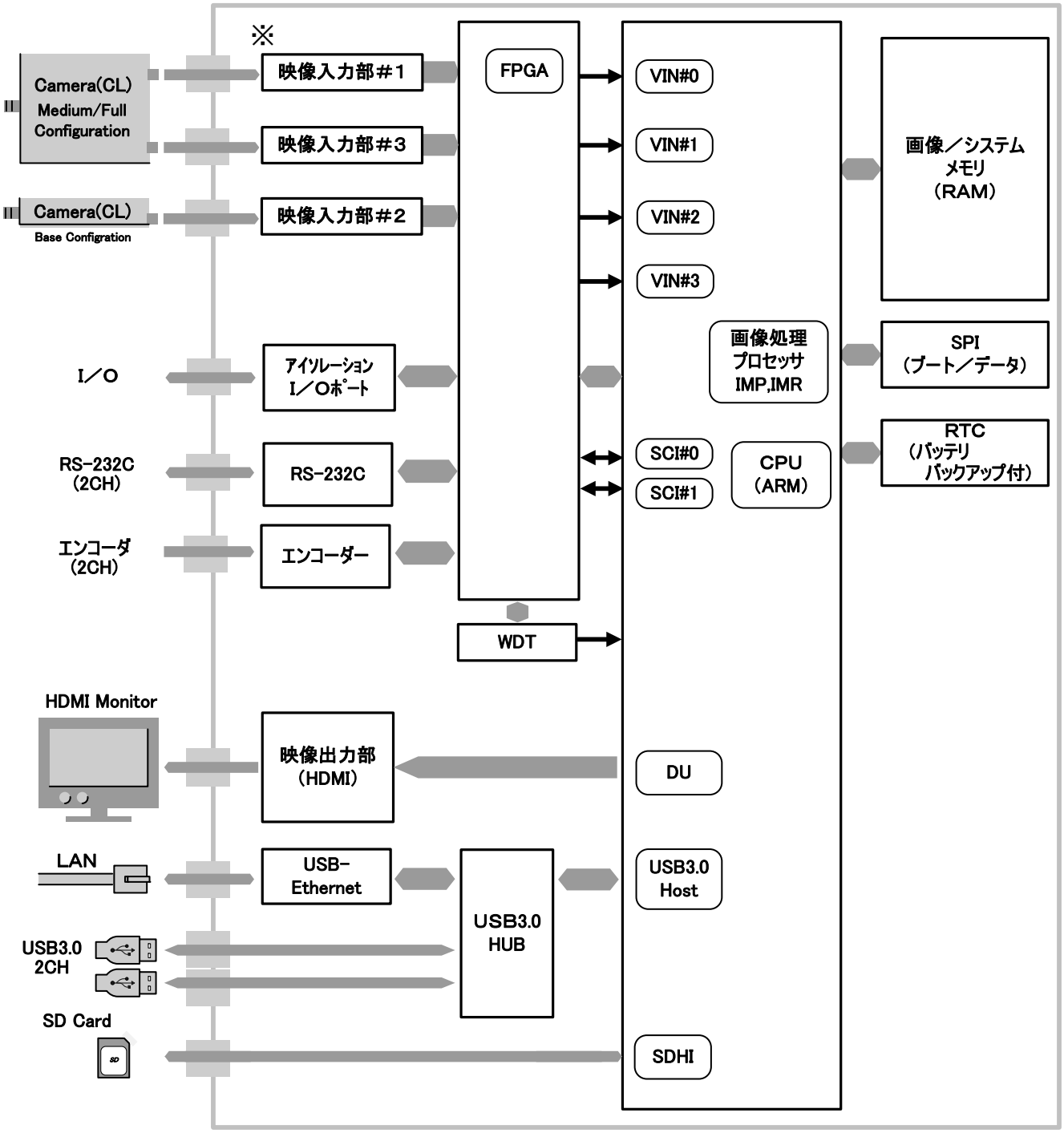


図1-4 NVP-Ax435CLハード構成



※映像入力部は、Medium/Full Configuration 1CH 又は
Base Configuration 2CHの接続が可能です。

図1-5 NVP-Ax435FCLハード構成

2. パッケージ内容

2.1 インストールCD

インストールCDの構成と主な内容を図2-1に示します。

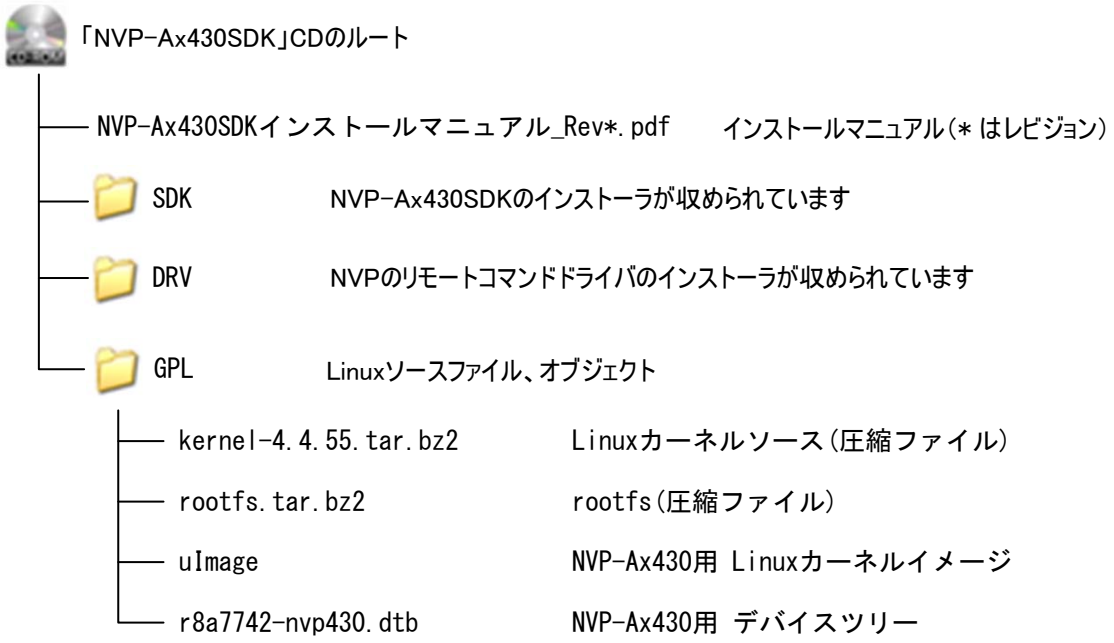


図2-1 主なCDの内容

2.2 パッケージ概要

本SDKは、NVPアプリケーション開発で使用する以下の内容を提供します。

・リモートコマンドライブラリ(Lib、DLL、ヘッダファイル)
・NVP-Linuxアプリケーション用コマンドライブラリ(共有ライブラリシンボリックリンクファイル、ヘッダファイル)
・NVP-Linuxアプリケーション用共有ライブラリ、ドライバ
・NVP-Linuxリモートコマンドデーモン

(1) リモートコマンドライブラリ

本ライブラリは、Microsoft Visual C/C++用のライブラリです。このライブラリは、LibファイルとDLL(Dynamic Link Library)で提供され、Libファイルをアプリケーションとリンクすることでアプリケーションを構築できます。また、リモートコマンドAPIは、シングルユニット構成やマルチユニット構成でシステムに対応するため、Windows側でユニットを識別するためのユニット識別子(デバイスID)無しのAPIとデバイスID付きのAPIを用意しています。

(2) NVP-Linuxアプリケーション用コマンドライブラリ

本ライブラリは、NVP-Linuxアプリケーションの構築用のGCCで使用できる共有ライブラリです。このライブラリは、共有ライブラリのシンボリックリンクファイル(.so)で提供され、シンボリックリンクファイルをアプリケーションとリンクすることでアプリケーションを構築できます。

(3) NVP-Linux共有ライブラリ、ドライバ

共有ライブラリ本体と画像処理プロセッサ等を制御するためのカーネルドライバです。

(4) NVP-Linuxリモートコマンドデーモン

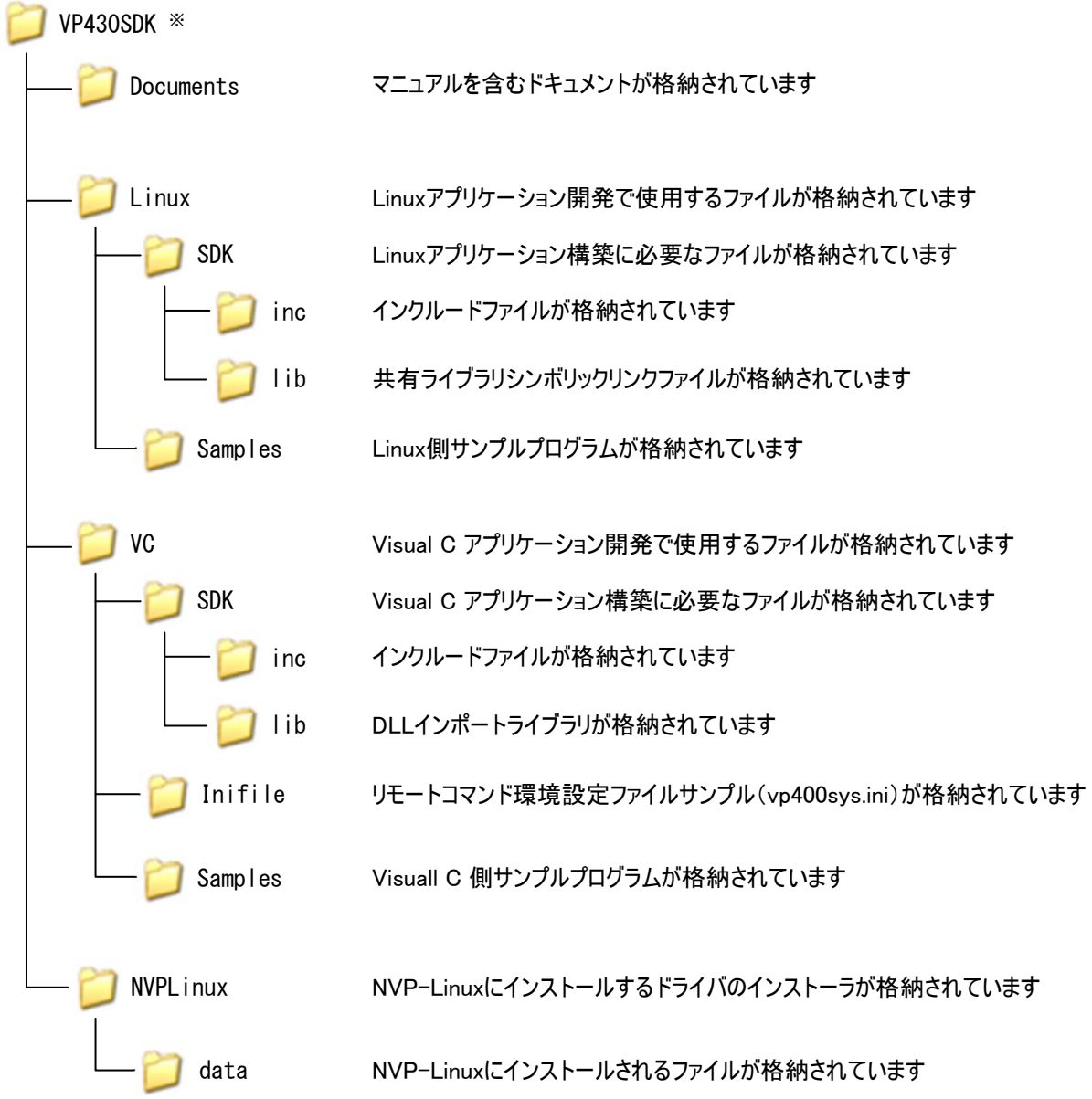
Windows側からのリモートコマンドのみでNVPを制御するためのサービスアプリケーションです。

2.3 インストール内容

本SDKのインストールで以下の構成でファイルがインストールされます。

(1) フォルダ構成

SDKインストールドライブ



※インストールフォルダが、「VP430SDK」の場合

図2-2 SDKフォルダ構成

(2) ファイル内容

本SDKで提供する主なファイルを以下に示します。

表2-1 SDKで提供する主なファイル

項目	ファイル名	内容
リモートコマンド ライブラリ	vp400mul.lib	Visual C 用 画像処理ライブラリインポートライブラリファイル （デバイスID付き版）
	vp400sgl.lib	Visual C 用 画像処理ライブラリインポートライブラリファイル （デバイスID無し版）
	ipxdef.h	Visual C 用 画像処理ライブラリ ヘッダファイル
	ipxsys.h	Visual C 用 画像処理ライブラリ ヘッダファイル
	ipxprot.h	Visual C 用 画像処理ライブラリ ヘッダファイル
GCC用 コマンドライブラリ (NVP-Linuxアプリケーション構築用)	libipxcmds.so	画像処理コマンド 共有ライブラリシンボリックリンクファイル
	ipxdef.h	GCC用 画像処理ライブラリ ヘッダファイル
	ipxsys.h	GCC用 画像処理ライブラリ ヘッダファイル
	ipxprot.h	GCC用 画像処理ライブラリ ヘッダファイル
DLL	vp400mul.dll	リモートコマンド制御DLLファイル （デバイスID付き版）
	vp400sgl.dll	リモートコマンド制御DLLファイル （デバイスID無し版）
	vp400drv.dll	リモートコマンド制御DLLファイル
	vp400tcp.dll	イーサネット通信コントロールDLLファイル
NVP-Linux 共有ライブラリ、 ドライバ	ipxmem.ko	画像処理プロセッサ用メモリ管理ドライバ
	ipxdrv.ko	画像処理プロセッサ用制御ドライバ
	libipxcmds.so.*.*.*	画像処理コマンド共有ライブラリ(*.*.* はバージョン)
リモートコマンド デーモン	ipxcmdd	NVP-Linux リモートコマンドデーモン

3. NVP-Linuxシステムと操作方法

3.1 Linuxベースシステム

NVPのLinuxシステムを、表3-1に示します。

表3-1 NVP-Linuxベースシステム

項目	内容	備考
Linuxカーネル	Kernel 4.4.55cip3	
リファレンス・ビルドシステム	Poky 2.0.1 (Yocto Project)	

3.2 Linuxハードウェアサポート

NVPのLinuxがサポートするハードウェアを表3-2に示します。

表3-2 Linuxサポートハードウェアー覧

項目		内容
プロセッサアーキテクチャ		ARM v7-A (ARM Cortex-A15)
インタフェース	USB	USB1.0/1.1(OHCI)、USB2.0(EHCI)、USB3.0(xHCI)
入出力	シリアルポート	RS232C(COM1,COM2) / CameraLink(COM3) 対応ビットレート(bps): 1200,2400,4800,9600,19200,38400,57600,115200
	ディスプレイ	HDMI
記憶装置	MTD	eMMC
	SDメモリカード	SD、SDHC、SDXC
	USBメモリ	USB mass-storage device class
その他		RTC(real-time clock)
汎用ネットワーク		Ethernet(eth0) ネットワークプロトコル : IPv4 接続インタフェース:1000BASE-T/100BASE-TX/10BASE-T (オートネゴシエーション対応、固定接続には非対応)

3.3 eMMCの構成

NVPのブートローダは、NVP搭載のeMMCの第1パーテーションのVFATフォーマット領域からデバイスツリーとカーネルイメージをRAMに展開しLinuxをブートします。

eMMCは、2つのパーテーションに分割し、第1パーテーションは、VFATでフォーマットされた領域でデバイスツリーとカーネルイメージのファイルが書込まれ、第2パーテーションは、EXT4でフォーマットされた領域で、ルートファイルシステムが書き込まれています。

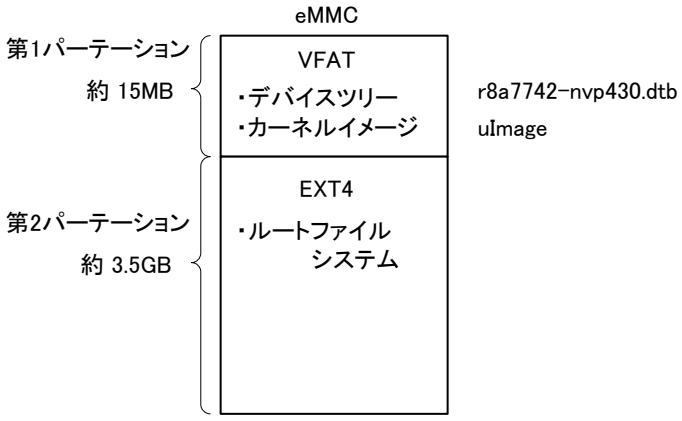


図3-1 eMMCの構成

3.4 ファイルシステム

3.4.1 対応ファイルシステム

NVP Linuxのマウント可能なファイルシステム一覧を表3-3に示します。

表3-3 ファイルシステム一覧

ファイルシステム	説明	備考
ext4	ジャーナル機能付きext2ファイルシステム	
vfat	MS-DOSファイルシステム(FAT16,FAT32)	
ntfs	WindowsNT ファイルシステム（圧縮、暗号化は未サポート）	
cifs	NAS接続やWindowsファイルの共有	

3.4.2 対応メモリデバイス

NVP Linuxの対応するメモリデバイス一覧を表3-4に示します。

表3-4 メモリデバイス一覧

メモリデバイス	説明	対応フォーマット
eMMC	Linuxのルートファイルシステムとしてマウントされています	ext4
SDメモリカード	SD、SDHC、SDXCに対応（CPRM未サポート）	ext4, vfat, ntfs
USBマスストレージ	USBメモリ、USB-HDDに対応	

3.4.3 SD、USBメモリのマウントポイント

SDメモリカードとUSBマスストレージのマウントポイントを表3-5に示します。SDメモリカードは、電源投入後の挿入および抜去はできません。USBマスストレージは、挿入時、表3-5 のマウントポイントを作成し自動的にマウントします。抜去する場合、抜去する前に手動でシェルコマンド「sync」及び「umount」を行うか、またはLinuxのシャットダウン終了後抜去してください。前記の操作を行わない場合、データが破壊される可能性がありますので注意してください。

表3-5 マウントポイント

メモリデバイス	デバイス名	マウントポイント	備考
SDメモリカード	/dev/mmcblk0p1	/media/card1	
USBマスストレージ(A)	/dev/sda1	/media/usb-sda1	USB(A),USB(B)は認識順。 USB(A)は最初に認識したUSBデバイス。 USB(B)は、後に認識したデバイス。
USBマスストレージ(B)	/dev/sdb1	/media/usb-sdb1	

※パーテーションは1パーテーションのみ対応

3.4.4 CIFSのマウント

CIFSでNASやWindows PC と接続可能です。接続先がNASの場合、NAS側の必要な設定を、Windows PCの場合は、接続先のフォルダの共有設定を接続前に行ってください。以下にシェルコマンド「mount」での接続例を示します。

```
# mount -t cifs -o username=xxxx,password=xxxx,domain=xxxx //192.168.0.199/share /media/net
```

※username=xxxx,passwd=xxxx,domain=xxxx は接続する相手先に合わせて指定してください。

※「//192.168.0.199/share」は接続先の共有フォルダ、「/media/net」は、Linux側のマウントポイントです 。

※接続する機器によっては上記以外のオプション指定が必要な場合があります。接続する機器の説明書等をご確認ください。

3.4.5 ファイルシステム使用での制限事項

(1) 全てのメディアやフォーマット対応を保証するものではありません。eMMC、SDカード、USBマスストレージデバイス、その他NVPにマウントされたディスクへの書込みは、Linuxのシャットダウン終了しないままの電源切断やディスクアクセス時の電源切断及びリセットなど、すべての状況においてのデータの確実性を保証するものではありません。また、破損データのリカバリは不可能ですのでご了承ください。

(2) アプリケーションでNVP-Linuxのファイルシステムを使用する場合は、CPUの使用率、スケジュールポリシーなどを考慮し使用してください。Linuxではそれぞれのディスクキャッシュ管理のデーモンが動作しており、CPU使用率が高い場合やリアルタイムスケジューリングでの使用などで、デーモンが動作できなくなると、ファイルシステムが正常に動作しなくなる場合がありますので、CPUアフィニティを活用するなど十分評価、検討した上でご使用ください。

(3) Linuxは、ファイルをクローズしても直ぐにディスク装置にデータが書き込まれず、カーネルのディスクキャッシュにデータが書き込まれます。Linuxの「sync」システムコールを使用しディスクキャッシュからディスク装置にデータを書き込んでください。また、NVPの電源を切断する際はLinuxのシャットダウン処理を実行してください。なお、ディスク装置自体にもバッファキャッシュを持っており、最終的に物理的な記憶部に書き込みが終了するまで時間がかかる場合がありますので電源切断の際は注意してください。

3.5 DIPSWとLED

NVP-Ax430シリーズののDIPSW(SW1)の設定とLED(RUN、ERR)について説明します。それぞれの配置を図3-2、図3-3に示します。他のDIPSW、LEDについては、ハードウェアマニュアルを参照してください。

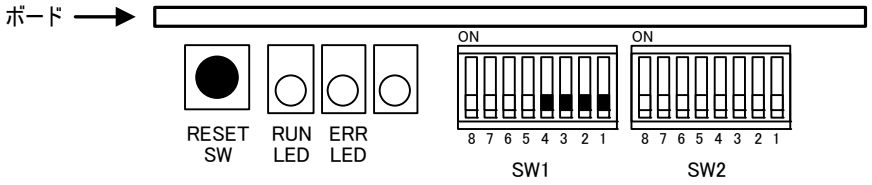


図3-2 NVP-Ax430CL DIPSW、LED配置

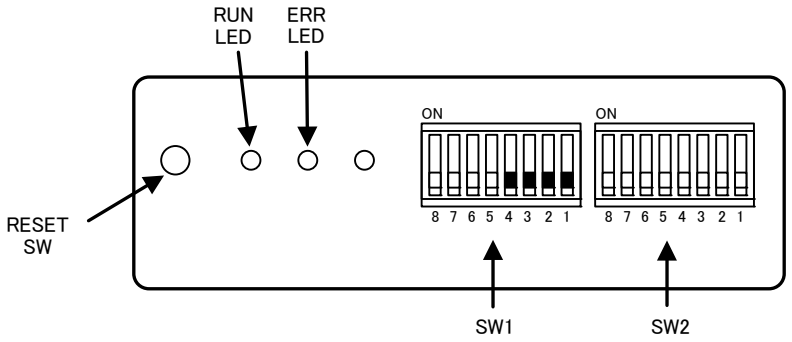


図3-3 NVP-Ax430ACL/Ax435CL/Ax435FCL DIPSW、LED配置

3.5.1 DIPSW(SW1)

NVPのDIPSW:SW1でリモートコマンドデーモン、COM1メッセージ出力、NVPセットアップモードの選択が可能です。設定値を表3-6に示します。なお、DIPSW:SW1の5～8は、全てOFFにして下さい。

表3-6 DIPSW : SW1の設定

SW1					リモートコマンドデーモン	COM1メッセージ出力	備考
8 ～ 5	4	3	2	1			
全て OFF	OFF	OFF	OFF	OFF	起動なし	あり	出荷設定
	OFF	OFF	OFF	ON	起動	あり	
	OFF	OFF	ON	OFF	起動なし	なし	
	OFF	OFF	ON	ON	起動	なし	
	ON	ON	OFF	OFF	NVPセットアップモード NVPセットアップ動作時以外では設定しないでください		

※表3-6以外の設定をしないでください。設定した場合、正しく動作しません。

3.5.2 LED(RUN,ERR)

表3-7にNVPのLED表示内容を示します。

表3-7 NVPのLED表示内容

LED名	点灯色	内容	備考
RUN	青	Linuxが動作中、点灯します。	
ERR	赤	CPUのオーバーヒート検出時、点灯します。	

3.6 ログイン

NVP-Linux上で開発作業を行う場合は、Linuxにログインする必要があります。COM1接続、SSH接続などのシュル環境などでプロセス毎にログインが必要です。表3-8にユーザID、パスワードの初期値を示します。なお、パスワードは初期値として設定していません。設定が必要な場合は、シェルコマンド「passwd」で設定してください。

表3-8 ユーザID、パスワード初期値

項目	設定値	備考
ユーザID	root	ルート権限のユーザ
パスワード	-	パスワードは設定していません

3.7 シャットダウンとリブート

NVPの電源を切断する場合、以下のコマンドでLinuxのシャットダウン処理を行ってください。ルートファイルシステムを含め、NVP-Linuxにマウントされているファイルシステムに書込みを行った場合、シャットダウン処理をしないで電源切断を行った場合、データが壊れるほか、システムが起動できなくなる可能性がありますので必ず実行してください。

```
# shutdown -h now
```

また、Linuxのリブートを行う場合は、以下のコマンドでリブート処理を行ってください。Linuxをシャットダウン後、再起動します。

```
# shutdown -r now
```

3.8 テキストエディタ

NVP-Linuxでテキストファイルを編集する場合「vi」、「nano」を使用することができます。

```
# vi <ファイル名>
```

```
# nano <ファイル名>
```

3.9 ログインシェルとCOM1

3.9.1 COM1でのログインシェルの動作

NVPに電源投入後、出荷設定（DIPSW:SW1が全てOFF）の場合、Linuxシステムが起動を開始し起動メッセージをCOM1に出力し、起動完了後、以下のログインシェルが起動し、ログインメッセージが出力されユーザIDの入力待ちになります。

```
Poky (Yocto Project Reference Distro) 2.0.1 nvp43x /dev/ttySC2

nvp43x login:
```

また、以下のCOM1(RS232C)の通信設定値でログインシェルが起動します。

表3-9 COM1 ログインシェル設定値

項目	値
ビットレート	115200 bps
データ長	8bit
パリティ	無し
ストップビット	1bit
フロー制御	無し

なお、COM1は、NVPユニット本体にD-SUBコネクタが直接出ていないため、使用する場合、ケーブルを作成頂く必要があります。

3.9.2 COM1でのログインシェルを無効にする方法

COM1を他の通信で使用する場合、Linux起動メッセージやログインシェルを無効にする必要があります。以下の設定で無効にすることができます。

① DIPSW:SW1[4:3:2] を[OFF:OFF:ON] に設定（詳細は3.5 DIPSWとLEDを参照）し、COM1へのLinuxのメッセージ出力を無効にする

② /etc/inittab を変更し、ログインシェルの実行を無効にする

テキストエディタを使用して /etc/inittab の 31行目の記述
「SC2:12345:respawn:/sbin/getty -L 115200 ttySC2」をコメントアウトするか削除します

/etc/inittab

```
# /etc/inittab: init(8) configuration.
# $Id: inittab,v 1.91 2002/01/25 13:35:21 miquels Exp $

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
# Normally not reached, but fallthrough in case of emergency.
z6:6:respawn:/sbin/sulogin
SC2:12345:respawn:/sbin/getty -L 115200 ttySC2
# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
```

この記述をコメントアウト又は削除する

※コメントは行先頭に「#」を付ける

3.10 ネットワークの設定

NVP-LinuxはIPv4に対応しています。以下にイーサーネットのIPアドレス等の初期値を示します。

表3-10 イーサーネット 初期値

項目	値
IPアドレス	192.168.0.205
サブネットマスク	255.255.255.0
デフォルトゲートウェイ	192.168.0.1

変更は、「/etc/network/interfaces」ファイルのeth0の設定値を変更してください。

/etc/network/interfaces

```
# /etc/network/interfaces -- configuration file for ifup(8),

# The loopback interface

# Wired or wireless interfaces
auto eth0
#iface eth0 inet dhcp
#iface eth1 inet dhcp
iface eth0 inet static
        address 192.168.0.205
        netmask 255.255.255.0
        gateway 192.168.0.1
```

この記述を変更してください

3.11 日付と時刻

NVP-Linuxの日付と時刻は、タイムゾーンが「Asia/Tokyo」のローカル時刻(JST)で管理されており、シェルコマンド「date」で扱う日時データは日本時間になります。また、NVPには、RTCが搭載されており、Linux起動時にRTCの時刻をLinuxのシステム時刻に反映するよう設定されています。RTCは、UTC時刻で管理されており、システム時刻と独立しているため、設定は、それぞれ行う必要があります。なお、NVP-Linuxは、32ビットのLinuxシステムのため、管理できる範囲は、UTC時刻で1970年1月1日0時00分00秒※1 ～2038年1月19日3時14分7秒までとなります。

(1) 日付と時刻の設定

date コマンドで日付と時刻(JST)を設定し、hwclockコマンドでRTC(UTC)を更新します。以下に設定例を示します。

```
# date -s "2019/2/2 12:45:20"     ←―― システム時刻を設定
# hwclock -w --utc                ←―― RTC を更新
```

(2) 日付と時刻の表示

date コマンドで日付と時刻の表示、hwclockコマンドでRTC時刻の表示を行います。

```
# date                            ←―― システム時刻を表示
# hwclock                         ←―― RTC時刻を表示※2
```

※1)「UTCで1970年1月1日9時0分0秒＋Linuxが起動してからの累計時間」以前の時刻は設定できません

※2) RTC時刻は、システム時刻から 9時間遅れています

3.12 ディスプレイ解像度の設定

NVPのHDMIポートからの映像出力のディスプレイ解像度は、デフォルトで1024×768です。「/boot/config.txt」ファイルに表3-11の設定(記述の変更)をすることで解像度を変更可能です。

表3-11 解像度設定

解像度 (pixel)	周波数 (Hz)	/boot/config.txt への記述内容	備考
640×480	60	video=640x480@60	
1024×768	60	video=1024x768@60	出荷設定
1280×720	60	video=1280x720@60	

※表3-11以外の記述及び項目削除をしないでください。指定した場合正しい動作ができません。

3.13 CPUオーバーヒート検知

NVPは、CPU内部が高温になると、ERRLEDを点灯し、自動的にLinuxがシャットダウンします。なお、NVPの動作温度条件に関してはハードウェアマニュアルを参照してください。

3.14 isolcpus オプション

NVPは、4コアのマルチCPU構成であり、プロセスやスレッドを実行する際は、Linuxのスケジューラーが自動的に4コアのうちどれかをCPUを割り付けます。isolcpuオプションは、スケジューラーが自動的に行うCPU割り当て可能リストから任意のCPUを除外するためのものです。設定は、「/boot/config.txt」ファイルに以下の記述を追加してください。

isolcpus=<除外対象CPU番号>[,<除外対象CPU番号>,<除外対象CPU番号>]

除外対象CPU番号は「0」～「3」を指定(昇順)

[ ]内は複数のCPUを除外指定する場合。「,」で区切ってください。

【例】CPU0、CPU1を除外する場合

isolcpus=0,1

「isolcpus」オプションで除外したCPUをアプリケーションのプロセス、スレッドで割り当てる場合は、アフィニティを設定でCPUを指定することで割り当て可能です。なお、「0」～「3」の全てのCPUを除外対象に指定しないでください。

4. NVP-Ax430SDKとリモートコマンド

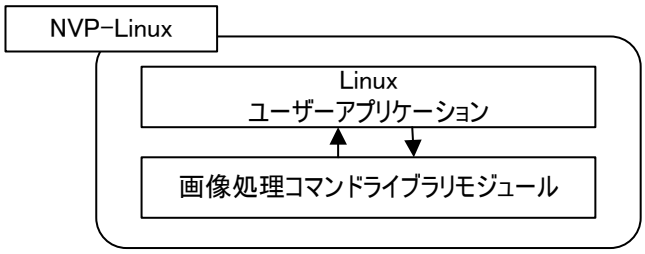
4.1 NVP-Ax430SDKが提供する機能

NVP-Ax430SDKは、NVP-Linux上で実行するユーザーアプリケーションからコールされる画像処理コマンドライブラリとそのLinux上で実行される画像処理コマンドライブラリや画像処理コマンド組合わせたモジュールをWindows上からリモートで実行するリモートモジュールのフレームワークを提供します。

以下に本SDKが実現可能なユーザーアプリケーションの構成を示します。

(1) スタンドアロン構成

NVP-Linux上で動作するWindows(PC)を使用しないスタンドアロンのユーザーアプリケーション



※インテリジェントモジュール以外のタスクモジュールの実行が可能

図4-1 スタンドアロン構成

(2) リモートコマンドのみの構成

ユーザーアプリケーションが、Windows側で実行され、タスクモジュールフレームワークを使用しない。この場合は、NVP-Linux側でリモートコマンドデーモンを動作させ、既定の画像処理コマンドを実行します。

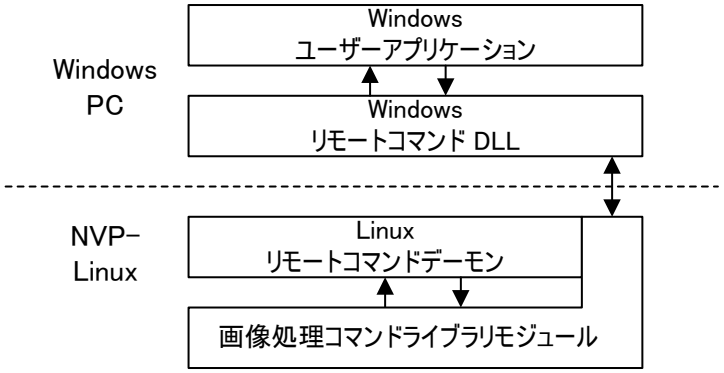


図4-2 リモートコマンドのみ構成

(3) タスクモジュールフレームワークを使用したLinuxユーザーアプリケーションの構成

ユーザーアプリケーションが、Windows側で実行され、タスクモジュールフレームワークを使用する。この場合は、NVP-Linux側のユーザーアプリケーション内でタスクモジュールを動作させます。

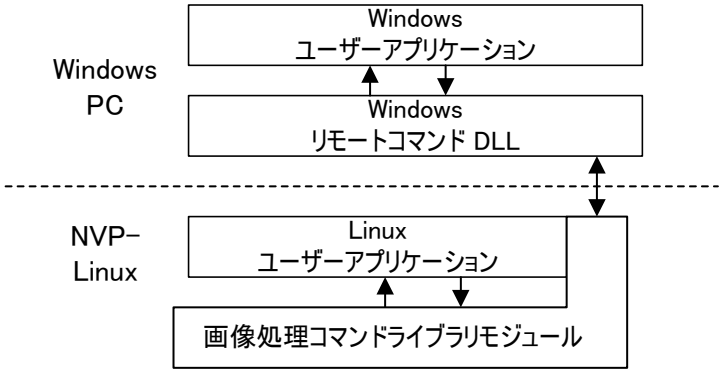


図4-3 タスクモジュールフレームワークを使用した構成

4.2 画像処理コマンドライブラリの構成

画像処理コマンドライブラリは、NVP-Linux上で動作する共有ライブラリである。画像処理コマンドライブラリは、C言語インタフェース部、各画像処理の固有処理部、システムテーブルから構成されます。画像認識コマンドライブラリは、独自のプロセス、スレッドは生成せず、上位の画像認識アプリケーションと同じプロセス、スレッドで実行されます。但し、リモートモジュールフレームワークは、起動すると画像認識アプリケーションのプロセス内に独自のスレッドを生成しそのスレッドで処理を実行します。

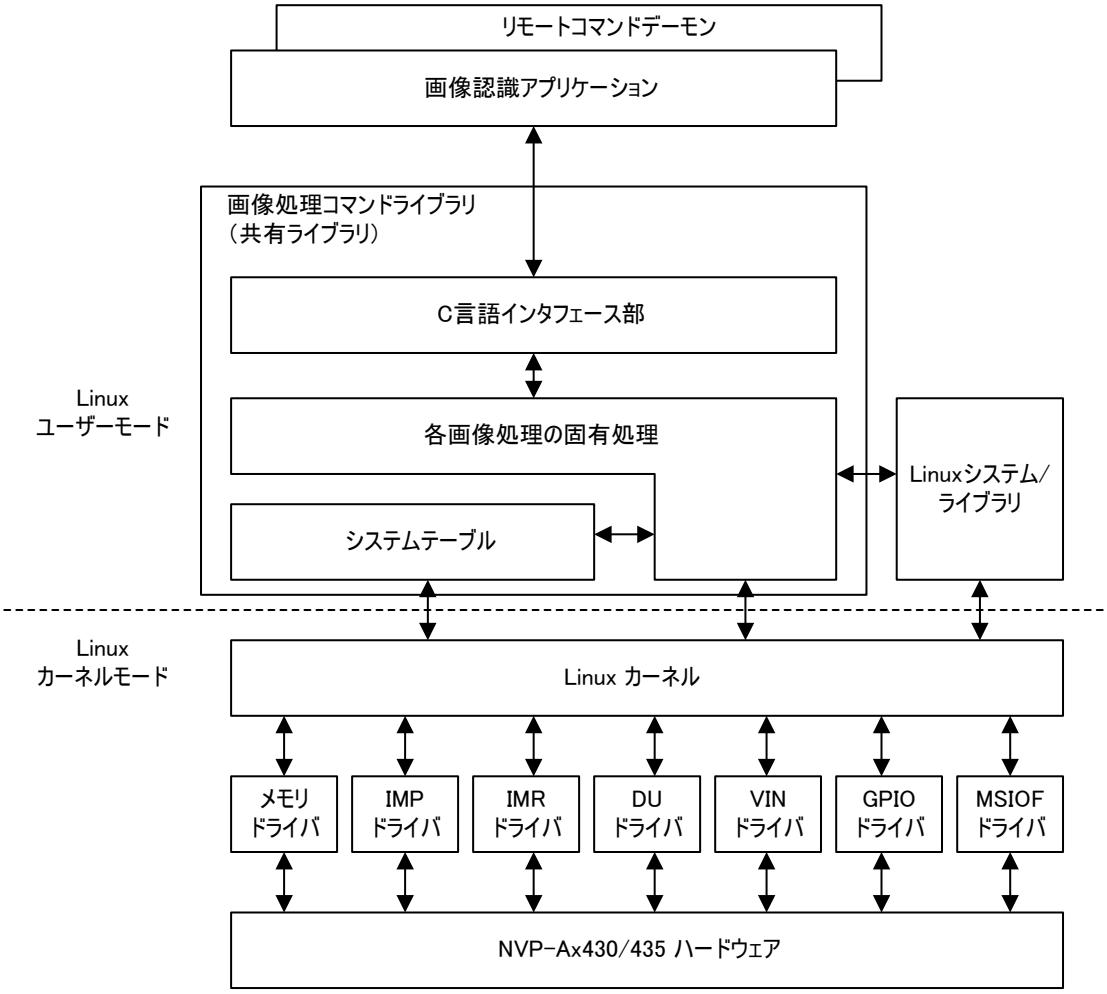


図4-4 画像処理コマンドライブラリの構成

4.3 Windows リモートコマンドの構成

Windows上のリモートコマンドは、図4-5に示すように、リモートコマンドDLL(Dynamic Link Library)、NVP-Linux上で動作するリモートコマンドデーモンやユーザーアプリケーションプロセスで構成されます。Windowsからリモートコマンドを使用する場合は、あらかじめ対応するリモートコマンドデーモンやユーザーアプリケーションを動作させる必要があります。

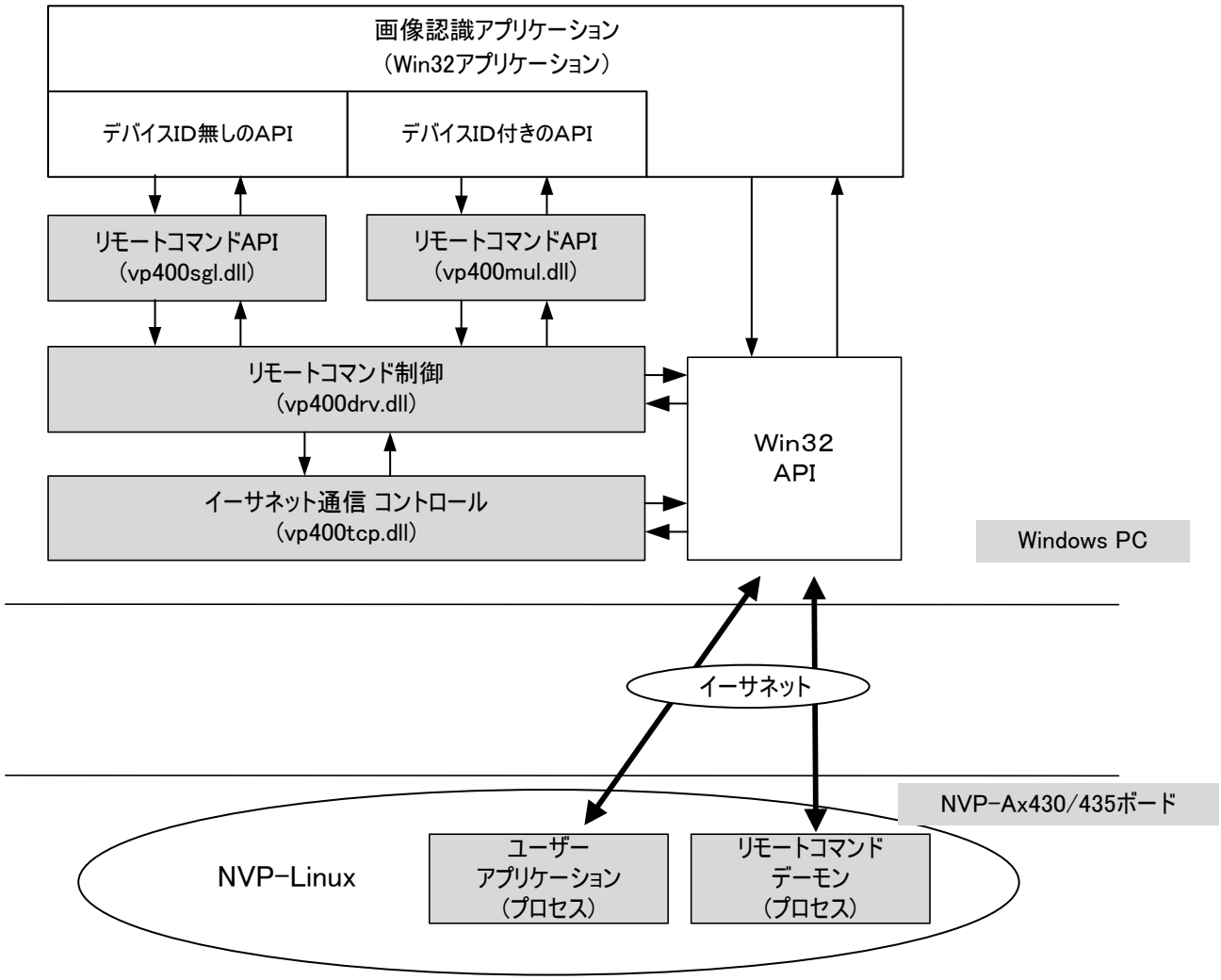


図4-5 リモートコマンドの構成

4.4 リモートコマンドAPI

リモートコマンドAPIは、シングルユニット構成やマルチユニット構成でシステムに対応するため、VisualC/C++で次の2種類のインタフェースを提供します。

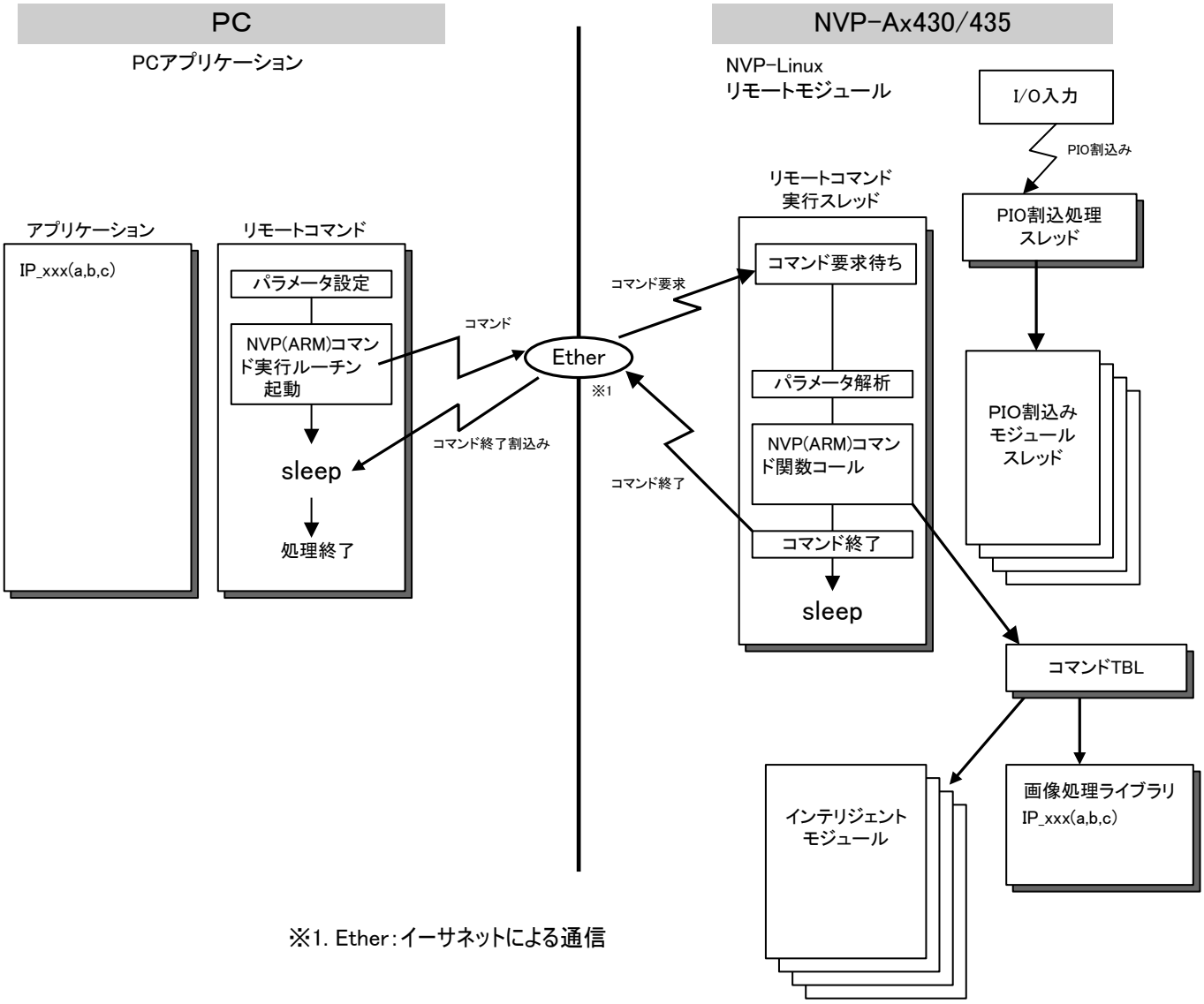
(1)ユニットを識別するためのユニット識別子(デバイスID)無しのAPI

(2)ユニットを識別するためのユニット識別子(デバイスID)付きのAPI

シングルユニット構成のシステムには、ユニットを識別するためのユニット識別子(デバイスID)無しのAPIを使用します。画像処理コマンドAPIのベーシックな形式で、NVPのLinux上ので動作するモジュールは、常にこの形式のAPIで動作する。ユニットを識別するためのユニット識別子(デバイスID)付きのAPIは、マルチユニット構成でNVPを使用する場合にデバイスIDで常に複数のユニットを切換えながら画像処理を行う場合に使用します。ユーザアプリケーションに応じてデバイスID無しのAPI(vp400sgl)と有りのAPI(vp400mul)は、それぞれのライブラリを選択することで使い分けることができます。

4.5 リモートコマンドの概要

NVPの画像処理コマンドライブラリモジュールは、NVP-Linux上で動作するCPU(ARM)と画像処理プロセッサにより処理が行われます。リモートコマンドは図4-6に示すようにPCアプリケーションとNVP-Linuxリモートモジュールとの間でコマンド通信することにより画像処理を実行します。



※1. Ether:イーサネットによる通信

図4-6 リモートコマンドの構成

4.6 Windowsアプリケーションの構築

4.6.1 構築の概要

画像処理コマンドライブラリを使用したWindowsアプリケーションを作成する場合、Microsoft Visual C/C++を使用しアプリケーションのソースファイルをコンパイルした後、ライブラリファイルとリンクします。

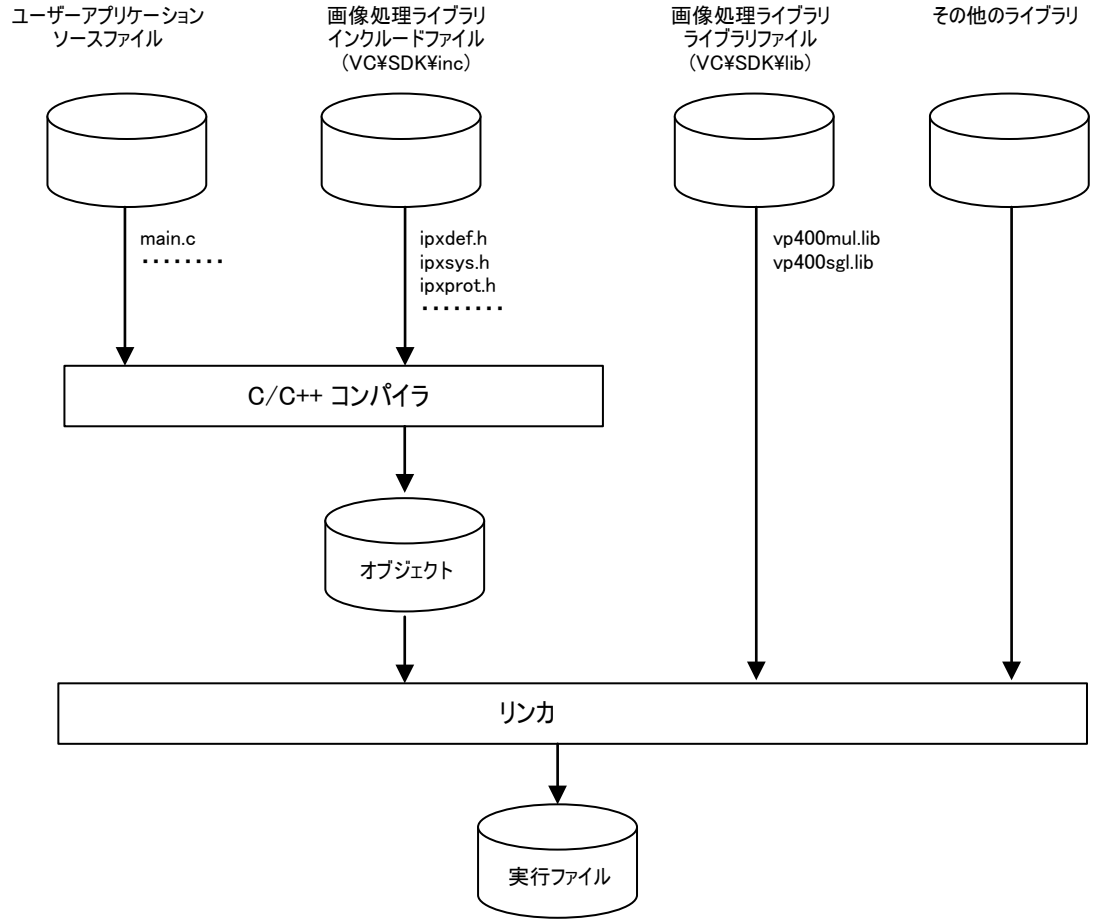


図4-7 Windowsアプリケーション構築の概要

4.6.2 コーディング

(1) インクルードパスとファイル

コンパイルのインクルードパスにSDKインストール先のディレクトリ「VC¥SDK¥inc」を追加し、以下に示すインクルードファイルを以下の順番でソースコードにインクルードしてください。

表4-1 インクルードファイル一覧

ファイル名	内容	記述の順番
ipxdef.h	define,enum定義など	1
ipxsys.h	構造体定義など	2
ipxprot.h	関数プロトタイプなど	3

(2) プリプロセッサの設定

デバイスID付きのリモートコマンドでNVPを使用する場合は、以下の定義をコンパイル時のプリプロセッサに定義してください。

```
MULTI_BOARD_CONFIG
```

デバイスID無しのリモートコマンドでNVPを使用する場合は、定義しないでください。

(3) ライブラリコマンドの開始と終了

リモートコマンドを使用する場合は、Windowsのユーザアプリケーションから必ず下記のNVPのセットアップを行う必要があります。また、NVPのセットアップは、デバイスID無しのリモートコマンドとデバイスID付きのリモートコマンドでそれぞれ異なります。

① デバイスID無しリモートコマンドでのNVPの初期化

デバイスID無しのリモートコマンドでNVPを使用する場合は、NVP起動（StartIP）を実行し初期化します。そしてアプリケーションの終了時にNVPの停止処理（StopIP）を実行してください。

```
#include <stdio.h>
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

void main( )
{
    /* ローカル変数の設定  */
    int  ret,ImgID1,ImgID2,ImgID3;

    /*  プログラム          */

    /* NVPの起動 */
    ret = StartIP( IPBOARD_0, CONNECT_PORT_DEAMON );
    if(ret < 0){
        printf("ボードの起動に失敗しました¥n");
        return;
    }

    /* 画像処理コマンドの初期化 */
    InitIP( );
    ActiveVideoPort( VIDEO_PORT1 );
    SelectCamera( CAMERA_PORT0, BW_CAMERA );
    SetVideoFrame( INTERLACE, VIDEO_FS_512H_480V );
    ImgID1 = AllocImg( IMG_FS_512H_512V );
    ImgID2 = AllocImg( IMG_FS_512H_512V );
    ImgID3 = AllocImg( IMG_FS_512H_512V );
    GetCamera( ImgID1 );
    IP_AddConst( ImgID1, ImgID2, 20 );
    IP_Binarize( ImgID2, ImgID3, 120 );
    DispImg( ImgID3 );

    /* NVPの停止 */
    StopIP( IPBOARD_0 );
}
```

このインクルードファイルは必ずインクルードして下さい

NVPの起動

画像処理コマンド初期化

NVPの停止

図4-8 デバイスID無しリモートコマンドでのコーディング例

② デバイスID付きリモートコマンドでのNVPの初期化

デバイスID付きのリモートコマンドでNVPを使用する場合は、NVPのオープン処理（OpenIPDev）を実行しデバイスIDを取得して下さい。そしてプログラムの終了時にNVPのクローズ処理（CloseIPDev）を実行してください。

```
#include <stdio.h>
#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

void main( )
{
    /* ローカル変数の設定  */
    DEVID   devID;
    int     ImgID1,ImgID2,ImgID3;

    /*  プログラム          */

    /* NVPのオープン */
    devID = OpenIPDev( IPBOARD_0, CONNECT_PORT_DEAMON );
    if( devID == ISPX_NULL ){
        printf("ボードの起動に失敗しました¥n");
        return;
    }

    /* 画像処理コマンドの初期化 */
    InitIP(devID);
    ActiveVideoPort(devID,VIDEO_PORT1 );
    SelectCamera( devID,CAMERA_PORT0, BW_CAMERA );
    SetVideoFrame(devID,INTERLACE,VIDEO_FS_512H_480V);
    ImgID1 = AllocImg(devID,IMG_FS_512H_512V);
    ImgID2 = AllocImg(devID,IMG_FS_512H_512V);
    ImgID3 = AllocImg(devID,IMG_FS_512H_512V);
    GetCamera(devID,ImgID1);
    IP_AddConst(devID,ImgID1,ImgID2,20);
    IP_Binarize(devID,ImgID2,ImgID3,120);
    DispImg(devID,ImgID3);

    /* NVPのクローズ */
    CloseIPDev(devID);
}
```

このインクルードファイルは必ずインクルードして下さい

NVPのオープン

画像処理コマンド初期化

NVPのクローズ

図4-9 デバイスID付きリモートコマンドでのコーディング例

4.6.3 リンク

画像処理リモートコマンドライブラリを使用する場合は、以下に示すライブラリファイル及びその他必要なライブラリをリンクしてください。

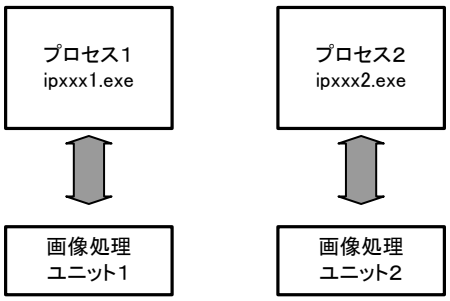
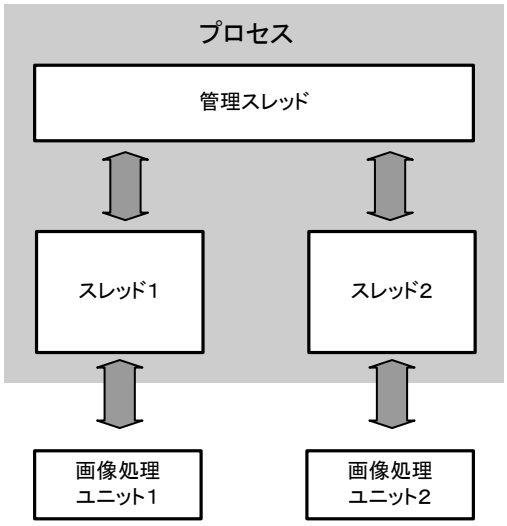
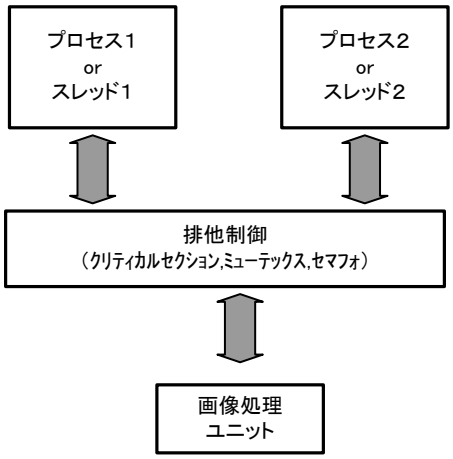
表4-2 リモートコマンドライブラリー覧

ライブラリ名	内容	備考
vp400mul.lib	Visual C 用 画像処理ライブラリインポートライブラリファイル（デバイスID付き版）	
vp400sgl.lib	Visual C 用 画像処理ライブラリインポートライブラリファイル（デバイスID無し版）	

4.7 マルチユニット構成時の注意点

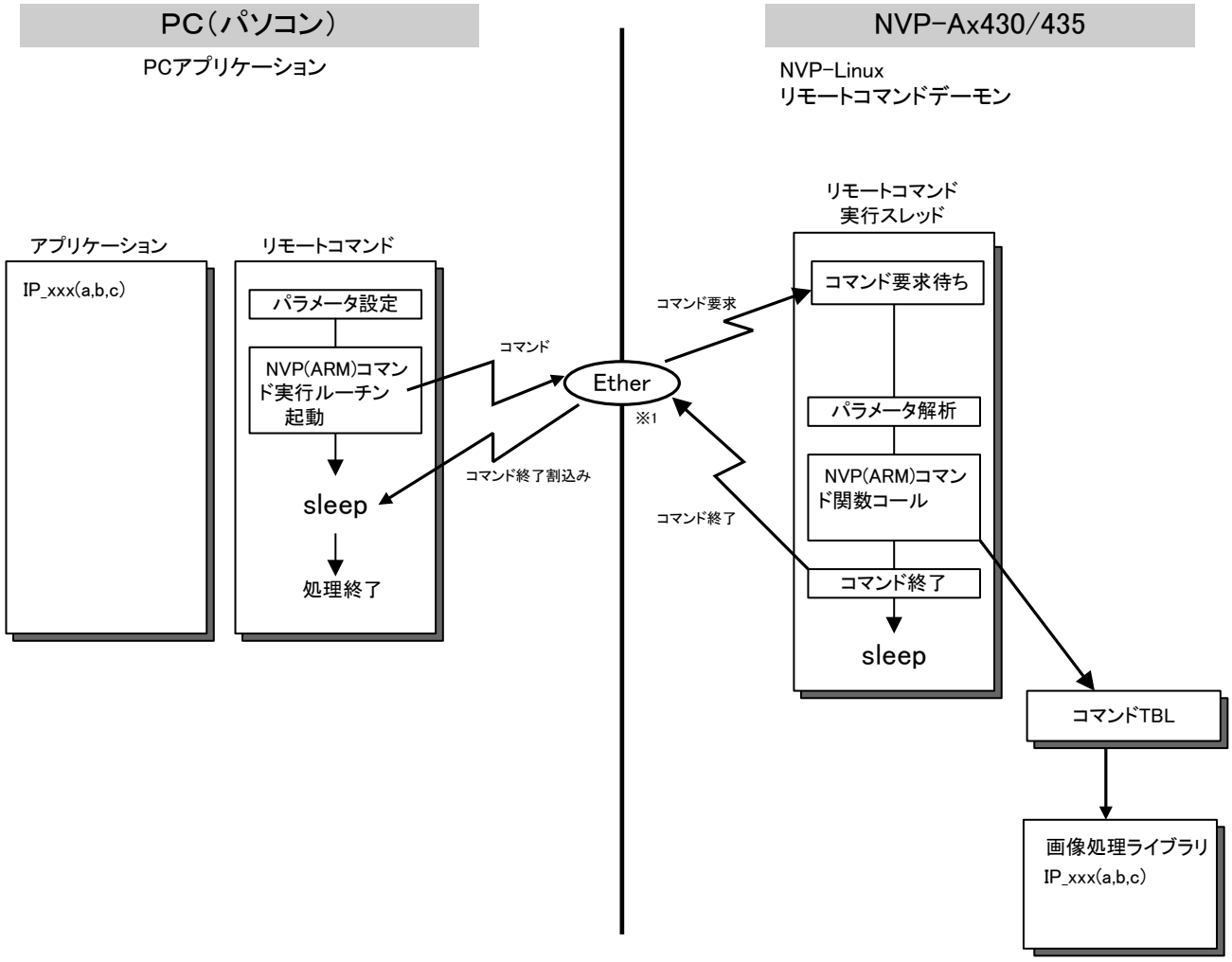
画像処理ユニットは1台のパソコンでNVP-Ax430/435は最大16台までコントロール可能です。Windows上でのマルチプロセス、マルチスレッドでのアプリケーションにも対応しています。1ユニットに2つ以上のプロセスやスレッドからアクセスする場合にはクリティカルセクション、ミューテックス、セマフォ等を使用して2つ以上のプロセスやスレッドから1ユニットに対するコマンド要求がお互いに中断されないように排他制御を行う必要があります。

表4-3 マルチユニットでの対応可能な構成

処理内容	ハード構成	備考
複数のプロセスから各々異なる画像処理ユニットをアクセスする		
複数のスレッドから各々異なる画像処理ユニットをアクセスする		
複数のプロセスやスレッドから1台の画像処理ユニットをアクセスする		排他制御を行う

4.8 リモートコマンドデーモン

PC側のリモートコマンドのみで実行される画像認識ユーザーアプリケーションでは、NVP側の画像処理コマンドを実行するためNVP-Linuxで動作するリモートモジュールを実装したプロセスが必要です。本SDKではリモートコマンドデーモンとして提供します。リモートコマンドデーモンは、NVP-Linuxリモートモジュールでリモートコマンド実行処理のみに特化したLinuxのプロセスです。リモートコマンドデーモンは、NVPのDIPSW:SW1-1をONにすることで実行されます。



※1. Ether：イーサネットによる通信

図4-10 リモートコマンドデーモンの構成

4.9 リモートモジュールフレームワーク

本SDKでは、リモートコマンドとLinuxユーザーアプリケーションのインタフェースやI/O割込み起動からの処理を実現するため、以下のフレームワークを提供します。

(1) インテリジェントモジュール

インテリジェントモジュールとは、画像処理コマンドを複数組み合わせて作成するユーザーオリジナルの画像認識コマンドのことです。WindowsのリモートコマンドからNVP搭載のCPUにより行っているユーザーの画像認識処理を実行するためのフレームワークです。リモートコマンドは、画像処理は画像処理プロセッサとNVP搭載のCPUにより行っているため、ユーザーの画像認識アプリケーションの一部をインテリジェントモジュールとして実装することにより、処理の分散化を実現できます。また、インテリジェントモジュールは画像処理コマンドスレッドの一部として登録し実行されます。

(2) 画像認識タスクモジュール

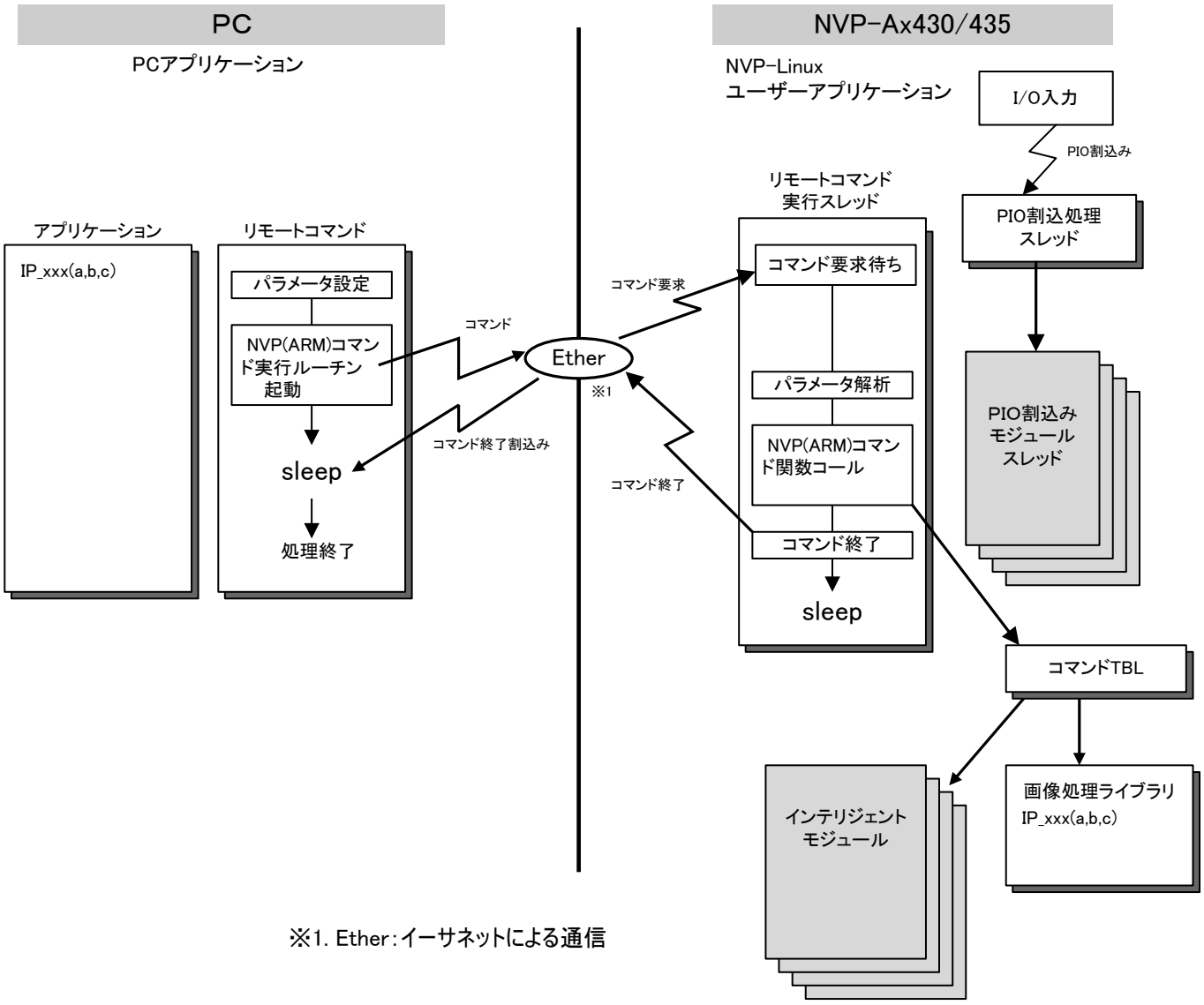
タスクとはアプリケーションを独立して並列に処理可能な単位で分割したプログラムのことです。画像認識タスクモジュールは、タスクとして登録されたユーザーの処理をユーザーアプリケーションプロセスのスレッドとして実行しリモートコマンドと同期して処理を実行するためのフレームワークです。割込み起動モジュールのベースとなるタスクも画像認識タスクモジュールとして登録します。

(3) 割込み起動モジュール

割込み起動モジュールとは、ボード上のアイソレーションパラレルI／O（PIO）からの割込みにより起動することができるモジュールで、ユーザーアプリケーションでPIOからの割込み処理を扱うことができるフレームワークです。画像処理コマンドを複数組み合わせて作成したユーザーオリジナルの画像認識モジュールをPIOの割込み起動モジュールとして登録し、PIOの割込みにより起動することができます。

4.10 リモートモジュールフレームワークの制御

リモートモジュールフレームワークはリモートコマンドモジュールをNVP-Linuxユーザーアプリケーションプロセス内で実行し、リモートコマンド実行、インテリジェントモジュール、割込み起動モジュールの制御を行います。



※1. Ether：イーサネットによる通信

図4-11 リモートモジュールフレームワークの構成

4.11 リモートコマンドでの実行環境設定

リモートコマンドでPC側とNVP側のコマンドデーモンやユーザーアプリケーションを実行する場合、各NVPのネットワークや構成情報等の設定が必要です。

4.11.1 PC側リモートコマンドの環境設定(vp400sys.ini)

PC側のリモートコマンドの実行には、各NVPのネットワークや構成情報等の設定が必要です。「vp400sys.ini」ファイルで管理され、以下に設定項目一覧を示します。「vp400sys.ini」は、アプリケーションのカレントディレクトリに作成下さい。
表4-4の内容の「vp400sys.ini」は、SDKをインストールしたフォルダの「¥VC¥Inifile」フォルダにサンプルとして入っています。

表4-4 vp400sys.ini ファイル設定値一覧

セクション	キー	値	内容	
VP400SYS	User	1	システム情報、変更禁止	
Config	BoardNum	1	NVP接続台数	
	Dll	vp400tcp.dll	システム情報、変更禁止	
Device0	start	1	NVPの検索 有/無	Board0 ボード情報
	ipaddr	192.168.0.205	IPアドレス	
	port	30000	リモートコマンドの ベースポート番号	
DeviceX ※1	指定したNVPの接続台数分のボード情報の設定を記述			

※1 「X」にはボード番号の数字(10進)を入れて下さい

4.11.2 NVP側リモートコマンドアプリケーション環境設定(ipxcmds.ini)

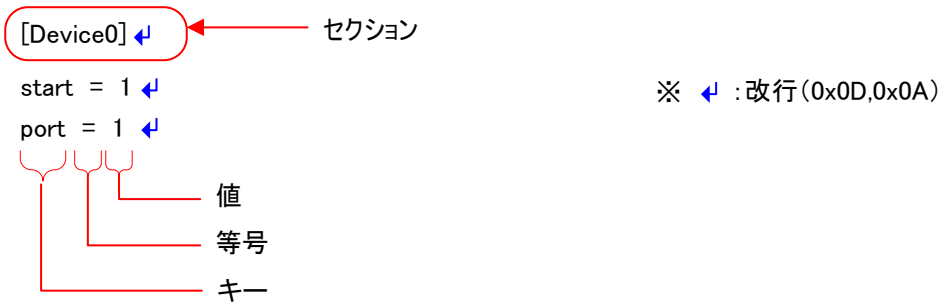
NVP側のリモートコマンドアプリケーションは、リモートコマンドのベースポート設定が必要です。
「/usr/share/nvpd/ipxcmds.ini」ファイルで管理され、以下に設定項目一覧を示します。

表4-5 ipxcmds.ini ファイル設定値一覧

セクション	キー	値	内容
remote	port	30000	リモートコマンドのベースポート番号 (ipxcmds.ini ファイルが無い場合は 「30000」になります)

4.11.3 環境設定ファイル（INIファイル）のフォーマット

INIファイルは、セクションという区画単位で構成され、各セクションは、角かっこ（[ ]）で囲まれた文字列（セクション名）で区切られます。各セクションには任意の数のキーを保存できます。キーは構成の設定値に名前を付けたものです。キーを設定するステートメントはキーの名前、等号(=)、キーの値で構成されます。１行に１つのステートメントを記述します。以下にINIファイルのセクションの例を示します。



キーの名前と等号の間、キーの値と等号の間は、スペースを入れない。セクション名とキー名は大文字と小文字を区別します。

5. Linuxユーザーアプリケーション

5.1 概要

Linux ユーザーアプリケーションは、リモートモジュールフレームワークを使用したアプリケーションとして、また、スタンドアロンのアプリケーションとしてNVP-Linux上でNVPに搭載されている画像処理プロセッサを使用し画像認識処理を実行することが可能です。本SDKを使用したLinux ユーザーアプリケーションの作成方法及び実行方法について説明します。

5.2 Linuxアプリケーションの構築

5.2.1 構築の概要

画像処理コマンドライブラリを使用したアプリケーションを作成する場合、アプリケーションのソースファイルをコンパイルした後、ライブラリファイルとリンクします。

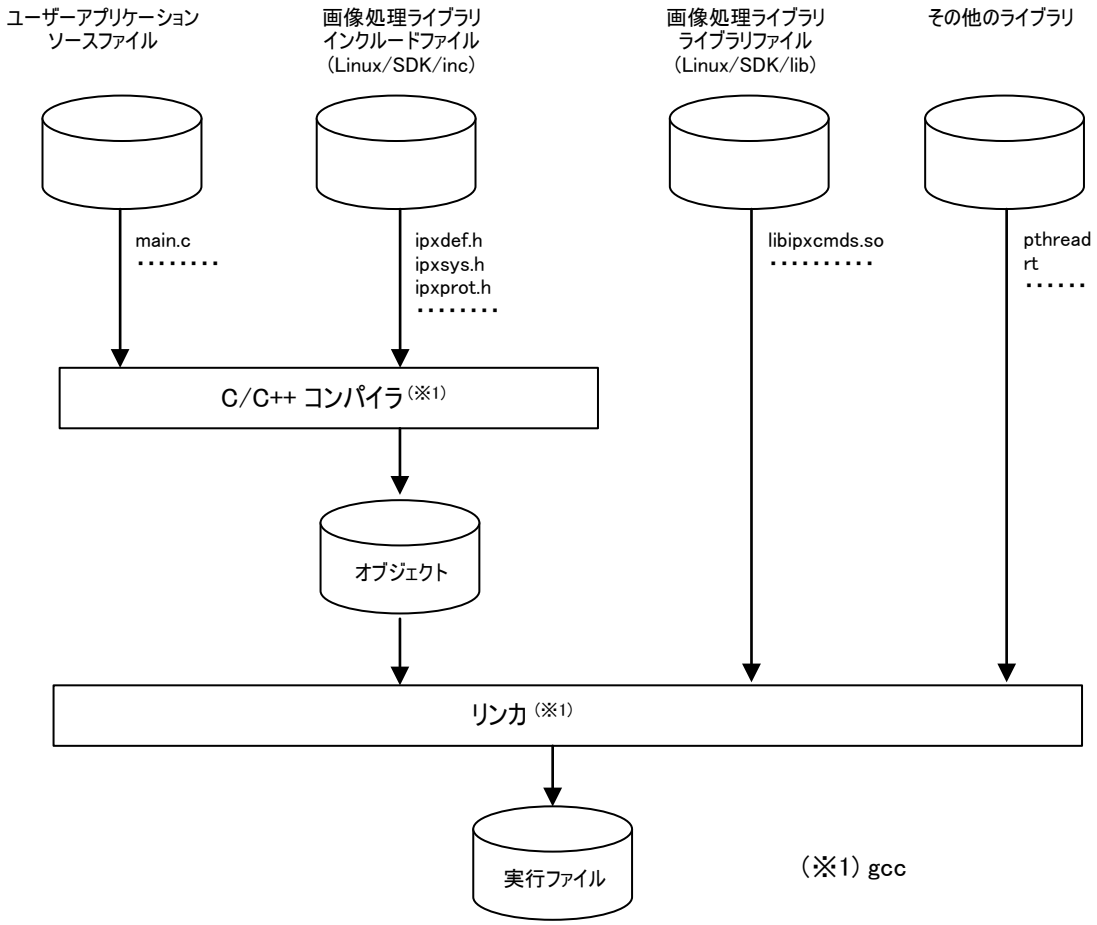


図5-1 Linuxアプリケーション構築の概要

5.2.2 コーディング

(1) インクルードパスとファイル

コンパイルのインクルードパスにSDKインストール先のディレクトリ「Linux/SDK/inc」を追加し、以下に示すインクルードファイルを以下の順番でソースコードにインクルードしてください。

表5-1 インクルードファイル一覧

ファイル名	内容	記述の順番
ipxdef.h	define,enum定義など	1
ipxsys.h	構造体定義など	2
ipxprot.h	関数プロトタイプなど	3

(2) プリプロセッサの設定

PC側でデバイスID付きのリモートコマンドでアプリケーションを作成し、そのソースコードを使用してLinux側でコンパイルする場合は、以下の定義をコンパイル時のプリプロセッサに定義してください。

MULTI_BOARD_CONFIG

上記以外の場合は、定義しないでください。

(3) ライブラリコマンドの開始と終了

画像処理コマンドライブラリを使用する場合、ドライバ動作開始処理 **StartIP()** を実行し、実行に必要なリソースの確保、初期化を行います。そしてアプリケーションの終了時にドライバの停止処理 **StopIP()** を実行します。
また、ライブラリコマンドの初期化処理として、コマンドを使用開始する前に「InitIP」を実行します。

```
#include <stdio.h>

#include "ipxdef.h"
#include "ipxsys.h"
#include "ipxprot.h"

void main( )
{
    /* ローカル変数の設定  */
    int  ret;

    /*  プログラム          */

    /* 画像処理コマンド開始 */
    ret = StartIP( IPBOARD_0, 0 );
    if(ret < 0){
        printf("Error StartIP¥n");
        return;
    }

    /* ライブラリコマンドの初期化 */
    ret = InitIP( );
    if(ret < 0){
        printf("Error InitIP¥n");
        return;
    }

    /* 画像処理コマンド停止 */
    StopIP( IPBOARD_0 );
}
```

コンパイルに必要なヘッダファイルを記述

実行に必要なリソースを確保、オープン

ライブラリコマンド初期化

ここからライブラリコマンドを使用したアプリケーションを記述してください

ドライバクローズ

図5-2 コーディング例

5.2.3 リンク

画像処理コマンドライブラリを使用する場合は、以下に示すライブラリファイル及びその他必要なライブラリをリンクしてください。

表5-2 画像処理ライブラリー覧

ライブラリ名	内容	備考
libipxcmds.so	画像処理コマンドライブラリ(共有ライブラリ)	本SDKライブラリ
pthread	POSIXスレッドライブラリ	gcc 付属
rt	POSIXリアルタイムライブラリ	gcc 付属

5.3 マルチIMPでの動作方法

NVP-Ax430シリーズは、画像認識プロセッサ(IMP)を4個搭載しており、個々に実行することが可能です。NVP-Ax430シリーズでは、1プロセス内でスレッド毎に使用するIMPを指定し管理することで複数のIMPを制御する仕組みになっています。複数の画像認識プロセッサを複数スレッドで同時に使用する動作モードをマルチプロセッサモード、1つの画像認識プロセッサを1スレッドのみで使用する場合の動作モードをシングルプロセッサモードを呼びます。

マルチプロセッサモードを使用する場合、**OpenIMP()**コマンドでスレッド毎に複数のIMP動作に対するリソースを確保、初期化する必要があります。複数の画像認識プロセッサを使用する場合、**OpenIMP()**を発行することで、発行したスレッドに対しIMP管理テーブルを割り当て、複数のIMPを実行する準備をします。**OpenIMP()**コマンドで割り当てたスレッドに対するデフォルトのアクティブなIMP番号は、IMP#0になっています。アクティブなIMPを変更する場合、**ActiveIMP()**コマンドにより設定します。また、割り当てたIMP管理テーブルは、スレッド削除前に**CloseIMP()**コマンドで解放して下さい。

なお、詳細については、コマンドリファレンス 2.1.9 ～ 2.1.11 を参照してください。

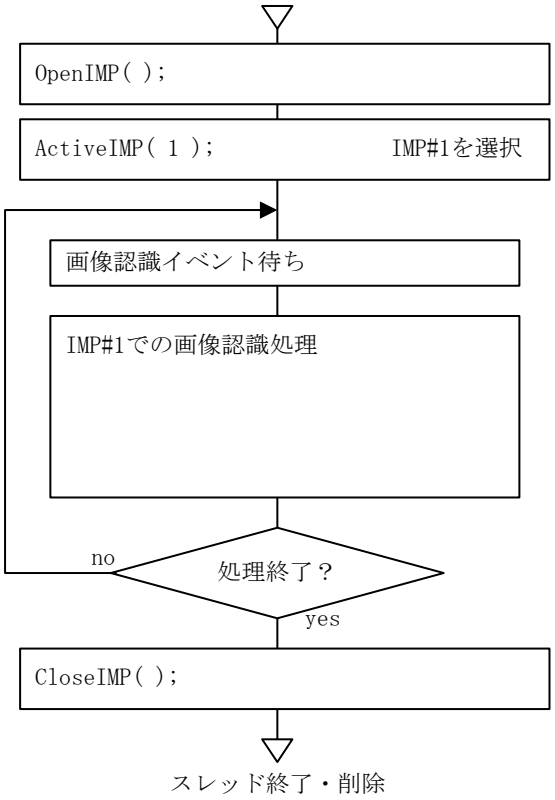


図5-3 マルチIMPでの処理フロー例

5.4 NVP画像処理コマンドアプリケーションの制限事項

5.4.1 マルチタスク動作に対する制限事項

NVP-Linuxで動作している画像処理コマンド(ライブラリ、ドライバ)は、Linuxのシングルプロセスのマルチスレッドに対応していますが、マルチプロセスでの動作には対応していません。リモートコマンドデーモン動作中に画像処理コマンドを使用したユーザーアプリケーションを実行したり、画像処理コマンドを使用したユーザーアプリケーションを複数のプロセスで実行した場合、正しい動作ができません。そのため、そのような動作が必要なアプリケーションには使用できませんのでご了承ください。

(1) リモートデーモン動作中のユーザーアプリケーション実行

リモートコマンドデーモン動作中(DIPSW:SW1-1 ON)に画像処理コマンドを使用したユーザーアプリケーションの実行はできません。

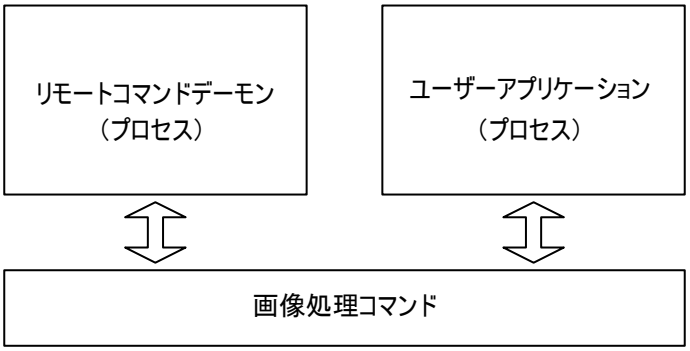


図5-4 画像処理コマンドが使用できない例(1)

(2) 2つ以上のユーザーアプリケーションプロセス実行

画像処理コマンドを使用したユーザーアプリケーションの複数のプロセスで実行や複数の画像処理コマンドを使用したユーザーアプリケーションの同時実行はできません。

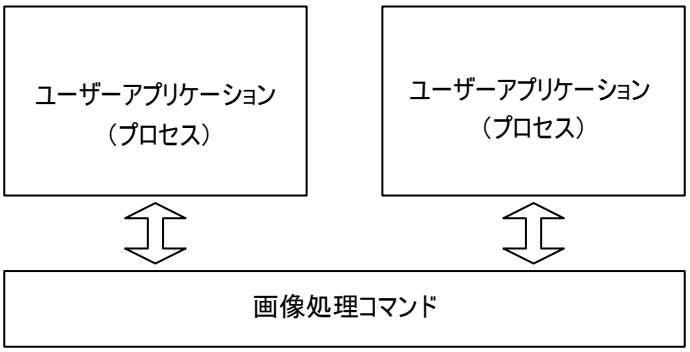


図5-5 画像処理コマンドが使用できない例(2)

5.4.2 リアルタイム動作に対する制限事項

NVPの画像処理コマンドやユーザーアプリケーションは、NVPに搭載しているLinux上で動作します。プロセスやスレッドはLinuxのスケジューラーが管理しており、それらのスケジューリングは、Linuxの動作に依存します。

NVPに搭載しているLinuxはリアルタイムOSではありません。アプリケーションで使用されるユースケースにて性能評価を実施頂き、実製品への適用可能性を十分検討頂くようお願い致します。

5.5 アプリケーションの実行

5.5.1 COM1シェルからの実行

NVPは電源投入又はリセットでLinuxが起動しCOM1ポート(RS-232C)にログインシェルが起動します。シェル起動時COM1の設定値等は「3.9 ログインシェルとCOM1」を参照ください。

ユーザー名及びパスワードを入力しログイン後、シェルプロンプトが表示されますので、RS-232Cのターミナル上から対話形式でコマンドを実行できるようになります。

例：実行ファイル:sample が、「/home/root/」ディレクトリにある場合

```
# /home/root/sample ↵
```

5.5.2 SSHシェルからの実行

NVPでLinuxが起動している状態でTeraTerm等のターミナルソフトでNVPにSSH接続することが可能です。なお、NVPのIPアドレスの初期値は「192.168.0.205」です。COM1シェルと同様にユーザー名及びパスワードを入力しログイン後、シェルプロンプトが表示されますので、TeraTermのターミナル上から対話形式でコマンドを実行できるようになります。

5.5.3 システム起動時の自動実行

NVPのLinuxはランレベル「5」で起動します。NVPの電源投入又はリセットでLinux起動時に自動的にアプリケーションを実行する場合は、アプリケーションを実行するシェルスクリプトを作成し、「/etc/rc5.d/」にコピーするか又はシンボリックリンクを作成しコピーして下さい。また、起動の順番が85～89になるように「S85～S89」で始まるファイル名で作成してください。

例：「/home/root/sample」アプリケーションを起動時、自動実行する場合

① シェルスクリプト(S85sample_boot.sh)を作成

S85sample_boot.sh

```
#!/bin/sh
/home/root/sample &
```

② S85sample_boot.sh を「/etc/rc5.d」にコピーするか「/etc/rc5.d」にシンボリックリンクを作成してください。

6. リモートモジュールフレームワーク

6.1 概要

リモートコマンドを使用しWindows PC とNVP-Linuxアプリケーションを連動させて制御するアプリケーションの構成で、NVP-Linuxで動作させるアプリケーションをリモートモジュールと呼びます。
リモートモジュールには動作の特徴から

- ・インテリジェントモジュール
- ・画像処理タスクモジュール
- ・割込み起動モジュール

という3通りのモジュール化の方法があります。なお、前記のモジュールは、Linuxの1つのプロセス内でのみ動作します。複数のプロセスには対応しません。

NVPのシステムでは、画像処理コマンドでの処理は画像処理プロセッサとNVPに搭載されているCPU(以降NVP-ARMと呼ぶ)とLinuxにより行われています。
Windows PCから実行される画像処理コマンドでは、コマンド1つ1つについて以下のフローに従いNVP-ARMで処理が実行されます。そのため、画像処理コマンドを実行するたびにNVP-ARMの処理時間以外にデータ転送やコマンド起動に数mS程度のオーバヘッドが加わります。

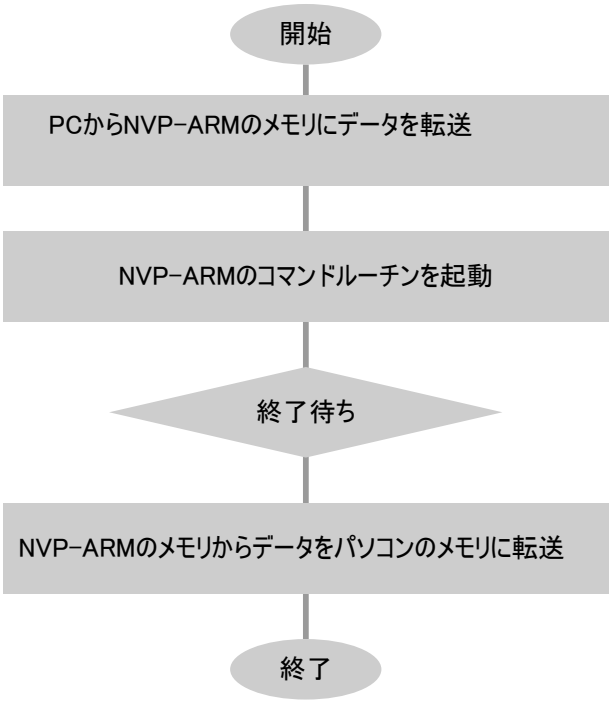


図6-1 コマンド実行フロー

そこで、画像処理コマンドでは、パソコンとのI／Fでのオーバヘッド削減と処理の分散化のために、ユーザーオリジナルの画像処理アプリケーションをNVP-ARMで実行できるように

- ・インテリジェントモジュール
- ・画像処理タスクモジュール
- ・割込起動モジュール

という3つのフレームワークを実装しています。このフレームモデルの特徴は、C言語の関数形式でパソコンからNVP-ARM側に簡単にデータを渡すことができることと、モジュールの実行制御を簡単に行うことができるということです。

6.1.1 リモートモジュールの開始と終了

画像処理コマンドライブラリをリモートモジュールとして使用する場合、ドライバ動作開始処理 StartIP() のオプションに「REMOTE_SERVER_BOOT | CONNECT_PORT_USRAPL」を設定して実行し、リモートコマンドサーバーを起動してください。また、PIOの割込み起動モジュールを使用する場合は、「PIOINT_SERVER_BOOT」を論理和で設定しPIO割込処理サーバーを起動してください。

リモートモジュールの構築については、「5.2 Linuxアプリケーションの構築」を参照してください。

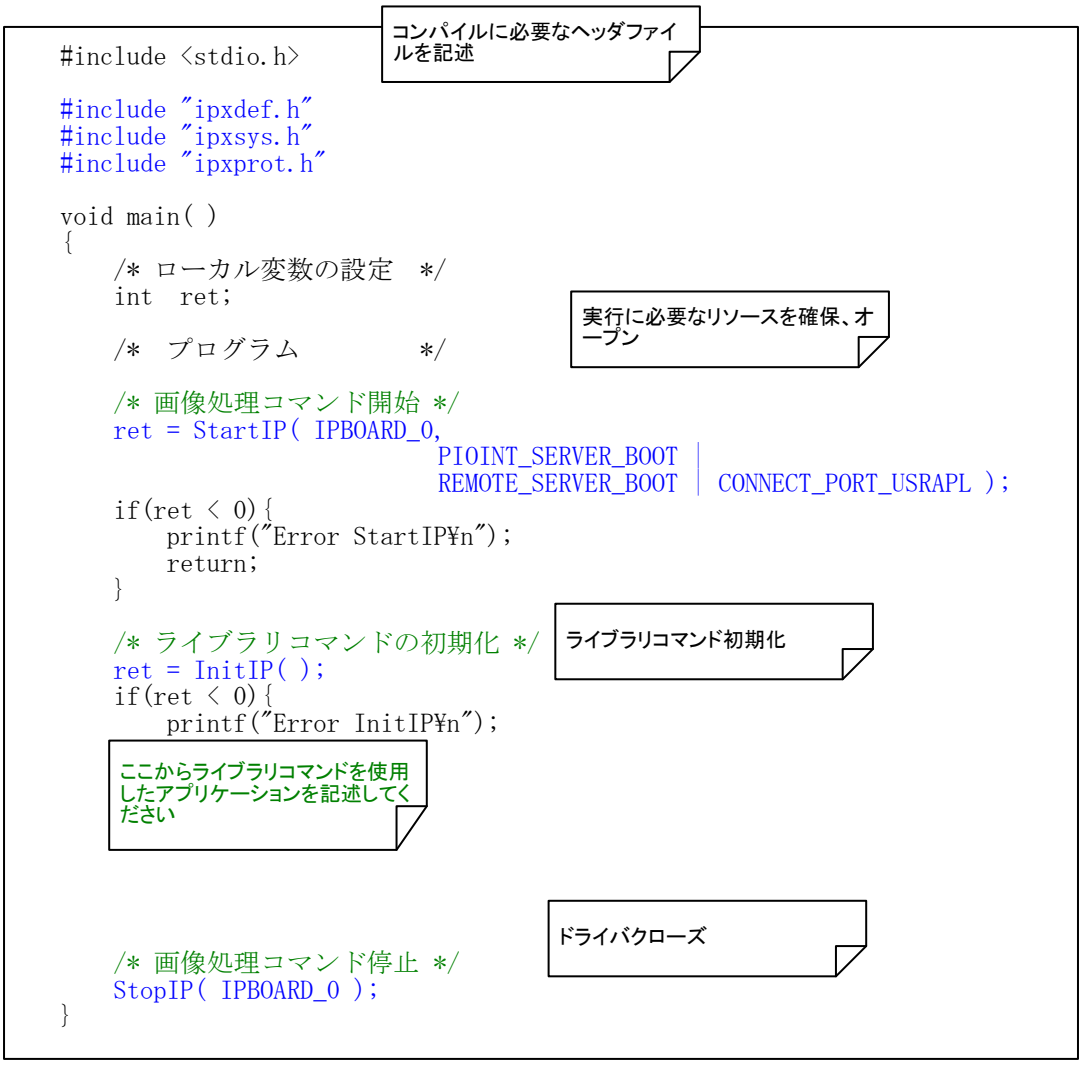


図6-2 コーディング例

6.1.2 スタック領域

リモートモジュールのスタック領域は、それぞれのモジュールにより割り当てられるスタックサイズが異なります。画像処理タスクモジュール、割込み起動モジュールで画像処理コマンドを使用する場合は、64Kバイト以上のスタックを確保してください。

表6-1 スタックサイズ

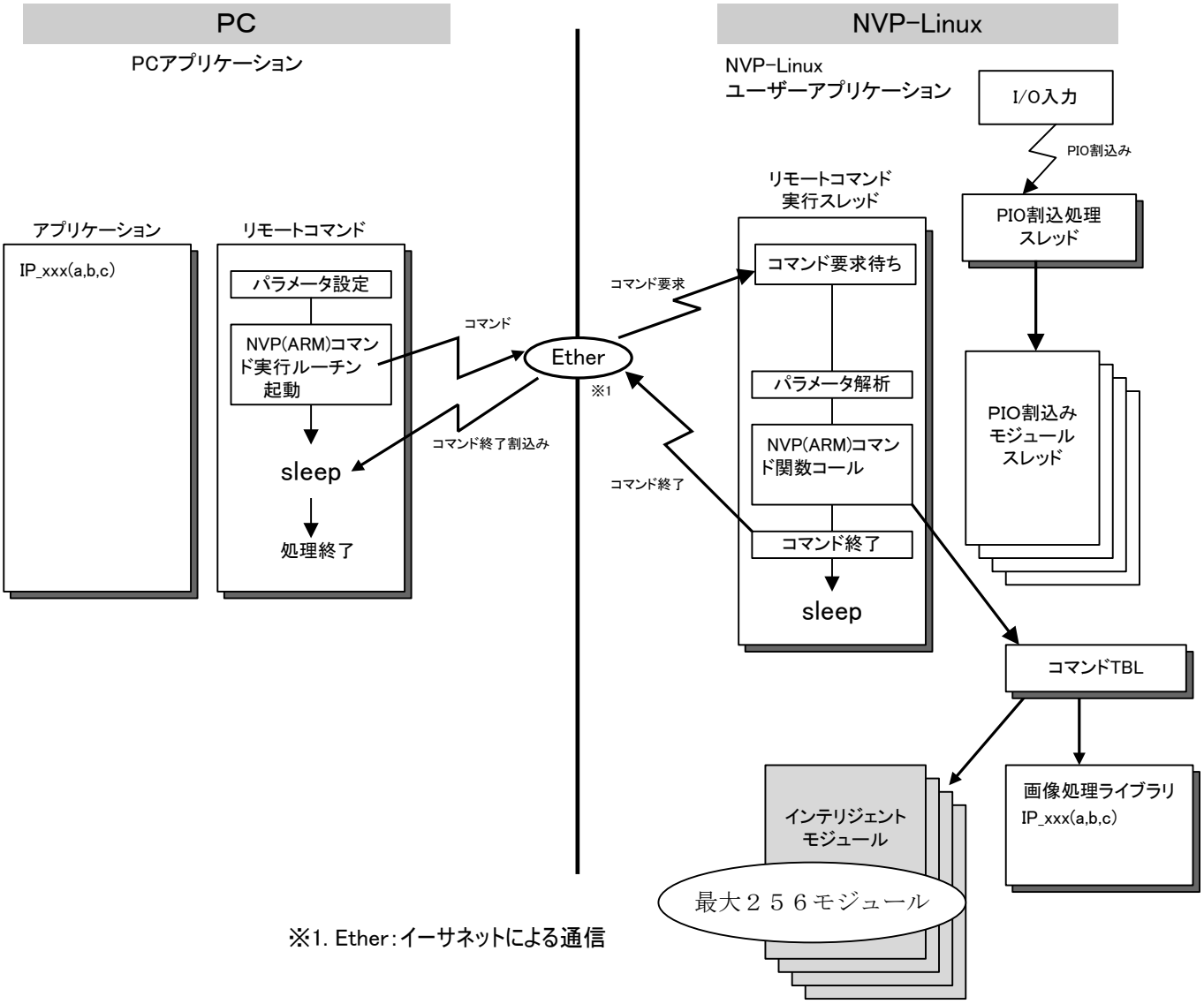
モジュールの属性	スタックサイズ	備考
インテリジェントモジュール	64Kバイト	
画像処理タスクモジュール	タスク生成時に指定	最低でも16KB必要。16KB未満指定時、16KBに設定される。
割込み起動モジュール	タスク生成時に指定	

6.2 インテリジェントモジュール

インテリジェントモジュールとは、画像処理コマンドを複数組み合わせて作成するユーザーオリジナルの画像処理コマンドのことです。NVPのシステムでは、画像処理は画像処理プロセッサとNVP-ARMにより行っているため、ユーザーの画像処理アプリケーションの一部をインテリジェントモジュールとして実装することにより、処理の分散化を実現できます。

インテリジェントモジュールは画像処理コマンドの一部として登録し実行されます。最大256モジュールの登録が可能です。

なお、インテリジェントモジュールを使用する場合、NVPユーザーアプリケーションでのドライバ動作開始処理 **StartIP()** のオプションに「**REMOTE_SERVER_BOOT | CONNECT_PORT_USRAPL**」を設定して実行し、リモートコマンドサーバーを起動してください。



※1. Ether:イーサネットによる通信

図6-3 インテリジェントモジュールの仕組み

6.2.1 インテリジェントモジュールのコーディング

Windows PCから起動されるインテリジェントモジュールのプログラムは、C言語の関数（サブルーチン）形式で記述します。関数名は任意でパラメータはint型、float型、ポインタ型で 0から最大16個まで指定できます。これらのパラメータは、PC側のプログラムで「モジュールサポートコマンド」によりデータが設定されます。それ以外は、PCでのプログラムと同一にしてください。

```
void inteli_module0(int a,float b,short *tbl)
{
   ..........
   ..........
}
```

6.2.2 インテリジェントモジュールの実行

本項では、リモートモジュールフレームワークを用いてNVP-Linuxユーザーアプリケーション内のインテリジェントモジュールを実行するためのWindows PC側のプログラムについて説明します。
以下に実行フローを示します。

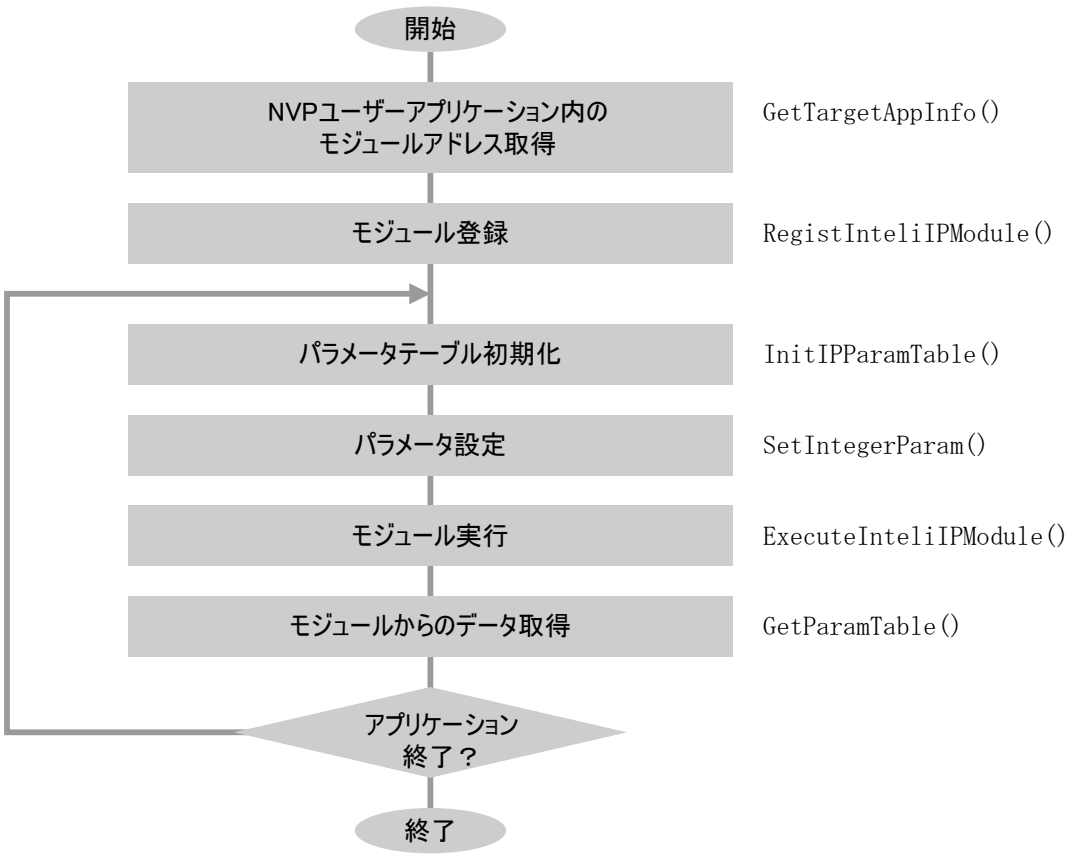


図6-4 モジュール実行フロー

(1) NVPユーザーアプリケーション内のモジュールアドレスアドレス取得

NVPユーザーアプリケーション内で**AllocTargetAppInfo()**を使用して設定したモジュールアドレスを取得します。

NVPユーザーアプリケーション

```
unsigned long   madr[4];

memset( madr, 0, sizeof( madr ) );

madr[0] = (unsigned long)inteli_module0;
madr[1] = (unsigned long)inteli_module1;
madr[2] = (unsigned long)inteli_module2;
madr[3] = (unsigned long)inteli_module3;

/* モジュールアドレス情報設定 */
AllocTargetAppInfo( madr, sizeof( madr ) );
```

PC側アプリケーション

```
unsigned long   madr[4];

memset( madr, 0, sizeof( madr ) );

/* モジュールアドレス情報取得*/
GetTargetAppInfo( madr, sizeof( madr ) );
```

(2) インテリジェントモジュールの登録

ダウンロードしたインテリジェントモジュールを **RegistInteliIPModule()** コマンドで登録します。

```
RegistInteliIPModule( MODULE_0, madr[0], 0 );
```

inteli_module0のアドレス

(3) インテリジェントモジュールへのパラメータ渡し

登録したインテリジェントモジュールにパラメータを渡す方法を説明します。

(a) パラメータテーブルの初期化

InitIPParamTable() コマンドでパラメータテーブルを初期化する。

```
MODULE_PARAM_TBL    prmtbl;

InitIPParamTable( &prmtbl, 3, INTELI_MODULE, 0, 0 );
```

(b) パラメータの設定

インテリジェントモジュールのパラメータがint型やfloat型の場合、**SetIntegerParam(),SetFloatParam()**コマンドでパラメータテーブルにデータを設定します。

```
SetIntegerParam( &prmtbl, PARAM_1, 123 );
SetFloatParam( &prmtbl, PARAM_2, 4.567 );
```

(c) テーブル(配列)パラメータの設定

インテリジェントモジュールのパラメータが配列の場合、**SetParamTable()**コマンドでパラメータテーブルにデータを設定します。

```
SetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

(4) インテリジェントモジュールの実行

ExecuteInteliIPModule()コマンドにより、パラメータのデータを転送し、登録したインテリジェントモジュールを実行します。

```
ExecuteInteliIPModule( MODULE_0, &prmtbl, NULL );
```

このコマンドはユーザーアプリケーション内のインテリジェントモジュールを起動し、その処理が終了するまでウェイトします。そして、処理が終了した時点でスリープが解除されこのコマンドが終了します。

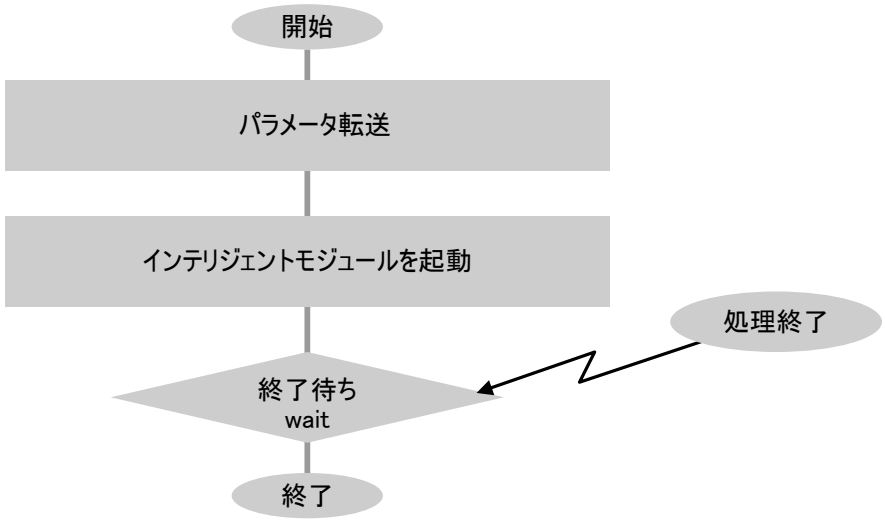


図6-5 コマンド実行フロー

(5) インテリジェントモジュールからのデータ取得

インテリジェントモジュール実行後にデータ取得する方法を説明します。

(a) 指定した配列パラメータが入出力データの場合

インテリジェントモジュールのパラメータが配列でインテリジェントモジュールにデータを渡してなおかつインテリジェントモジュール実行終了後にデータを取得する場合、**SetParamTable()**コマンドでパラメータテーブルにデータを設定し、インテリジェントモジュール実行終了後、**GetParamTable()**コマンドでデータを取得して下さい。

```
SetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
ExecuteInteliIPModule( MODULE_0, &prmtbl, NULL );
GetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

(b) 指定した配列パラメータが出力のみのデータの場合

インテリジェントモジュールのパラメータが配列でインテリジェントモジュール実行終了後にデータを取得する場合、**AllocParamTable()**コマンドでパラメータテーブルの領域を確保し、インテリジェントモジュール実行終了後、**GetParamTable()**コマンドでデータを取得して下さい。

```
AllocParamTable( &prmtbl, PARAM_3, sizeof(tbl) );
ExecuteInteliIPModule( MODULE_0, &prmtbl, NULL );
GetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

6.3 画像処理タスクモジュール

画像処理タスクモジュールは、Linuxの1つのタスクとして生成、実行されるため、パソコンとは独立して動作することが可能です。ユーザータスクは最大32モジュール(割込み起動モジュール含む)の生成が可能です。

6.3.1 画像処理タスクモジュールの動作

画像処理タスクモジュールはLinuxの1つのスレッドとして生成し実行します。タスクモジュールフレームワークは**CreateIPTask()**コマンドでLinuxスレッド生成し、**StartIPTask()**でします。また、モジュールは、Linuxスレッドの動作に適合するようにプログラムする必要があります。

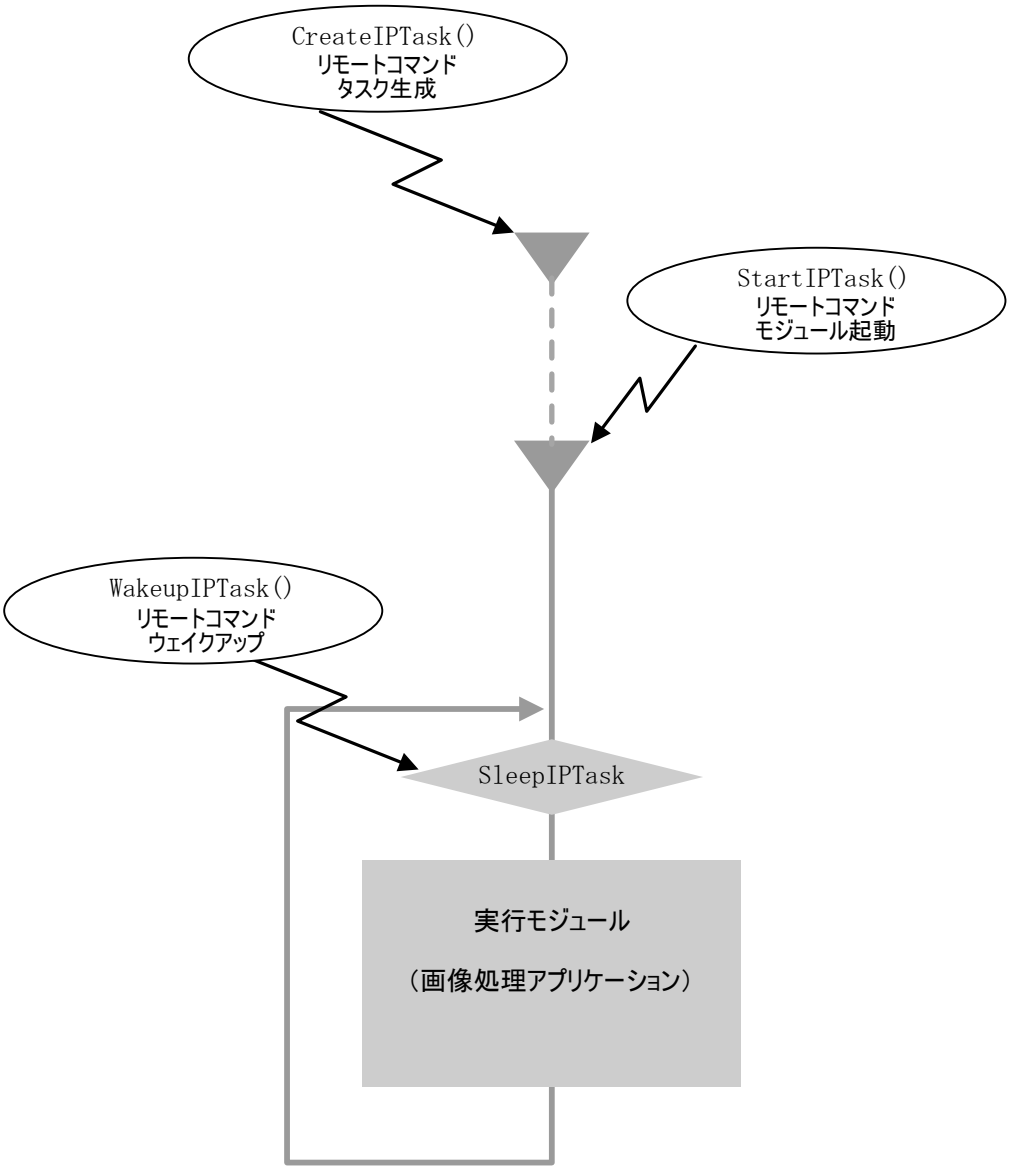


図6-6 画像処理タスクモジュールの動作

6.3.2 画像処理タスクモジュールのコーディング

画像処理タスクは、C言語の関数（サブルーチン）形式で記述します。

```
void task_module0( )
{
   ...........
   ...........

   for( ;; ){
      ret=SleepIPTask();
      if(ret < 0){
         /* タスククリーンアップ処理 */
         return; /* タスク終了 */
      }
      /* 画像処理 */
      ...........
      ...........
   }

}
```

6.3.3 PCとの同期

画像処理タスクモジュールに対して、Windows PC側の処理とNVPユーザーアプリケーションとの同期をとる方法として**WaitforIPTaskSignal()**と**SendSignaltoPC()**コマンドを用意しています。

まず、PC側は**WaitforIPTaskSignal()**コマンドでNVPユーザーアプリケーションの処理が終了するのを待ちます。そして、画像処理タスクモジュール側から**SendSignaltoPC()**コマンドを実行すると、PC側に終了シグナルが送信され、PC側は**WaitforIPTaskSignal()**コマンドのウェイト処理から抜けます。

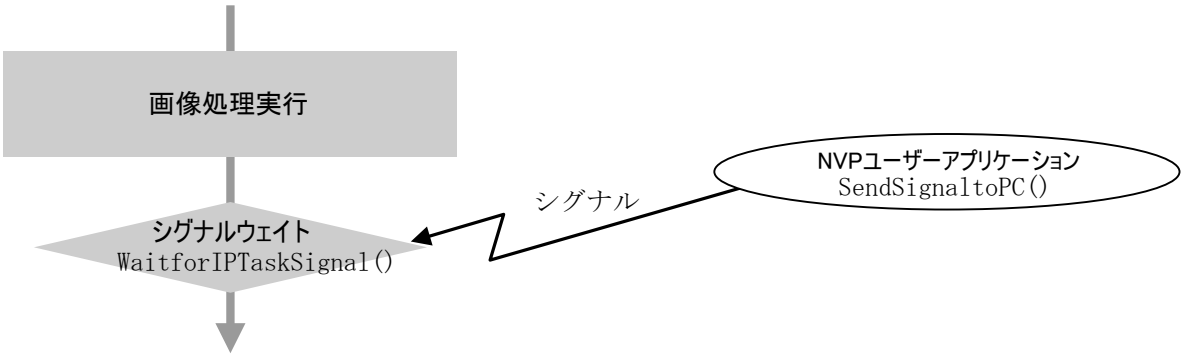


図6-7 PCとの同期

6.3.4 画像処理タスクモジュールの制御

画像処理タスクモジュールを制御するためのPC側の制御フローを示します。

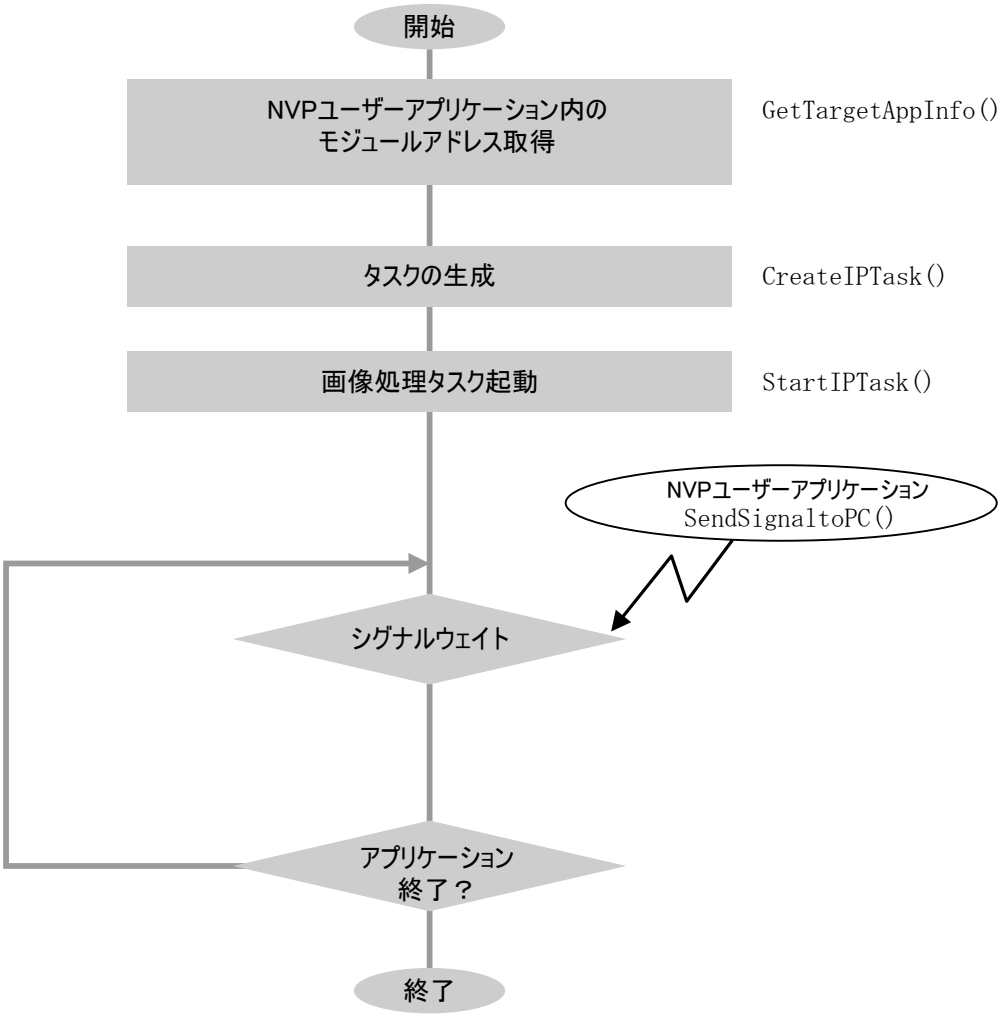
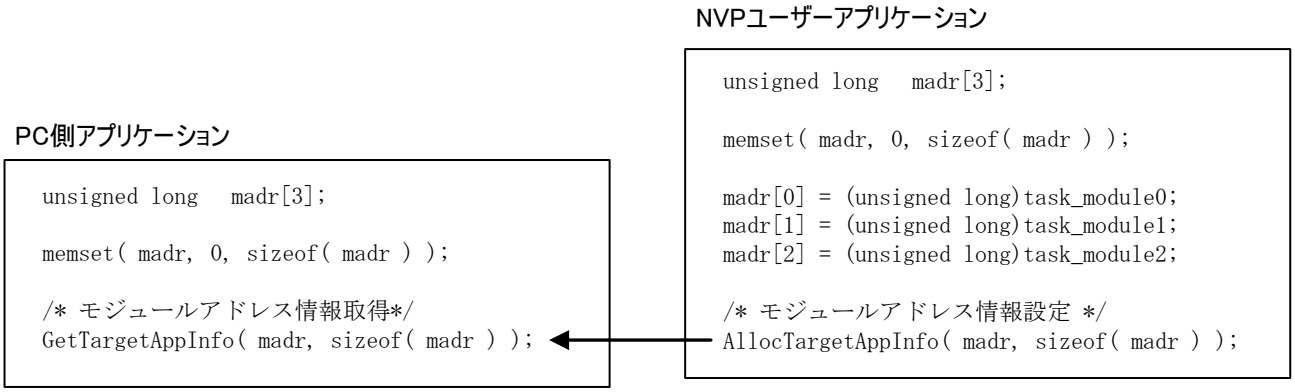


図6-8 画像処理タスクモジュール制御フロー

(1) NVPユーザーアプリケーション内の画像処理タスクモジュールのアドレス取得

NVPユーザーアプリケーション内でAllocTargetAppInfo()を使用して設定した画像処理タスクモジュールアドレスを取得します。



(2) タスクの生成

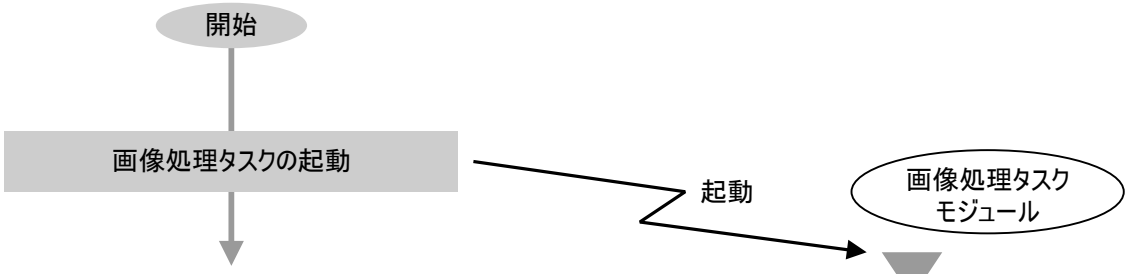
ダウンロードした画像処理タスクモジュールを生成します。

```
int tskid;
CREATE_TASK_TBL   iptsk;

iptsk.task_addr  = madr[0];    /* task_module0のアドレス */
iptsk.priority   = TASK_PRI_NORMAL;
iptsk.pbuff_size = 0;
iptsk.stack_size = 64 * 1024;  /* スタック64KB */
iptsk.task_opt   = 0;
iptsk.param_opt  = 0;

tskid= CreateIPTask( &iptsk );
```

(3) 画像処理タスクモジュールの起動

CreateIPTask()で生成した画像処理タスクを起動します。

```
StartIPTask( tskid );
```

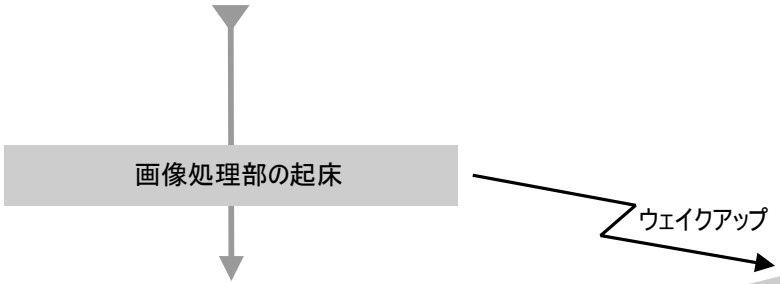


(4) 画像処理タスクモジュールのウェイクアップ

WakeupIPTask()コマンドにより、画像処理タスクをウェイクアップします。

このコマンドは画像処理タスクモジュールのスリープを
をウェイクアップするだけです。処理終了ウェイトは行いません。

```
WakeupIPTask(  tskid  );
```



(5) 画像処理タスクモジュールの終了ウェイト

WaitforIPTaskSignal()コマンドにより、画像処理タスクモジュールの**SendSignaltoPC()**コマンドによる終了シグナルをウェイトします。そして、画像処理タスクモジュールで**SendSignaltoPC()**コマンドを実行した時、PCでのウェイトが解除されこのコマンドが終了します。

```
WaiIPSigInf    SigInf

WaitforIPTaskSignal( PCI_SIGNAL_USER_0, 0, 0, &SigInf );
```

7. 割込み起動モジュール

7.1 PIO割込について

NVPは、フォトカプラによりアイソレートされたパラレルの入力／出力ポート（PIO）を搭載しています。
入力ポートはビット毎に割込に対応しており、入力ポートのレベルが「L」→「H」になる時の立ち上がりエッジによりNVP-Linuxに対して割込が発生します。

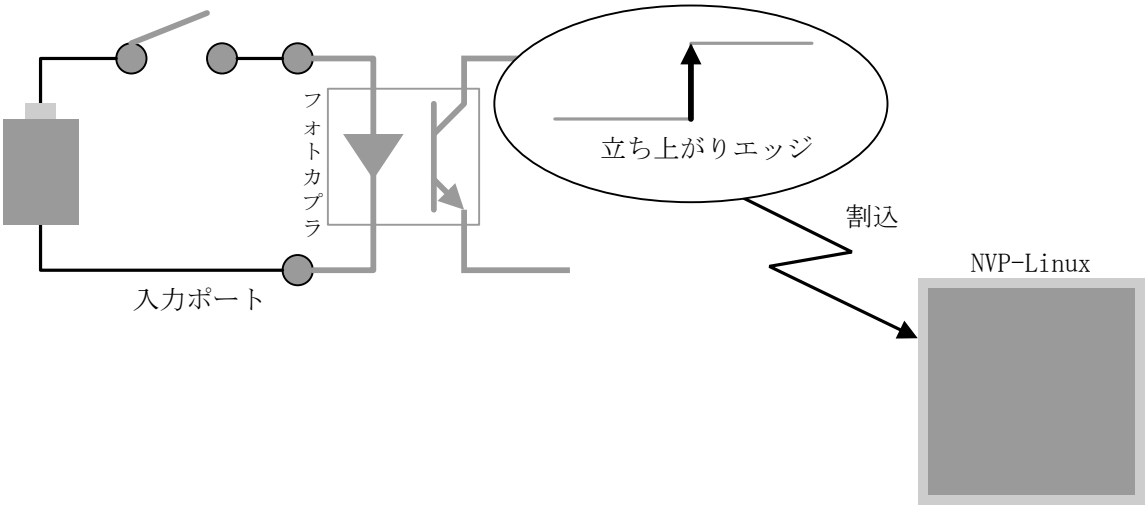


図7-1 入力ポート割込

PIO割込に対する処理をNVP-Linuxユーザーアプリケーションで簡単に実行できるように割込起動モジュールというフレームワークを実装しています。そのフレームワークにより、ユーザーアプリケーションのモジュールを登録するだけで、任意のPIO割込でそのモジュールを起動することができます。なお、PIOの割込み起動モジュールを使用する場合は、ドライバ動作開始処理**StartIP()**のオプションに「**PIOINT_SERVER_BOOT**」を論理和で設定しPIO割込処理サーバーを起動してください（「DI0～DI7」の割込み起動モジュールが動作するようになります）。
NVP-Ax430CLは、図7-2に示すようにPIOの入力ポート8ビット（DI0～DI7）を割込起動モジュールに割り当てられます。

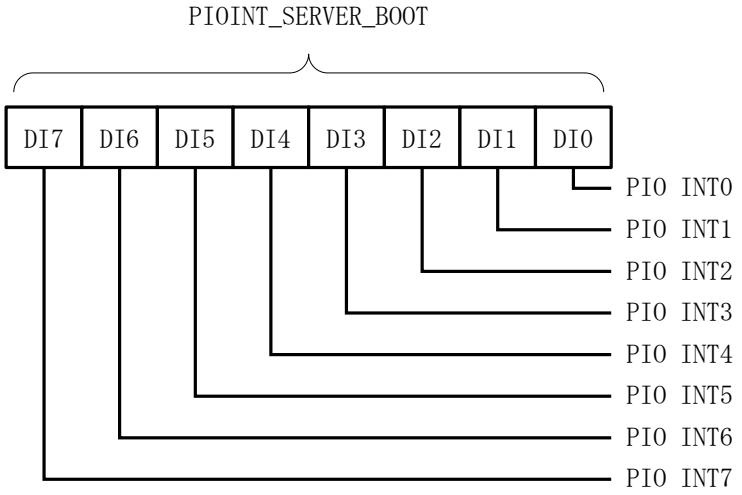


図7-2 NVP-Ax430CLのPIO入力ポートの割り当て

NVP-Ax430ACL/Ax435CL/Ax435FCLは、図7-3に示すようにPIOの入力ポート16ビット(DI8～DI15)を割込起動モジュールに割り当てられます。

なお、NVP-Ax430ACL/Ax435CL/Ax435FCLの「DI8～DI15」を割込み起動モジュールに割り当てる場合は更に「**PIOINT1_SERVER_BOOT**」を論理和で設定してください。

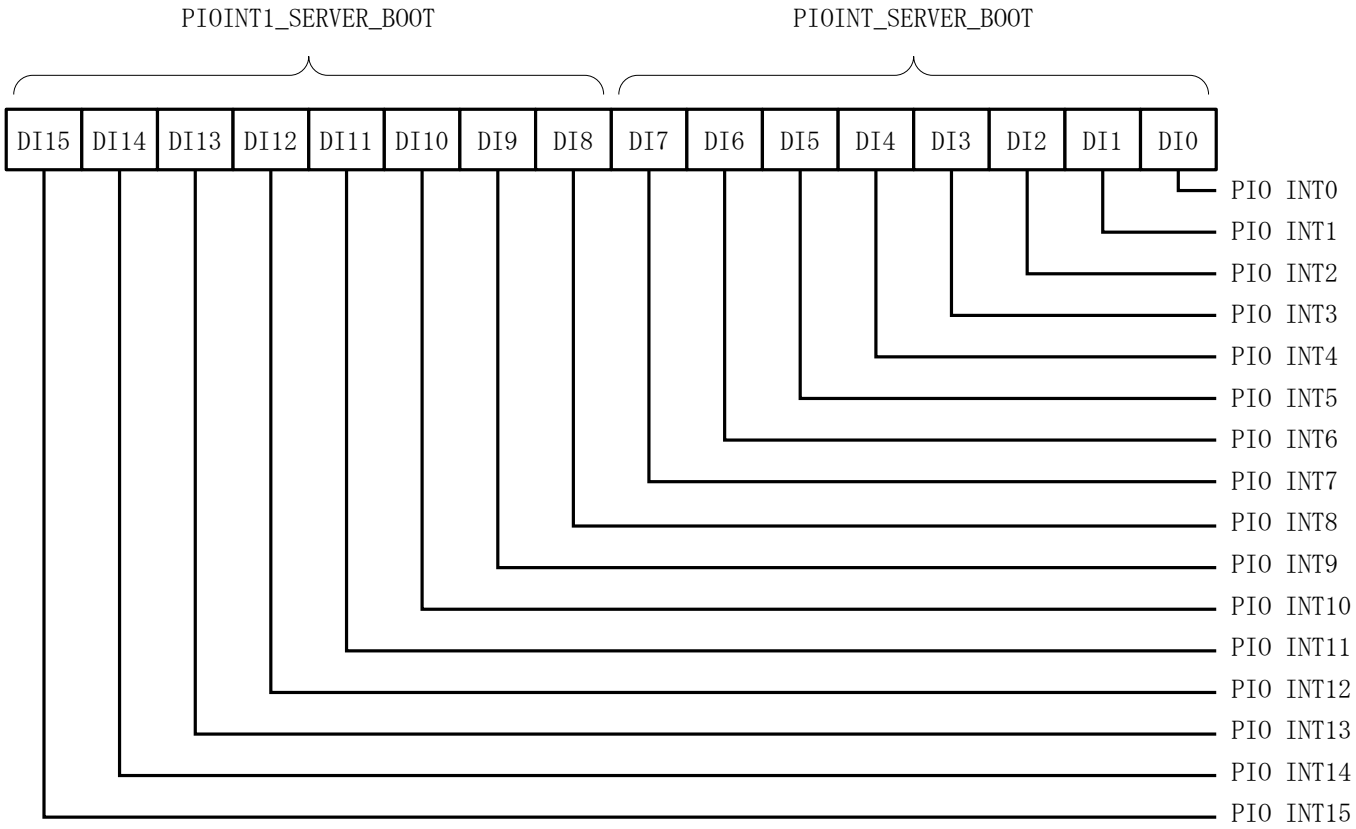
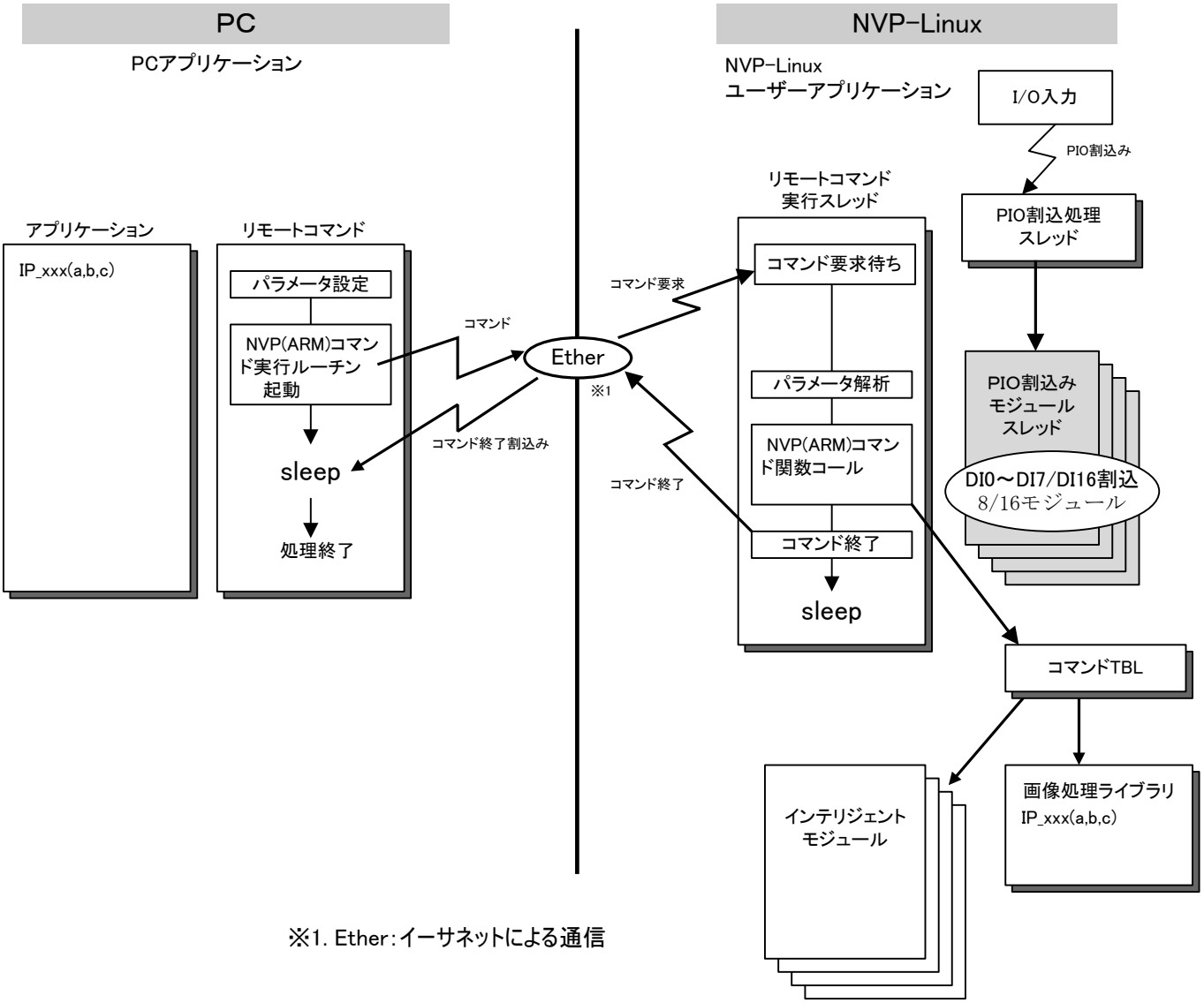


図7-3　NVP-Ax430ACL/Ax435CL/Ax435FCLのPIO入力ポートの割り当て

7.2 割込起動モジュールの概要

割込み起動モジュールとは、NVPのパラレルI／O（PIO）等の割込みにより起動することができるモジュールです。画像処理コマンドを複数組み合わせて作成したユーザーオリジナルの画像処理モジュールをPIOの割込み等により起動するモジュールとして登録することができます。

PIOからの割込で起動できる割込起動モジュールは画像処理タスクモジュールの1つとして登録され実行されます。NVPはPIO入力ポートのデータビットの割込にそれぞれ任意のタスクを対応させたモジュールとして登録することができます。NVPでは、ユーザは合計32タスクを生成することができ、そのうちの8タスク（NVP-Ax430ACL/Ax435CL/Ax435FCLは16タスク）を割込起動モジュールとして管理することが可能です。



※1. Ether：イーサネットによる通信

図7-4 割込モジュールの仕組み

7.3 割込起動モジュールの動作

割込起動モジュールは画像処理タスクモジュールとして登録し実行します。割込起動モジュールは、初期化部モジュールと実行部モジュールのペアのモジュールで1つのタスクを構成します。

初期化部モジュールは、**StartIPTaskwithParam()**コマンドで起動されタスクが終了するまで1度だけ動作します。初期化部は、ユーザアプリケーションの変数などの初期化に使用します。

実行部モジュールは、**WakeupIPTaskwithParam()**コマンドや対応するPIOの入力割込によりウェイトが解除され、実行部が実行されます。実際の画像処理アプリケーションはこの実行部に登録します。

また、初期化部モジュール、実行部モジュールには、インテリジェントモジュールと同様にパラメータを渡すことが可能です。

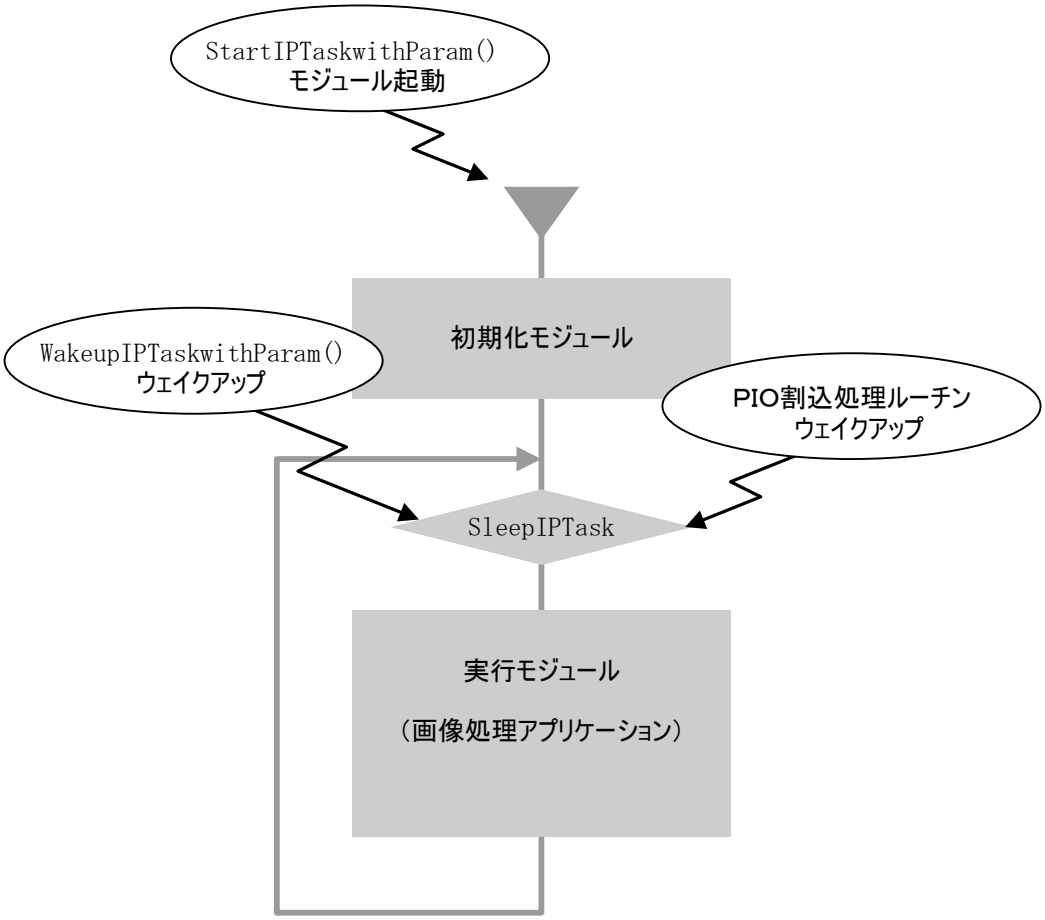


図7-5 割込モジュールの動作

7.4 割込み起動モジュールのコーディング

割込起動モジュールでは、初期化部モジュールと実行部モジュールの2つのモジュールを作成します。
Windows PC側から起動されるNVPユーザーアプリケーションは、C言語の関数（サブルーチン）形式で記述します。

初期化部モジュール

```
void pioint0_task_init(int a,float b)
{
   ..........
   ..........
}
```

実行部モジュール

```
void pioint0_task(int a,float b,short *tbl)
{
   ..........
   ..........
   SendSignaltoPC(.........);
}
```

関数名は任意でパラメータはint型、float型、ポインタ型で0から最大16個まで指定できます。これらのパラメータは、PC側のプログラムで「モジュールサポートコマンド」によりデータが設定されます。それ以外は、PC側でのコーディングと同一にしてください。

7.5 割込起動モジュールの制御

本項では、NVP-Linux側の割込起動モジュールを制御するためのPC側のプログラムについて説明します。
以下に基本的な制御フローを示します。

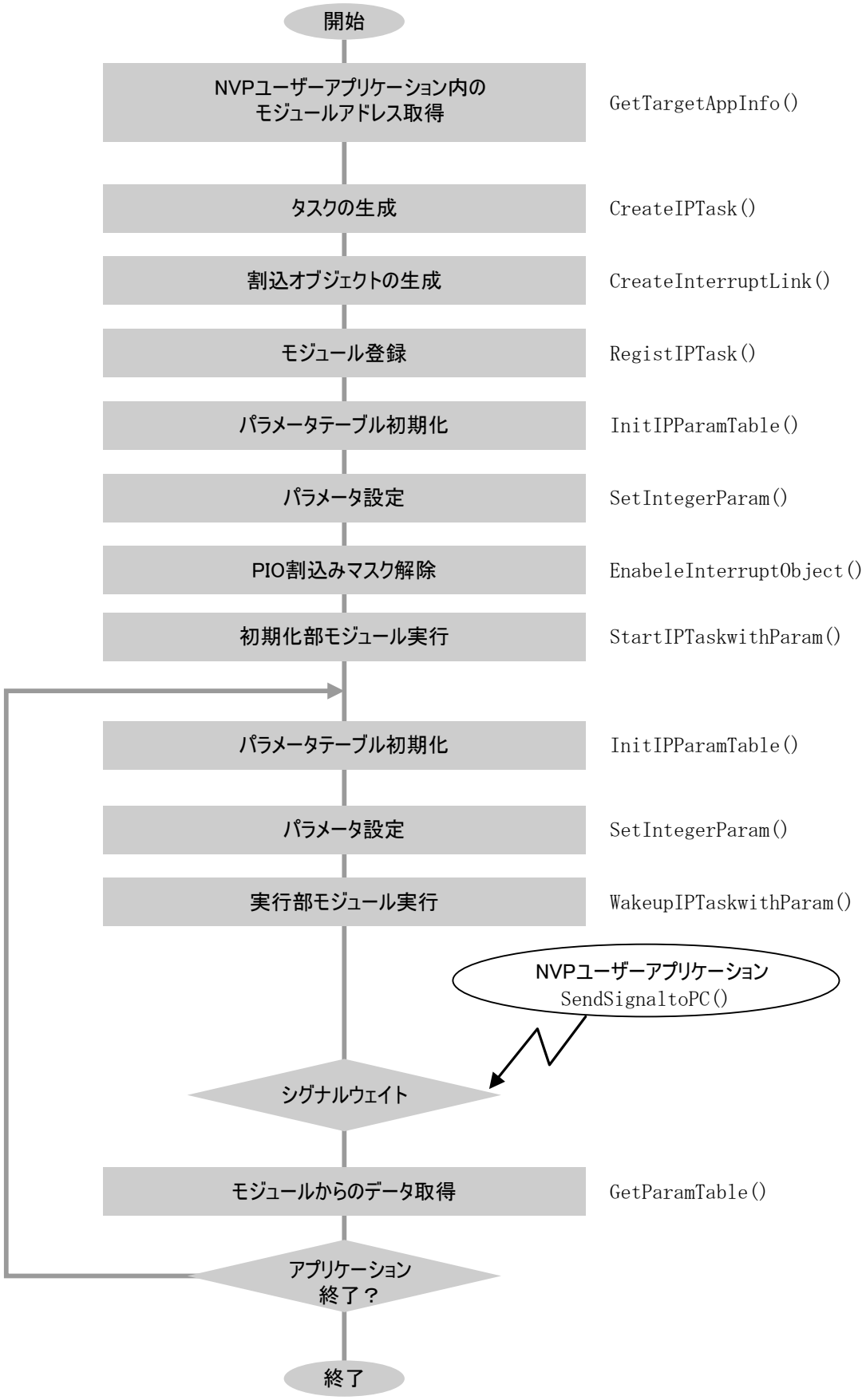
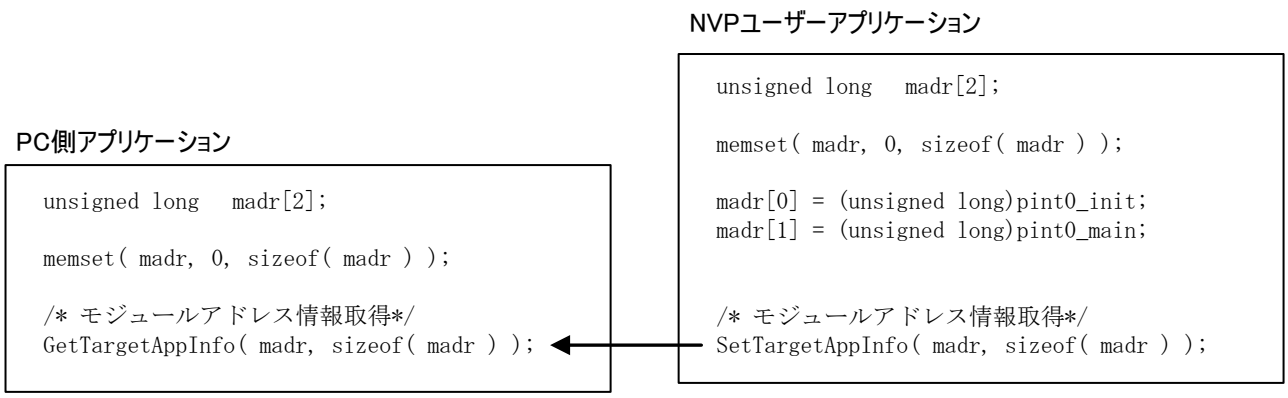


図7-6 割込起動モジュール制御フロー

(1) NVPユーザーアプリケーション内の割込み起動モジュールのアドレス取得

NVPユーザーアプリケーション内で**SetTargetAppInfo()**を使用して設定した割込み起動モジュールアドレスを取得します。



(2) タスクの生成

ダウンロードした割込み起動モジュールのタスクを生成します。

```
int tskid;
CREATE_TASK_TBL   iptsk;

iptsk.task_addr  = 0; /* 割込み起動モジュールの場合は「０」 */
iptsk.priority   = TASK_PRI_NORMAL;
iptsk.pbuff_size = 0;
iptsk.stack_size = 64 * 1024;  /* スタック64KB */
iptsk.task_opt   = 0;
iptsk.param_opt  = 0;

tskid= CreateIPTask( &iptsk );
```

(3) 割込オブジェクトの生成

```
INT_DEVICE_OBJ   intobj;

intobj.intdev = INTDEV_PIO;
intobj.evtpri = 0;
intobj.intbit = 0;        /* PIO[0]の割込みの場合 */
intobj.opt    = 0;

CreateInterruptLink( &intobj, tskid, INTEVENT_WAKEUP, 0 );
```

(4) 割込起動モジュールの登録

ダウンロードした割込起動モジュールを**RegistIPTask()**コマンドで登録します。

```
RegistIPTask( tskid, madr[0], madr[1], TASK_NO_OPTION );
```

(5) 割込起動モジュールへのパラメータ渡し

登録した割込起動モジュールにパラメータを渡す方法を説明します。初期化モジュール、実行モジュールの両方が同じ方法でパラメータの設定ができます。

(a) パラメータテーブルの初期化

InitIPParamTable()コマンドでパラメータテーブルを初期化します。

```
MODULE_PARAM_TBL   prmtbl;

InitIPParamTable( &prmtbl, 3, PIOINT_MODULE, tskid, 0 );
```

(b) パラメータの設定

割込起動モジュールのパラメータがint型やfloat型の場合、**SetIntegerParam()**,**SetFloatParam()**コマンドでパラメータテーブルにデータを設定します。

```
SetIntegerParam( &prmtbl, PARAM_1, 123 );
SetFloatParam( &prmtbl, PARAM_2, 4.567 );
```

(c) テーブル(配列)パラメータの設定

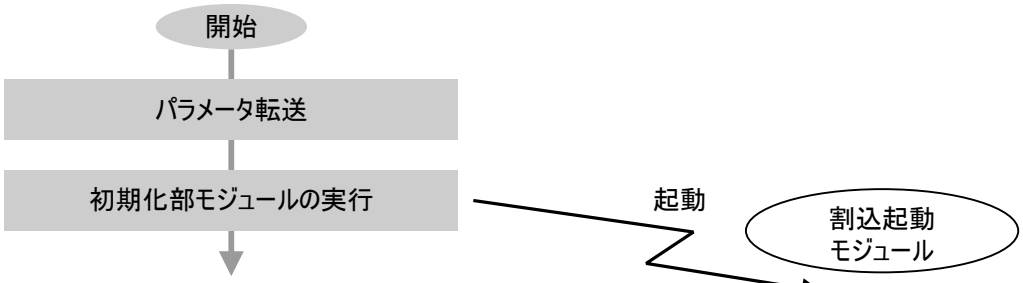
割込起動モジュールのパラメータが配列の場合、**SetParamTable()**コマンドでパラメータテーブルにデータを設定します。

```
SetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

(6) 初期化部モジュールの実行

StartIPTaskwithParam()コマンドにより、パラメータのデータを転送し、登録した割込起動モジュールの初期化部モジュールを実行します。

```
StartIPTaskwithParam( tskid, &prmtbl );
```

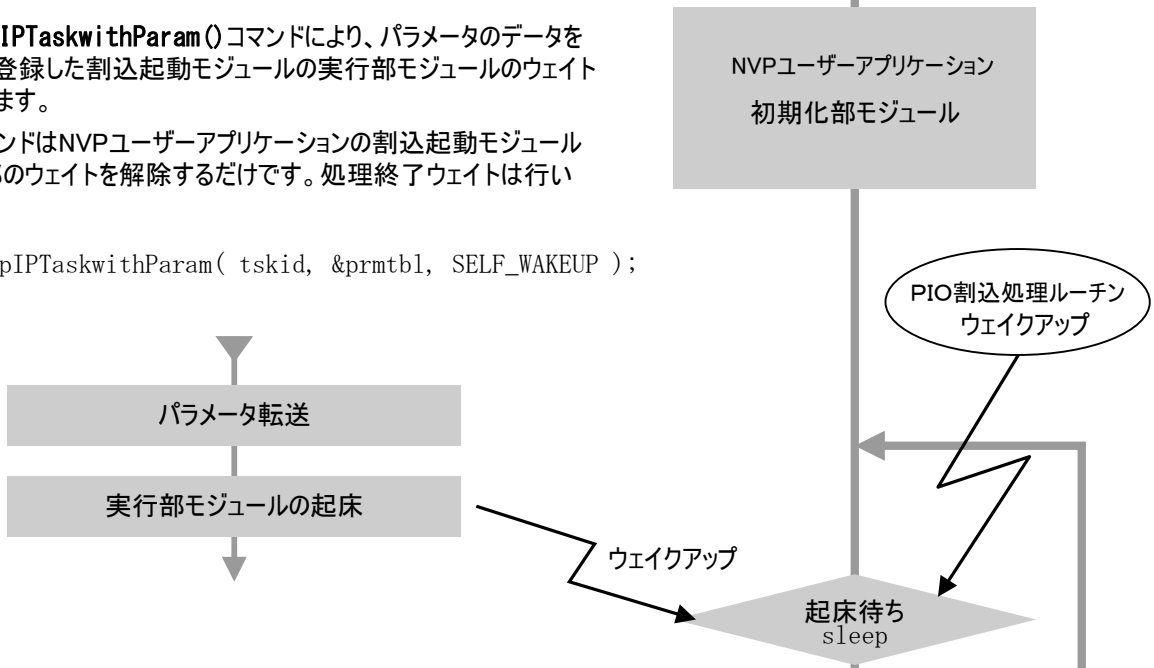


(7) 実行部モジュールのウェイクアップ

WakeupIPTaskwithParam()コマンドにより、パラメータのデータを転送し、登録した割込起動モジュールの実行部モジュールのウェイトを解除します。

このコマンドはNVPユーザーアプリケーションの割込起動モジュールの実行部のウェイトを解除するだけです。処理終了ウェイトは行いません。

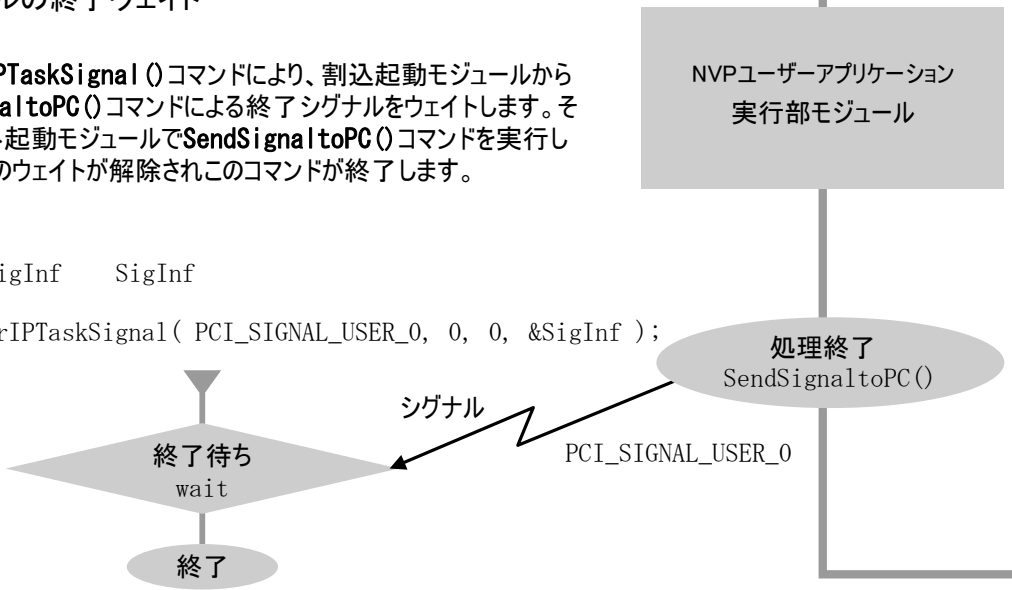
```
WakeupIPTaskwithParam( tskid, &prmtbl, SELF_WAKEUP );
```



(8) モジュールの終了ウェイト

WaitforIPTaskSignal()コマンドにより、割込起動モジュールからの**SendSignaltoPC()**コマンドによる終了シグナルをウェイトします。そして、割込み起動モジュールで**SendSignaltoPC()**コマンドを実行した時、PCでのウェイトが解除されこのコマンドが終了します。

```
WaiIPSigInf    SigInf

WaitforIPTaskSignal( PCI_SIGNAL_USER_0, 0, 0, &SigInf );
```



(9) 割込起動モジュールからのデータ取得

割込起動モジュール実行後にデータ取得する方法を説明します。

(a) 指定した配列パラメータが入出力データの場合

割込起動モジュールのパラメータが配列で割込起動モジュールにデータを渡し、かつ割込起動モジュール実行終了後にデータを取得する場合、**SetParamTable()**コマンドでパラメータテーブルにデータを設定し、割込起動モジュール実行終了後、**GetParamTable()**コマンドでデータを取得します。

```
SetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
WakeupIPTaskwithParam( tskid, &prmtbl, PIO_WAKEUP );
WaitforIPTaskSignal( PCI_SIGNAL_USER_0, &prmtbl, 0, 0, NULL );
GetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

(b) 指定した配列パラメータが出力のみのデータの場合

割込起動モジュールのパラメータが配列で割込起動モジュールにデータを渡し、かつ割込起動モジュール実行終了後にデータを取得する場合、**AllocParamTable()**コマンドでパラメータテーブルの領域を確保し、割込起動モジュール実行終了後、**GetParamTable()**コマンドでデータを取得します。

```
AllocParamTable( &prmtbl, PARAM_3, sizeof(tbl) );
WakeupIPTaskwithParam( tskid, &prmtbl, PIO_WAKEUP );
WaitforIPTaskSignal( PCI_SIGNAL_USER_0, &prmtbl, 0, 0, NULL );
GetParamTable( &prmtbl, PARAM_3, &tbl, sizeof(tbl) );
```

8. 画像処理コマンド

8.1 画像処理コマンドの構成

画像間演算、空間フィルタリングなどの画像処理コマンドは、サブルーチン形式のコマンドとしてC言語で利用できます。
詳細は画像処理コマンドリファレンスを参照してください。

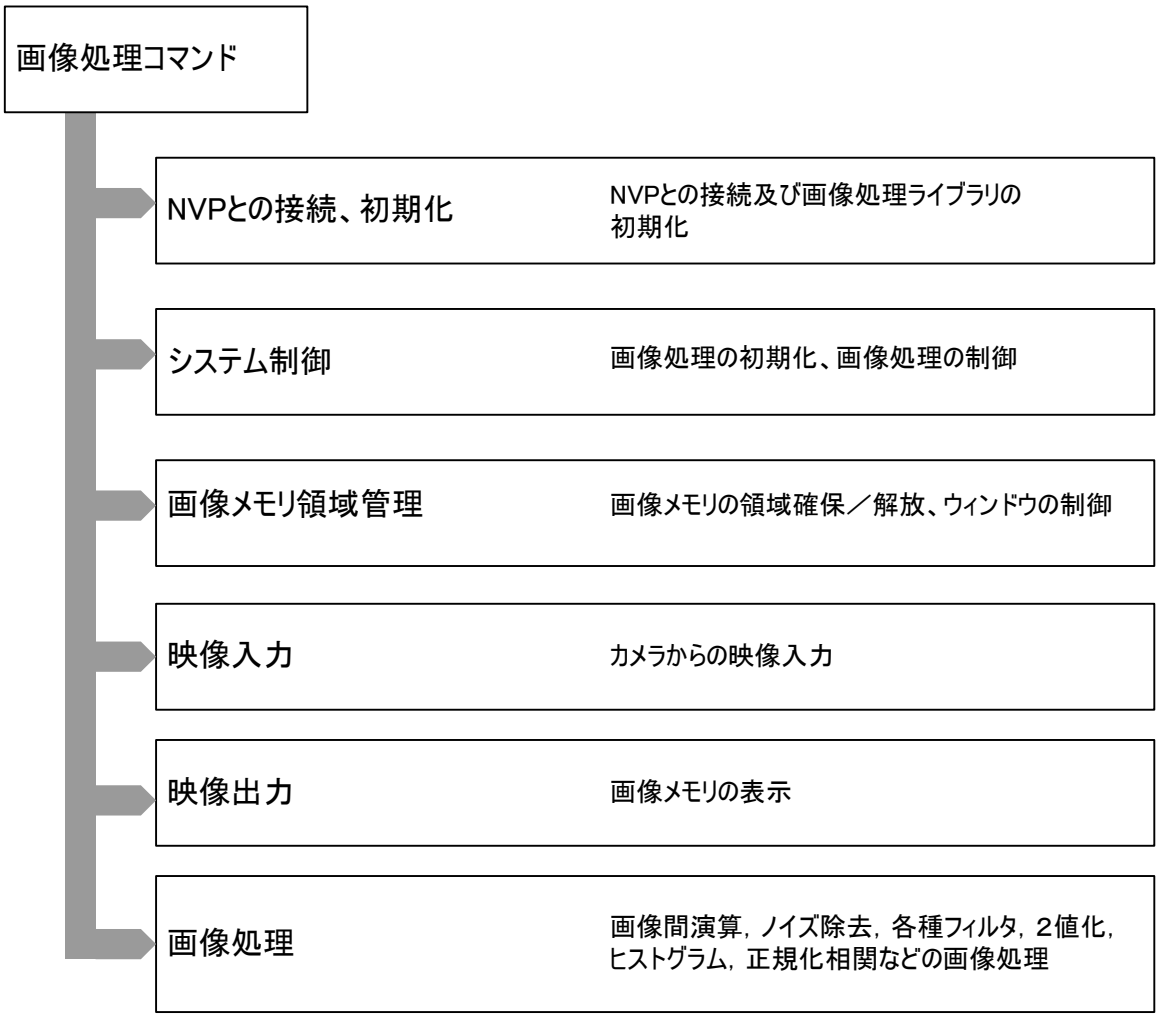


図8-1 画像処理コマンドの構成

8.2 NVPとの接続及び初期化

リモートコマンドを使用する場合は、**StartIP()**または**OpenIPDev()**でNVPとの接続、NVP-Linuxユーザーアプリケーションでは、**StartIP()**でNVP-Linuxの画像処理ライブラリの初期化を行う必要があります。

8.2.1 リモートコマンドでのNVPとの接続

Windows PCでリモートコマンドから画像認識コマンドを実行する場合、最初にリモートコマンドデーモン又はNVP-Linuxユーザーアプリケーションとの接続を行いリモートコマンドが動作できる状態にする必要があります。

(1) デバイスID無しリモートコマンドでのNVPとの接続

デバイスID無しのリモートコマンドで画像認識ボードを使用する場合は、**StartIP()**を実行しNVPと接続します。このとき、NVP-Linuxのリモートコマンドデーモンに接続する場合は、オプションパラメータに「**CONNECT_PORT_DAEMON**」を設定し、Linuxユーザーアプリケーションの場合、オプションパラメータに「**CONNECT_PORT_USRAPL**」を設定します。NVPと接続する場合は、**StartIP()** 実行前に接続先のリモートコマンドデーモン及びユーザーアプリケーションが実行されている必要があります。

アプリケーションの終了時は**StopIP()** を実行しNVP接続を停止します。

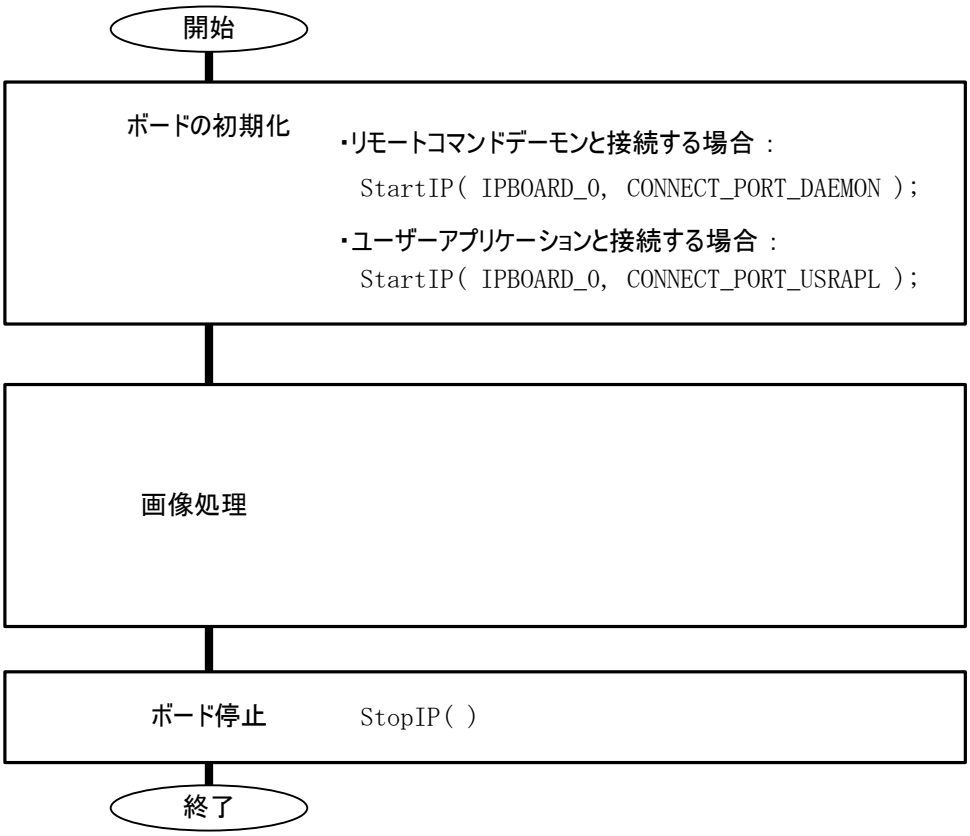


図8-2 デバイスID無しリモートコマンドでのNVPとの接続

1台のPCに複数のボードを使用する場合は、Windowsのスレッド毎に各ボード番号でStartIP()を実行し、ActiveIP()でボードを切り換えて下さい。

(2) デバイスID付きリモートコマンドでのNVPとの接続

デバイスID付きのリモートコマンドで画像認識ボードを使用する場合は、ボードのオープン処理**OpenIPDev()**を実行しNVPと接続しデバイスIDを取得します。このとき、NVP-Linuxのリモートコマンドデーモンに接続する場合は、オプションパラメータに「**CONNECT_PORT_DAEMON**」を設定し、Linuxユーザーアプリケーションの場合、オプションパラメータに「**CONNECT_PORT_USRAPL**」を設定します。NVPと接続する場合、**OpneIPDev()**実行の前に接続先のリモートコマンドデーモン及びユーザーアプリケーションが実行されている必要があります。

プログラムの終了時は画像認識ボードのクローズ処理**CloseIPDev()**を実行しNVP接続を停止します。

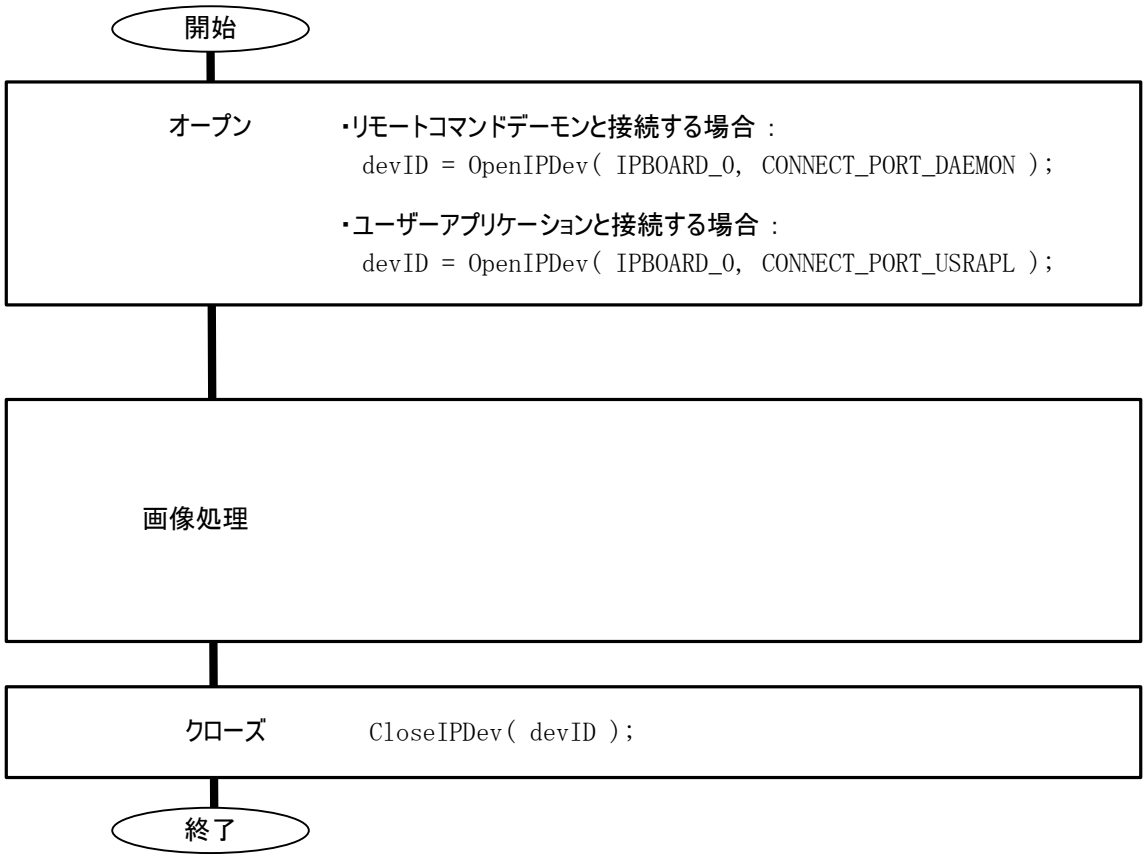


図8-3 デバイスID付きリモートコマンドでのNVPとの接続

8.2.2 リモートコマンドデーモンの起動

リモートコマンドデーモンと接続する場合は、リモートコマンドデーモンが実行されている必要があります。リモートコマンドデーモンはDIPSW:SW1-1をONにしてLinuxを起動することで実行されます。なお、リモートコマンドデーモン実行中は、他の画像処理コマンドを使用したユーザーアプリケーションのプロセスを実行しないでください。お互いの処理が正常に実行できません。

8.2.3 Linuxユーザーアプリケーションでの画像処理ライブラリの初期化

スタンドアロンの画像認識アプリケーションやリモートモジュールフレームワークを使用したアプリケーションでは、**StartIP()**を実行し画像処理ライブラリの初期化を行う必要があります。このとき、リモートモジュールフレームワークを使用したアプリケーションの場合、オプションパラメータに「**REMOTE_SERVER_BOOT | CONNECT_PORT_USR**」を設定します。また、アプリケーションの終了時に**StopIP()**を実行しクリーンアップを行います。

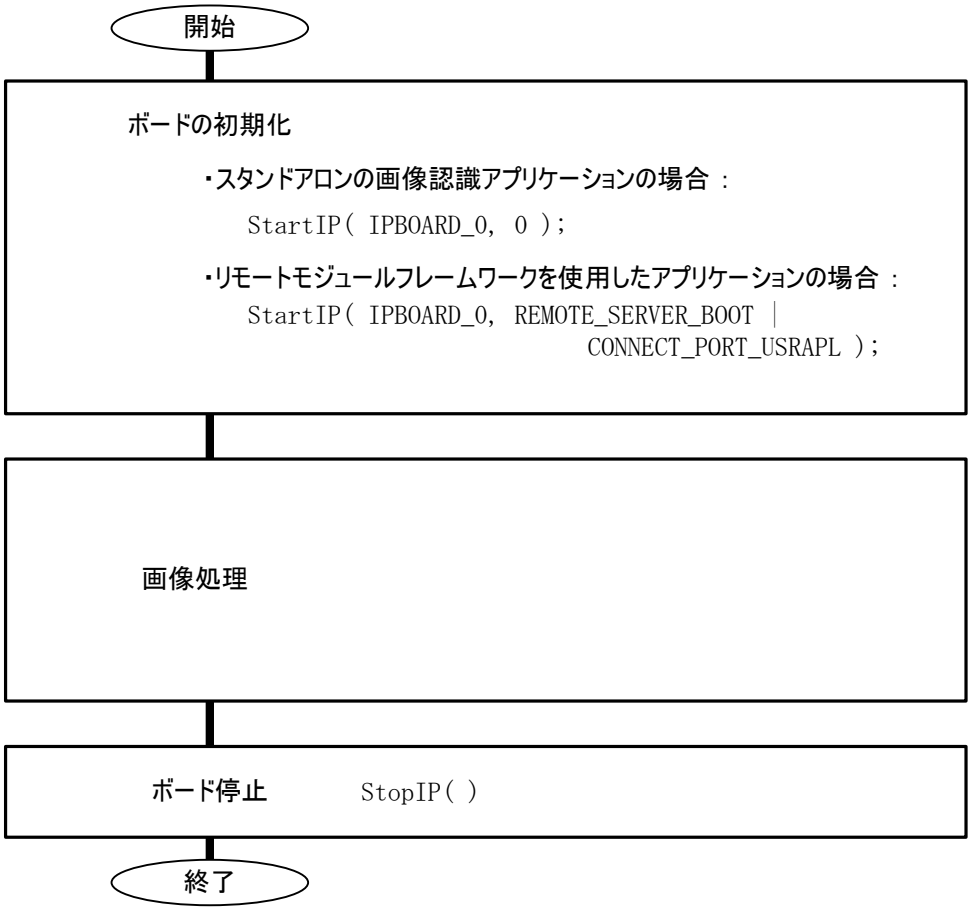


図8-4 Linuxユーザーアプリケーションでの画像処理ライブラリの初期化

8.3 システム制御

システム制御コマンドは、システムの初期化やエラー情報読み出し等、システム全体を制御します。

8.3.1 システムの初期化

電源投入時は、画像処理コマンドが初期化されていません。アプリケーションの初期化部で InitIP() コマンドを実行し、画像処理コマンドの初期化を行って下さい。

8.3.2 システムの状態遷移図

画像認識ライブラリには、画像処理システム未初期化状態、動作可能状態、エラー状態の3種類の状態が存在します。

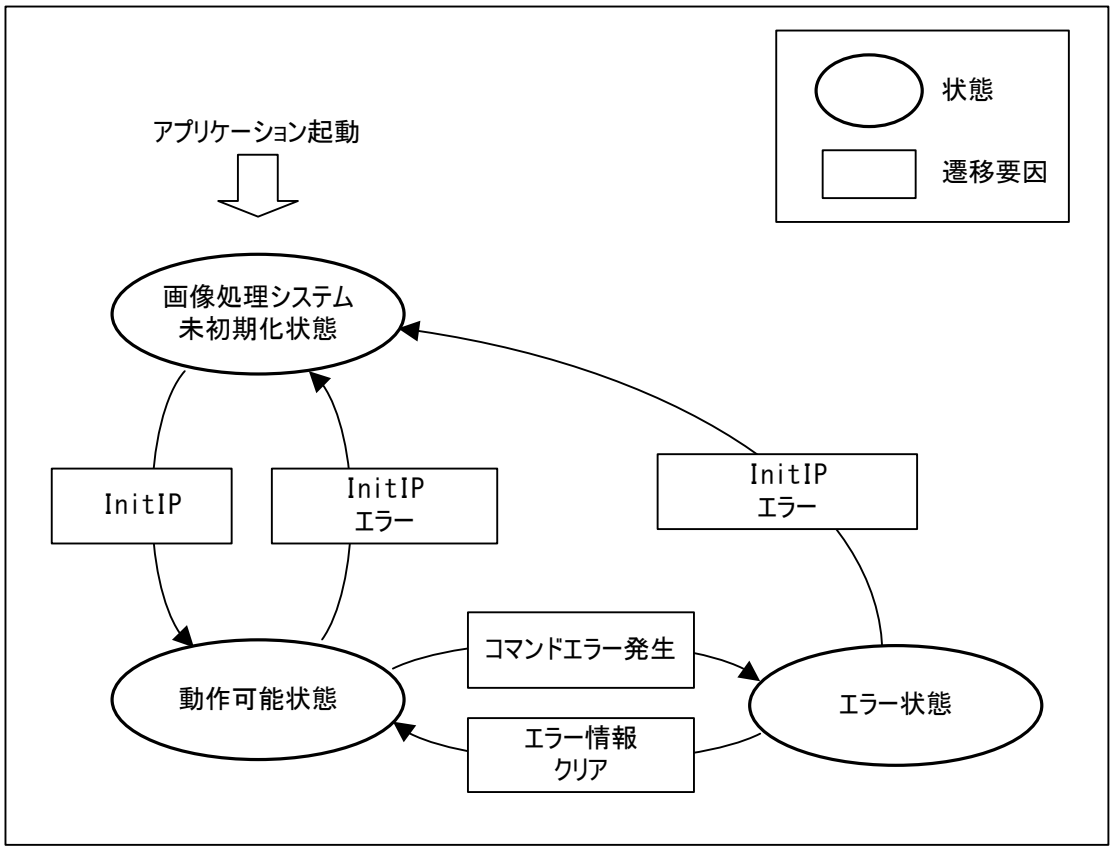


図8-5 システムの状態遷移図

8.3.3 エラー情報管理

画像処理コマンドでエラーが発生すると、エラー情報のダイアログボックスが表示され、OKボタンを押すまで処理が中断します(ダイアログ表示はWindows側のリモートコマンドのみの機能です)。画像処理コマンド以外のコマンドでは、下記のエラー情報管理は行われません。

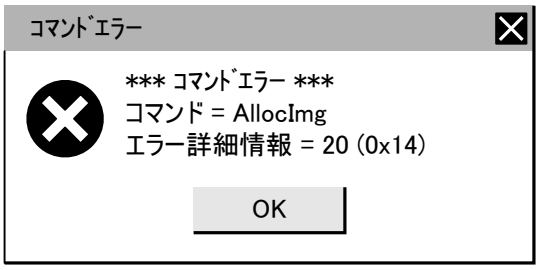


図8-6 システムの状態遷移図

このエラー情報のダイアログボックスは**DisableIPErrorMessage()**及び**EnableIPErrorMessage()**で出力を制御できます。画像処理コマンドでエラーが発生すると、コマンドのリターンコードにはエラーコードが返されますが、詳細エラー情報はリターンコードに反映されずシステムに登録されます。また、エラー発生後の画像処理は、エラー情報をクリアしない限り全て実行することができません。そのため、ユーザは**ReadIPErrorTable()**コマンドでエラー情報を読み出すか、または**ClearIPError()**コマンドでエラー情報をクリアする必要があります※1。

以下に画像処理コマンドでエラーが発生した際の処理手順を示します。

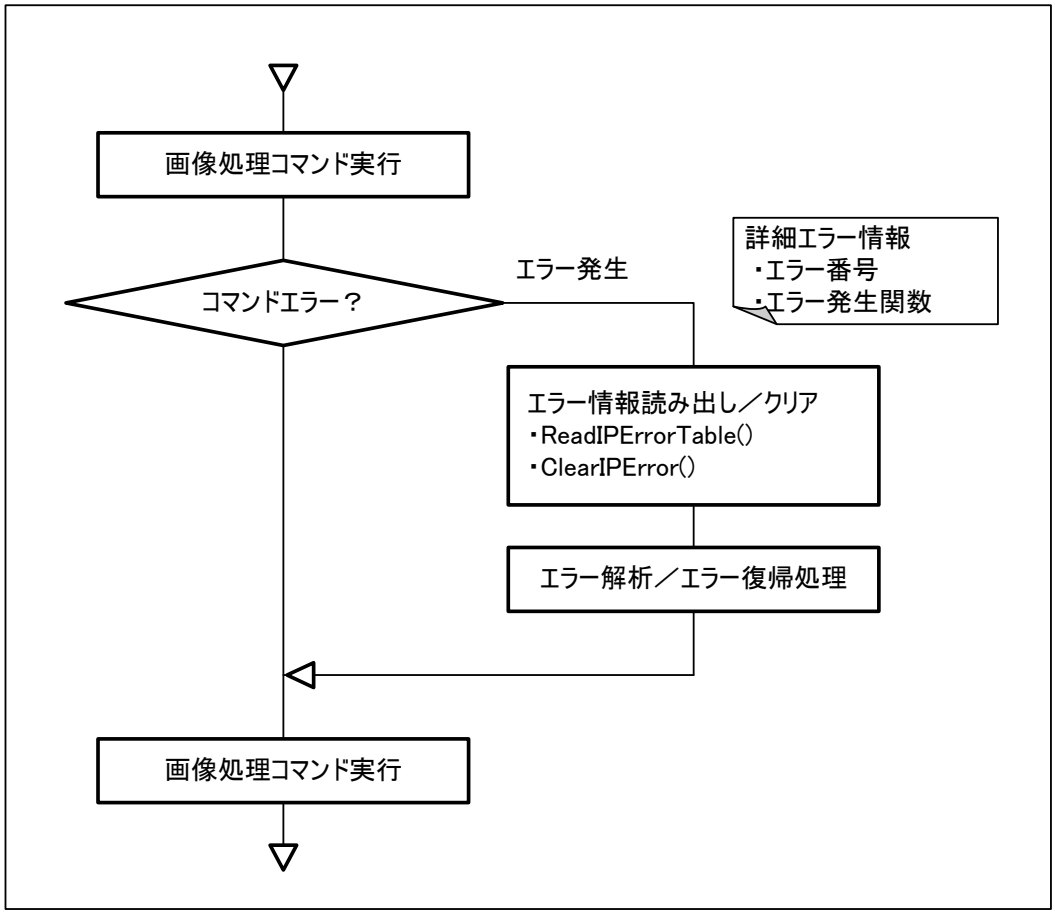


図8-7 コマンドエラー処理手順

※1 リモートコマンド、NVP-Linuxのコマンドはスレッド毎に管理されているためスレッド毎にエラー情報読出し及びエラークリアの発行が必要です。

8.4 画像メモリ領域管理

画像メモリを使用するための領域の確保、解放と、ウィンドウの制御を行います。

8.4.1 画像メモリの概要

画像メモリは、カメラ映像の入力データや画像処理演算結果を格納するメモリです。
画像メモリは、画像メモリ確保コマンドでモノクロ画面やRGB画面を確保します。画像メモリには濃淡画像メモリと2値画像メモリの区別はなく、2値画像として使用するときは、黒は0、白は255の値で使用します。

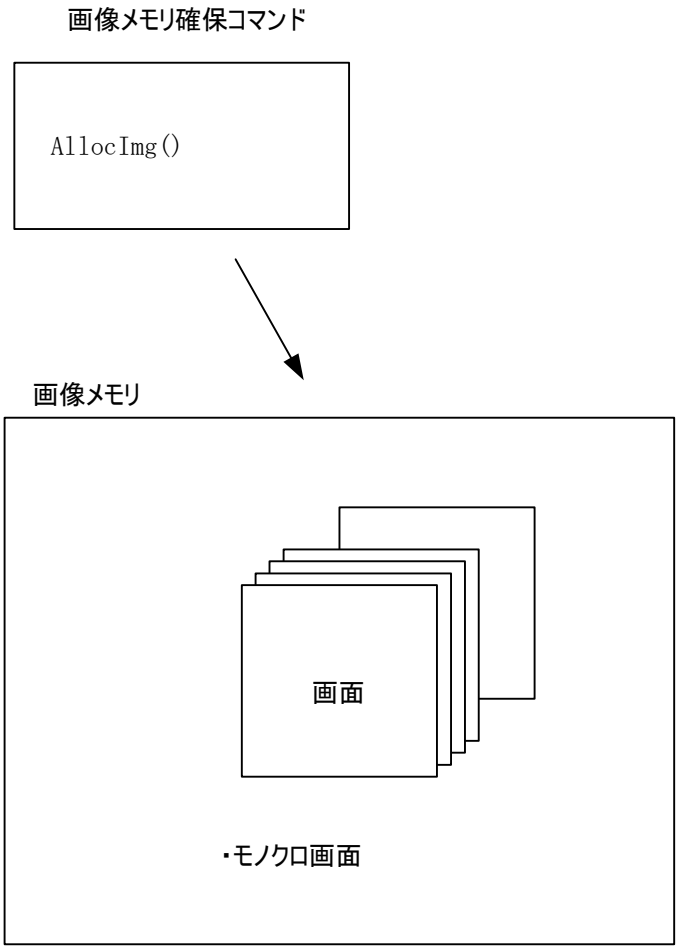
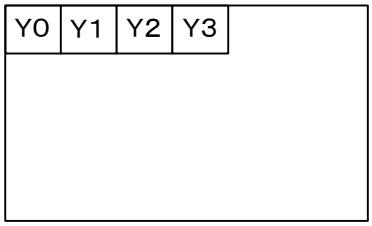
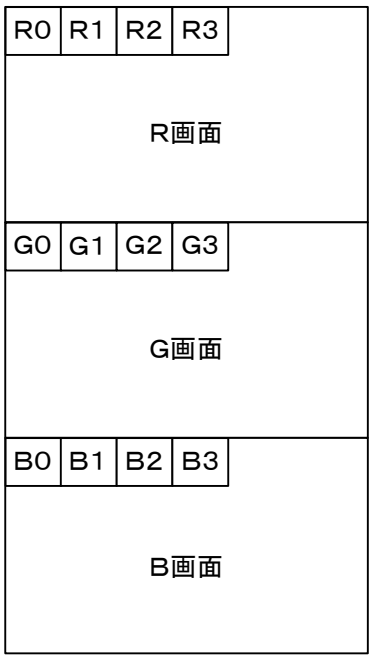
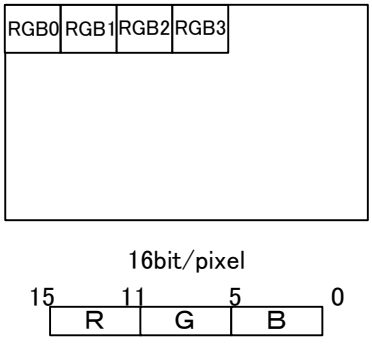


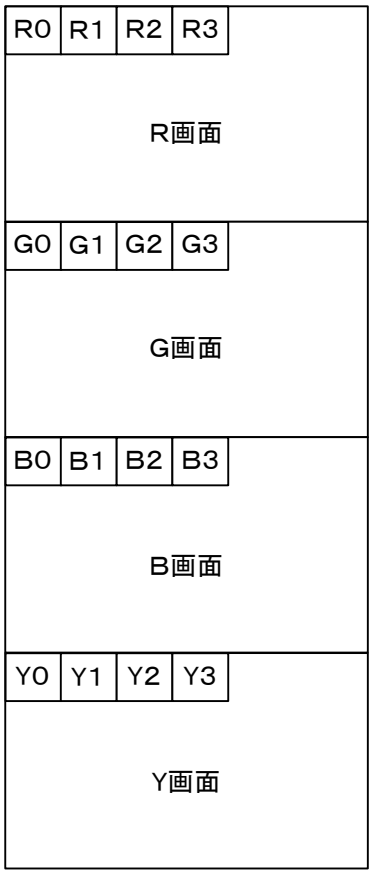
図8-8 画像メモリ

8.4.2 画像メモリの画面タイプ

画像処理コマンドでは、モノクロ画面（Y画面）とRGBカラー画面の画像メモリを確保可能です。Y画面は映像入力、画像処理および映像表示に使用できます。コンポーネントRGB画面は映像入力(※)と画像処理、RGB16画面は映像表示に使用できます。以下に画面タイプの一覧を示します。

表8-1 画像メモリの画面タイプ

画面タイプ	フォーマット	画面確保コマンド	詳細
Y	 8bit/pixel	AllocImg()	8bit/pixelの画面。濃淡及び2値の画像処理、モノクロ映像入力表示に使用できます。
コンポーネントRGB	 24bit/pixel	AllocRGBImg()	8bit/pixelのR画面、G画面、B画面で構成される画面。R、G、Bの3画面でRGBカラー情報を構成します。RGBカラーの映像入力に使用できます。直接、映像表示はできません。 R画面、G画面、B画面はそれぞれモノクロ画面として画像処理に使用できます。
RGB16	 16bit/pixel 	AllocRGB16Img()	16bit/pixelの画面。16bitデータは上位からRデータ5bit、Gデータ6bit、Bデータ5bitで、1画面でRGBカラー情報を構成します。RGBカラーの映像表示に使用できます。映像入力、画像処理はできません。

コンポーネント RGBY		AllocRGBYImg()	8bit/pixelのR画面、G画面、B画面と8bit/pixelの濃淡Y画面で構成される画面。R、G、Bの3画面でRGBカラー情報、RGBLUT変換データ等のY画面で構成します。マルチコンポーネント映像入力に使用できます。RGB部分は、直接、映像表示はできません。 R、G、B画面はRGB画面、Y画面はモノクロ画面として画像処理に使用できます。

8.4.3 画像メモリのサイズ

画像処理に使用できる画面サイズは、最大で16352×16384で画像メモリを確保できます。なお、画像処理ハードウェアの制限により、横幅は32バイトで割り切れるサイズを指定する必要があります。
また、1024×1024を超える大きさの範囲では画像処理ができないコマンドがありますので注意してください。

8.4.4 画像メモリの座標系

画面の座標系は、画面の左上隅を原点とした下図のような座標系となります。

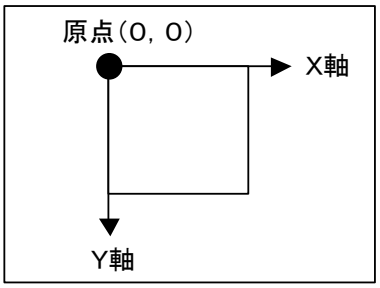


図8-9 画像メモリの座標系

8.4.5 画像メモリのデータタイプ

画像メモリに格納されるデータは、濃淡データと2値データの2種類があります。濃淡データは256階調を持ち、符号なし8ビット(0～255)と符号付8ビット(－128～127)の2種類があります。基本設定は符号なし8ビットです。
2値データは、黒と白の2階調のデータです。黒は0、白は255の値で画像メモリに格納されます。

8.4.6 ウィンドウの概要

ウィンドウとは、処理対象画面、または結果格納画面内の処理領域のことです。画像処理は画像メモリで確保したウィンドウのサイズで行われます。よって、画面のサイズより小さいウィンドウを設定することにより、処理を高速に行うことができます。

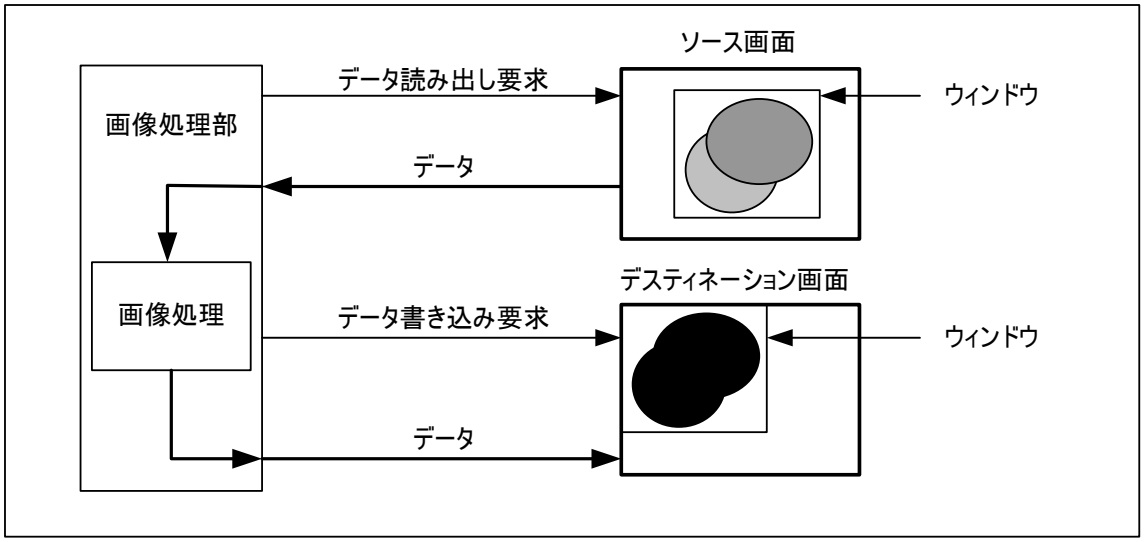


図8-10 ウィンドウの役割

8.4.7 ウィンドウの座標系

ウィンドウの座標系は、画面の左上隅を原点とした下図のような座標系となります。ウィンドウ設定値は、画面の原点相対の座標を指定します。また、ヒストグラム等の画像処理はウィンドウで指定した始点を原点として抽出座標を出力します。

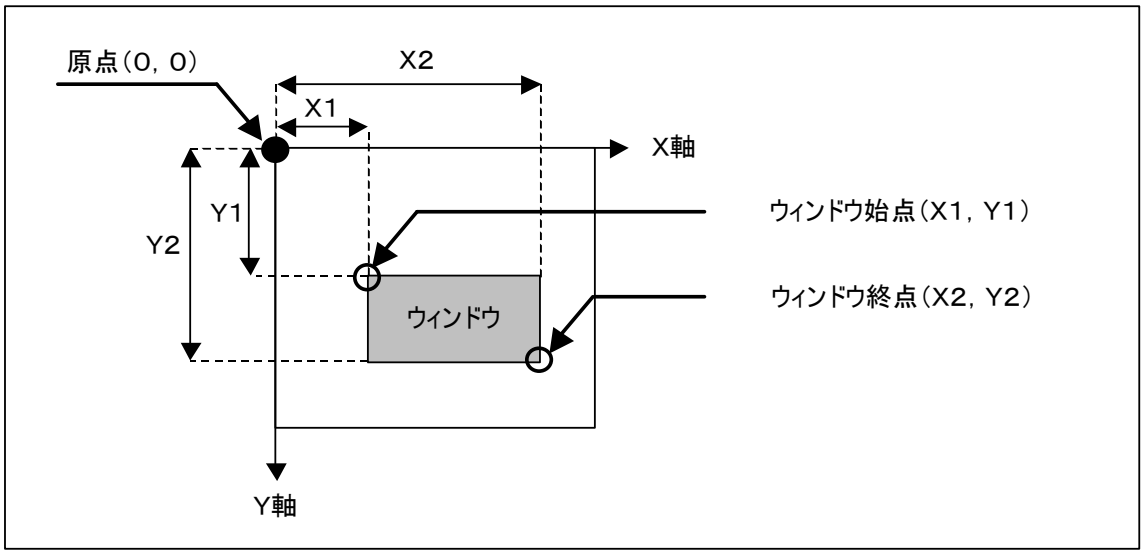


図8-11 ウィンドウの座標系

8.4.8 ウィンドウの種類

ウィンドウには、以下6つの種類定義されています。

表8-2 ウィンドウの種類

ウィンドウ種類	用　　途
ソースO画面 （SRC0_WIN）	画像処理でのソース画面に使用します
ソース1画面 （SRC1_WIN）	画像間演算処理でのソース画面に使用します （ソース画面を2画面使用するときの第2ソース画面）
ソース拡張画面 （SRC2_WIN）	画像処理でのソース拡張画面に使用します（未使用）
デスティネーション画面（DST_WIN）	画像処理でのデスティネーション画面に使用します
画像メモリアクセス（SYS_WIN）	画像メモリ制御コマンド（WriteImg, ReadImg, WritePixel,ReadPixel, LoadBMPImg, SaveBMPImg）等で有効なウィンドウ
デスティネーション拡張画面(DST_EXT_WIN)	画像処理でのデスティネーション拡張画面に使用します （未使用）

ソース画面とデスティネーション画面で異なるサイズのウィンドウを設定した場合、それぞれのウィンドウサイズ内の最小のサイズで処理を行います。

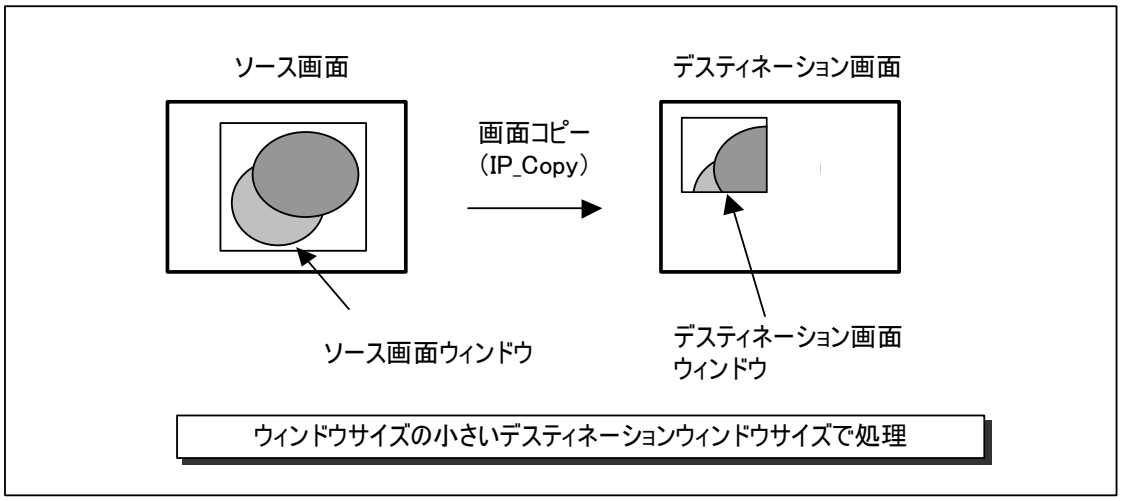


図8-12 ウィンドウ処理サイズ

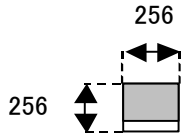
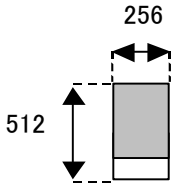
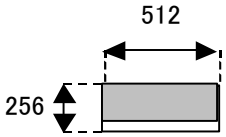
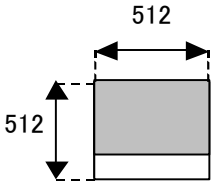
8.4.9 ウィンドウの有効／無効

画像処理は、ビデオフレームサイズ、またはウィンドウのサイズで行われます。どちらで動作するかはウィンドウの有効／無効状態で決まり、ウィンドウが無効のときはビデオフレームサイズ、有効のときは設定したウィンドウサイズとなります。この設定は、**EnableIPWindow()**および**DisableIPWindow()**コマンドによって行います。

ウィンドウ無効は、**DisableIPWindow()**コマンド、有効は**EnableIPWindow()**コマンドで行います。

ビデオフレームサイズとは、カメラから取り込む映像サイズのことであり、**SetVideoFrame()**コマンドで現在設定されている値です。表8-3に、ウィンドウ無効時のウィンドウサイズの例を示します。

表8-3 ウィンドウ無効時のウィンドウサイズ例

画面サイズ	256×256	256×512	512×256	512×512
ビデオ フレーム サイズ	256×220	256×440	512×220	512×440
画面サイズ ウィンドウ サイズ				

8.4.10 データタイプの属性

本ライブラリでは、映像入出力、画像処理等の処理データを以下のいずれかのタイプであるとみなして処理します。

表8-4 データタイプ

データタイプ		値の範囲	オーバーフロー／アンダーフロー
符号なし 8ビット	UNSIGN8_DATA	0～255	処理結果がアンダーフローの場合、0（0x00） 処理結果がオーバーフローの場合、255（0xFF）
符号付 8ビット	SIGN8_DATA	－128 ～127	処理結果がアンダーフローの場合、－128（0x80） 処理結果がオーバーフローの場合、127（0x7F）
2値	BINARY_DATA	0または 255	なし

※ デフォルト設定は、符号なし8ビット

また、データタイプの種類として、システムデータタイプと画面データタイプの2種類があります。

(1) システムデータタイプ

システムとしてのデフォルト設定のデータタイプ。カメラからの映像取り込み、画像処理結果のデスティネーション画面データタイプの設定等に使用されます。

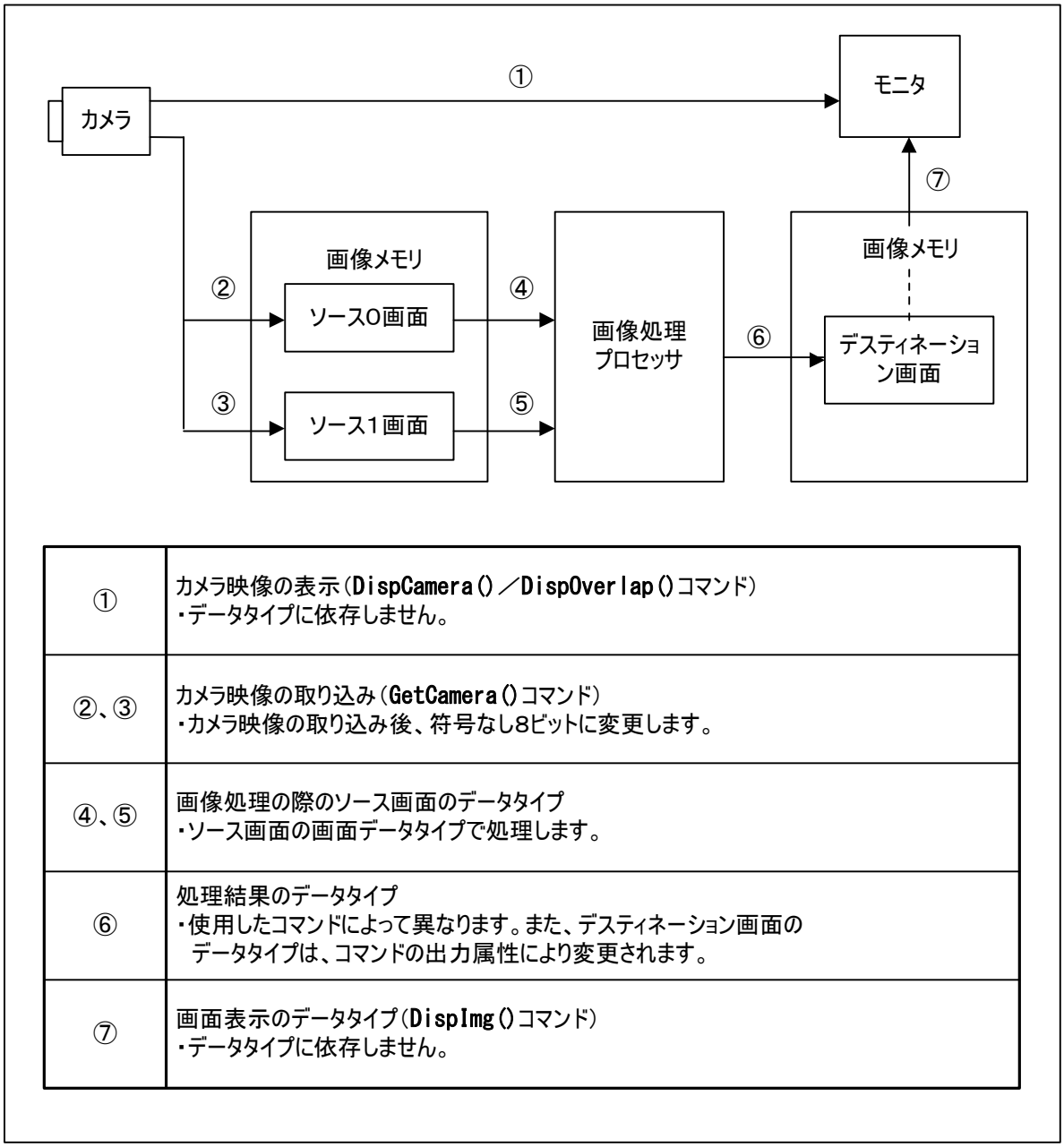
システムデータタイプは、システムイニシャライズ時に符号なし8ビットに設定されますが、**SetIPDataType()**コマンドで変更することも可能です。

(2) 画面データタイプ

画面ごとに設定されているデータタイプで、その画面のデータがどのタイプ（符号付8ビット、符号なし8ビット、2値）であるかを示します。

AllocImg()コマンドで確保された画面の画面データタイプは、符号なし8ビットに設定されます。
画面データタイプは、**ChangeImgDataType()**コマンドで変更できます。

以下に、各処理において使用されるデータタイプについて示します。
また、画像処理実行後の処理結果のデータタイプは、コマンドの出力属性により変更されます。
デスティネーション画面のデータタイプを次の表に示します。



①	カメラ映像の表示(DispCamera()／DispOverlap()コマンド) ・データタイプに依存しません。
②、③	カメラ映像の取り込み(GetCamera()コマンド) ・カメラ映像の取り込み後、符号なし8ビットに変更します。
④、⑤	画像処理の際のソース画面のデータタイプ ・ソース画面の画面データタイプで処理します。
⑥	処理結果のデータタイプ ・使用したコマンドによって異なります。また、デスティネーション画面の データタイプは、コマンドの出力属性により変更されます。
⑦	画面表示のデータタイプ(DispImg()コマンド) ・データタイプに依存しません。

図8-13 処理の流れとデータタイプ

8.5 映像入力

8.5.1 NVPでの映像入力の概要

NVPは、カメラリンクカメラの映像入力をサポートします。カメラのインタフェースはNVP-Ax430CL/ACLが2CH（BaseConfigration）、NVP-Ax435CLが4CH（BaseConfigration）、NVP-Ax435FCLが2CH（CH1：Base/Midium/FullConfigration, CH2:BaseConfigration）です。

画像処理コマンドは、以下の映像入力をサポートします。ただし、カメラによっては実現できない機能がありますので、本マニュアルの内容をご確認ください。

- ・複数カメラ同時入力
- ・キャプチャモード
- ・プリフェッチ映像入力
- ・ストロボ映像入力
- ・パーシャル映像入力
- ・マルチコンポーネント映像入力
- ・エンコーダ映像入力

RGBカラー映像入力については、「8.7 RGBカラー処理機能」を参照してください。

8.5.2 ビデオポートとカメラポートの選択

NVP-Ax430CL/Ax430ACL/Ax435FCLは、ビデオポートが2ポートあり最大で2台のカメラが接続できます。NVP-Ax435CLは、ビデオポートが4ポートあり最大で4台のカメラが接続できます。カメラによっては複数カメラの同時入力が可能です。

ビデオポートの切り替えは**ActiveVideoPort()**コマンドで行います。カメラポートは**SelectCamera()**コマンドで行いますが1ポート固定となります。

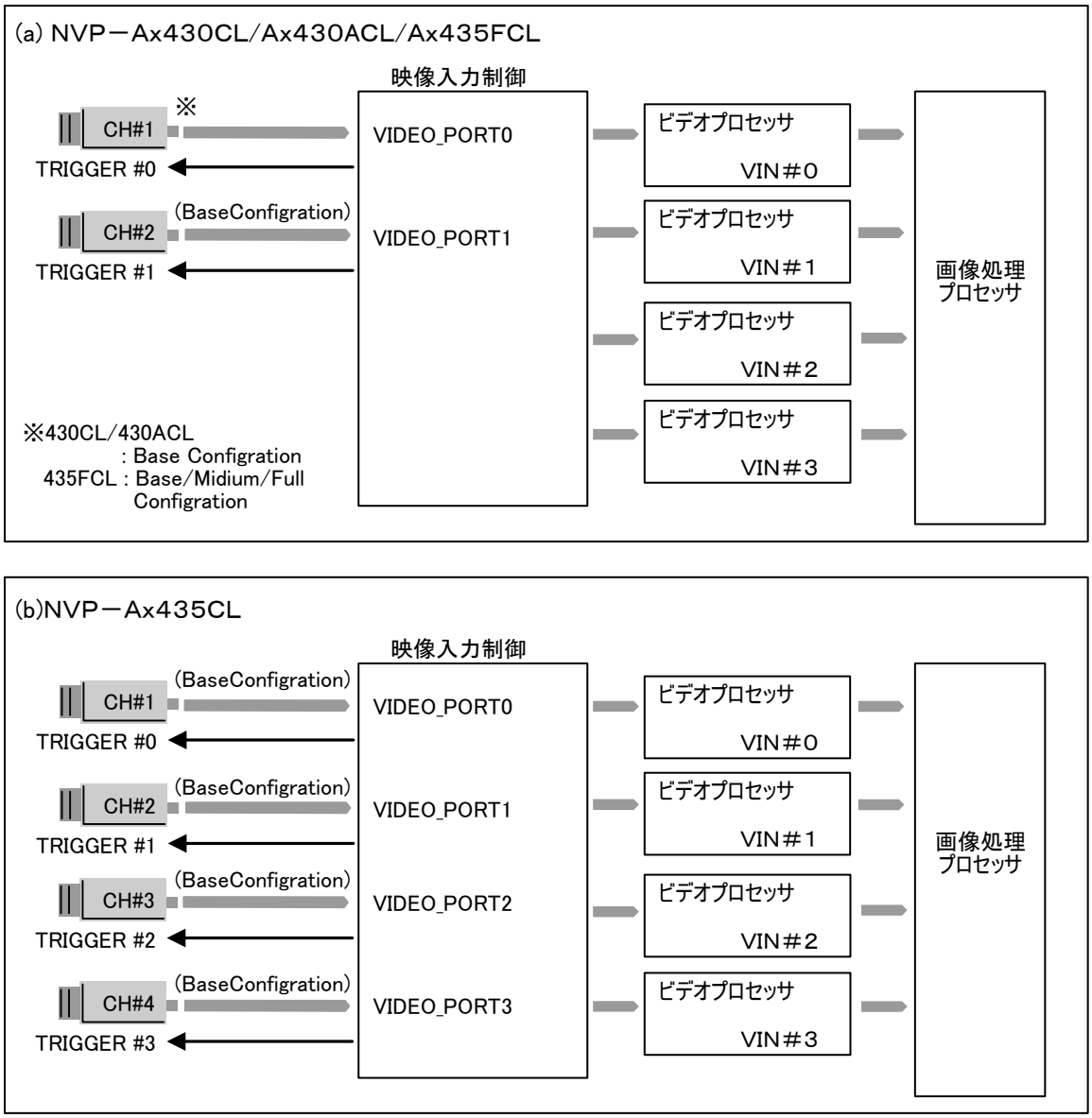


図8-14 映像入力ハードウェア構成

8.5.3 サポートカメラとカメラの選択

NVP-Ax430シリーズがサポートするカメラはカメラリンクカメラです。以下の表のいづれかの動作モード(VINモード)に分類されます。このモードによって映像入力のサイズ、同時入力の制限がありますので、カメラ選定の際にご確認ください。具体的なカメラとそのカメラ属性については、コマンドリファレンスの**SelectCamera()**の項を参照ください。

画像処理コマンドでは、**SelectCamera()**コマンドでカメラの選択と同時入力の有無、マルチコンポーネントモードの選択、**SetVideoFrame()**コマンドでビデオフレームサイズを設定します。

(1) エリアカメラ

表8-5 VINモードとカメラリンクTAP周波数対応

VIN拡張モード	VINモード	使用可能なカメラリンクTAP(周波数)
0:垂直拡張(0固定)	0:single	1TAP(85Mz以下),2TAP(50MHz以下)
	1:dual	2TAP(85Mz以下),3TAP/RGBカメラ(50MHz以下)
	2:triple	RGBカメラ(85Mz以下)
	3:quad	4TAP(85Mz以下),8TAP(50MHz以下)

※1 すべてのカメラの接続を保証するものではありません。
※2 4TAP/8TAPはNVP-Ax435FCLでのみ使用可能です。

表8-6 VINモードとビデオフレームサイズ

VIN拡張モード	VINモード	ビデオフレームサイズ 水平方向	ビデオフレームサイズ 垂直方向 ※1※2
0:垂直拡張(0固定)	0:single	有効画素数(最大4096)以下、128画素単位 ※3	有効画素数(最大4096)以下、1ライン単位
	1:dual		有効画素数(最大4096)以下、2ライン単位
	2:triple		有効画素数(最大4095)以下、3ライン単位
	3:quad	有効画素数(最大3968)以下、128画素単位	有効画素数(最大4096)以下、4ライン単位

※1 設定可能な最小値は single:2,dual:4,triple:6,quad:8となります。
※2 RGBカメラは、VINモード 0:singleと同じ制限になります
※3 RGB(50MHz以下,VINモード=1)カメラの場合水平方向は最大2048となります。

表8-7 接続可能CHと接続制限(マルチコンポーネントモードを使用しない場合)

VINモード	接続可能CH ※1	他CHの接続制限
0:single	CH1～CH4※2	なし
1:dual	CH1,CH2	CH1での使用でCH3が、CH2での使用でCH4が使用できません。
2:triple	CH1	CH3,CH4は使用できません。
3:quad	CH1 ※3	CH2,CH3,CH4は使用できません。

※1 マルチコンポーネントモードを使用しない(0:カメラ画像)場合です。
※2 CH3,CH4はNVP-Ax435CLでのみ接続可能です。
※3 NVP-Ax435FCLのCH1でのみ使用可能です。

（2）ラインカメラ

表8-8 VINモードとカメラリンクTAP周波数対応

VIN拡張モード	ＶＩＮモード	使用可能なカメラTAP（周波数）※1 ※2	使用可能カメラの有効画素最大
0：垂直拡張	0:single	2TAP(50MHz以下)	4096
	1:dual	2TAP(85Mz以下)	
	2:triple	使用できません。	
	3:quad	4TAP(85Mz以下),8TAP(50MHz以下)	
1：水平拡張	0:single	使用できません。	-
	1:dual	使用できません。	-
	2:triple	使用できません。	-
	3:quad	8TAP(50MHz以下),2TAP(85MHz以下)	16384

※1 すべてのカメラの接続を保証するものではありません。
※2 4TAP/8TAPはNVP-Ax435FCLでのみ使用可能です。

表8-9 VINモードとﾋﾞﾃﾞｵﾌﾚｰﾑｻｲｽﾞ

VIN拡張モード	ＶＩＮモード	ビデオフレームサイズ 水平方向	ビデオフレームサイズ 垂直方向 ※1 ※2
0：垂直拡張	0:single	有効画素数(最大4096)以下、128画素単位	(最大4096)以下、1ライン単位
	1:dual		(最大8192)以下、2ライン単位
	2:triple		(最大12288)以下、3ライン単位
	3:quad	有効画素数(最大3968)以下、128画素単位	(最大16384)以下、4ライン単位
1：水平拡張 ※5	3:quad	有効画素数(最大16352)以下、512画素単位 ※3,4	(最大4096)以下、1ライン単位

※1 設定可能な最小値は垂直拡張の場合、single:2,dual:4,triple:6,quad:8となります。
水平拡張の場合 2となります。
※2 RGBカメラは、ＶＩＮモード 0：singleと同じ制限になります
※3 水平拡張の場合、設定可能な値はquad:4096～16352です。
4096未満での使用は垂直拡張モードに設定してください。
※4 ビデオフレームサイズ16352未満の場合512画素単位です。
※5 水平拡張の場合はカメラ有効画素の水平方向(maxfs.Hsize)とビデオフレームサイズは同値である必要があります。
ただしビデオフレームサイズが水平16352の場合は、maxfs.Hsize=16384となります。

表8-10 接続可能CHと接続制限

ＶＩＮモード	接続可能CH	他CHの接続制限
0:single	CH1～CH4	なし
1:dual	CH1,CH2	CH1での使用でCH3が、CH2での使用でCH4が使用できません。
2:triple	CH1	CH3,CH4は使用できません。
3:quad	CH1	CH2,CH3,CH4は使用できません。

※1 マルチコンポーネントモードを使用しない（0：カメラ画像）場合です。
ラインカメラは、マルチコンポーネントモードは使用できません。
※2 CH3,CH4はNVP-Ax435CLでのみ接続可能です。
※3 NVP-Ax435FCLのCH1でのみ使用可能です。

(2) マルチコンポーネント接続制限

マルチコンポーネントモードによって接続制限があります。以下に示します。

接続制限は、以下の接続パターンA～Dの4パターンいずれかになります。

表8-11 接続可能CHと接続制限

接続パターン	接続可能CH	他CHの接続制限
A	CH1,CH2※1	なし
B	CH1,CH2	CH1での使用でCH3が、CH2での使用でCH4が使用できません。
C	CH1	CH3,CH4は使用できません。
D	CH1	CH2,CH3,CH4は使用できません。

※1 マルチコンポーネントモードを使用しない(0:カメラ画像)場合はCH3,CH4も接続可能です。

以下に各カメラにおける接続制限を示します。カメラデータ(TAP、周波数など)、マルチコンポーネントモードなどで異なります。(表記記載がないVIN拡張モードは0です)VINモード、VIN拡張モード、周波数はカメラデータに設定されています。**SetCameraData()**のVINモード詳細を参照してください。

表8-12 接続可能RAWカメラ 1TAP

VINモード	周波数	マルチコンポーネントモード	接続パターン
0:single	1:50MHz以下	0:カメラ画像(8bpp)	A
		1:カメラ画像(8bpp)＋RGBLUT変換(8bpp)	非対応(接続不可)
		2:カメラ画像(8bpp)＋RAW-RGB変換(24bpp)	
		3:RAW-RGB変換(24bpp)＋RGBLUT変換(8bpp)	
		4:RAW-RGB変換(24bpp)	
	0:85MHz以下	0:カメラ画像(8bpp)	A
		1:カメラ画像(8bpp)＋RGBLUT変換(8bpp)	B
		2:カメラ画像(8bpp)＋RAW-RGB変換(24bpp)	D
		3:RAW-RGB変換(24bpp)＋RGBLUT変換(8bpp)	D
		4:RAW-RGB変換(24bpp)	C

表8-13 RAWカメラ 2TAP

VINモード	周波数	マルチコンポーネントモード	接続パターン
0:single	1:50MHz以下	0:カメラ画像(8bpp)	A
		1:カメラ画像(8bpp)＋RGBLUT変換(8bpp)	B
		2:カメラ画像(8bpp)＋RAW-RGB変換(24bpp)	D
		3:RAW-RGB変換(24bpp)＋RGBLUT変換(8bpp)	D
		4:RAW-RGB変換(24bpp)	C
1:dual	0:85MHz以下	0:カメラ画像(8bpp)	B
		1:カメラ画像(8bpp)＋RGBLUT変換(8bpp)	非対応(接続不可)
		2:カメラ画像(8bpp)＋RAW-RGB変換(24bpp)	
		3:RAW-RGB変換(24bpp)＋RGBLUT変換(8bpp)	
		4:RAW-RGB変換(24bpp)	

表8-14 RAWカメラ 3TAP

VIN モード	周波数	マルチコンポーネントモード	接続 パターン
1:dual	1:50MHz以下	0:カメラ画像(8bpp)	B
		1:カメラ画像(8bpp)＋RGBLUT変換(8bpp)	非対応 (接続不可)
		2:カメラ画像(8bpp)＋RAW-RGB変換(24bpp)	
		3:RAW-RGB変換(24bpp)＋RGBLUT変換(8bpp)	
		4:RAW-RGB変換(24bpp)	

表8-15 RAWカメラ 4TAP

VIN モード	周波数	マルチコンポーネントモード	接続 パターン
3:quad	0:85MHz以下	0:カメラ画像(8bpp)	D
		1:カメラ画像(8bpp)＋RGBLUT変換(8bpp)	非対応 (接続不可)
		2:カメラ画像(8bpp)＋RAW-RGB変換(24bpp)	
		3:RAW-RGB変換(24bpp)＋RGBLUT変換(8bpp)	
		4:RAW-RGB変換(24bpp)	

表8-16 RAWカメラ 8TAP

VIN モード	周波数	マルチコンポーネントモード	接続 パターン
3:quad	1:50MHz以下	0:カメラ画像(8bpp)	D
		1:カメラ画像(8bpp)＋RGBLUT変換(8bpp)	非対応 (接続不可)
		2:カメラ画像(8bpp)＋RAW-RGB変換(24bpp)	
		3:RAW-RGB変換(24bpp)＋RGBLUT変換(8bpp)	
		4:RAW-RGB変換(24bpp)	

表8-17 RGBカメラ

VIN モード	周波数	マルチコンポーネントモード	接続 パターン
1:dual	1:50MHz以下	0:カメラ画像(24bpp)	B
		1:カメラ画像(24bpp)＋RGBLUT変換(8bpp)	B
		2:カメラ画像(8bpp)＋RAW-RGB変換(24bpp)	非対応 (接続不可)
		3:RAW-RGB変換(24bpp)＋RGBLUT変換(8bpp)	
		4:RAW-RGB変換(24bpp)	
2:triple	0:85MHz以下	0:カメラ画像(24bpp)	C
		1:カメラ画像(24bpp)＋RGBLUT変換(8bpp)	D
		2:カメラ画像(8bpp)＋RAW-RGB変換(24bpp)	非対応 (接続不可)
		3:RAW-RGB変換(24bpp)＋RGBLUT変換(8bpp)	
		4:RAW-RGB変換(24bpp)	

8.5.4 キャプチャモード

キャプチャモードは、単一キャプチャモード及び連続キャプチャモードをサポートします。
走査方式はノンインタレースモードです。
また、画像処理コマンドではプリフェッチ映像入力をサポートしますが、プリフェッチ映像入力は連続キャプチャモードで動作し、プリフェッチ映像入力以外は単一キャプチャモードで動作します。
プリフェッチ映像入力については「8.5.6 プリフェッチ映像入力」を参照してください。

表8-18 カメラの走査方式とキャプチャモード

No.	走査方式	キャプチャモード		備考
		単一	連続	
1	ノンインタレースモード	○	○	

(1) 単一キャプチャモード

単一キャプチャモードは、1フィールドまたは1フレームのみCPUバスが使用され、それ以外の場合はCPUバスが使用されません。バスの占有率を考えると効率が良いといえます。
単一キャプチャモードの場合、**GetCamera()**コマンドで映像入力を行います。

(2) 連続キャプチャモード

連続キャプチャモードは、キャプチャを開始すると停止するまでCPUバスが使用され、アプリケーションで映像入力の必要がない期間もキャプチャを行っているために、バスの占有率を考えると効率が良くないといえます。
連続キャプチャモードでは、常に映像が入力され、入力データをバッファリングすることができるので、取込開始をかけたフレームまたはフィールドのデータを取込データにすることができ、アプリケーションの処理時間を単一キャプチャモードに比べて短縮できます。
画像処理コマンドでは、**CaptureContinuous()**で連続キャプチャを有効にし、**StopCaptureContinuous()**で連続キャプチャを停止します。
連続キャプチャモードの場合、**GetCamera()**、**GetCameraReqScene()**、**GetCameraResScene()**コマンドで映像入力を行います。

8.5.5 ビデオフレームサイズ

ビデオフレームサイズはカメラ映像を入力する際の解像度のことで、デフォルトサイズは５１２×４８０に設定されています。
ビデオフレームサイズは**SetVideoFrame()**コマンドで設定します。使用するカメラの取得可能なサイズを超えない様に設定してください。
また、カメラ映像を入力する画像メモリの大きさは、そのビデオフレームサイズ以上の大きさである必要があります。ビデオフレームサイズに合わせて画像メモリを確保して下さい。

8.5.6 プリフェッチ映像入力

通常の画像処理アプリケーションでは、映像入力を行い、その入力した映像に対し画像処理を実行するため、映像入力と画像処理を順番に行っています。ビデオレートで画像処理アプリケーションを実行させるには、映像入力と画像処理をパイプラインで実行する必要があります。

プリフェッチ映像入力は、画像先行取得を行うことであり、映像入力と画像処理をパイプライン処理し、全体的な処理速度の向上を図ります。

以下にプリフェッチ映像入力による画像処理の例を示します。

(a) プリフェッチ映像入力(画像先行取得)をしない場合

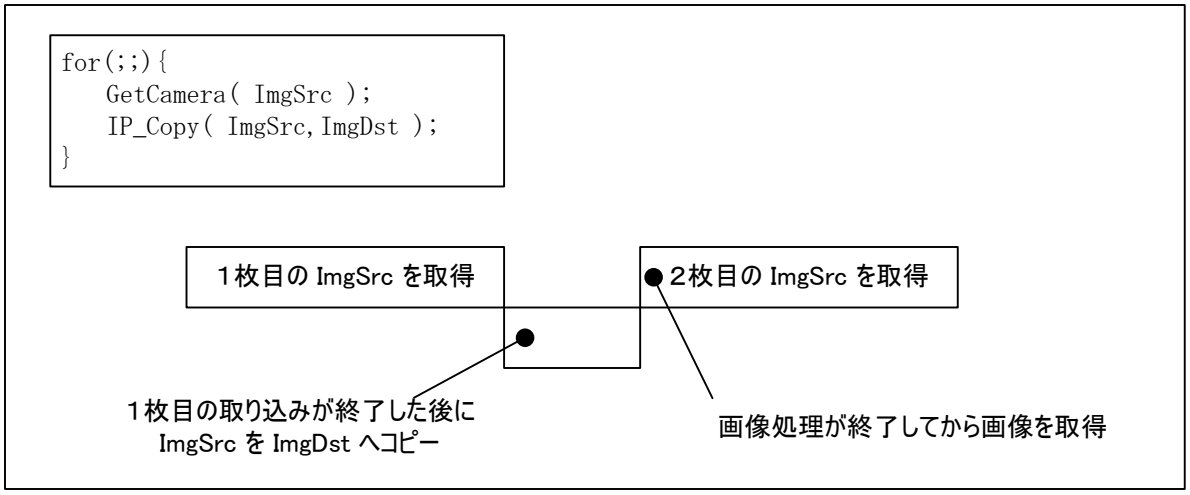


図8-15 通常の画像処理アプリケーション例

(b)プリフェッチ映像入力(画像先行取得)をした場合

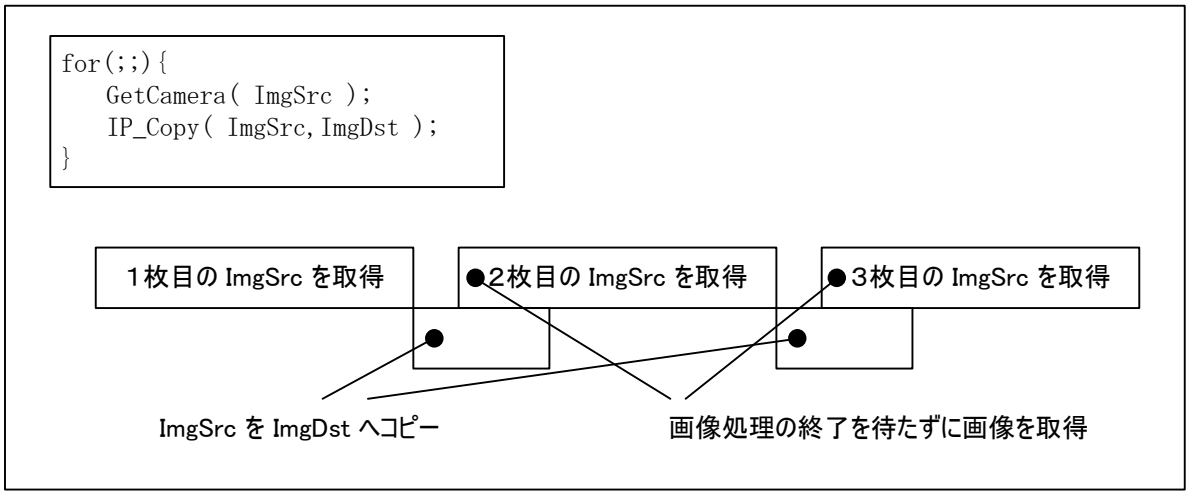


図8-16 プリフェッチ映像入力を行う画像処理アプリケーション例

（1）画像処理コマンドでの連続映像入力

連続映像入力モードでは、常に映像のシーン番号を管理しています。そのため映像のシーン番号を指定して入力したり、入力した映像のシーン番号を取得することができます。画像処理コマンドでは、以下のコマンドで連続映像入力を行います。

表8-19 連続映像入力コマンド

No.	コマンド名	機能
1	GetCamera	最新のフレームを入力します。コマンド実行時、映像入力が終了している最新のフレームを入力します。
2	GetCameraReqScene	指定されたシーン番号のフレームを入力します。コマンド実行時、指定されたシーン番号のフレームの映像入力が終了していない場合は、指定されたシーン番号のフレームの映像入力が終了するまでウェイトします。
3	GetCameraResScene	最新のフレームを入力し、そのフレームのシーン番号を返します。コマンド実行時、映像入力が終了している最新のフレームを入力します。

映像のシーン番号は内部カウンタで管理していますが、そのカウンタ値は、シーン番号カウンタ取得コマンド **GetSceneCounter()**、シーン番号カウンタ設定コマンド**SetSceneCounter()**で任意に設定 し、利用することができます。

連続映像入力での映像フレームの画像メモリへの転送とそれに対する画像処理可能フレームのタイムチャートを下図に示します。連続映像入力では、常に3画面の画像メモリバッファを切り替えながら映像入力しているため、常に映像転送中の画面の前2画面の映像を保持しています。映像フレームがシーン番号nのタイミングで映像入力コマンド **GetCameraResScene()**を実行すると取得シーン番号は（n－1）を返します。このとき、フレーム番号指定の映像入力 **GetCameraReqScene()**は（n－2）～32bitの範囲で有効になります。

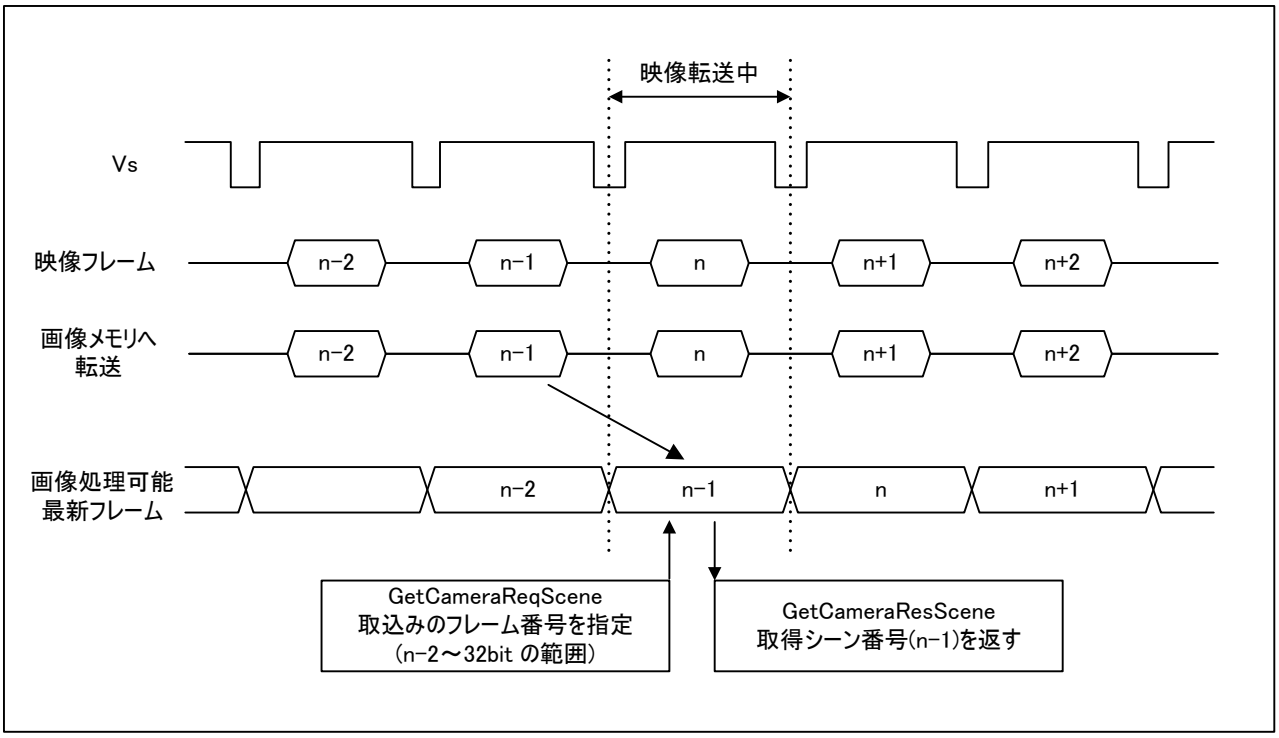


図8-17 連続映像入力のタイムチャート

映像入力は、映像の取りこぼしや取り込みタイミングのずれ等がないことをお客様のシステムで十分ご評価の上ご使用ください。

（2）画像処理コマンドでの連続映像入力方法

画像処理コマンドでは、CaptureContinuous()コマンドで連続映像入力モードの設定と起動を行ってから、StopCaptureContinuous()コマンドで連続映像入力を終了するまで、映像入力コマンドでの映像入力はすべてプリフェッチ映像入力が行われます。以下にフローを示します。

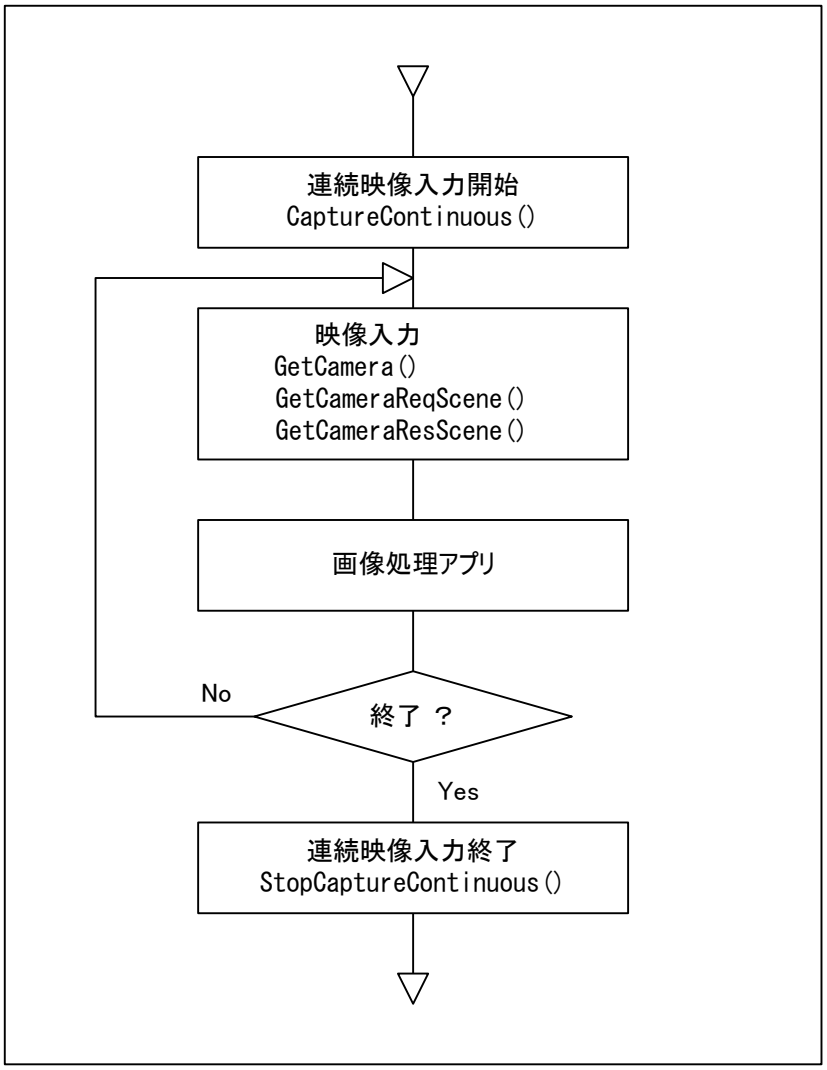


図8-18 プリフェッチ映像入力フロー

8.5.7 カメラ映像の入力方法

(1) 基本のカメラ映像入力

基本のカメラ映像入力は以下の順序で実行します。**ActiveVideoPort()**コマンド、**SelectCamera()**コマンド、**SetVideoFrame()**コマンドの実行順序が異なるとエラーになります。

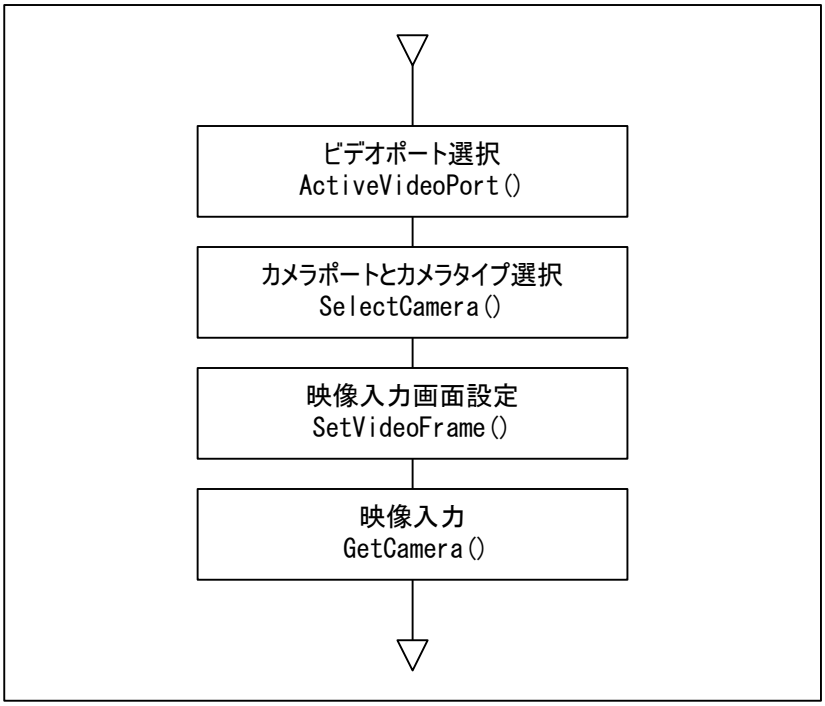


図8-19 基本の映像入力フロー

(2) フレームシャッタを使用するカメラ映像入力

フレームシャッタを使用する場合のカメラ映像入力は、以下の順序で実行します。基本の映像入力の順序でコマンドを実行し、**SetTriggerMode()**コマンドでトリガモードの設定とシャッタスピードの設定を行います。

トリガモードを解除する場合には、**SetTriggerMode()**コマンドでノーマルモードを設定するか、再度**SelectCamera()**コマンドと**SetVideoFrame()**コマンドを実行します。

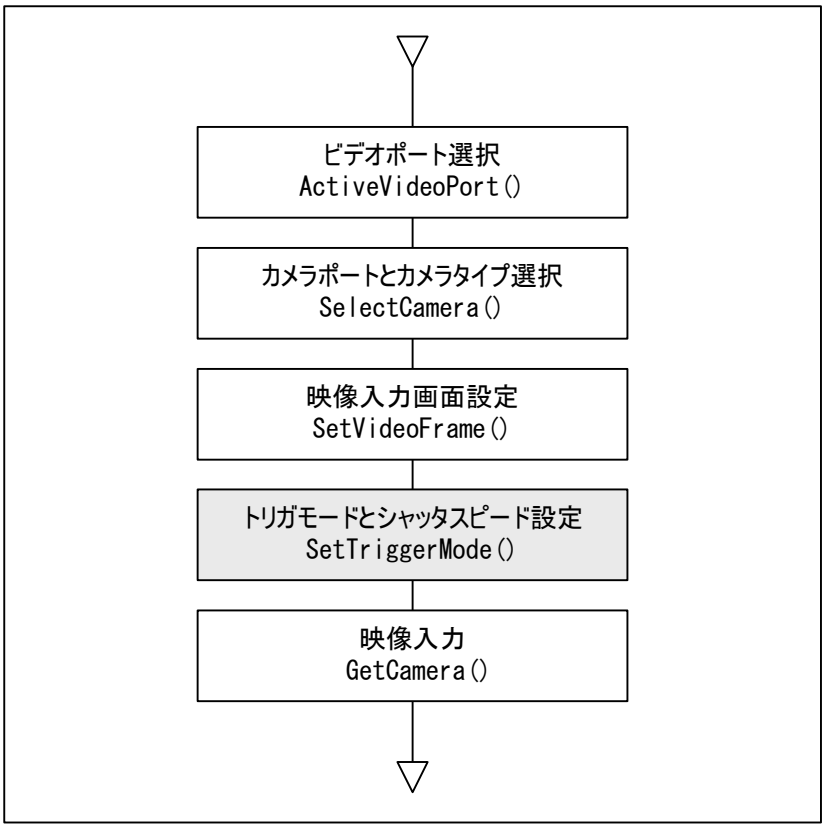


図8-20 フレームシャッタを使用する場合の映像入力フロー

(3) 複数カメラ同時入力

カメラの同時入力は、以下の順序で実行します。フレームシャッタを使用する場合と同じですが、同時入力情報を**SelectCamera()**コマンドで設定します。

カメラの同時入力を解除する場合には、1カメラ入力に設定して**SelectCamera()**コマンドと**SetVideoFrame()**コマンドを実行します。

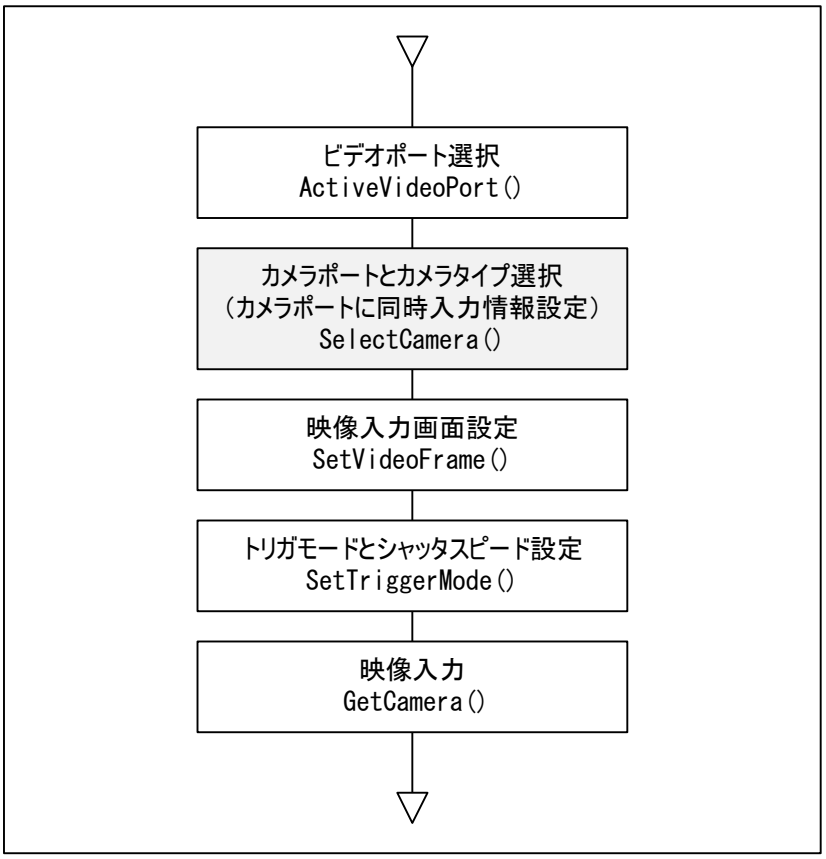


図8-21 カメラ同時入力時の映像入力フロー

(4) カメラ映像入力位置の調整

カメラ映像入力を行った際の映像は、画面中央に入力するようにカメラタイプ毎にデフォルト値が設定されています。この映像入力位置は、現位置からのオフセットを設定することで微調整することができます。

映像入力位置の微調整は、**SetVFDelay()**コマンドで水平／垂直方向の遅延サイズを設定します。画面中央の位置に戻す場合は、水平／垂直方向の遅延サイズを0に設定するか、再度、**SetVideoFrame()**コマンドを実行します。

また、フレームシャッタを使用する場合は、**SetTriggerMode()**コマンドを実行してから**SetVFDelay()**コマンドを実行してください。

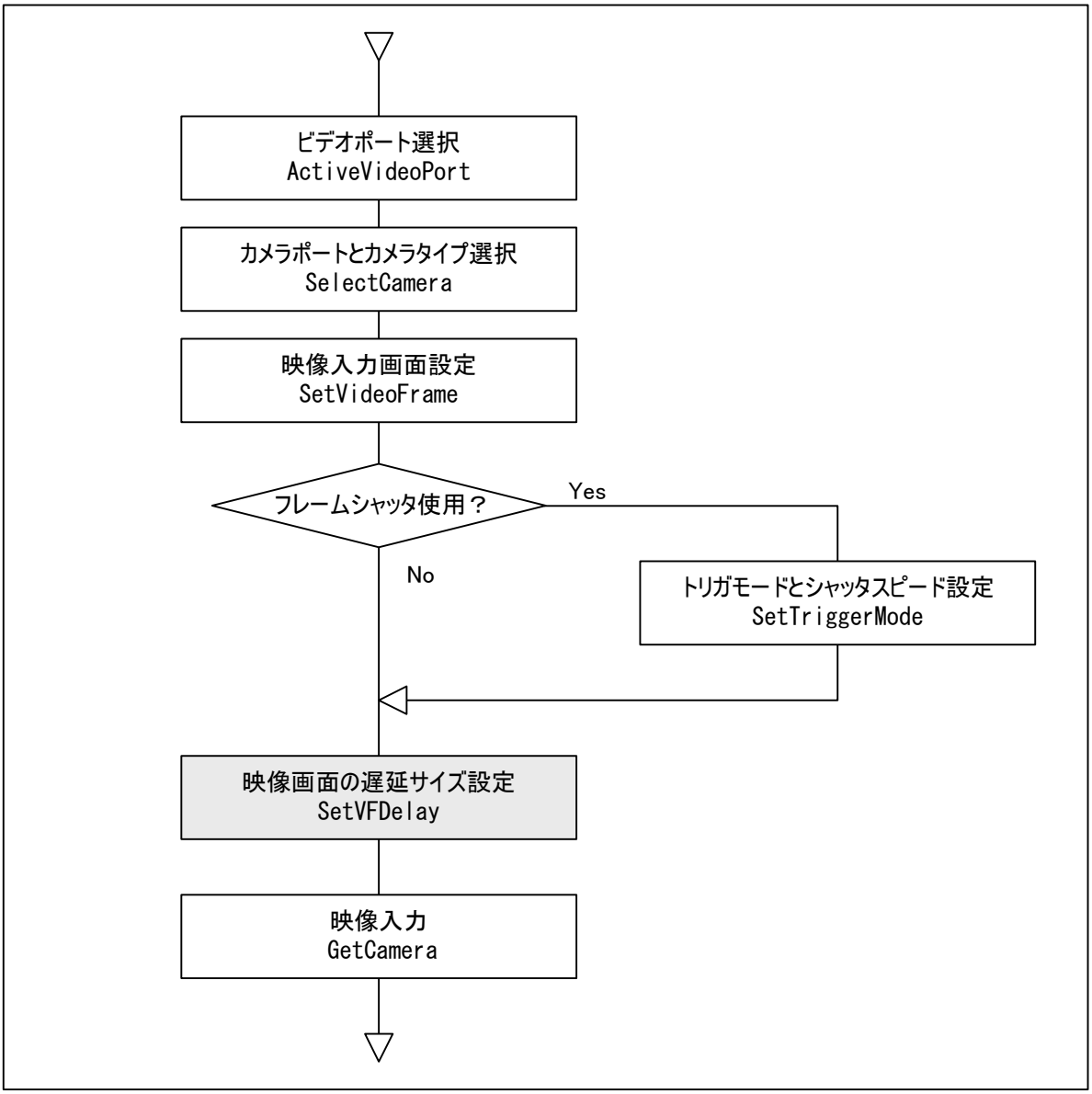


図8-22 カメラ映像入力位置調整フロー

(5) ストロボ映像入力

NVP－Ax430シリーズは、独立に4チャネルのストロボを設定することができます（但しNVP－Ax430CLは2チャンネル）。以下にトリガとストロボ出力のタイミングを示した図とストロボ情報テーブルを示します。ストロボは指定したビデオポートのトリガ出力からDelay時間後に、Lengthの長さで出力されます。ストロボ出力ディレイ、ストロボ出力長、ストロボ極性の設定値は使用するストロボにより異なります。

ストロボ映像入力は、トリガモードの映像入力設定に加えて**SetStrobeMode()**コマンドでストロボ情報を設定します。**SetStrobeMode()**コマンドの実行タイミングは特に制限されません。

ストロボモードを解除する場合には、**SetStrobeMode()**コマンドでストロボ無効に設定してください。このとき、ストロボ情報テーブルの設定値はストロボ有効に設定した値をそのまま設定してください。設定値が異なるとストロボモードを解除できないことがあります。

```
/* ストロボ情報テーブル */
typedef struct {
    int  VideoPort;    /* ビデオポート番号                          */
    int      Delay;    /* 出力ディレイ                              */
    int     Length;    /* ストロボ出力長                            */
    int       Pole;    /* ストロボ極性          (0:正極性,1:負極性) */
} STRB_INFO;
```

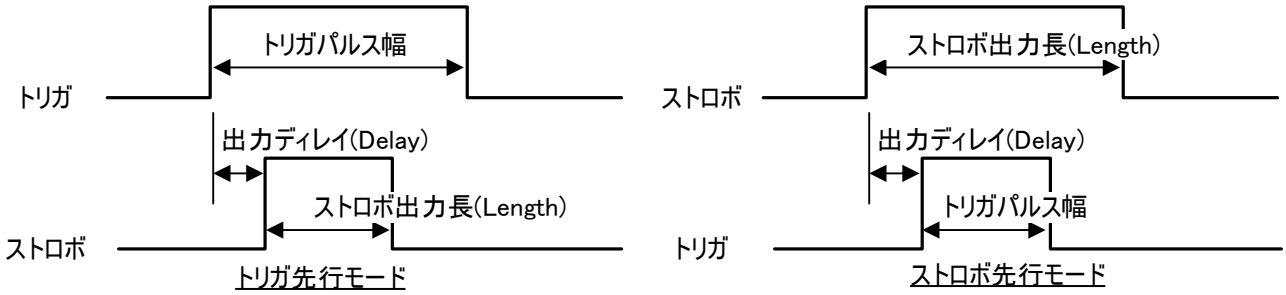


図8-23 トリガおよびストロボ出力タイミング

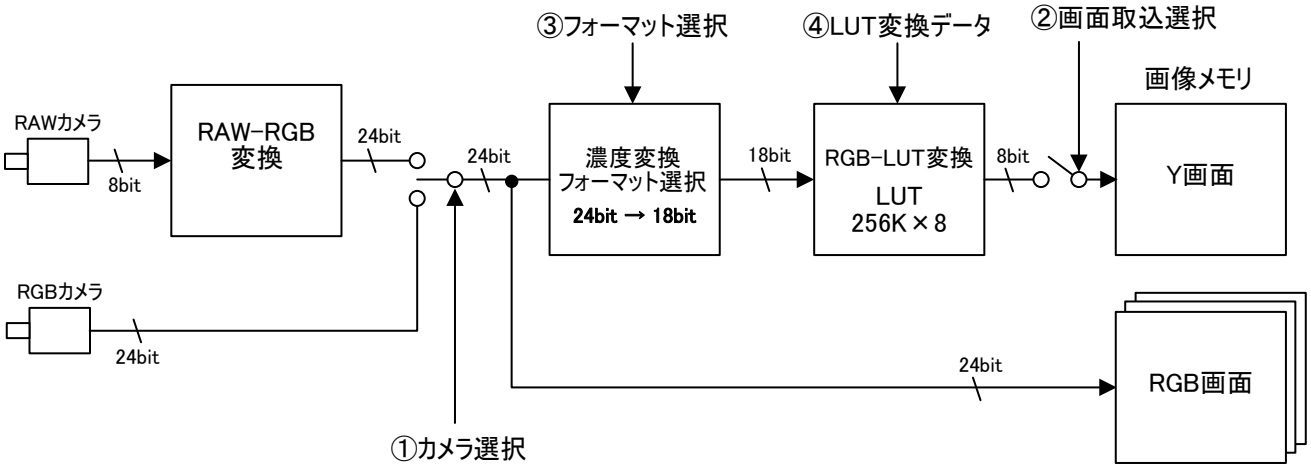
図8-24 ストロボを使用する場合の映像入力フロー

(6) パーシャル映像入力

パーシャル映像入力可能なカメラに対して、パーシャル映像入力をサポートします。
パーシャル映像入力を行う場合、カメラの有効サイズが変わるため、カメラデータ(8.5.8 カメラデータの設定)の有効画素をパーシャルモードに合わせて変更する必要があります。
パーシャルモードや設定するパーシャル情報はカメラにより異なります。使用するカメラの取扱説明書を確認の上、パーシャル映像入力を行ってください。

(7) マルチコンポーネント映像入力

RAWカメラ、RGBカメラに対してRAW-RGB変換、RGB-LUT変換を行う以下のハードウェアを搭載し、RAW-RGB変換、RGB-LUT変換を行いながら映像入力が可能です。カメラ毎に対応可能な映像入力機能の一覧を表8-17に示します。



①カメラ選択、②画面取込選択は、SelectCamera() コマンドで設定
③フォーマット選択、④LUT変換データ設定は、SetVideoLUTMode()で設定

図8-25 マルチコンポーネント映像入力ハードウェア構成

表8-20 マルチコンポーネント映像入力機能一覧

カメラ	変換処理		入力映像の画像メモリ画面タイプ	備考
	RAW-RGB変換	RGB-LUT変換		
RAWカメラ	有効	有効	コンポーネント（2CH）	RAWデータと RAW-RGB変換 ＋ RGBLUT変換
	有効	無効	コンポーネントRGB	RAW-RGB変換のみ
	有効	有効	コンポーネントRGBY	RAW-RGB変換 ＋ RGBLUT変換
	有効	有効	コンポーネントRGBY	RAWデータと RAW-RGB変換 ＋ RGBLUT変換
RGBカメラ	—	有効	コンポーネントRGBY	RGBLUT変換のみ

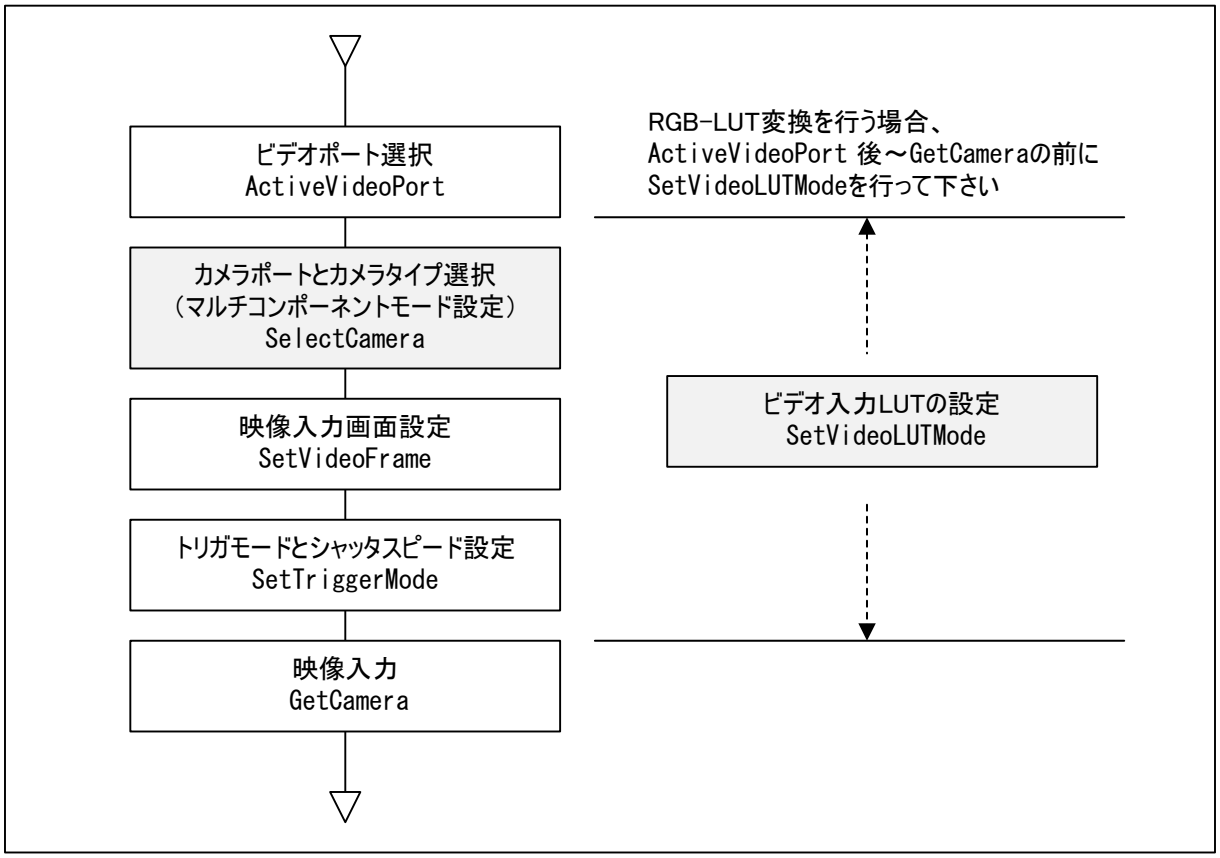


図8-26 マルチコンポーネント映像入力フロー

(8) エンコーダー映像入力

エンコーダーに同期した映像入力が可能です。エンコーダーパルスをカウントし、カメラに対しトリガを発行します。
SetEncoderMode()コマンド でエンコーダーのモードを設定します。エンコーダーからのトリガ出力は、**SetTriggerMode()**コマンドで指定します。

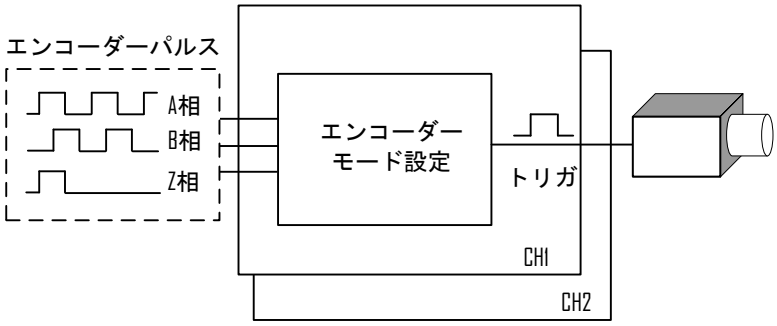


図8-27 エンコーダー映像入力

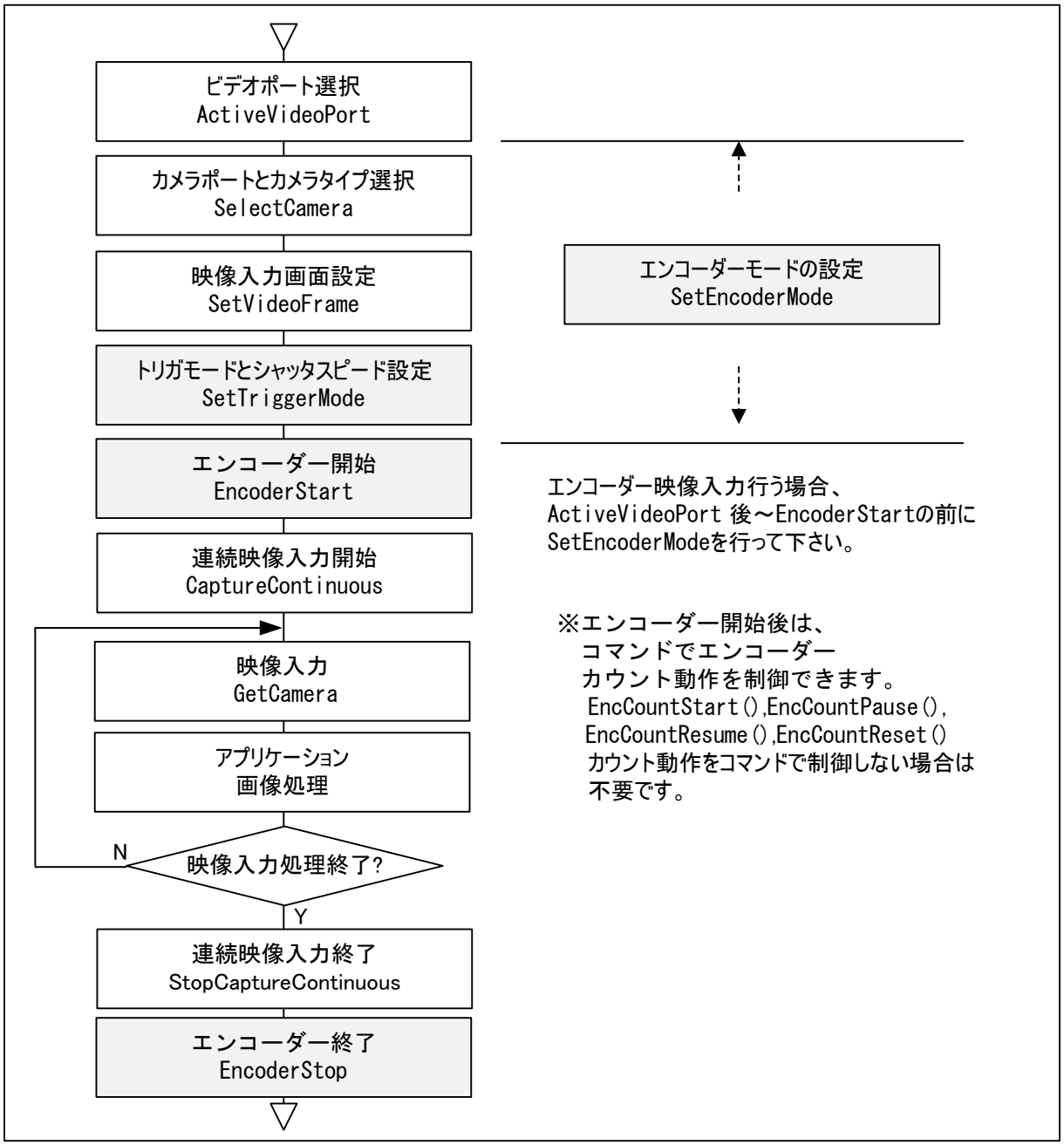


図8-28 エンコーダー映像入力フロー

8.5.8 カメラデータの設定

NVP-Ax430SDKは多種のカメラの映像入力をサポートします。そのため映像入力に必要なデータをデフォルトデータとして管理していますが、映像入力のアプリケーションによって、デフォルト値を変更したい場合があります。また、サポートカメラ以外のカメラで映像入力を行いたい場合があります。

画像処理コマンドは、ユーザカメラとして新規カメラを１２８台まで登録可能です。システムに設定されているカメラデータを取得するGetCameraData()コマンド、カメラデータを設定するSetCameraData()コマンド、および、テキストファイルに記述されたカメラデータをシステムに設定するLoadCameraFile()コマンドで、カメラデータの変更や新規カメラデータを設定することができます。

但し、カメラデータとして不適切な値を設定した場合には、故障・破損の原因になります。カメラデータおよびカメラデータファイルについては、弊社が提供するもの以外を設定しないでください。

8.6 映像出力

8.6.1 NVPでの映像出力の概要

NVPは、HDMIでの映像出力をサポートします。
画像処理コマンドでサポートする映像表示は、カメラ映像の表示(ライブ表示)、画像メモリの表示、画像メモリとカメラ映像の重ね合わせ表示(オーバレイ表示)です。
また、オーバレイのカラー表示やの画像メモリの擬似カラー表示をサポートします。

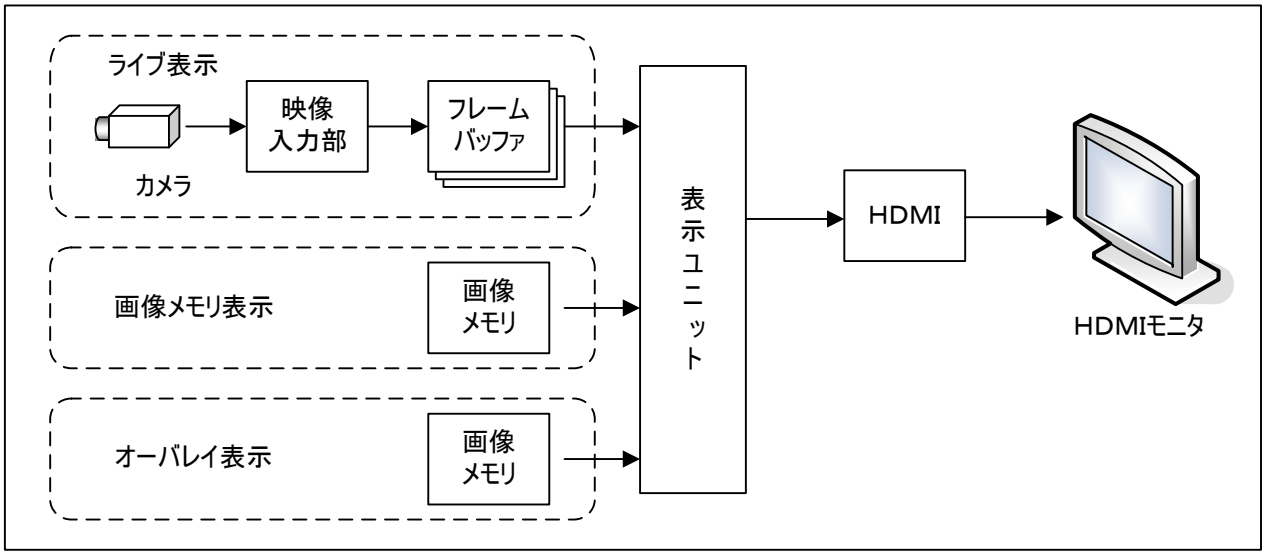


図8-29 映像表示機能概要

(1) 映像表示プレーン

NVPには、映像表示プレーンと呼ぶ表示用画面が8面あります。
各プレーンは重ね合わせて表示することが可能で、この機能を用いてオーバレイ表示を行います。映像表示プレーンは1～7(DISP_PLANE_1～DISP_PLANE_7)までの番号で表し、以下のようにプレーン番号が小さいほど表示の優先順位が高くなります。画像メモリ表示やオーバレイ表示のプレーン画面は、表示画面の構成制御コマンドで設定します。

DISP_PLANE_1 ＞ DISP_PLANE_2 ＞ （中略） ＞ DISP_PLANE_6 ＞ DISP_PLANE_7

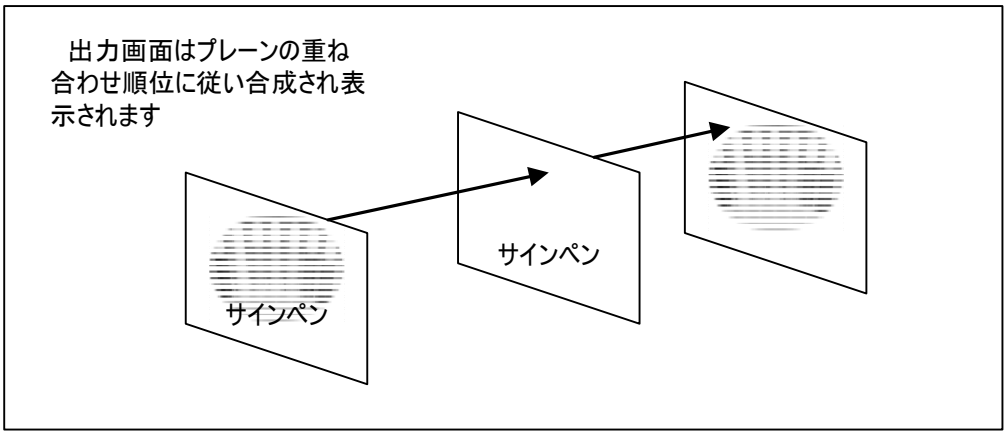


図8-30 プレーンの重ね合わせ

(2) カラーパレット

NVPには、26万色中、同時に256色が表示可能なカラーパレットが4面あり、オーバレイのカラー表示、カメラ映像や画像メモリの擬似カラー表示が可能です。カラーパレットは、1～4(DISP_PALLET_1～DISP_PALLET_4)までの番号で表し、表示画面の構成制御コマンドで使用するパレット番号を設定します。

(3) 映像表示機能

画像処理コマンドが制御する映像表示機能一覧を以下に示します。

表8-21 映像表示機能一覧

機能名	コマンド名	表示画面	
ライブ表示	DispCamera()	Y ※2	モノクロカメラ映像表示 8bit/pixelモード
		Y ※2	モノクロカメラ擬似カラー映像表示 ※1 8bit/pixelモード
画像メモリ表示	DispImg()	Y	モノクロ画像メモリ表示 8bit/pixelモード
		Y	モノクロ画像メモリ擬似カラー映像表示 8bit/pixelモード
		RGB16 ※3	カラー画像メモリ表示 16bit/pixelモード
オーバレイ表示	DispOverlap()	Y	モノクロ画像メモリ擬似カラー映像表示 8bit/pixelモード
		RGB16	カラー画像メモリ表示 16bit/pixelモード

※1 SelectDisp()コマンドで設定
※2 ライブ表示の場合は、内部確保画面タイプ
※3 コンポーネントRGB画面はRGB16画面に変換して表示

8.6.2 表示画面の構成制御設定

画像処理コマンドでサポートするライブ表示、画像メモリ表示、オーバレイ表示は、表示画面の構成制御設定 **SetConfigDisp()**コマンドで設定された情報で行います。以下に表示画面の構成制御パラメータを示します。

(1) 表示フレームサイズ

基本のカメラ映像入力は以下の順序で実行します。**ActiveVideoPort()**コマンド、**SelectCamera()**コマンド、**SetVideoFrame()**コマンドの実行順序が異なるとエラーになります。

表8-22 表示フレームサイズ

設定内容	動作	設定範囲	備考
xsize	X方向の表示サイズ	1～4095　※	
ysize	Y方向の表示サイズ	1～2047　※	

(2) 表示位置

表8-23 表示位置

設定内容	動作	設定範囲	備考
dpx	X方向の表示開始位置	1～4095　※	
dpy	Y方向の表示開始位置	1～2047　※	

※ 対象システムが実際に表示可能な範囲内で設定してください。

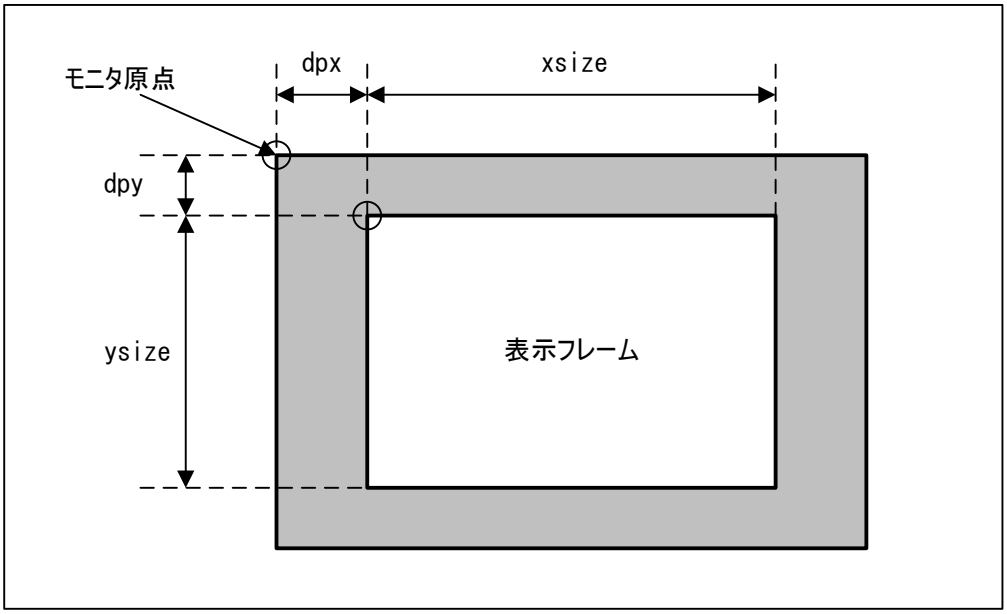


図8-31 表示画面の構成制御パラメータ

(3) 画像表示プレーン番号

ライブ表示／画像メモリ表示に使用するプレーン番号を指定します。

(4) オーバレイ表示プレーン番号

オーバレイ表示に使用するプレーン番号を指定します。

(5) オーバレイ表示パレット番号

オーバレイ表示に使用するカラーパレット番号を指定します。

8.6.3 表示フレームサイズ

画像処理コマンドでサポートする映像入力フレームサイズ、表示フレームサイズ、画面サイズの組合せ例を以下に示します。
映像入力フレームサイズは**SetVideoFrame()**コマンド、表示サイズは**SetConfigDisp()**コマンドで指定します。画面サイズはAllocImgコマンドで領域確保した画像メモリのサイズです。
なお、HDMIモニタへの表示できる最大表示サイズは、NVP-Linuxの「/boot/config.txt」で設定した値になります。

表8-24 映像入力フレームサイズと表示サイズ例

No.	映像入力 フレームサイズ	画面サイズ	表示サイズ	備考
1	320H×240V	320H×240V	320(X)×240(Y)	
2	512H×480V	512H×480V	512(X)×480(Y)	
3	640H×480V	640H×480V	640(X)×480(Y)	
4	1024H×768V	1024H×768V	1024(X)×768(Y)	video=1024x768@60　設定
5	1280H×960V	1280H×960V	1280(X)×720(Y)	video=1280x720@60　設定

8.7 RGBカラー処理機能

RGBカラー処理機能には、以下の３点の機能があります。

- コンポーネントRGBカラーカメラからの映像取り込み
- カラー画像のLUT変換
- カラー画像表示

8.7.1 コンポーネントＲＧＢ信号について

コンポーネントRGBカラー信号は、R（赤）、G（緑）、B（青）の信号で構成され、RGBカラーカメラからは別々の信号線で送られます。以下に主なカラーバーに対するRGB値を示します。

表8-25 カラーバーに対するRGB値

	R	G	B		R	G	B
White	255	255	255	Blue	0	0	255
Black	0	0	0	Yellow	255	255	0
Red	255	0	0	Cyan	0	255	255
Green	0	255	0	Magenta	255	0	255

8.7.2 ＲＧＢカラー時の画像メモリ使用方法

RGBカラー映像データは、R(赤)、G(緑)、B(青)の3つのデータで構成されるため、画像メモリは3画面分を使用します。

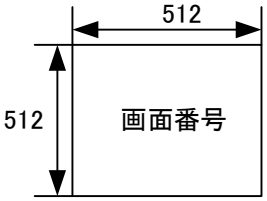
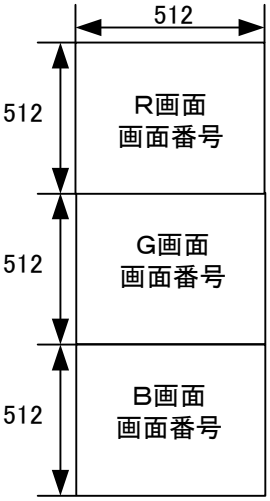
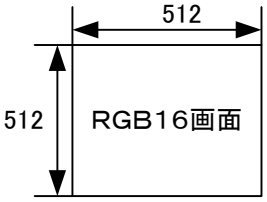
RGBカラー用の画像メモリを確保するコマンドが**AllocRGBImg()**です。**AllocRGBImg()**では、カラー映像用にR画面とG画面とB画面が連続する3画面を確保します。ユーザにはこのうちR画面の画面番号を返し、G、B用の画面はシステム内部で管理されます。

RGB画面は、RGBカラー映像入力、画像処理に使用できますが、映像表示に直接使用できません。映像表示用には、RGB16画面(16bit/pixel)を使用します。

RGB16画面を確保するコマンドが**AllocRGB16Img()**です。RGB16画面は、映像入力、画像処理には使用できません。

また、コンポーネントRGBY画面をRGBカラー映像入力、画像処理に使用することができます。コンポーネントRGBY画面については、「8.4.2 画像メモリの画面タイプ」を参照してください。

表8-26 画面の種類と確保メモリ

画面の種類	コマンド	確保例
モノクロ画面	AllocImg()	(例)AllocImg (IMG_FS_512H_512V)  512 ×512 の画面を 1画面確保
RGB カラー画面	AllocRGBImg()	(例)AllocRGBImg (IMG_FS_512H_512V)  512 ×512 の画面を 3画面連続した領域に 確保 R画面の画面番号を ユーザに返す
RGB16 カラー画面	AllocRGB16Img()	 16bit/Pixelの 512 ×512 の画面を 1画面確保

8.7.3 コンポーネントＲＧＢカメラからの映像取り込み

コンポーネントRGBカラー映像では、R信号、G信号、B信号それぞれ別々の信号として伝送されます。R信号、G信号、B信号はディジタル化され、画像メモリ上に連続したR画面/G画面/B画面の3画面として取り込まれます。

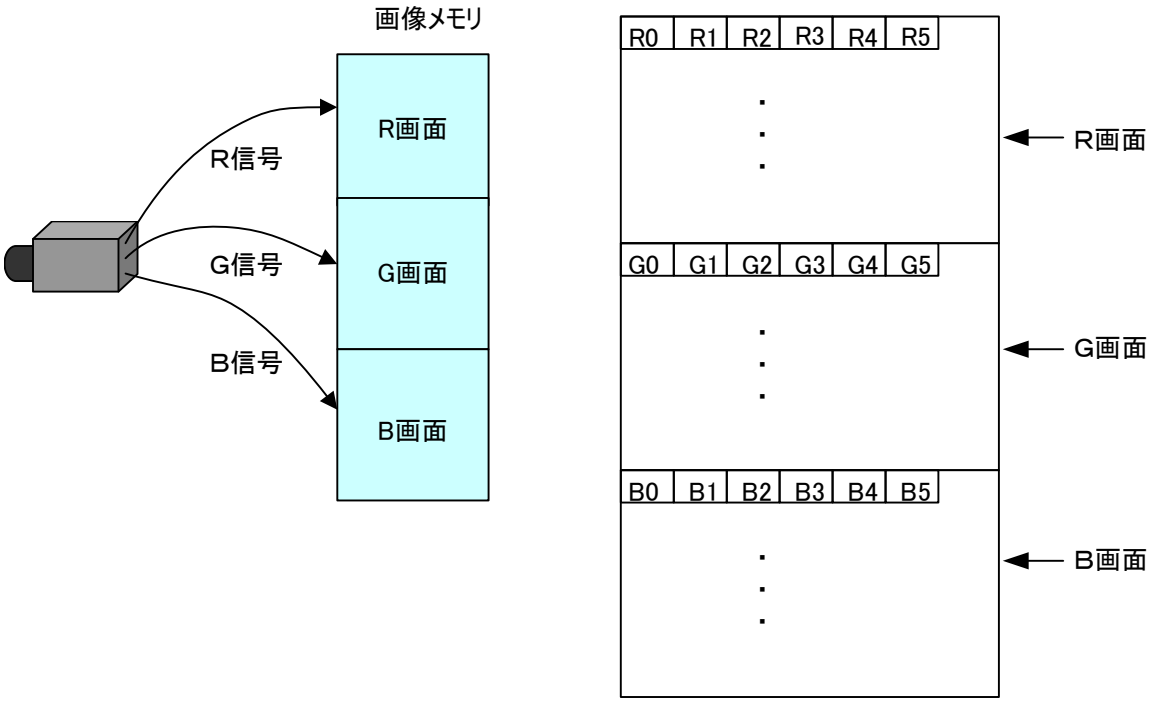


図8-32 RGB画面構成

カメラリンクRGBカメラからの映像入力構成を示します。

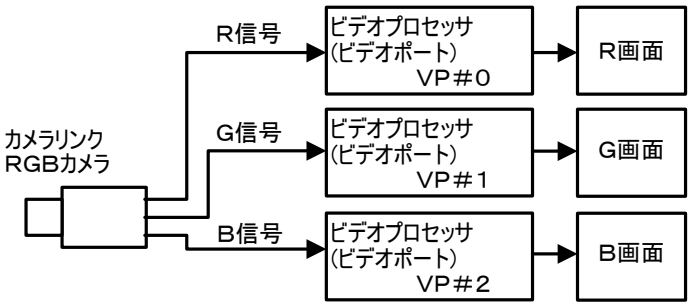


図8-33 RGB映像入力部のハード構成

以下に、RGB映像入力・出力の例を示します。

```
// 画面確保

      // ＲＧＢ画面確保
      ImgRGB = AllocRGBImg(IMG_FS_512H_512V);
      // ＲＧＢ１６表示画面確保
      ImgRGB16 = AllocRGB16Img(IMG_FS_512H_512V);


// ＲＧＢ映像入力

      // カメラポート配置の設定
      ActiveVideoPort(VIDEO_PORT0);
      // ＲＧＢカメラの選択
      SelectCamera(CAMERA_PORT0, 0x81); //0x81ユーザー定義RGBカメラの場合
      // 映像入力画面設定
      SetVideoFrame(NONINTERLACE,VIDEO_FS_512H_480V);
      // ＲＧＢカメラからの映像入力
      GetCamera(ImgRGB);

// ＲＧＢカラー映像表示

      // RGB -> RGB16 変換
      IP_CombineRGB(ImgRGB,ImgRGB16);
      // RGB カラー映像表示
      DispImg(ImgRGB16);
```

8.7.4 RGBカラー映像表示

NVPではRGBカラー映像を直接HDMIモニタに表示できません。そのため、**CombineRGB()**コマンドを使用してRGB画像をRGB16画像に変換する必要があります。

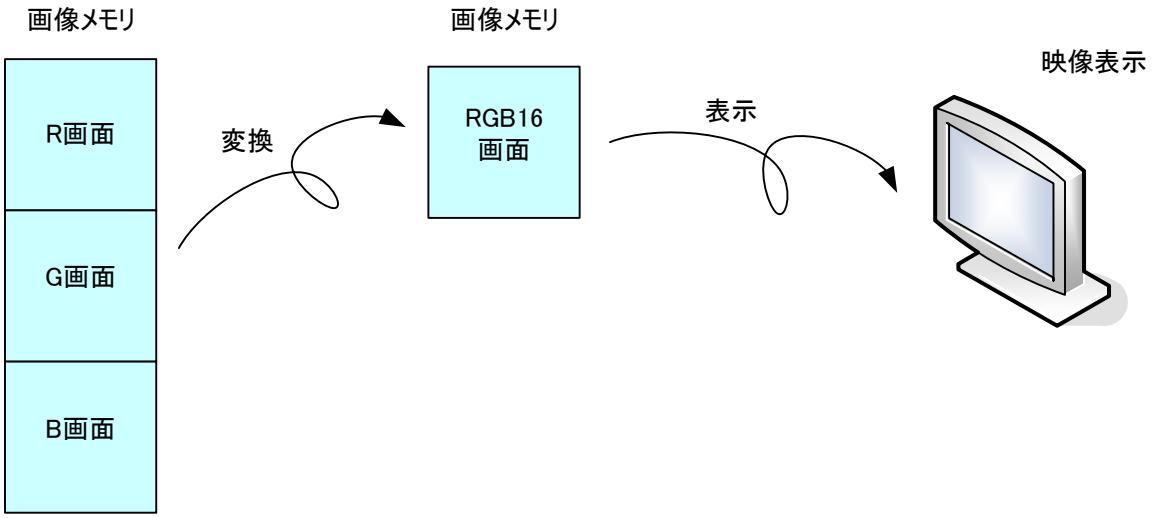


図8-34 RGB画像の表示

8.7.5 ＲＧＢカラー画面の画像処理

AllocRGBImg()コマンドで確保したRGBカラー画面は、モノクロ画面として画像処理をすることができます。ただし、RGBLUTコマンド等の一部のコマンドは、R，G，Bの全ての画面を必要とするコマンドもありますので詳細はコマンドリファレンスで確認ください。

RGBカラー画面に対して画像処理する場合は、R画面のみ処理します。G、B画面に対して処理する場合は、R画面の画面番号で管理するRGBカラー画面のG、B画面番号を取得し、このG、B画面番号を使用して実行できます。**GetGImgID()**コマンドでカラー画面のG画面番号を、**GetBImgID()**コマンドでカラー画面のB画面番号を取得します。

AllocRGB16Img()コマンドで確保したカラー画面に対しては画像処理することができません。ただし、**ReadImg()**、**WriteImg()**、**IP_Copy()**コマンド等、一部のコマンドを使用することが出来ますので、詳細はコマンドリファレンスでご確認ください。

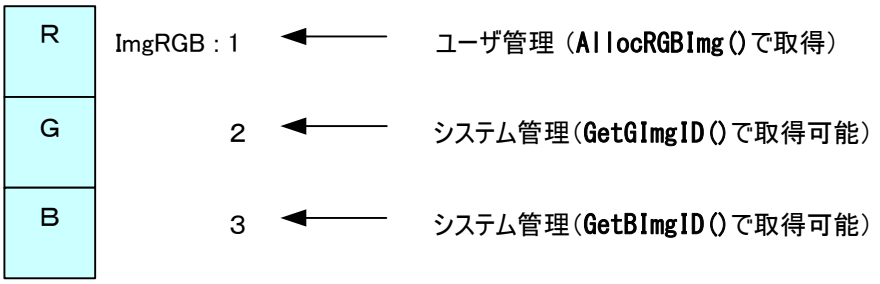
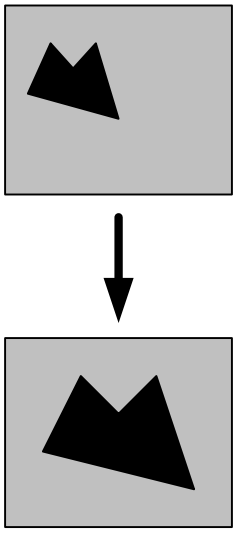
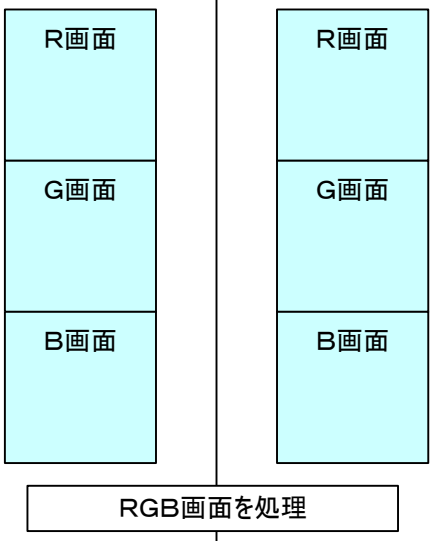
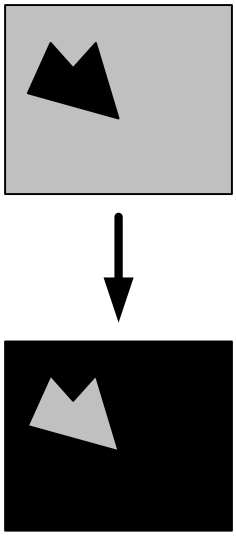
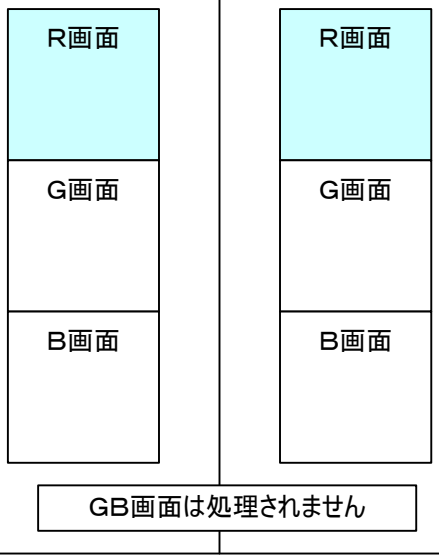
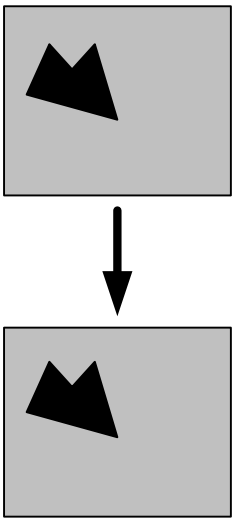
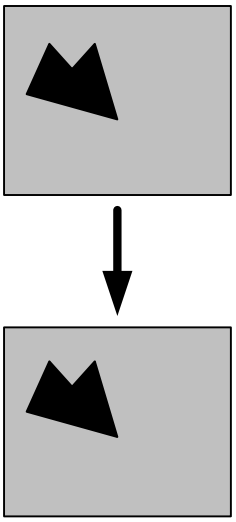
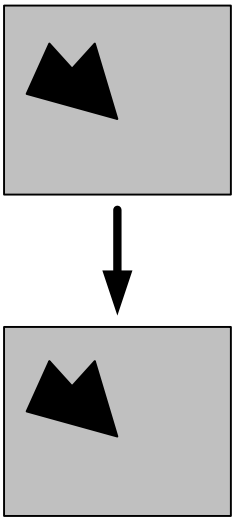


図8-35 RGB画像の画面番号の取得

表8-27 RGB画像処理の対象メモリ画面

画像処理例	画像処理	ソース	デスティネーション
拡大 IP_Zoom()			
論理反転 IP_Invert()			

画像処理例	画像処理	ソース	デスティネーション
RGBLUT変換 IP_ConvertRGBLUT()		R画面 G画面 B画面 RGB画面をY画面へ変換	Y画面
RGBからRGB16へ 変換 CombineRGB()		R画面 G画面 B画面 RGB画面からRGB16画面へ変換	RGB16画面
コピー IP_Copy()		RGB16画面 RGB16画面をRGB16画面へコピー	RGB16画面

９. 画像処理

画像処理は、機能ごとに下記のように分類されます。

（1）アフィン変換
（2）2値化
（3）濃度変換
（4）画像間算術演算
（5）画像間論理演算
（6）2値画像形状変換
（7）コンボリューション
（8）ランクフィルタ
（9）ラベリング
（10）ヒストグラム
（11）画像メモリアクセス
（12）2値パイプラインフィルタ
（13）パイプライン制御
（14）2値マッチングフィルタ
（15）正規化相関
（16）グラフィックス
（17）線分化
（18）穴埋め
（19）VP－910A互換
（20）直線抽出
（21）イメージキャリパ
（22）エッジファインダ
（23）RGBLUT変換
（24）歪み補正

9.1 画像処理の概要

画像処理は、処理対象画像データの画像メモリからの読み出し、画像処理プロセッサによる演算、処理 結果画像の画像メモリへの書き込みの順番で行われます。
処理対象画像は、ソース画像またはソース画面と呼びます。
処理結果画像は、デスティネーション画像またはデスティネーション画面と呼びます。

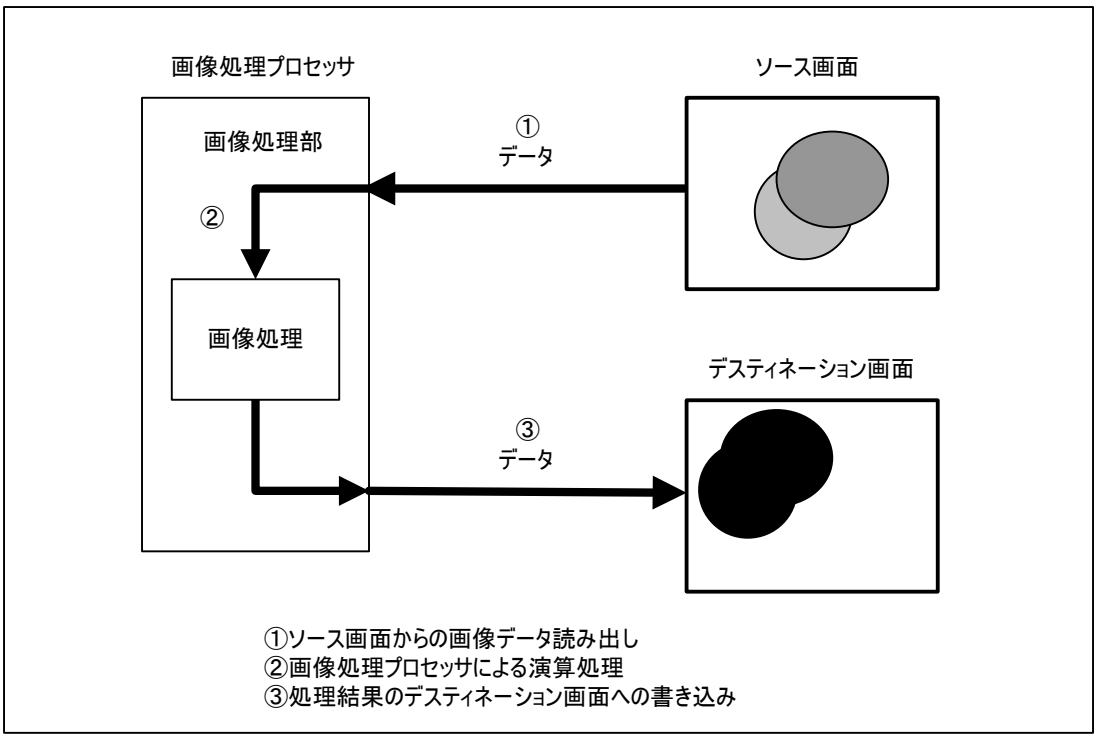
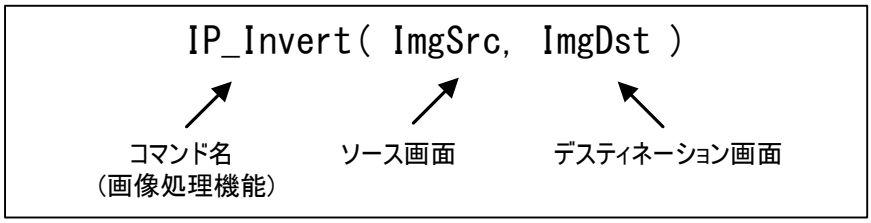


図9-1 画像処理の概要

本ライブラリでは、画像処理の機能をC言語のサブルーチン形式で提供します。画像処理のプログラムでは、コマンド（画像処理機能）の選択、ソース画面とデスティネーション画面の選択によって、画像処理機能を実行します。



画像処理の手順と各機能との関係は次のとおりです。ただし、これは一般的な場合の例であり、アプリケーションごとに、若干前後することがあります。

表9-1 画像処理の手順と各機能の対応

画像処理の手順	システム制御	画像メモリ領域管理	画像メモリ制御	映像入力表示制御	画像処理	パターン作成
開始 ↓ イニシャライズ（初期化） ↓	○					
処理画像領域の設定 ↓		○				
画像データ入力 ↓				○		
画像間演算・微分等の画像前処理 ↓					○	
面積等の特徴量を抽出 ↓					○	
画像メモリの内容を読み出し／書き込み ↓			○			
処理結果を表示する際の文字等の書き込み／重ね合せ表示 ↓ 終了				○		○

9.2 アフィン変換

アフィン変換とは、拡大／縮小、移動、回転などの幾何学変換のことです。本ライブラリでは、下の式の2次元のアフィン変換をハード処理とソフト処理によりサポートします。2～8倍及び1／2～1／8倍の整数倍の拡大／縮小と移動はハード処理で実現し、それ以外はソフト処理で実現します。

$$X = ax + by + c$$
$$Y = dx + ey + f$$

x,y	変換前の座標
X,Y	変換後の座標
a,b,c,d,e,f	任意定数

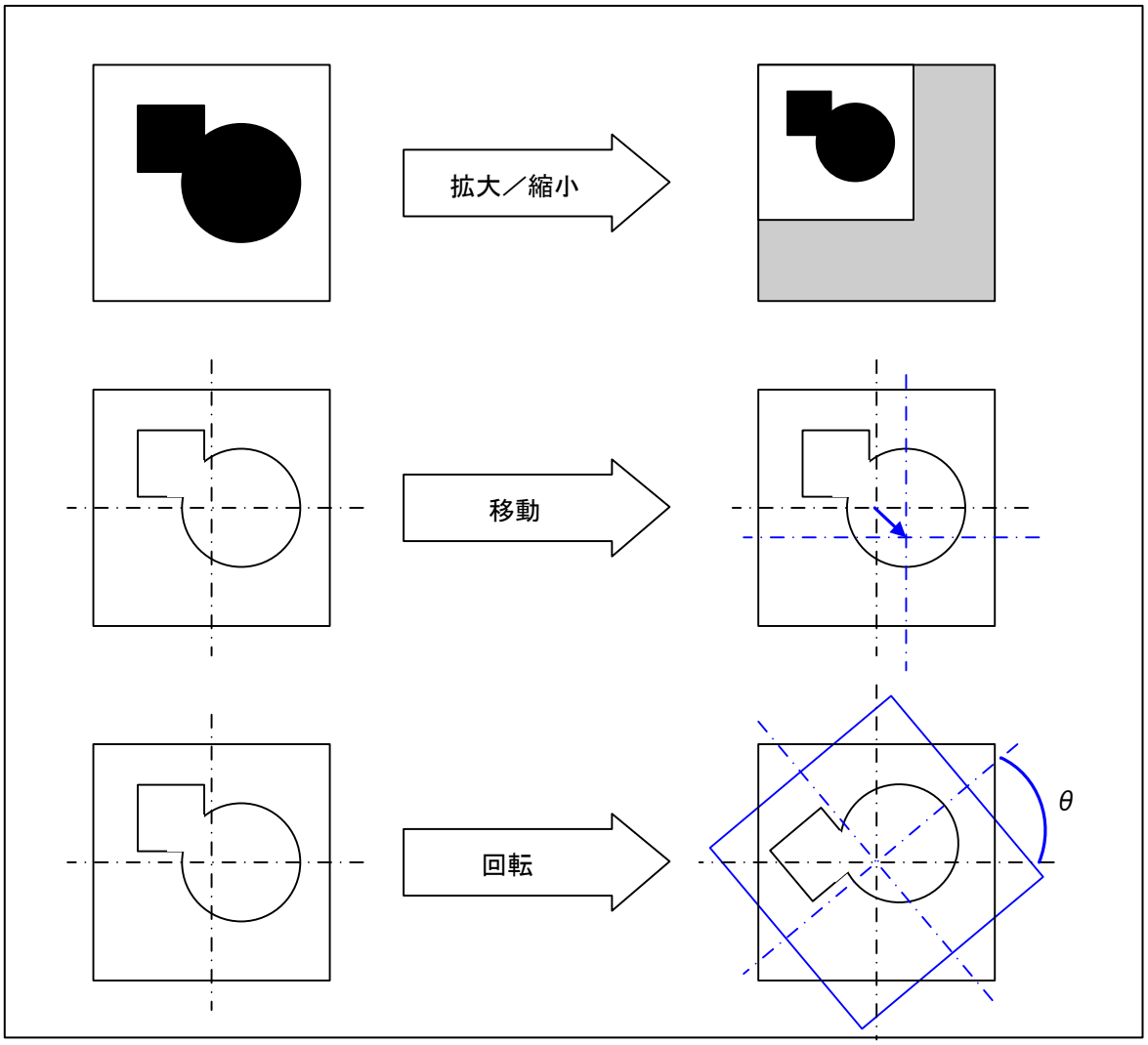


図9-2 アフィン変換

9.3 2値化

2値化とは、濃淡画像（256階調）を黒（濃度0）と白（濃度255）の2階調の画像に変換することです。本ライブラリコマンドでは、任意のしきい値以上の濃度値を白、それ以外の濃度値を黒とする通常の2値化と、任意の濃度値の範囲内と範囲外で白か黒にする範囲・反転付2値化をサポートします。

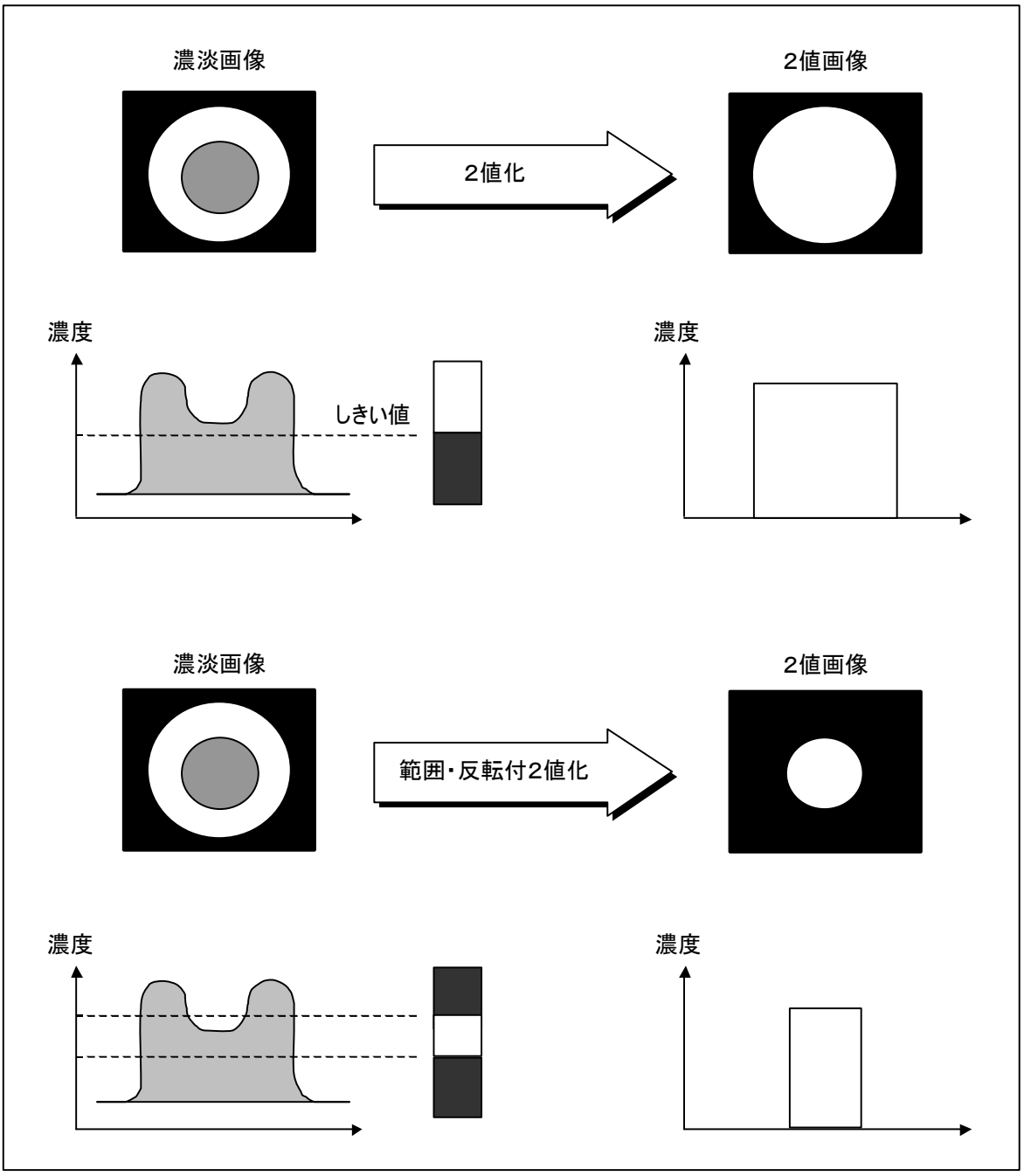


図9-3 2値化

9.4 濃度変換

濃度変換とは、濃度（輝度・明るさ）を変換する処理です。

濃度変換のパターンを換えることにより、暗い部分の濃度を高くしたり、逆に明るい部分の濃度を低くしたりする画質改善が可能です。画質改善以外にも、等濃度縞（ストライプ）のデータを使用すれば、曲面の曲がり、くぼみなどを見つけることができます。さらに、この考えを拡大していけば、3値化、4値化という処理が行えます。濃度変換のパターンは、自由に設定可能です。

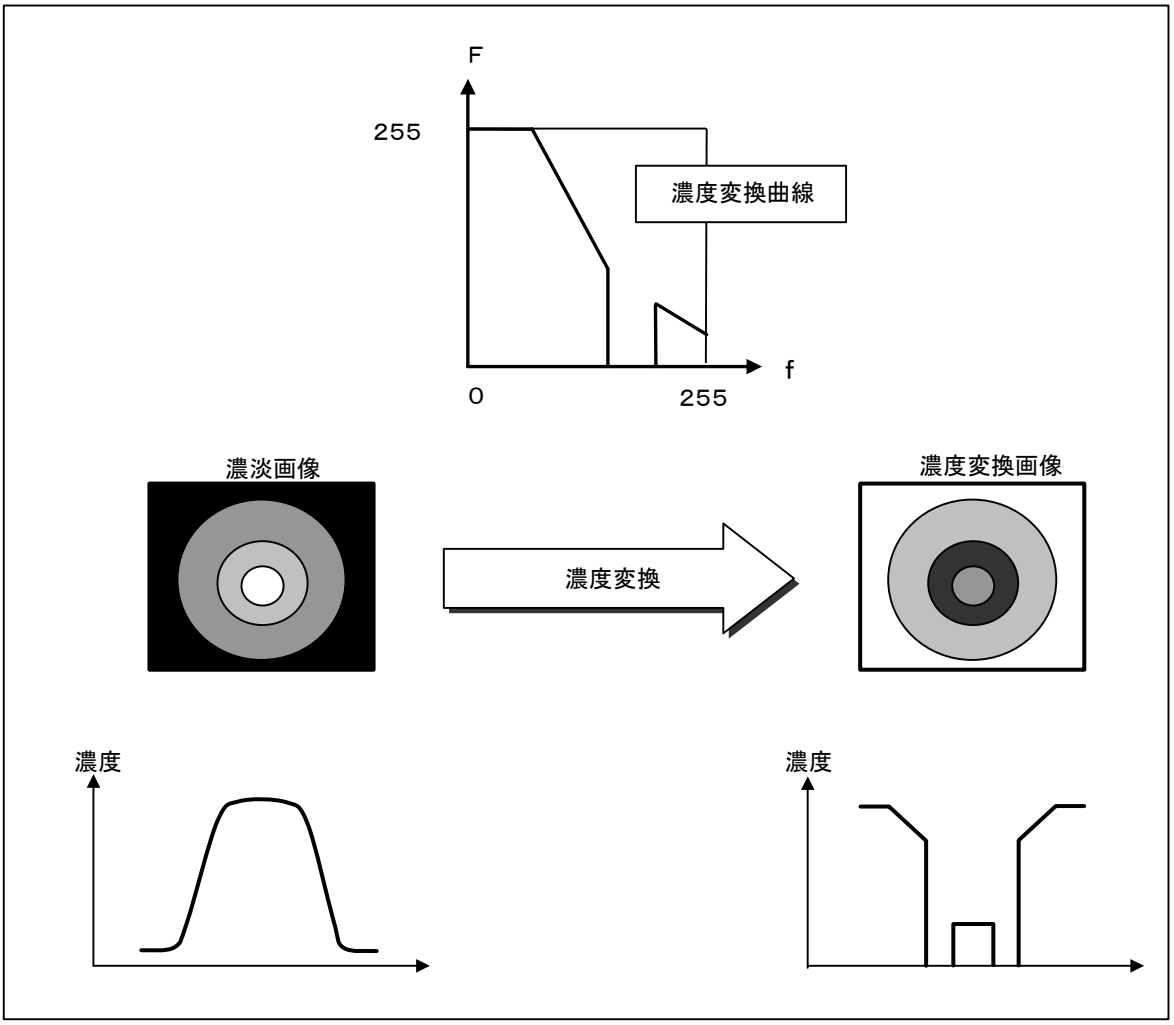
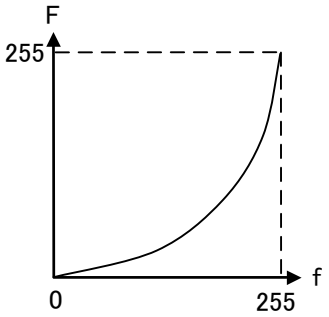
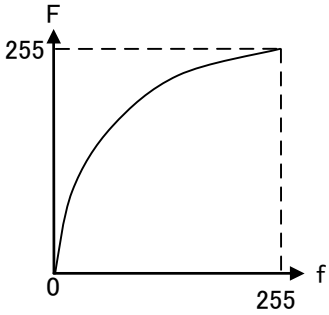
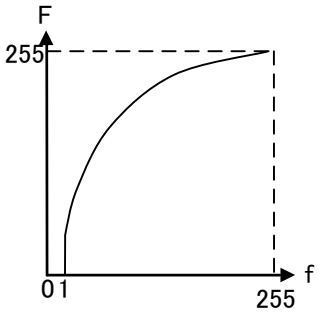
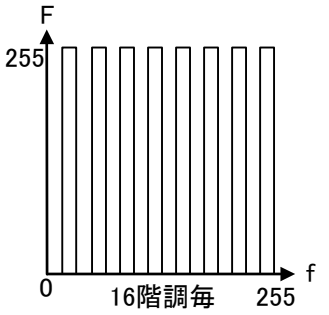
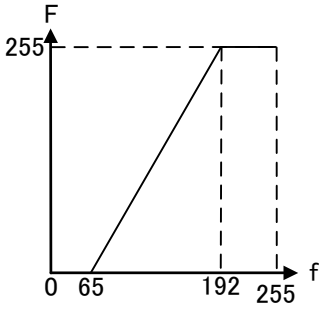


図9-4 濃度変換

表9-2 濃度変換詳細

種類	濃度変換曲線	詳細	効果
γ補正①		0≦f≦255において $F = af^{1.2}$ （但し $a = 255^{-0.2}$ ） f : 処理対象画像濃度 F : 処理結果画像濃度	低濃度部の強調
γ補正②		0≦f≦255において $F = af^{1/1.2}$ （但し $a = 255^{-0.2/1.2}$ ） f : 処理対象画像濃度 F : 処理結果画像濃度	高濃度部の強調
log変換		1≦f≦255において F = a*log(f) （但し a = 255/log(255)） f = 0 において F = 0 f : 処理対象画像濃度 F : 処理結果画像濃度	1以下を0とし高濃度部を強調
等濃度縞		0≦f≦255において 16階調毎にF=0 or F=255 f : 処理対象画像濃度 F : 処理結果画像濃度	16階調単位に黒と白に変換
強調		0≦f≦255において f = 0 ～ 64 において F = 0 f = 65 ～ 191 において F = 255/127*(f-65) f = 192 ～ 255において F = 255 f : 処理対象画像濃度 F : 処理結果画像濃度	低濃度部と高濃度部の強調 （コントラスト改善）

9.5 画像間算術演算

画像間演算とは、2画面の画像で演算を行い演算後別の1画面に合成することです。画像間演算には加算、減算、乗算などの画像間算術演算と論理和、論理積などの画像間論理演算があります。

画像間算術演算の用途としては、基準パターンとの減算による差画像から欠陥を検出したり、何度も同一画像を取込み加算することでランダムノイズ（ホワイトノイズ）を除去するといったことがあります。また、映像入力したフレーム毎に2画面の画像の最大値をとることで、明るい物体を次から次へと1画面の画像に合成することができます。これは、移動物体の軌跡や移動量を検出するために有効です。

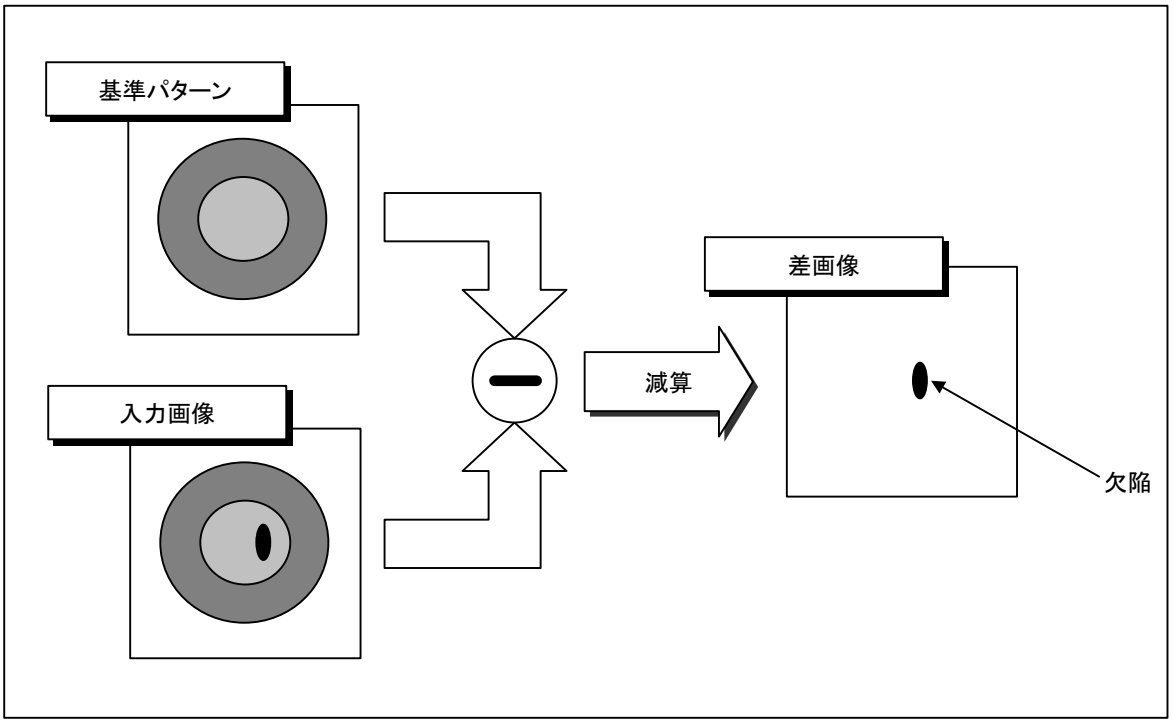


図9-5 画像間算術演算

9.6 画像間論理演算

2画面間で1画素毎に論理和、論理積などの論理演算を行い結果を別の画像に格納します。
2値画像での基準パターンと検査パターンとのチェック（XOR演算）、画像の合成（OR演算）、キズの検出に有効です。
また、画像に任意のパターンでマスクをかける場合にも任意パターンとターゲット画像のAND演算を行います。

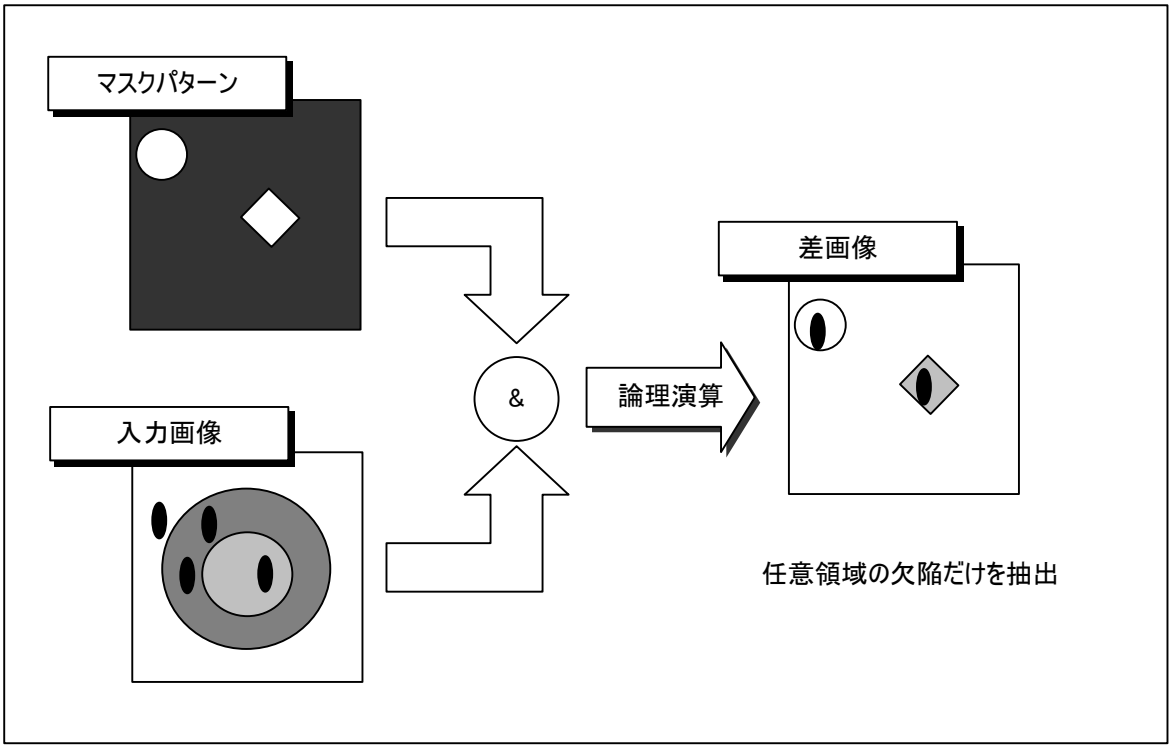


図9-6 画像間論理演算

9.7 2値画像形状変換

2値画像に対してノイズ除去、輪郭抽出、膨張、収縮、細線化、縮退化といった処理を行うことができます。これらの処理は、目的に応じて4連結と8連結を選択できます。

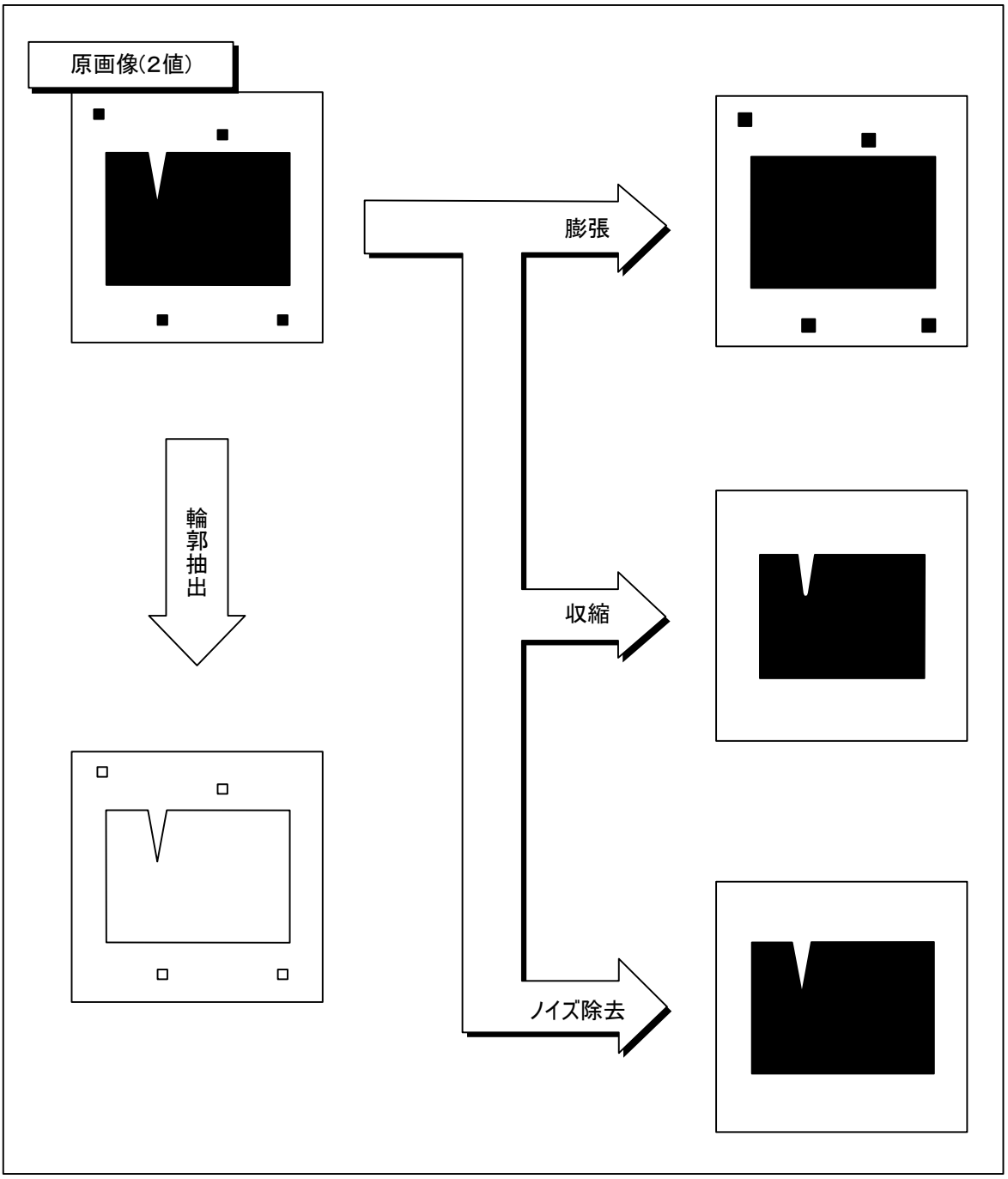


図9-7 2値画像形状変換

9.8 コンボリューション

コンボリューションは、近傍領域の積和演算です。下図に示すように、ソース画面の指定領域内の全画素に対して、注目画素を中心とした３×３近傍の局所領域で与えられた荷重係数との積和演算を行います。

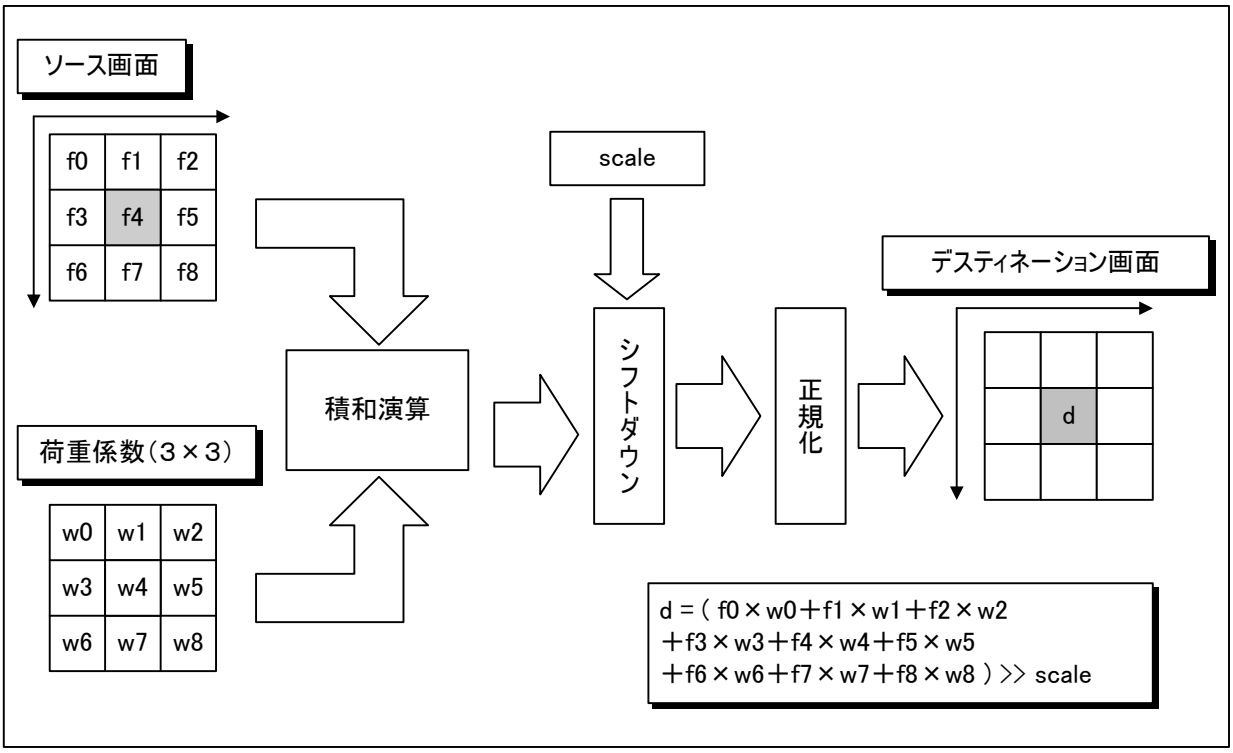


図9-8 コンボリューション

この演算は下表に示す荷重係数の設定により、濃淡画像での平滑化や輪郭強調などを行うことができます。

表9-3 コンボリューション詳細

種類･名称		荷重係数	scale	効果
平滑化	―	1 1 1 1 1 1 1 1 1	3	平滑化。画像の輪郭を滑らかにする。
1次微分 グラディエント	微分	0 0 0 0 1 −1 0 0 0	0	輪郭強調（X方向）
		0 0 0 0 1 0 0 −1 0	0	輪郭強調（Y方向）

種類･名称		荷重係数	scale	効果
1次微分 グラディエント	Roberts	0 0 0 0 1 0 0 0 −1	0	輪郭強調（X方向）
		0 0 0 0 0 1 0 −1 0	0	輪郭強調（Y方向）
	Sobel	−1 0 1 −2 0 2 −1 0 1	0	輪郭強調（X方向）
		−1 −2 −1 0 0 0 1 2 1	0	輪郭強調（Y方向）
2次微分 ラプラシアン	4連結	0 −1 0 −1 4 −1 0 −1 0	0	輪郭強調
	8連結	−1 −1 −1 −1 8 −1 −1 −1 −1	0	輪郭強調

9.9 ランクフィルタ

ミニマムフィルタ、マックスフィルタ、メディアンフィルタなどをランクフィルタと呼びます。ランクフィルタ処理は、ソース画面の指定領域内の全画素に対して、注目画素を中心とした３×３近傍の局所領域内の画素データをソート（順に並べる）し、任意の順番の画素データを抽出します。また、３×３近傍の局所領域内で任意のカーネルマスクでマスク処理を行い、有効となった画素データ内でソートすることができます。

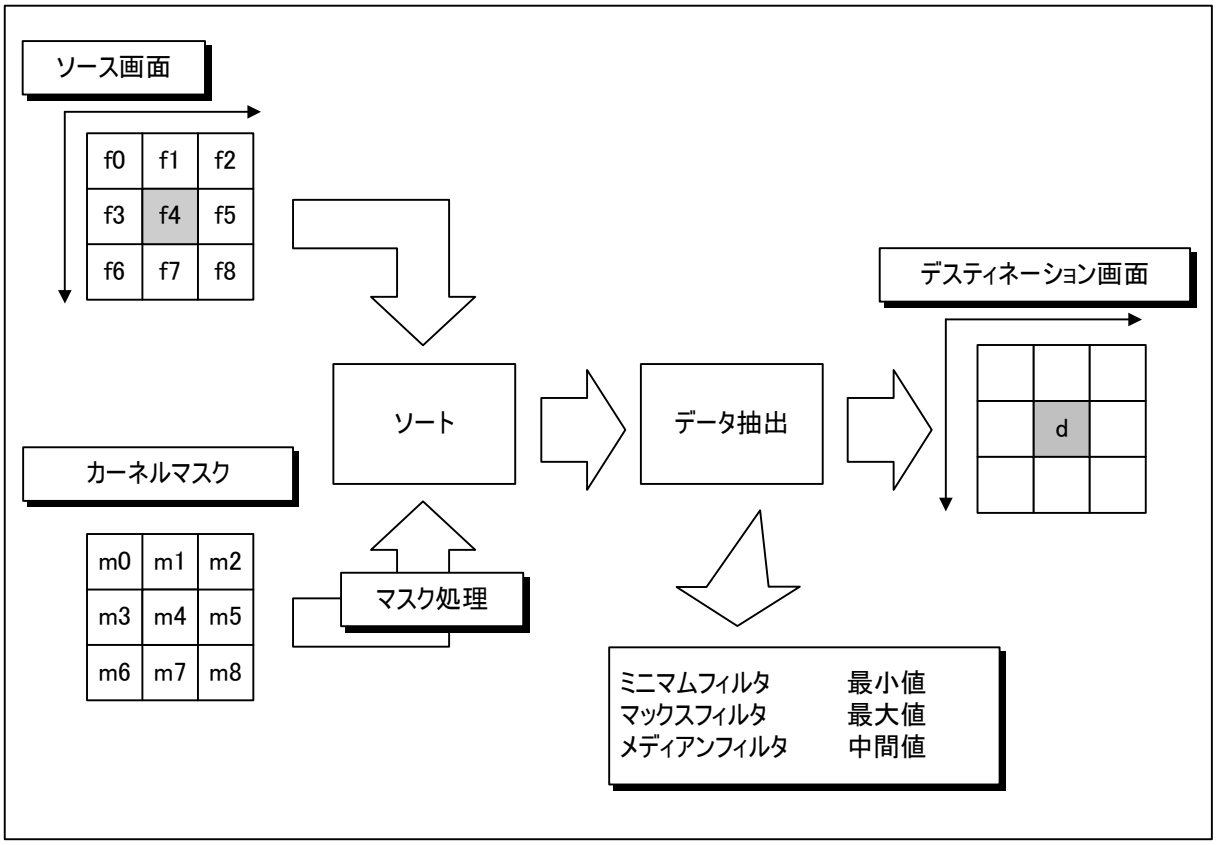


図9-9 ランクフィルタ

9.10 ラベリング

ラベリング処理では、2値画像に対して連結している物体ごとに番号を付けます。ラベリング処理の種類には、すべての物体をラベリングするものと、一定範囲内の面積物体にのみラベリングするものがあり、それぞれにオプションとして白をラベリングするか、黒をラベリングするかが選択できます。

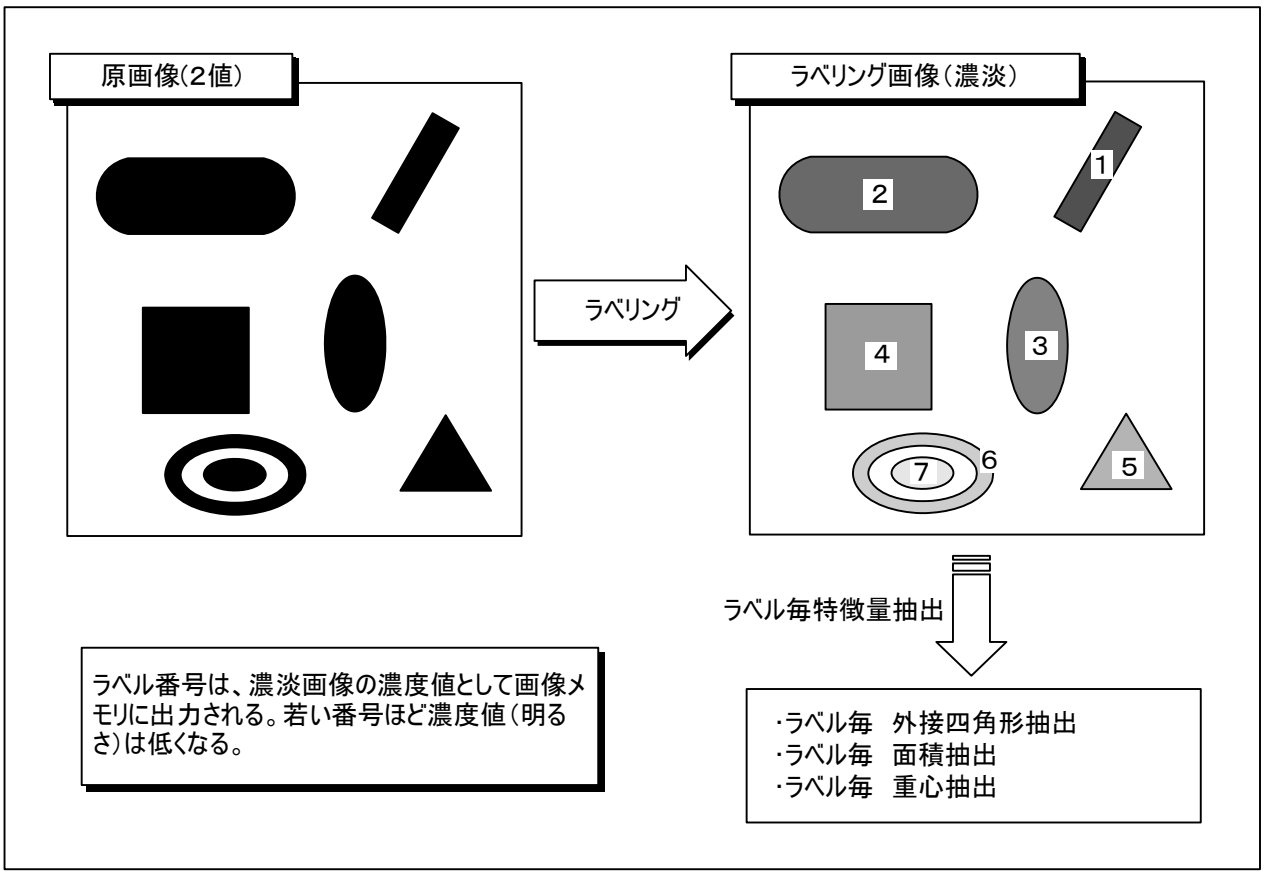


図9-10 ラベリング概要

ラベリング処理は、仮ラベル付け、合流対検出、真ラベル付けの3段階で行われます。

次にそれぞれの処理について説明します。
図9-11(a)の原画像に対してラベリング処理を起動すると、画面左上から仮ラベル付けが始まります。ラスタスキャンにて番号付けを行うため、図9-11(b)に示すように同じ物体を異なるラベルとして認識します。この段階では物体Aに対して1、2、3の3つの仮ラベル番号が、物体Bに対しては4、5の2つの仮ラベル番号が割り当てられます。そこで次の段階として合流対検出を行います。合流対検出とは同じ物体に対して複数の仮ラベル番号が付いていないかを調べる処理であり、図9-11(b)に対しては図9-11(c)に示すように、1、2、3は同じ、4、5は同じという結果が得られます。この結果をもとに真ラベル付けを行うことにより、最終的に図9-11(d)に示す結果が得られます。
なお、各々の処理には、
・仮ラベル付け番号　：最大　4094 まで
・合流対　　：最大　2046 まで
・真ラベル　　：最大　255 個まで
との制約があり、この値を超えるとオーバーフローとなり、正しい結果が得られません。

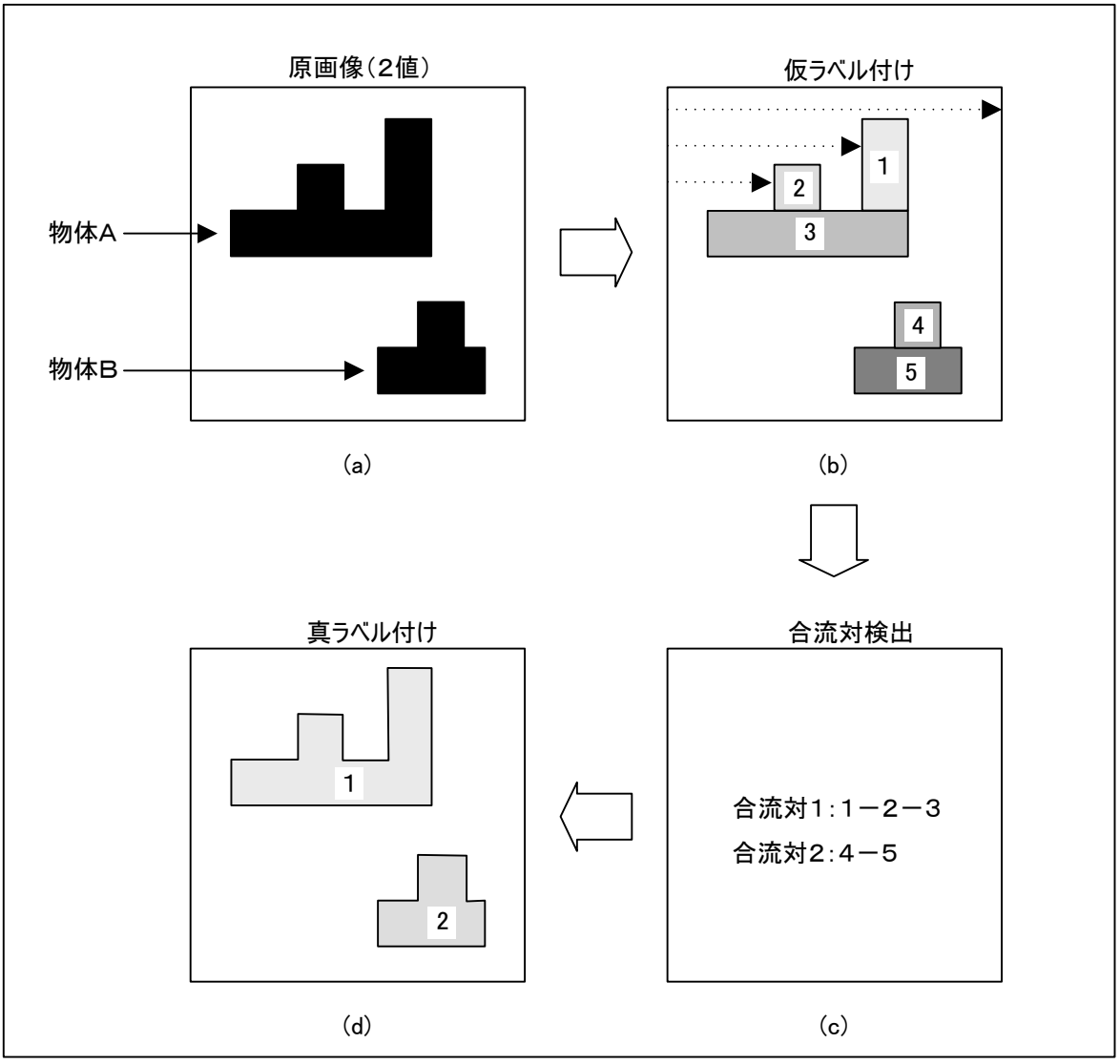


図9-11　ラベリング処理

9.11 濃淡画像特徴量抽出(ヒストグラム)

ヒストグラム処理では、2値画像と濃淡画像に対し、最大／最小濃度、濃度頻度分布抽出等の統計処理が行えます。

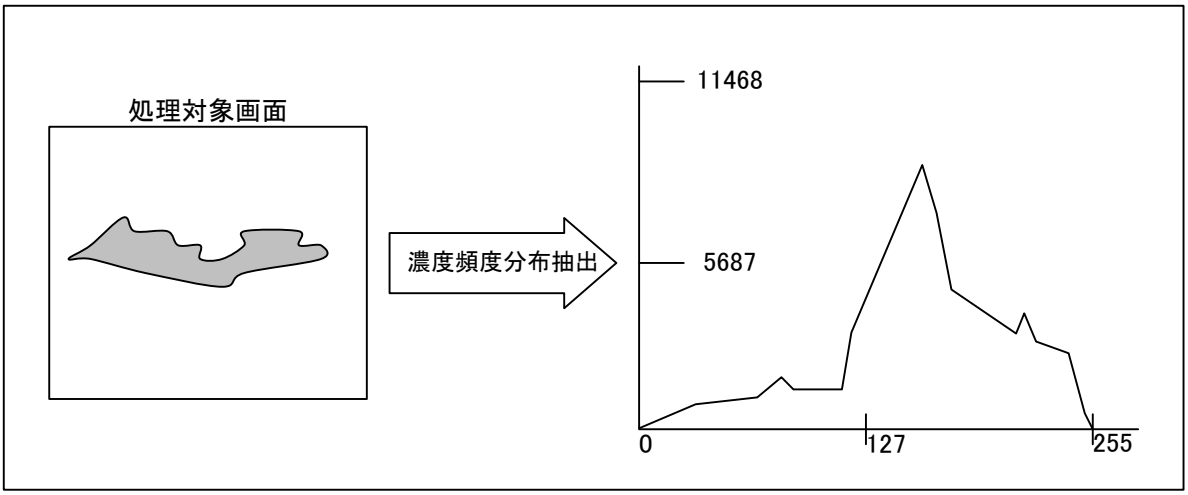


図9-12 濃度頻度分布抽出

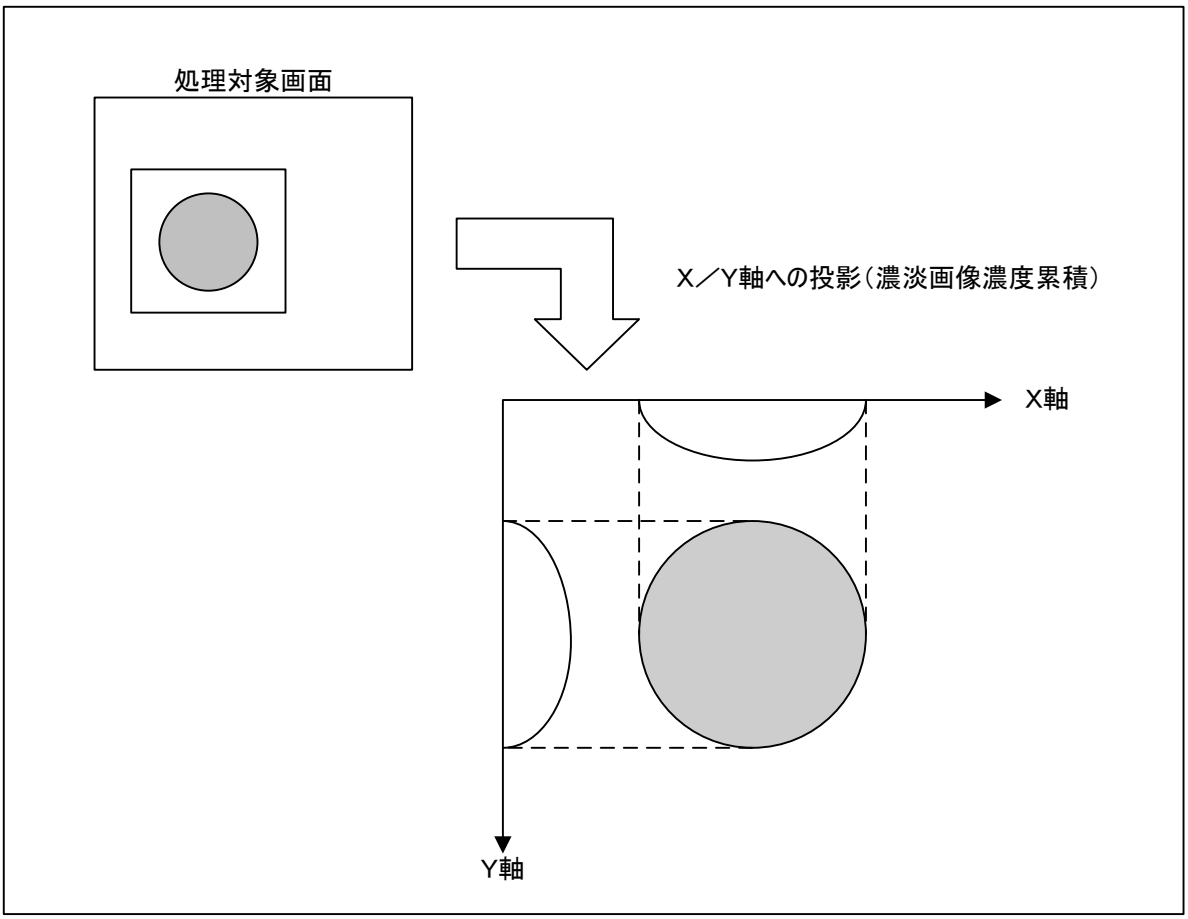


図9-13 X／Y軸への投影

ヒストグラム処理での座標系は、ウィンドウ相対であり、ウィンドウの始点が座標0になります。
よって、処理結果の座標はウィンドウ始点を基準とした座標であり、処理結果のテーブルはウィンドウ内の有効データがテーブルの先頭から格納されます。

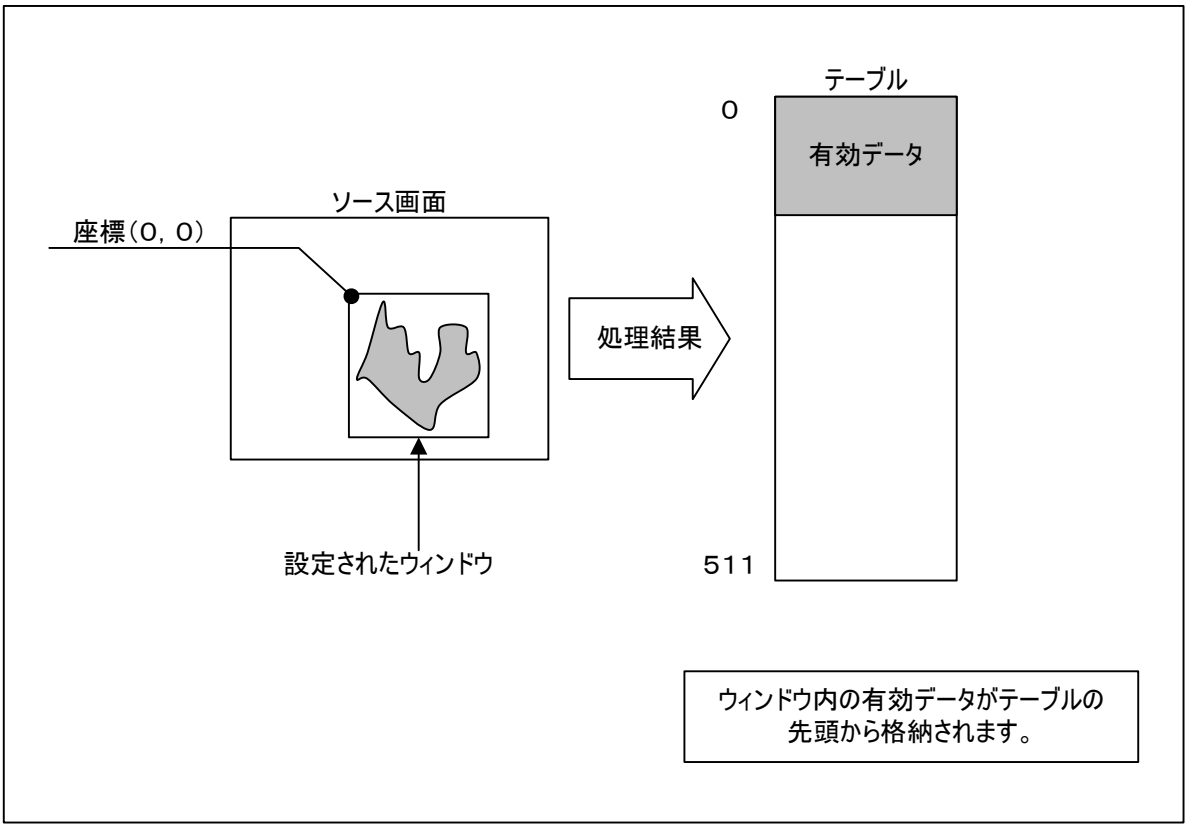


図9-14　処理結果テーブル

9.12 画像メモリアクセス

画像メモリアクセスコマンドでは、画像メモリのデータを1画素単位または、ブロック単位でアクセスすることができます。

9.12.1 画像メモリアクセス手順フロー

以下に、画像メモリアクセス手順フローを示します。

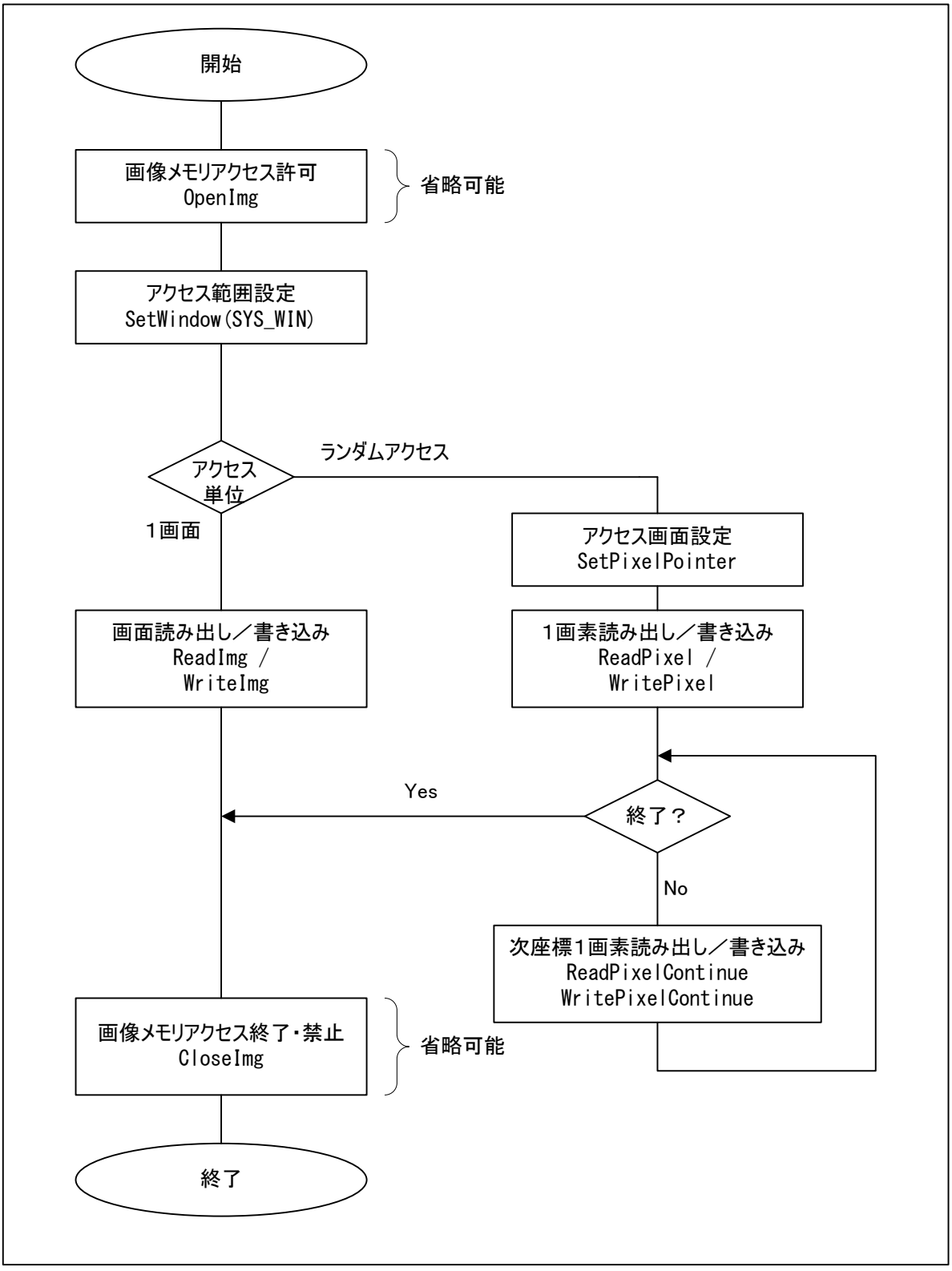


図9-15 画像メモリアクセス手順

9.12.2 ウィンドウの設定

画像メモリアクセスコマンドは任意のウィンドウが設定できます。設定するウィンドウの種類はSYS_WINです。**ReadImg()**コマンドはSYS_WINで設定された矩形領域の画素データをローカルメモリにブロック単位に読込み、**WriteImg()**コマンドはローカルメモリ上のデータをSYS_WINで設定された矩形領域の画像メモリにブロック単位に書込みます。

また、画像メモリ制御コマンドでの座標系はウィンドウ相対であり、ウィンドウの始点が座標0となります。

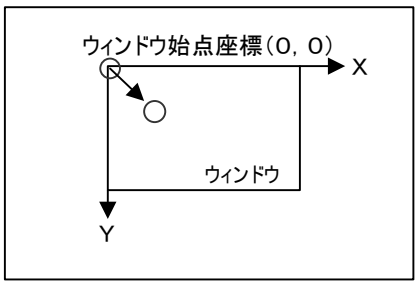


図9-16 ウィンドウ座標系

9.12.3 画面データタイプ

画像メモリアクセスを行う画面の画面データタイプは符号なし8ビット、符号付8ビット、2値です。なお、2値データ0は画素データ「0」、2値データ1は画素データ「255」です。

9.12.4 画像メモリ直接アクセス

本ライブラリのNVP-Linux側の処理では、**OpenImgDirect()**コマンドにより、画像メモリ直接アクセスコマンドにより画像メモリの先頭アドレスを取得し、画像メモリを直接アクセスすることができます。

PC側のリモートコマンドでは、**OpenImgDirect()**コマンドはパラメータで指定した画面の画面サイズでメモリを確保し、その領域に指定画面の画像データを格納します。**OpenImgDirect()**コマンドが返すアドレスはこの領域の先頭アドレスです。**CloseImgDirect()**コマンドは、**OpenImgDirect()**コマンドで確保したメモリの画像データを**OpenImgDirect()**コマンドで指定した画面の画像メモリに書き戻してから、確保メモリを解放します。そのため、**CloseImgDirect()**コマンドの実行で画像データの変更が反映されます。NVP-Linux側、PC側リモートコマンドでの実行時どちらでも**OpenImgDirect()**コマンド実行後は必ず**CloseImgDirect()**コマンドを実行して下さい。

以下に、画像メモリ直接アクセス手順フローを示します。

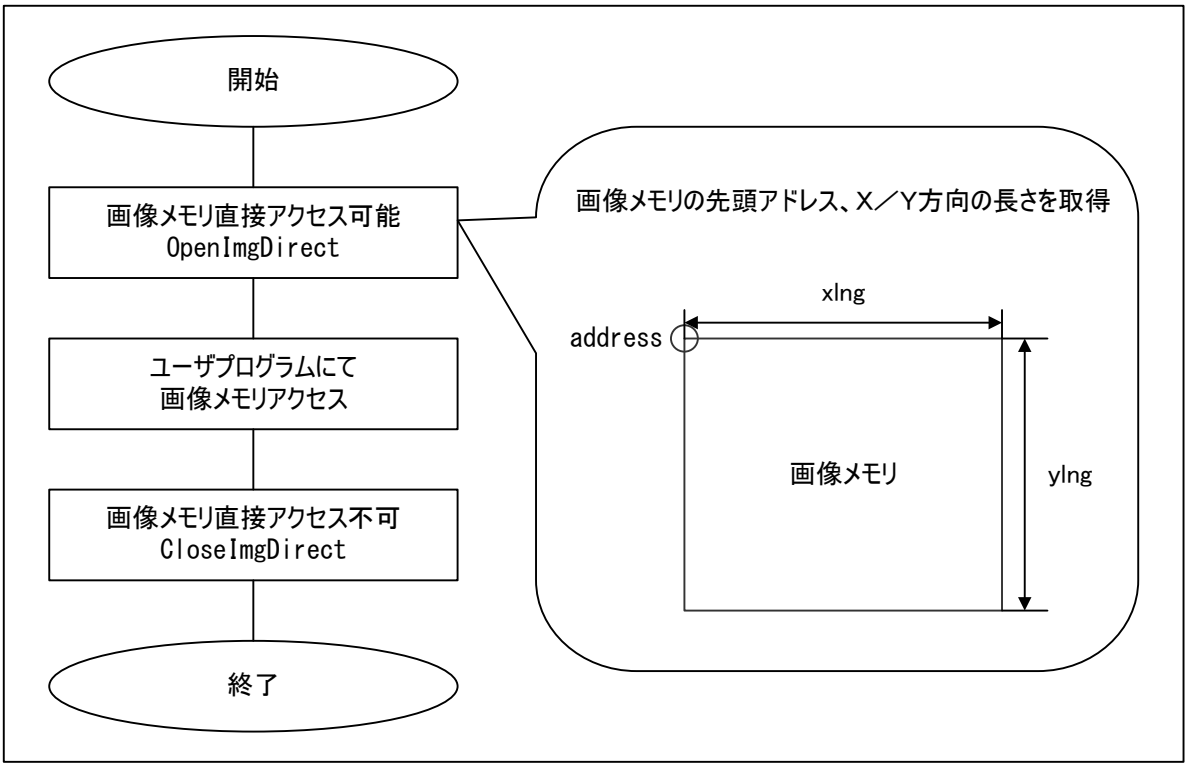


図9-17 画像メモリ直接アクセス手順

9.13 2値パイプラインフィルタ

2値パイプラインフィルタコマンドは、2値画像に対し、3×3カーネルを用いてノイズ除去、膨張、収縮、輪郭抽出の4つの画像処理を組み合わせて、最大8段までのフィルタ処理のパイプライン処理を行います。また、8段パイプラインを4段×2の構成にして、4段目の出力結果を論理演算して出力することもできます。

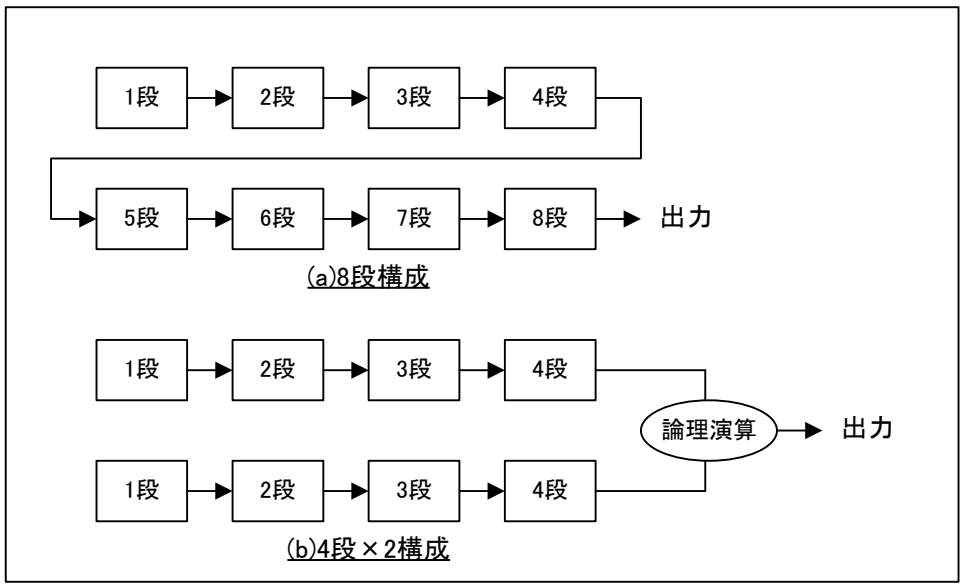
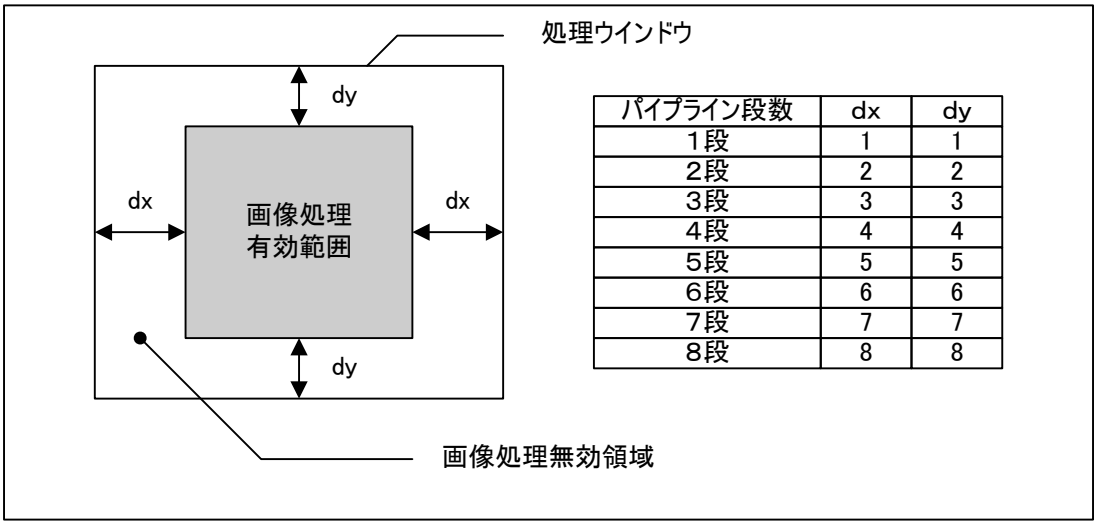


図9-18 2値パイプラインフィルタ

9.13.1 2値パイプラインフィルタ処理領域

3×3カーネルを用いて画像処理を行う為、画像処理対象エリアのエッジ部分(周辺)に画像処理結果無効の範囲が生じます。また、この画像処理無効範囲は、パイプラインの段数によって決まります。
段数による無効範囲を以下に示します。



パイプライン段数	dx	dy
1段	1	1
2段	2	2
3段	3	3
4段	4	4
5段	5	5
6段	6	6
7段	7	7
8段	8	8

図9-19 処理領域

9.14 パイプライン制御

画像処理プロセッサは、画像処理、2値化、ヒストグラムのプロセッサを独立して持っています。
そして、その3つの処理を画像処理→2値化→ヒストグラムの順番でパイプライン処理を行うことが可能であり、画像処理、2値化、ヒストグラムプロセッサの処理を1画面の処理時間で実行できます。

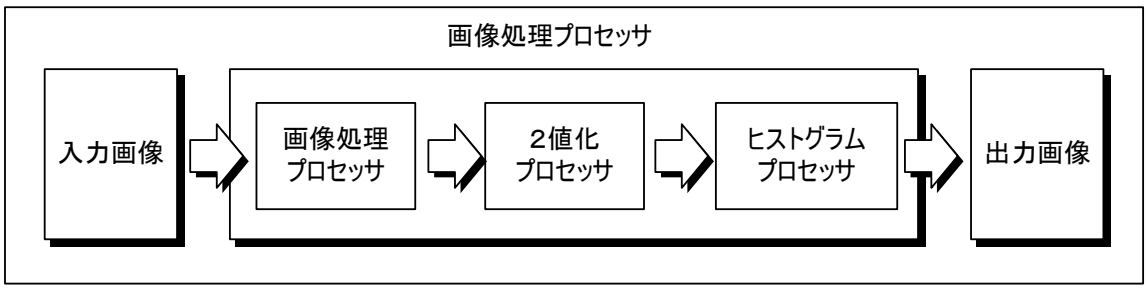


図9-20 画像処理プロセッサパイプライン処理部

また、下記の組み合わせでパイプライン処理が実行可能です。
各処理については、「パイプライン処理の実行条件」を参照ください。

（1）画像処理(濃淡) ＋ 2値化 ＋ ヒストグラム処理(2値)
（2）画像処理(濃淡) ＋ 2値化
（3）2値化 ＋ ヒストグラム処理(2値)
（4）画像処理(濃淡) ＋ ヒストグラム処理(濃淡)

9.14.1 パイプライン処理の実行方法

EnablePipeline(),**DisablePipeline()**コマンドでパイプラインモード制御を行います。

EnablePipeline()コマンドを実行すると、パイプラインモードになります。パイプラインモードでは、最初に発行された画像処理コマンドを実行せず、次のコマンド処理へ移行します。そのコマンドの処理がパイプラインで実行できない場合は、パイプライン制御により実行を待たされていた(ペンディング状態)画像処理コマンドを実行します。つまり、コマンドの処理がパイプラインでつながる場合、その時点での処理を行わずペンディング状態にして、コマンドバッファにバッファリングを行い、パイプラインのチェインが切れた時点で処理を行うわけです。

DisablePipeline()コマンドを実行すると、コマンドバッファにバッファリングされていたペンディング処理を実行し、パイプラインモードを解除し、通常の処理に戻ります。

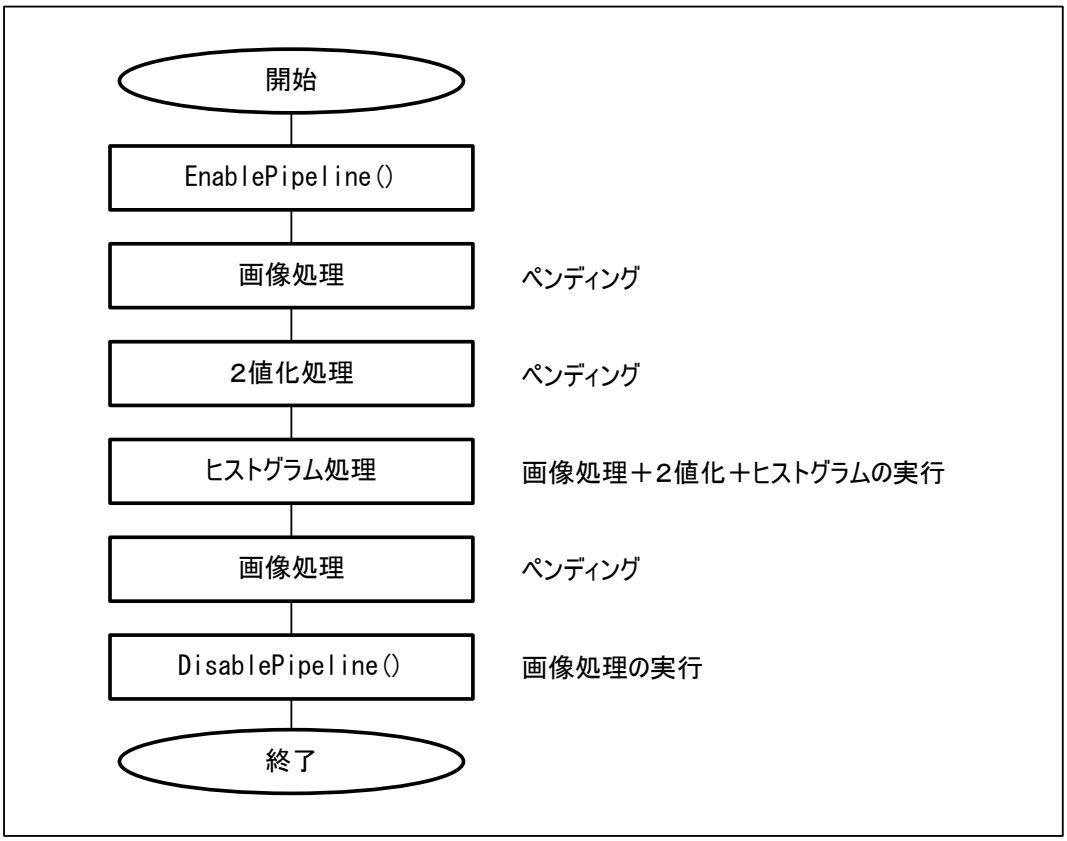


図9-21 パイプライン処理の動作

9.14.2 パイプライン処理の実行条件

パイプライン処理が行われるには、以下の条件があります。

(1) 画像処理の対象画面

先に実行されるコマンドのデスティネーション画面と、次に実行されるコマンドのソース画面が同一であることです。

(2) パイプラインが構成可能な処理と順番

下の表にパイプライン処理構成として、パイプラインが構成可能な処理と順番を示します。
他の組み合わせについては結果を保証いたしません。

表9-4 パイプライン処理構成

No.	処理と順番	備考
1	画像処理(濃淡) ⇒ ヒストグラム(濃淡)	注意事項２
2	画像処理(２値) ⇒ ヒストグラム(２値)	
3	２値化 ⇒ ヒストグラム(２値)	
4	画像処理(濃淡) ⇒ ２値化 ⇒ ヒストグラム(２値)	注意事項５

パイプライン処理を構成するコマンドである画像処理(濃淡)、画像処理(２値)、２値化処理、ヒストグラム処理(濃淡)、ヒストグラム(２値)、 および、パイプライン処理とは無関係に動作するコマンドを以下に示します。

表9-5 画像処理(濃淡)コマンド

項目	コマンド名	備考
画像転送／アフィン変換	IP_Copy	
	IP_ZoomOut	注意事項2
	IP_ZoomOutExt	注意事項2
	IP_ZoomIn	注意事項2
	IP_ZoomInExt	注意事項2
画素変換	IP_Invert	
	IP_Minus	
	IP_Abs	
	IP_AddConst	
	IP_SubConst	
	IP_SubConstAbs	
	IP_MultConst	
	IP_MinConst	
	IP_MaxConst	
	IP_ShiftDown	
画像間算術演算	IP_Add	
	IP_Sub	
	IP_SubAbs	
	IP_Comb	
	IP_CombAbs	
	IP_Mult	
	IP_Average	
	IP_Min	
	IP_Max	
	IP_SubConstAbsAdd	
	IP_SubConstMultAdd	
	IP_SubConstMult	
	IP_CombDrop	
画像間論理演算	IP_And	
	IP_Or	
	IP_Xor	
	IP_InvertAnd	
	IP_InvertOr	
	IP_Xnor	

項目	コマンド名	備考
コンボリューション	IP_SmoothFLT	
	IP_EdgeFLT	
	IP_EdgeFLTAbs	
	IP_Lapl4FLT	
	IP_Lapl8FLT	
	IP_Lapl4FLTAbs	
	IP_Lapl8FLTAbs	
	IP_LineFLT	
	IP_LineFLTAbs	
ミニ／マックスフィルタ	IP_MinFLT	
	IP_MinFLT4	
	IP_MinFLT8	
	IP_MaxFLT	
	IP_MaxFLT4	
	IP_MaxFLT8	
	IP_LineMinFLT	
	IP_LineMaxFLT	
ランクフィルタ	IP_RankFLT	
	IP_Rank4FLT	
	IP_Rank8FLT	
	IP_MedFLT	
	IP_Med4FLT	
	IP_Med8FLT	
2値マッチングフィルタ	IP_BinMatchFLT	
拡張コンボリューション	IP_SmoothFLT5x5	
	IP_SmoothFLT7x7	
	IP_EdgeFLT5x5	
	IP_EdgeFLT7x7	
	IP_MinFLT5x5	
	IP_MaxFLT5x5	
	IP_SmoothFLT5x5Ext	
	IP_SmoothFLT7x7Ext	
	IP_EdgeFLT5x5Ext	
	IP_EdgeFLT7x7Ext	

表9-6 画像処理(2値)コマンド

項目	コマンド名	備考
2値画像形状変換	IP_PickNoise4	
	IP_PickNoise8	
	IP_Outline4	
	IP_Outline8	
	IP_Dilation4	
	IP_Dilation8	
	IP_Erosion4	
	IP_Erosion8	
	IP_Thin4	
	IP_Thin8	
	IP_Shrink4	
	IP_Shrink8	
2値パイプラインフィルタ	IP_TrsPipelineFLT	

表9-7 2値化処理コマンド

項目	コマンド名	備考
2値化	IP_Binarize	
	IP_BinarizeExt	

表9-8 ヒストグラム処理(濃淡)コマンド

項目	コマンド名	備考
濃淡画像特徴量抽出	IP_ExtractGOFeatures	注意事項3
	IP_Histogram	
	IP_HistogramShort	
	IP_HistogramFeatures	
	IP_ProjectGO	
	IP_ProjectGOonX	
	IP_ProjectGOonY	
	IP_ProjectGOMaxValue	
	IP_ProjectGOMinValue	
	IP_ProjectBlockGO	
	IP_ProjectBlockGOMinMaxValue	
ラベリング	IP_ExtractLORegionX	注意事項3
	IP_ExtractLORegionY	注意事項3
	IP_ExtractLOArea	注意事項3
	IP_ExtractLOAreaExt	注意事項3
	IP_ExtractLOGravity	注意事項3

表9-9 ヒストグラム処理(2値)コマンド

項目	コマンド名	備考
2値画像特徴量抽出	IP_ExtractBOArea	
	IP_ExtractBOFeatures	
	IP_ProjectBO	
	IP_ProjectBORegionX	
	IP_ProjectBORegionY	
	IP_ProjectBlockBO	

パイプライン処理と無関係に動作するコマンドには、ペンディングされているコマンドを実行(パージ)してから動作するコマンドと、ペンディングされているコマンドを実行せずに単独で動作するコマンドがあります。
ペンディングされているコマンドを実行(パージ)してから動作するコマンドを以下に示します。

表9-10 パイプライン処理と無関係に動作するコマンド

項目	コマンド名	備考
映像出力	CombineRGB	
	CombineRGBEx	
画像クリア	IP_ClearAllImg	
	IP_ClearImg	
	IP_Const	
画像転送／アフィン変換	IP_Zoom	
	IP_ZoomExt	
	IP_ZoomS	
	IP_Shift	
	IP_Rotate	
画素変換	WriteConvertLUT	
	IP_ConvertLUT	
	IP_ShiftUp	
	IP_DivConstNume	
	IP_ModConstNume	
	IP_DivConstDenum	
	IP_ModConstDenum	
	IP_ConstSub	
画素間算術演算	IP_Div	
	IP_Mod	
	IP_SubSqare	
濃淡画像特徴量抽出	IP_ProjectLabelGO	
	IP_ProjectLabelGOMinMaxValue	

項目	コマンド名	備考
画像メモリアクセス	OpenImg	
	OpenImgExt	
	CloseImg	
	ReadImg	
	WriteImg	
	SetPixelPointer	
	RefreshImg	
	OpenImgDirect	
	CloseImgDirect	
パイプライン制御	DisablePipeline	
正規化相関	SetCorrTemplate	
	SetCorrTemplateExt	
	IP_Corr	
	IP_CorrPrecise	
グラフィックス	RefreshGraphics	
	DrawString	
	DrawLine	
	DrawSegments	
	DrawLines	
	DrawRectangle	
	DrawPolygon	
	DrawArc	
拡張コンボリューション	IP_SmoothFLTExt	
	IP_EdgeFLTAbsExt	
線分化	ExtractPolyline	
RAW/RGB変換	IP_ConvertRAWtoRGB	
	IP_ConvertRAWtoRGBEx	
VP互換 正規化相関	vpxSetCorrTemplate	
	vpxSetCorrTemplateExt	
	vpxIP_CorrStep	
	vpxIP_CorrPoint	
	vpxIP_CorrPrecise	

【注意事項】

1. パイプラインモードでパイプライン処理とは無関係に動作するコマンドを実行した場合、パイプライン有効／無効で処理の順序が変わる為、処理結果が一致しない場合があります。

2. パイプラインモードで以下のアフィン変換コマンドを実行した場合、パイプライン有効／無効で処理領域が変わる為、処理結果が一致しない場合があります。
　パイプライン有効時には、アフィン変換コマンド実行後の処理領域はアフィン変換の結果領域になります。

　IP_ZoomOut、IP_ZoomOutExt、IP_ZoomIn、IP_ZoomInExt

3. ヒストグラム処理(濃淡)対象が2値画像である場合、以下コマンドは2値画像を符号なし8ビット濃淡画像として処理します。

　IP_ExtractGOFeatures、IP_ExtractLORegionX、IP_ExtractLORegionY、
　IP_ExtractLOArea、IP_ExtractLOAreaExt、IP_ExtractLOGravity

4. パイプライン処理では、1段目のデスティネーション画面と2段目のソース画面を同一の画面に設定しますが、この画面には1段目の処理結果が格納されません。そこで、有効データが格納されている画面と区別する為に、画面データタイプを不定画面としています。
　不定画面は、映像入出力画面およびデスティネーション画面に使用することは可能ですが、ソース画面として使用するとエラーになります。
　不定画面をソース画面として使用する為には、ChangeImgDataType()コマンドで画面データタイプを変更する必要があります。

　以下に不定画面の発生例を示します。ImgSrc0とImgSrc1を加算処理後、2値化処理を実行し、結果をImgDstに格納する処理です。ImgIDは加算処理の処理結果が格納されず、不定画面になります。

```
EnablePipeline();                  // パイプラインモード設定
IP_Add(ImgSrc0, ImgSrc1, ImgID);   // 加算処理
IP_Binarize(ImgID, ImgDst, thr);   // ２値化処理
DisablePipeline();                 // パイプラインモード解除
```

5. パイプラインモードで、2値化に先行する画像処理(濃淡)のデスティネーション画面データタイプが符号付となる論理演算である場合、パイプライン処理しないで実行(パージ)されます。

6. パイプラインモード中の以下コマンドによるウィンドウ操作は禁止します。

　SetWindow、SetAllWindow、ResetAllWindow、EnableIPWindow、DisableIPWindow

9.14.3 パイプライン処理の処理例

以下にパイプライン可能／不可能の例を3例示します。

(1) 画像処理 ＋ 2値化

```
＜パイプライン動作可能な場合＞
EnablePipeline();                   パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);    IP_Addコマンドはペンディング
IP_Binarize(ImgID, ImgDst, thr);    IP_AddとIP_Binarizeをパイプライン実行
DisablePipeline();                  パイプラインモード終了

・IP_Addのデスティネーション画面とIP_Binarizeのソース画面が同じなので、パイプライン処理を行います。
```

```
＜パイプライン動作不可能な場合＞
EnablePipeline();                       パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);        IP_Addコマンドはペンディング
IP_Binarize(ImgSrc0, ImgDst, thr);      パイプライン実行できず、IP_Addを実行後にIP_Binarizeを実行
DisablePipeline();                      パイプラインモード終了

・IP_Addのデスティネーション画面とIP_Binarizeのソース画面が異なるので、パイプライン処理できません。
```

(2) 画像処理 ＋ ヒストグラム

```
＜パイプライン動作可能な場合＞
EnablePipeline();                           パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);            IP_Addコマンドはペンディング
IP_Histogram(ImgID, &Tbl, &Regs, opt);      IP_AddとIP_Histogramをパイプライン実行
DisablePipeline();                          パイプラインモード終了

・IP_Addのデスティネーション画面とIP_Histogramのソース画面が同じなので、パイプライン処理を行います。
```

```
＜パイプライン動作不可能な場合＞
EnablePipeline();                           パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);            IP_Addコマンドはペンディング
IP_Histogram(ImgSrc0, &Tbl, &Regs, opt);    パイプライン実行できず、IP_Addを
                                            実行後にIP_Histogramを実行
DisablePipeline();                          パイプラインモード終了

・IP_Addのデスティネーション画面とIP_Histogramのソース画面が異なるので、 パイプライン処理できません。
```

(3) 画像処理 ＋ 2値化 ＋ ヒストグラム

```
＜パイプライン動作可能な場合＞
EnablePipeline();                       パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);        IP_Addコマンドはペンディング
IP_Binarize(ImgID, ImgID2, thr);        IP_Binarizeコマンドはペンディング
IP_ProjectB0(ImgID2, &TblX, &TblY);     IP_AddとIP_BinarizeとIP_ProjectB0をパイプライン実行
DisablePipeline();                      パイプラインモード終了

・IP_Addのデスティネーション画面とIP_Binarize のソース画面、IP_Binarizeのデスティネーション画面と
IP_ProjectB0のソース画面が同じなので、パイプライン処理を行います。
```

```
＜パイプライン動作不可能な場合＞
EnablePipeline();                       パイプラインモード起動
IP_Add(ImgSrc0, ImgSrc1, ImgID);        IP_Addコマンドはペンディング
IP_Binarize(ImgID, ImgID2, thr );       IP_Binarizeコマンドはペンディング
IP_ProjectB0(ImgID3, &TblX, &TblY);     IP_BinarizeとIP_ProjectB0が
                                        パイプライン実行できないため、IP_AddとIP_Binarizeを
                                        パイプライン実行後に IP_ProjectB0を実行
DisablePipeline();                      パイプラインモード終了

・IP_Binarizeのデスティネーション画面とIP_ProjectB0のソース画面が異なるので、パイプライン処理できません。
```

9.14.4 パイプライン処理による不定画面

パイプライン処理で、結果が格納されずデータが不定になった画面を不定画面と言います。不定画面が発生する手順を除いて、不定画面を画像処理ソース画面として使用するとエラーになります。
また、不定画面をデスティネーション画面に使用することは可能です。

```
＜不定画面の発生例＞
ImgSrc0とImgSrc1をADD処理後、2値化処理を実行し、結果をImgDstに格納します。
このとき、ImgIDには処理結果が格納されず、ImgIDは不定画面となります。

EnablePipeline();
IP_Add(ImgSrc0, ImgSrc1, ImgID);
IP_Binarize(ImgID, ImgDst, thr);
DisablePipeline();
```

9.15 ２値マッチングフィルタ

２値マッチングフィルタコマンドは、２値画像に対してテンプレートでマッチングフィルタ処理を行い、その処理結果を出力します。
入力画像に対し、９×９画素のテンプレートでAND／XNORのマッチングフィルタ処理を行います。処理結果は、”1”となる画素の総数となります。

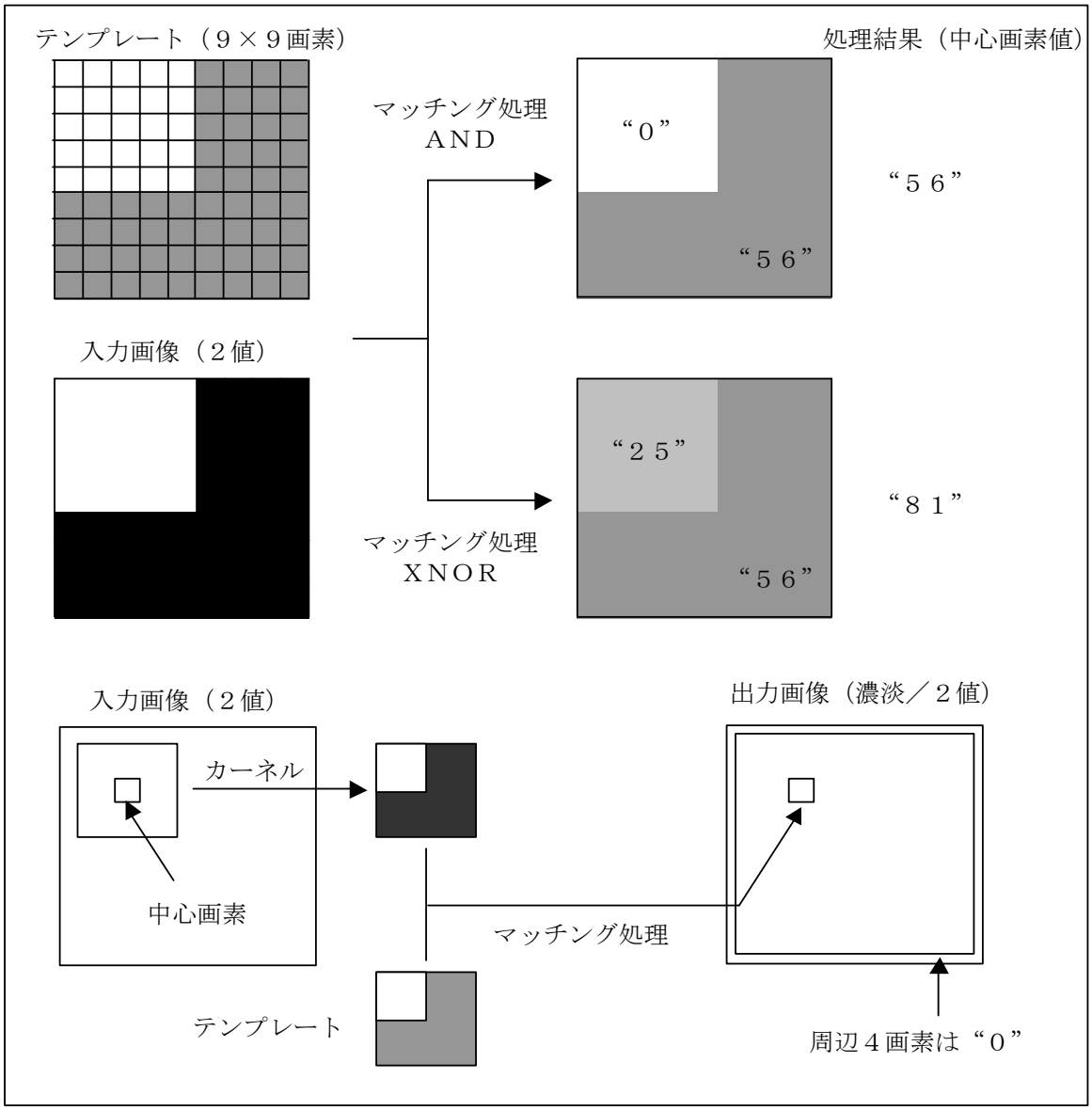


図9-22　２値マッチングフィルタ処理

２値マッチングフィルタ処理は、９×９テンプレートを用いて画像処理を行うため、画像処理対象領域の周辺４画素に画像処理結果無効の領域が生じます。この画像処理無効領域は”０”となります。

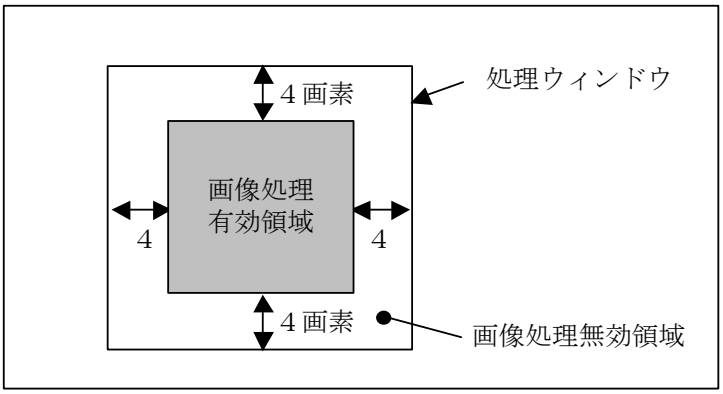


図9-23　２値マッチングフィルタ領域

9.16 正規化相関

正規化相関処理とは、濃淡画像によるパターンマッチングのことであり、ユーザの登録した濃淡テンプレートを、処理対象の画面の中から探し当てる(座標、相関値)ことができます。

正規化相関では、テンプレートと対象画面間で下記演算処理を行います。相関値(r)は0～1の値を取り、濃淡テンプレート(g)とサーチ対象画面(f)が完全に一致すると1となります。

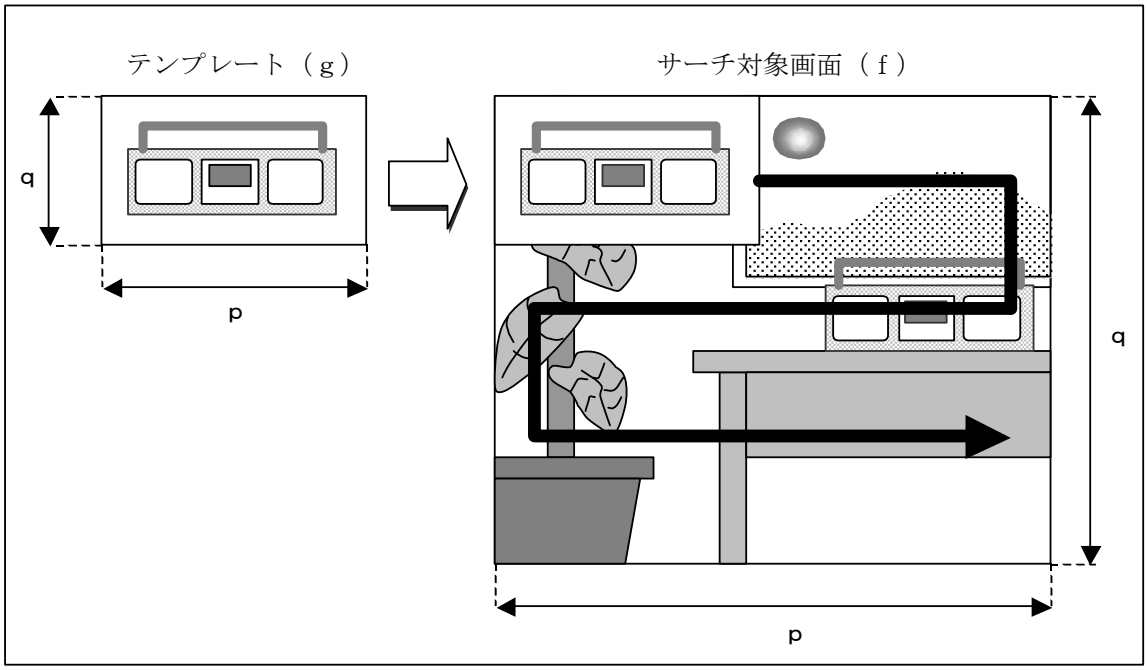


図9-24 正規化相関の概要

$$r^2 = \frac{\{ N\sum_{u}^{p}\sum_{v}^{q} f(u,v)g(u,v) - \sum_{u}^{p}\sum_{v}^{q} f(u,v) \times \sum_{u}^{p}\sum_{v}^{q} g(u,v) \}^2}{[N\sum_{u}^{p}\sum_{v}^{q} f(u,v)^2 - \{ \sum_{u}^{p}\sum_{v}^{q} f(u,v) \}^2] [N\sum_{u}^{p}\sum_{v}^{q} g(u,v)^2 - \{ \sum_{u}^{p}\sum_{v}^{q} g(u,v) \}^2]}$$

上記の式のうち、以下の項の計算はハードウェアによって局所並列的に実行されます。
相関値は、これらの結果を用いてソフトウェアで計算することにより求められます。

(1) $\sum_{u}^{p}\sum_{v}^{q} f(u,v)$　　(2) $\sum_{u}^{p}\sum_{v}^{q} g(u,v)$　　(3) $\sum_{u}^{p}\sum_{v}^{q} f(u,v)^2$　　(4) $\sum_{u}^{p}\sum_{v}^{q} g(u,v)^2$

(5) $\sum_{u}^{p}\sum_{v}^{q} f(u,v)\, g(u,v)$

r ： 相関値
u ： X座標
v ： Y座標
N ： テンプレートの有効画素数

9.16.1 正規化相関実行手順

正規化相関処理の処理手順は、

①テンプレート登録
②サーチモード指定（サーチ個数、サーチ方向やサーチ精度等）
③正規化相関実行

の3段階です。

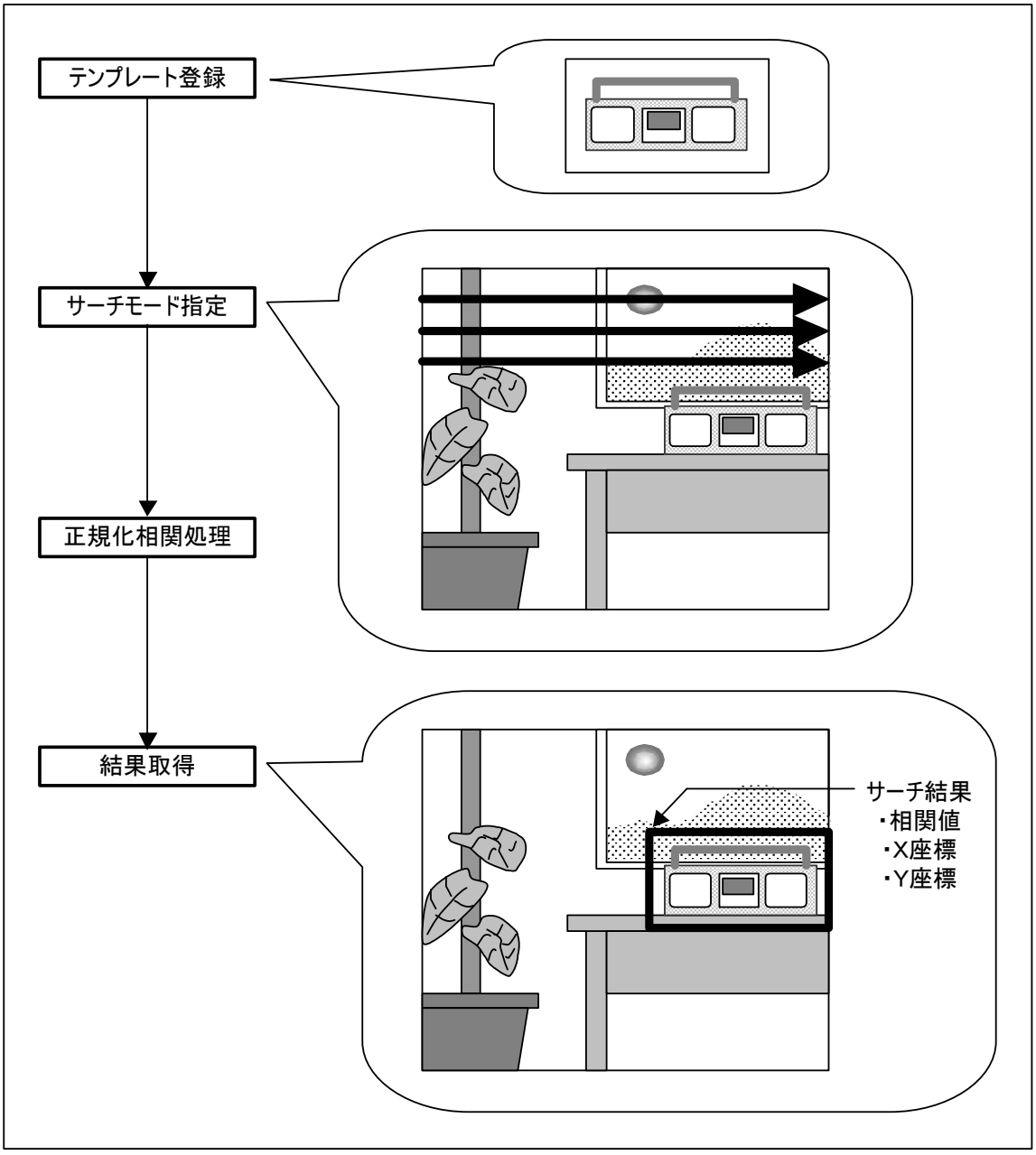


図9-25 正規化相関実行手順

以下に正規化相関の実行手順フローを示します。

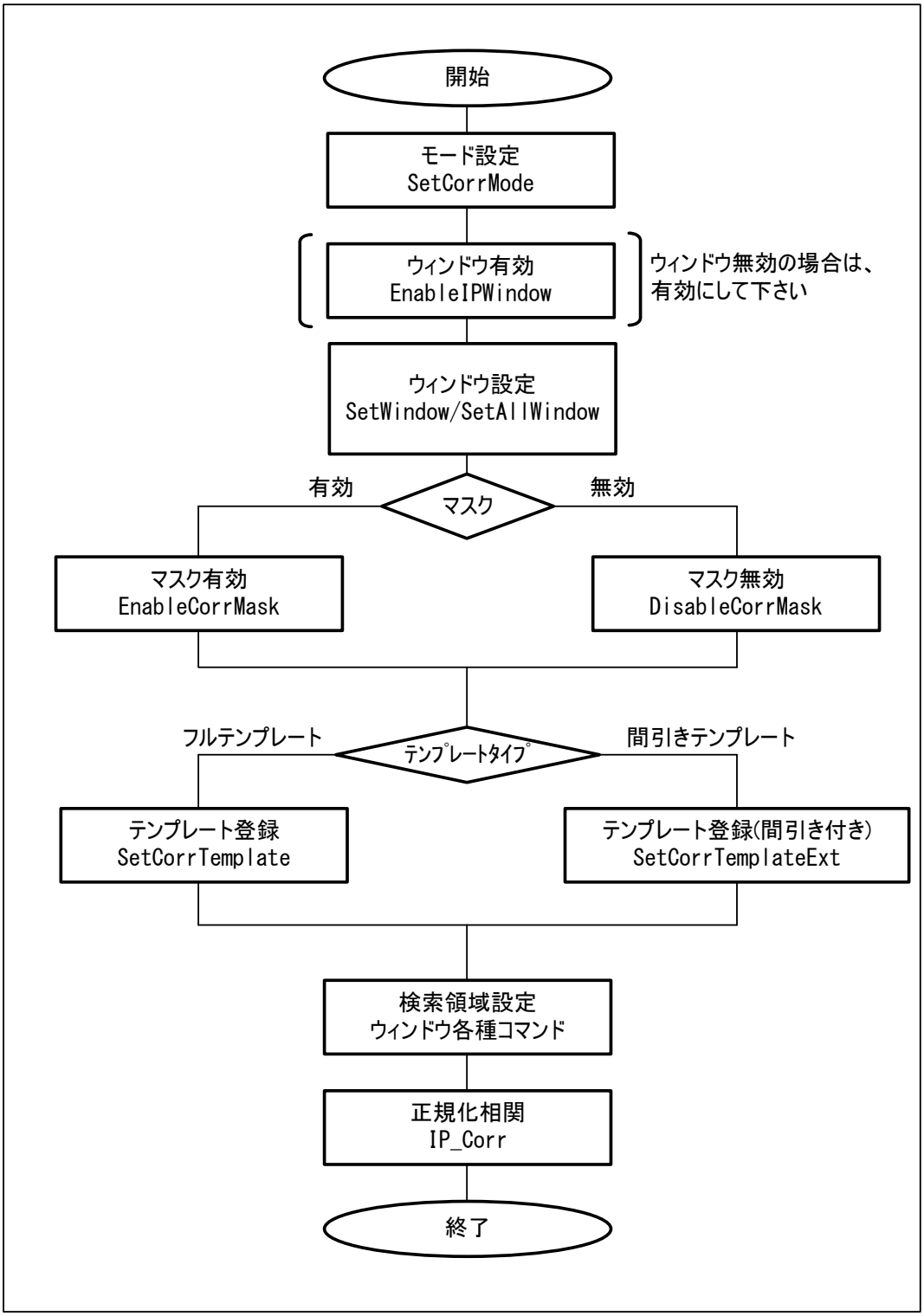


図9-26 正規化相関実行手順フロー

9.16.2 テンプレートタイプ

正規化相関のテンプレートには、2種類のタイプがあります。

(1) フルテンプレート

SetCorrTemplate()コマンドで登録されたテンプレート。フルテンプレートは、IP_Corr()実行時に、処理対象画面内全画素を処理する(フルサーチ)ので、高精度な処理が可能です。

(2) 間引きテンプレート

SetCorrTemplateExt()コマンドで登録されたテンプレートです。間引きテンプレートは、X／Y方向の間引き率を登録でき、この間引き率により、テンプレートデータおよび処理対象画面を間引いて演算処理するので、精度はフルテンプレートに対して落ちるが演算回数が減るため、高速処理が可能です。

処理対象画像に応じて、これら2つのテンプレートタイプを評価し使用します。

9.16.3 正規化相関マスク

正規化相関マスクは、テンプレート内で濃度0の画素を処理対象外にします。よって、矩型でない任意の形をテンプレートとしたい場合に有効です。矩型のテンプレート内で必要物体部以外を「0」で塗りつぶしておき、IP_Corr()を実行する前に、EnableCorrMask()を実行すれば、「0」画素部分は処理されないため、任意形のテンプレートで処理したことと同じになります。正規化相関マスクはEnableCorrMask()コマンドで設定を行います。

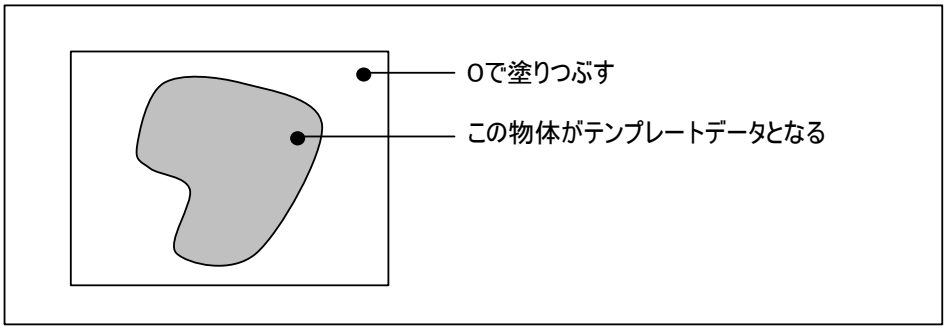


図9-27 正規化相関マスク

9.17 グラフィックス

グラフィックス処理では、文字列、図形描画等を行うことができます。本コマンドでの座標系は画像メモリアクセス（SYS_WIN）ウィンドウ相対であり、ウィンドウ始点が座標0となります。
以下にグラフィックス処理の実行手順フローを示します。

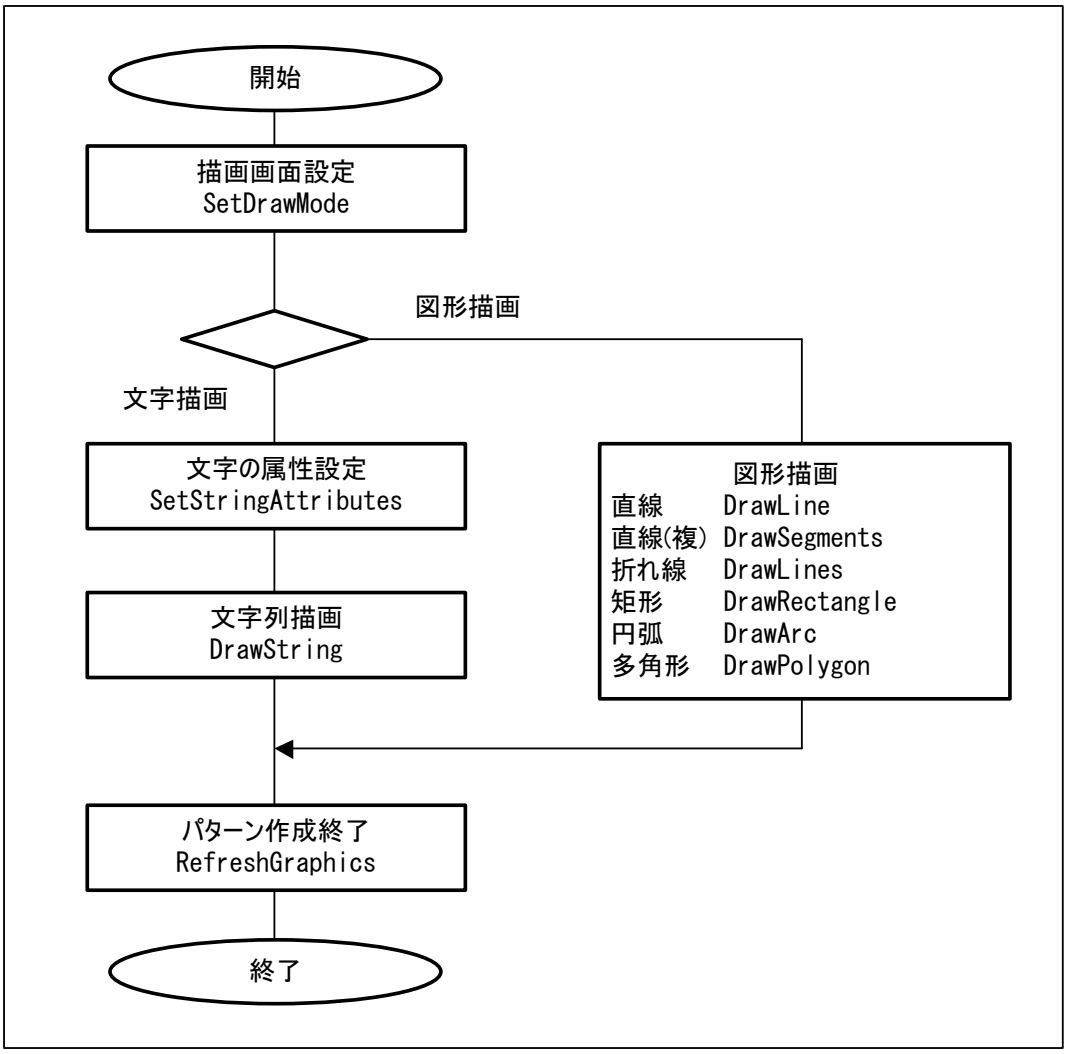


図9-28 グラフィックス処理の実行手順

9.18 線分化

画像処理コマンドでは線分化コマンドを用意しています。線分化は、2値画像の物体に対して線分列外周座標を抽出し、線分列外周座標から特徴量の抽出を行います。

線分列外周座標の抽出は、**ExtractPolyline()**コマンドで行います。**ExtractPolyline()**コマンドでは、指定画面の探索開始座標(sx,sy)からラスタースキャンで指定した濃度の探索対象物体の外周探索開始座標を取得し、取得した外周探索開始座標から時計回りに探索対象物体の外周を探索して、線分列外周座標を取得します。

対象物体の特徴量は、**PolyArea()**、**PolyPerm()**、**PolyGrav()**、**PolyFeature()**コマンドで抽出します。

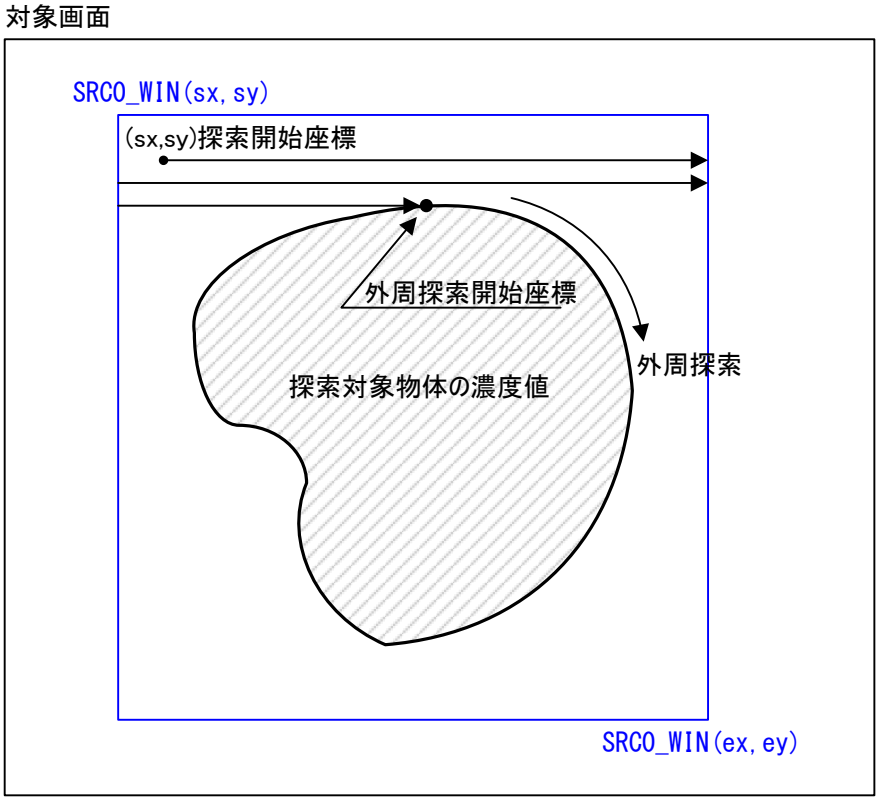


図9-29　線分化処理

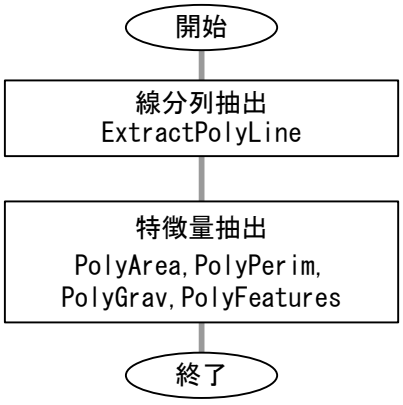


図9-30　線分化処理フロー

9.19 ２値画像の穴埋め

画像処理コマンドでは穴埋めコマンドを用意しています。穴埋めは、２値画像の物体に対してラベリング処理を行い、指定した面積しきい値の条件を満たすオブジェクト（穴）を塗りつぶす処理を行います。

下の図は、対象画面中の穴埋め対象物が”白”オブジェクト、穴が”黒”オブジェクトの例です。
穴埋め処理は、”黒”オブジェクトに対してラベリング処理を行い、面積が最小面積以上最大面積以下のものを塗りつぶします。

尚、穴埋め処理はラベリング処理を用いているため、２５５個以上の穴埋めはできません。

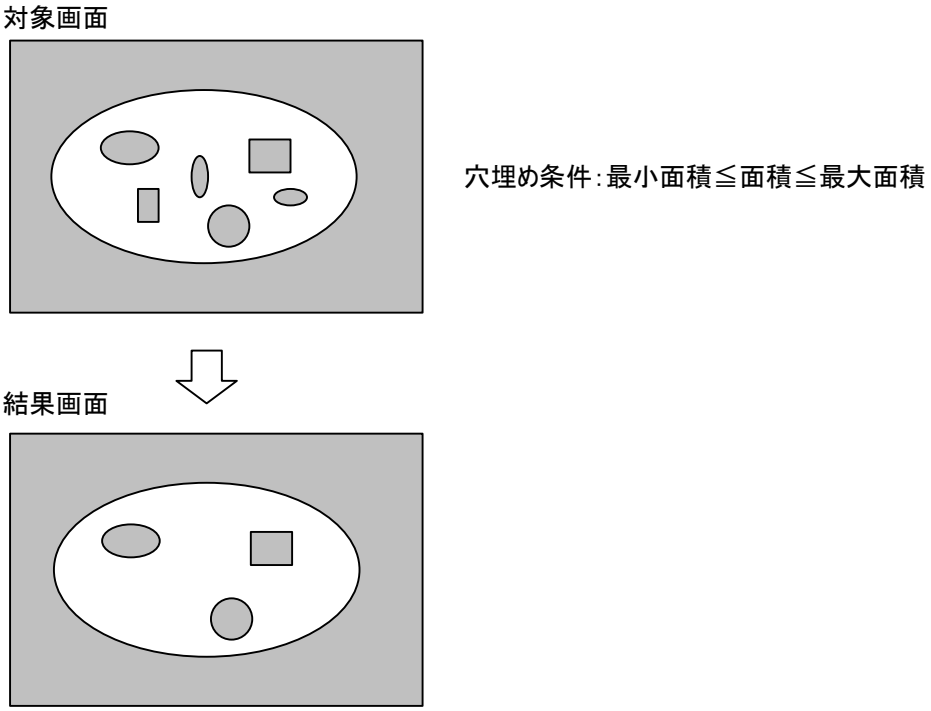


図9-31 ２値画像穴埋め処理

9.20 高速ラスタスキャン正規化相関サーチ

本SDKでは「9.16 正規化相関」の正規化相関コマンド以外にラスタースキャン専用でサーチを高速に実行できるコマンドが使用可能です。

以下に高速ラスタスキャン正規化相関サーチの実行手順フローを示します。

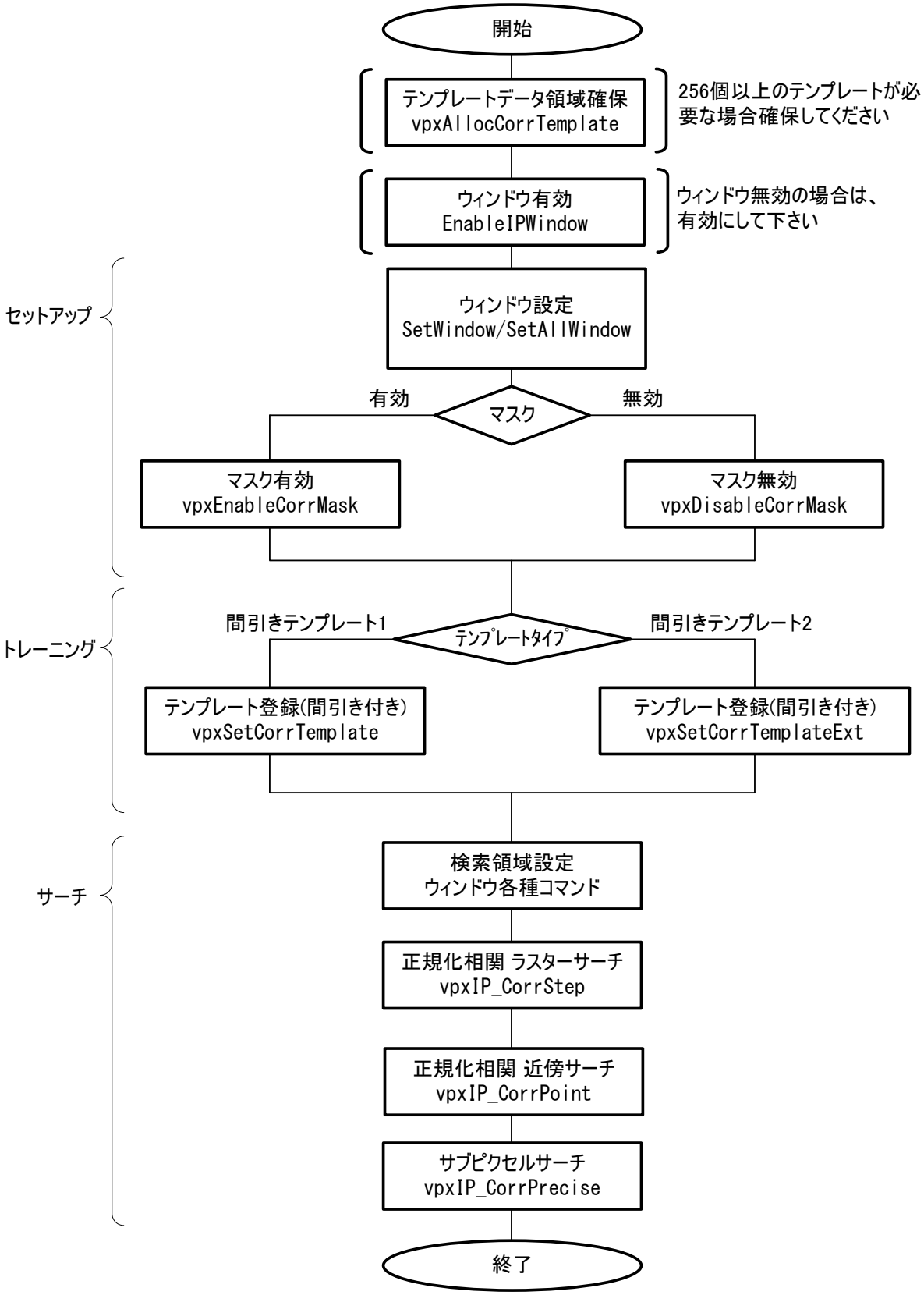


図9-32 高速ラスタスキャン正規化相関サーチの実行手順フロー

9.20.1 テンプレートデータ領域管理コマンド

正規化相関サーチを行う際、使用するテンプレートの個数分、テンプレート特徴量データ領域が必要です。デフォルトでは、256個の領域が確保されていますが、それ以上のテンプレートを使用する場合は、テンプレート特徴量データ領域確保コマンドにより、テンプレート特徴量データ領域を確保してください。

(1) テンプレートデータについて

トレーニングによりテンプレートが登録されると正規化相関サーチに必要なデータはテンプレート番号（テンプレートID）で管理されます。

正規化相関サーチのテンプレートのデータは、テンプレート特徴量と画像が必要です。トレーニングを行うと指定された画像からテンプレート特徴量を計算し、テンプレート特徴量データ領域にセーブされます。指定された画像は、そのまま画像メモリにあり、その画面番号とウィンドウのデータだけがテンプレート特徴量データ領域にセーブされます。つまり、テンプレート特徴量データと画像データは別々の領域にあるということで、特に画像データは誤って別の画像に書換えられる可能性があります。テンプレートデータ内の特徴量データと画像メモリの内容が異なると正常にサーチできなくなるのでテンプレートに指定する画面の管理には十分注意が必要です。

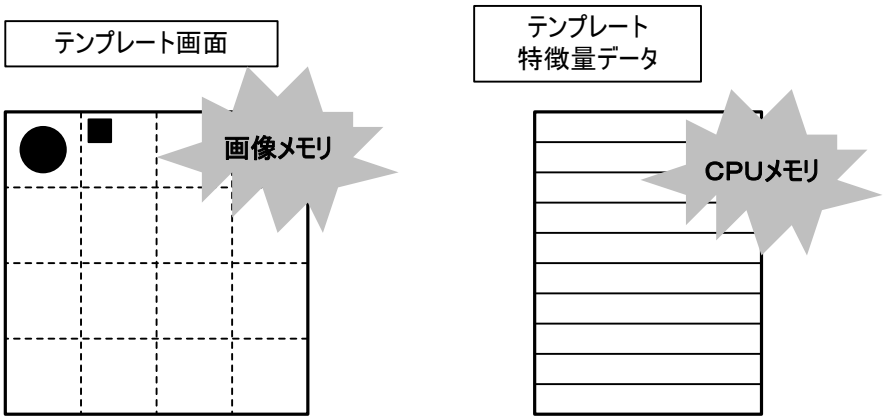


図9-33 テンプレートデータ

9.20.2 セットアップとトレーニング

正規化相関サーチは、セットアップ、トレーニング、サーチという3つのステップで実行されます。ここでは、セットアップとトレーニングのコマンドについて説明します。

(1) セットアップと画像の切出し

正規化相関サーチのセットアップとは、サーチを行う際のテンプレートを準備する操作のことです。
テンプレートとして登録したい画像を入力画像から切出したり、目的とする画像を画像描画コマンドで描画したりしてテンプレートを用意します。
正規化相関サーチでは、セットアップコマンドとして特別に用意しているものはありません。画像転送、画像縮小コマンドで画像をテンプレート画面として確保した画面に画像を切出し(転送)して下さい。また、画像描画コマンドを使用して目的の図形の画像をテンプレート画面に直接描画するか、別の画面に描画したものを画像転送、画像縮小コマンドでその画像を切出して下さい。

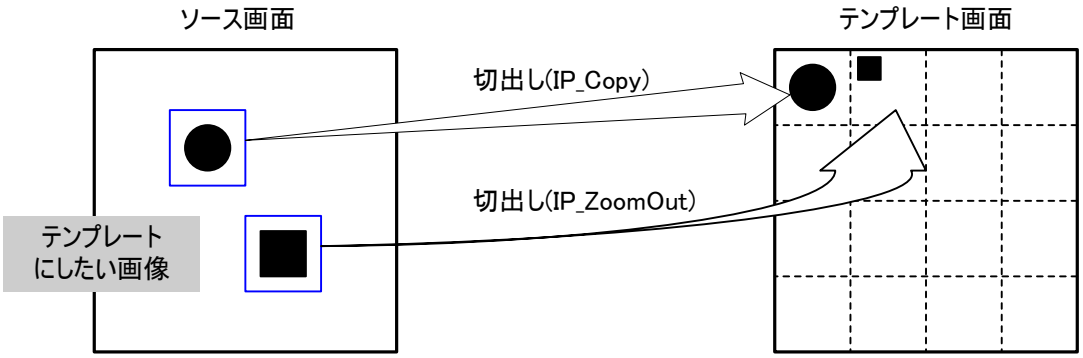


図9-34 画像の切出し

(2) テンプレートの情報量と処理時間

正規化相関サーチは、画像処理プロセッサとNVP-LinuxのCPUにより行っています。画像処理プロセッサではテンプレートカーネル内の積和演算を行います。NVP-LinuxのCPUでは、積和演算の結果から相関値を計算し、テンプレートカーネルを移動します。

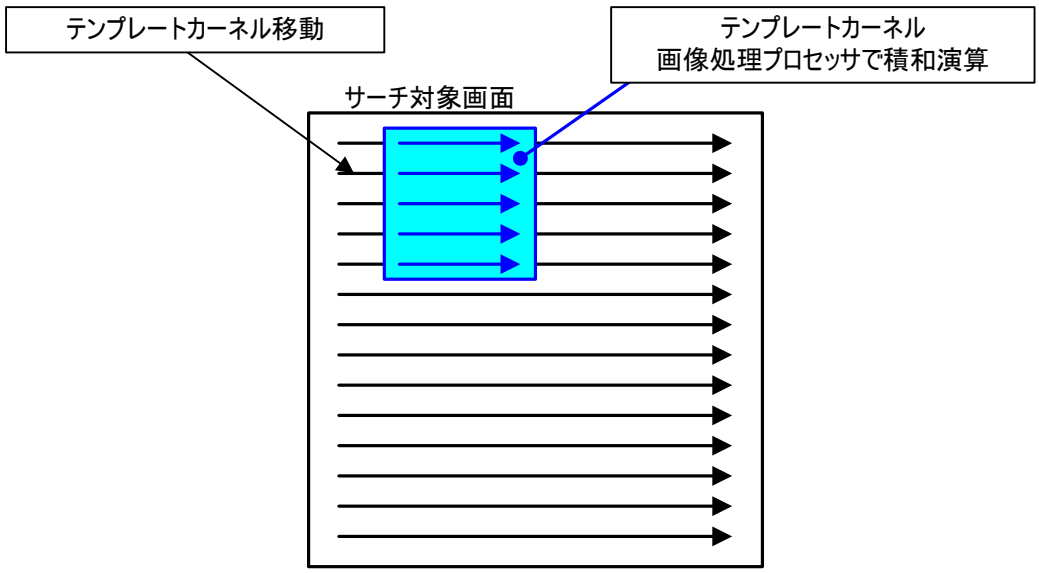


図9-35 正規化相関処理

画像処理プロセッサは、テンプレートカーネル内の積和演算を実行します。そしてNVP-LinuxのCPUでテンプレートカーネルを移動しながら、それぞれのポイントでの相関値を求め、最大の相関値とその座標を求めます。例えば、積和演算の実行が4ns/画素だとするとサーチ時間はハードのオーバヘッド等を無視して単純に計算すると

サーチ時間 ＝ 4ns × テンプレートの大きさ × カーネルの移動回数

の式で計算できます。例えば、テンプレートの大きさが128×128画素で512×480のサーチ対象画面をサーチするとすると

4e-9 × (128 × 128) × (512 × 480) = 16秒

16秒もかかることになります。
処理時間を縮めるにはテンプレートの大きさを小さくするかカーネルの移動回数を減らせばいいことになります。
画像処理コマンドでは、テンプレートに関しては、テンプレートの大きさを小さくする代わりに情報量を減らすことで等価的にテンプレートの大きさを小さくできるようにしています。情報量を減らすということは、サーチのときのカーネル内の積和演算のデータのサンプリングをハード的に間引いて実行するということです。また、カーネルの移動も間引いて実行できるようになっています。
上記の例で、テンプレート情報量を縦8画素、横8画素間引き、カーネル移動を縦8画素、横8画素間引くと

4e-9 × (128/8 × 128/8) × (512/8 × 480/8) = 0.001秒

16秒の処理が4096分の1の4ミリ秒でできることになります。
また、このときのテンプレートの情報量は

128/8 × 128/8 = 256

になります。

このように、実際の処理は情報量を減らして高速化を図ります。

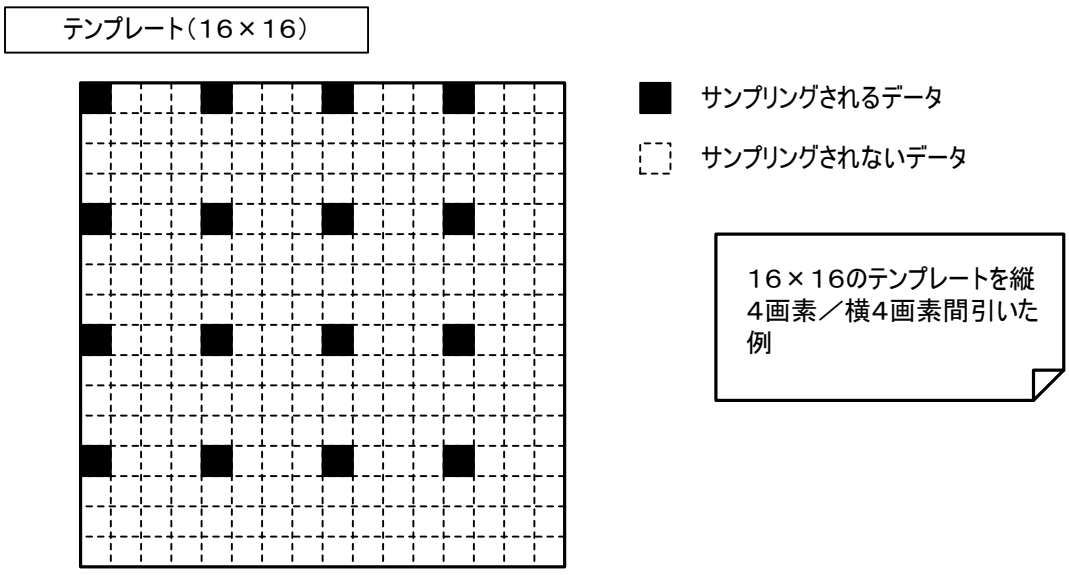


図9-36 テンプレートの間引き

(3) テンプレートマスクの設定

テンプレート画像は矩形領域しか設定できません。しかし、テンプレート画像として矩形以外の形の領域を設定したい場合があります。そこで、テンプレート画像にはマスク（不感帯）領域を設定できる機能があります。テンプレート画像のマスク設定された領域は、ハード処理される積和演算で計算の対象から除外されます。

設定したいマスク領域のテンプレート画像の値を「0」にすることでその領域がマスク領域になります。通常は、円等の簡単な図形でマスクしますが、複雑なマスクも設定可能です。

下図にマスクの設定例を示します。

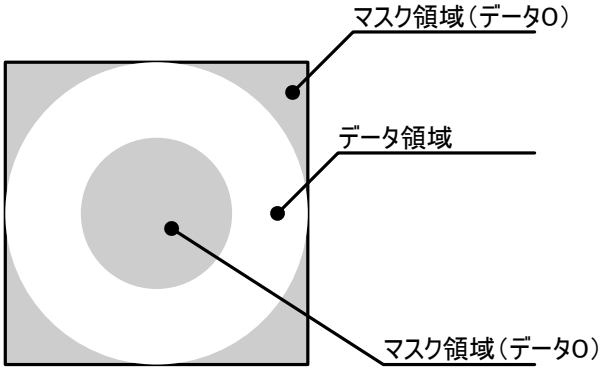


図9-37 マスクの設定例

次にマスクの設定手順を説明します。まず、テンプレートを切出し、テンプレートのマスクしたい領域に「0」を書込みます。
次に**EnableCorrMask**()コマンドでマスクモードを有効にします。
そして、**SetCorrTemplate**()または、**SetCorrTemplateExt**()コマンドでトレーニング（テンプレートの登録）を行います。
マスクを解除する場合は、**DisableCorrMask**()コマンドでマスクモードを解除して下さい。
マスクなしのテンプレートに戻したい場合は、テンプレート画像をもう一度切出し、トレーニングを行って下さい。

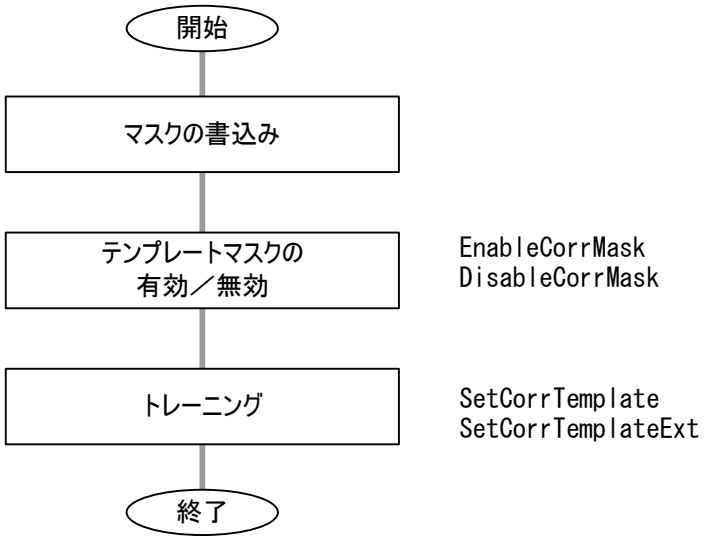


図9-38 テンプレートマスクの設定フロー

9.20.3 正規化相関サーチ

正規化相関サーチは、セットアップ、トレーニング、サーチという3つのステップで実行されます。ここでは、サーチのコマンドについて説明します。

(1) 高速サーチの方法

サーチコマンドには、**vpxIP_CorrStep()**, **vpxIP_CorrPoint()**, **vpxIP_CorrPrecise()**コマンドがあります。
サーチコマンドは、テンプレートカーネルを移動しながら相関値を演算し最大の相関値と座標を見つけ出す処理を行っています。
最も単純なサーチでは、前項で説明しているように処理時間がユーザの要求に応えられるものではありません。そのため、テンプレートカーネルの間引きとサーチの間引きを行うわけです。前項ではテンプレートカーネルの間引きについて説明しましたが、ここではテンプレートカーネルを移動する時の間引きについてと高速化の方法を説明します。

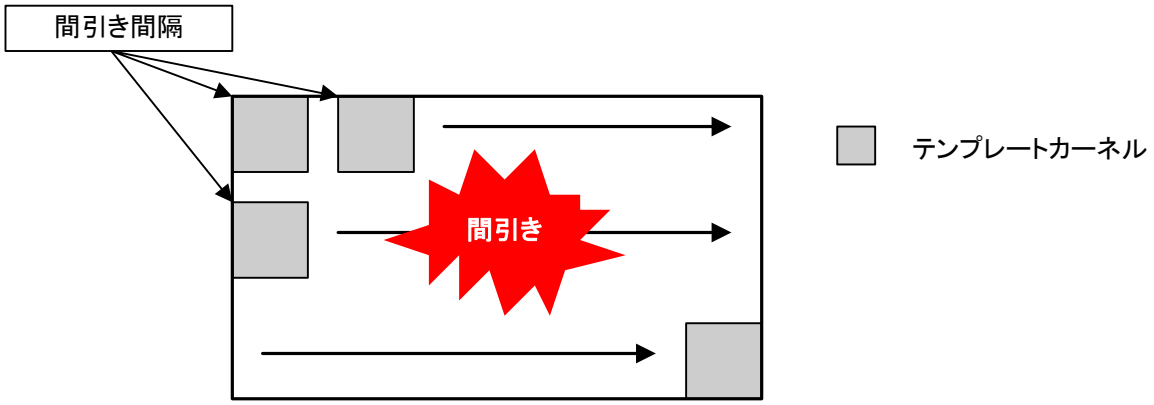


図9-39 vpxIP_CorrStepコマンド

vpxIP_CorrPoint()コマンドは、入力された座標の近傍領域でサーチを行います。通常は、**vpxIP_CorrStep()**コマンドで間引いたぶんの補間をするためのサーチとして使用します。

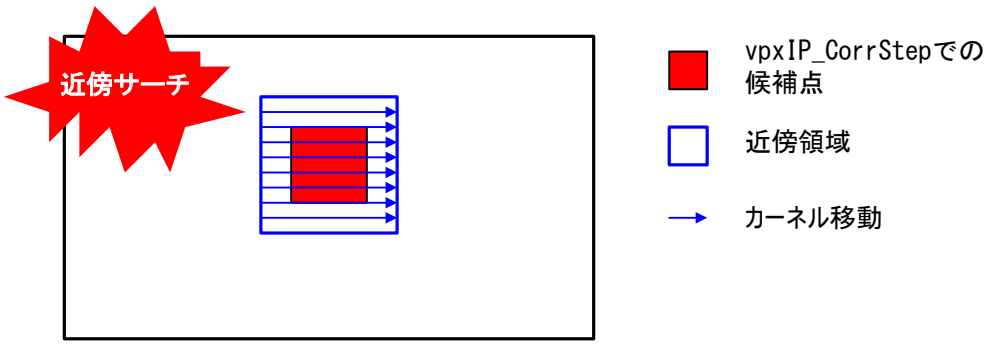


図9-40 vpxIP_CorrPointコマンド

vpxIP_CorrPrecise()コマンドは、入力された座標の近傍領域の相関値からサブピクセルの位置を計算します。
通常は、vpxIP_CorrPoint()コマンドで得られた最終位置座標をvpxIP_CorrPrecise()コマンドに入力します。

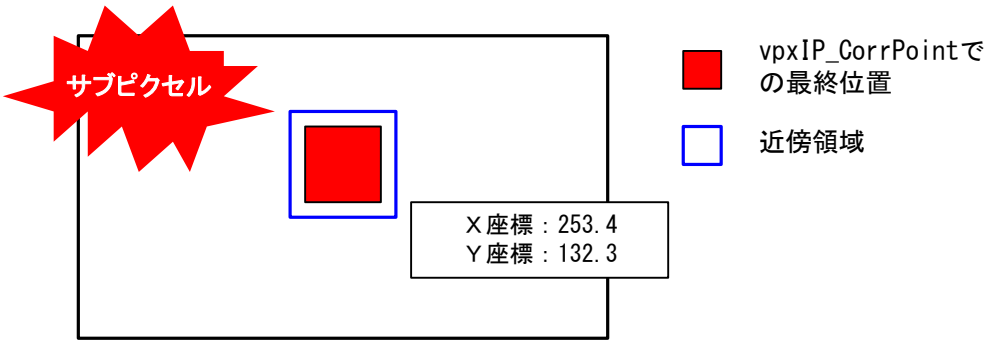


図9-41　vpxIP_CorrPreciseコマンド

(2) 並列演算カーネルによるサーチの高速化

画像処理プロセッサは、正規化相関演算用の積和演算器を16組内蔵しています。その並列演算カーネルを使用して1つのテンプレートカーネルに対して16組(4×4)の積和演算を並列に実行することができます。
並列演算カーネルを使用すると処理時間は原理的に並列演算カーネルを使わない時とくらべ1／16になります。
また、原理的に4×4カーネルの間隔は通常は1画素ですが、テンプレートカーネルの間引きを行うと4×4カーネルの間隔はテンプレートカーネルの間引き間隔になります。

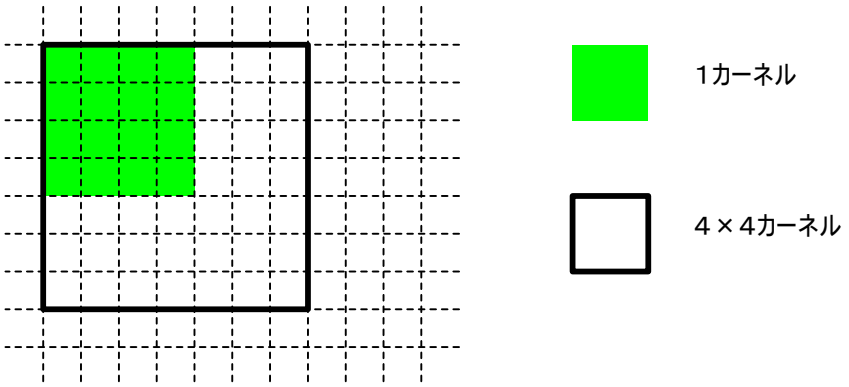


図9-42　4×4カーネル

サーチの高速化には上記の並列演算カーネルを使用する方法とテンプレートとサーチ移動の間引きによる方法がありますが、両方の方法を使用することで画像処理ボードの最大の性能を引き出すことができます。なお、並列演算カーネルとテンプレートとサーチ移動の間引きを同時に使用するには以下に示す条件を満足しなければなりません。

① サーチ移動の間引き間隔がテンプレートカーネルの間引き間隔と同じ
　または、その整数分の1
　または、その2倍

①の条件は、テンプレートカーネルの間引きを行うと4×4カーネルの間隔はテンプレートカーネルの間引き間隔になることに起因しています。
vpxIP_CorrStep(), vpxIP_CorrPoint()コマンドは、上記①の条件のとき自動的に並列演算カーネルモードで処理を実行します。

(3) サーチ除外領域設定とマッチング候補点

vpxIP_CorrStep()コマンドでは、テンプレートカーネルを移動しながら相関値の計算を行い、マッチングポイントを探します。通常は、間引いてサーチを行うので、探し出したポイントは正確なマッチングポイントではありません。その正確ではないマッチングポイントをマッチング候補点と呼びます。

vpxIP_CorrStep()コマンドでは、探し出すマッチング候補点の個数を任意に設定できるようになっています。しかし、複数のマッチング候補点をサーチすると、近傍領域にその候補点が集中します。

そこで、マッチング候補点がその近傍領域に集中しないようにサーチ除外領域を設定します。サーチ除外領域として設定した距離以内にマッチング候補点が2つ以上存在しないようにポイントを探します。

サーチ除外領域を設定／解除は、vpxSetSearchDistance()コマンドで行います。

9.21 直線抽出

画像処理コマンドではハフ変換による直線・矩形抽出コマンドを用意しています。以下にその使用方法を説明します。

9.21.1 ハフ変換による直線抽出

画像処理コマンドでは、ハフ変換を用いて離散的な座標データから直線を抽出します。ハフ変換の演算はNVP-LinuxのCPUで高速に処理します。

また、直線抽出コマンドでは下図のようにρ－θで示される直線が抽出されます。

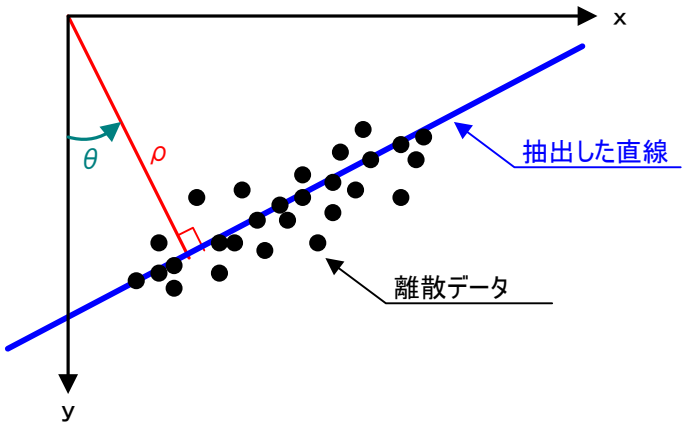


図9-43 離散データからの直線抽出

離散データからの直線抽出では、まず、上の図に示すような離散データの座標を2値化した画像から**IP_ProjectBORegionX()**, **IP_ProjectBORegionY()**コマンドなどの座標抽出を行い抽出します。

次に、その座標データ群からハフ変換によりρ－θで示される直線を抽出します。

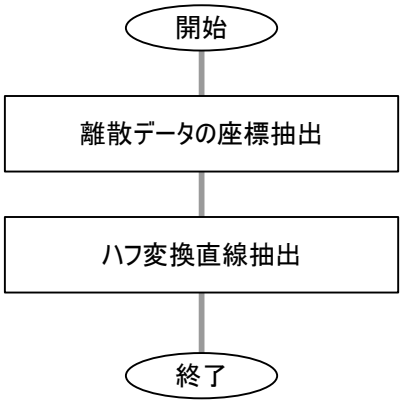


図9-44 離散データからの直線抽出フロー

9.21.2 ハフ変換直線からの矩形算出

画像処理コマンドでは、ハフ変換で抽出されたρ－θで示される4つの直線からそれぞれの交点座標を算出し、矩形の頂点座標や角度等を算出することができます。

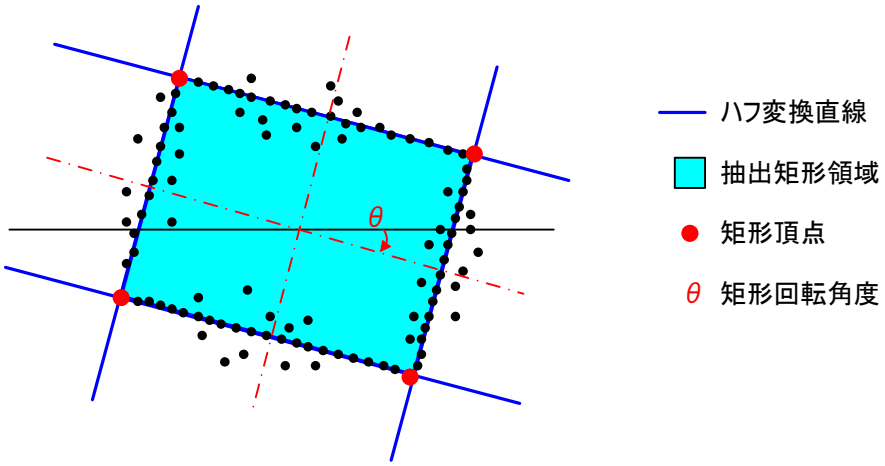


図9-45 ハフ変換直線からの矩形算出

GetCrossPoint(),GetRectPoint(),GetRectCenter(),GetAnglePoint4(),GetAnglePoint2()コマンドにより、ハフ変換で抽出されたρ－θで示される2つまたは4つの直線からそれぞれの交点座標の算出、矩形の頂点座標や角度等の算出を行うことができます。

9.22 イメージキャリパ

画像処理コマンドではキャリパコマンドを用意しています。キャリパとはノギスのことで、画像処理により対象物のエッジを検出し、その対象物のエッジで平行なペアをサーチし、2つのエッジ間の距離やその中心位置を計測することができます。
以下にその使用方法を説明します。

9.22.1 イメージキャリパによる寸法計測

以下にイメージキャリパコマンドによる寸法計測のフローを示します。例は、集積回路（IC）パッケージのリード幅やリード間隔を求める場合です。

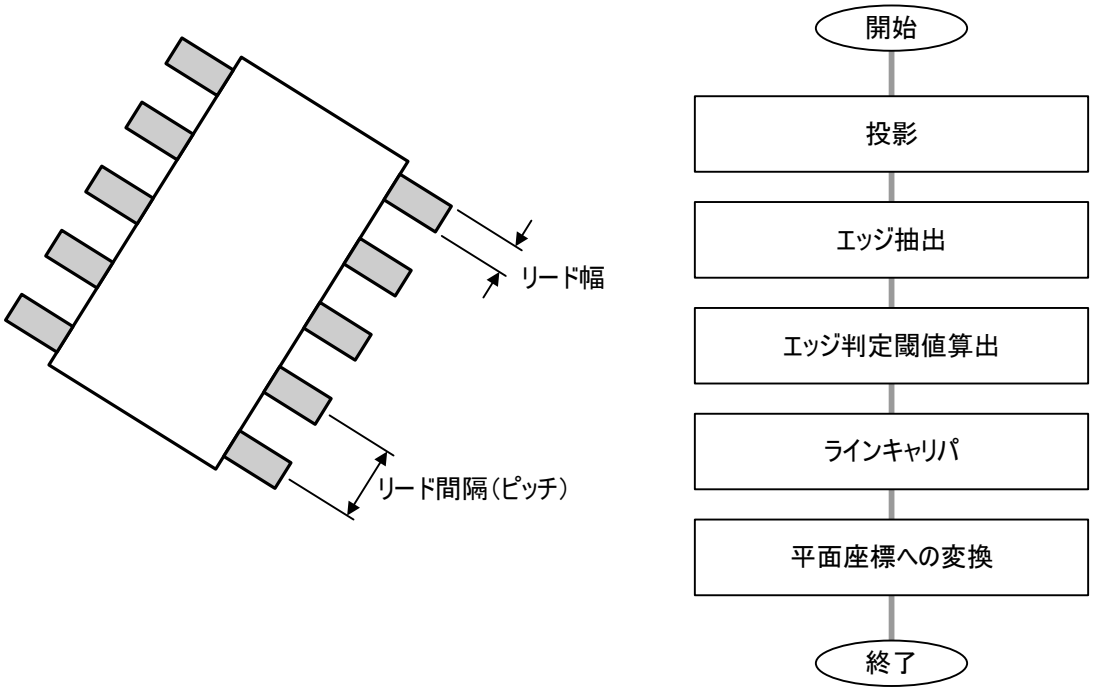


図9-46 寸法計測フロー

まず、ProjectLine()コマンドにより、ICのリードに沿って投影を行い、平面のデータをラインのデータに変換します。次にLineEdgeFilter(),LineEdgeFilterExt()コマンドによりエッジ抽出を行います。エッジデータからGetCaliperScore()コマンドでエッジ判定閾値を算出し、LineCaliper()コマンドでエッジ判定閾値に従い、アップエッジとダウンエッジのペアをサーチします。そして、CaliperLPtoSP()コマンドによりラインデータを平面のデータに変換して処理が完了します。そのデータからリード幅とリード間隔を計算します。

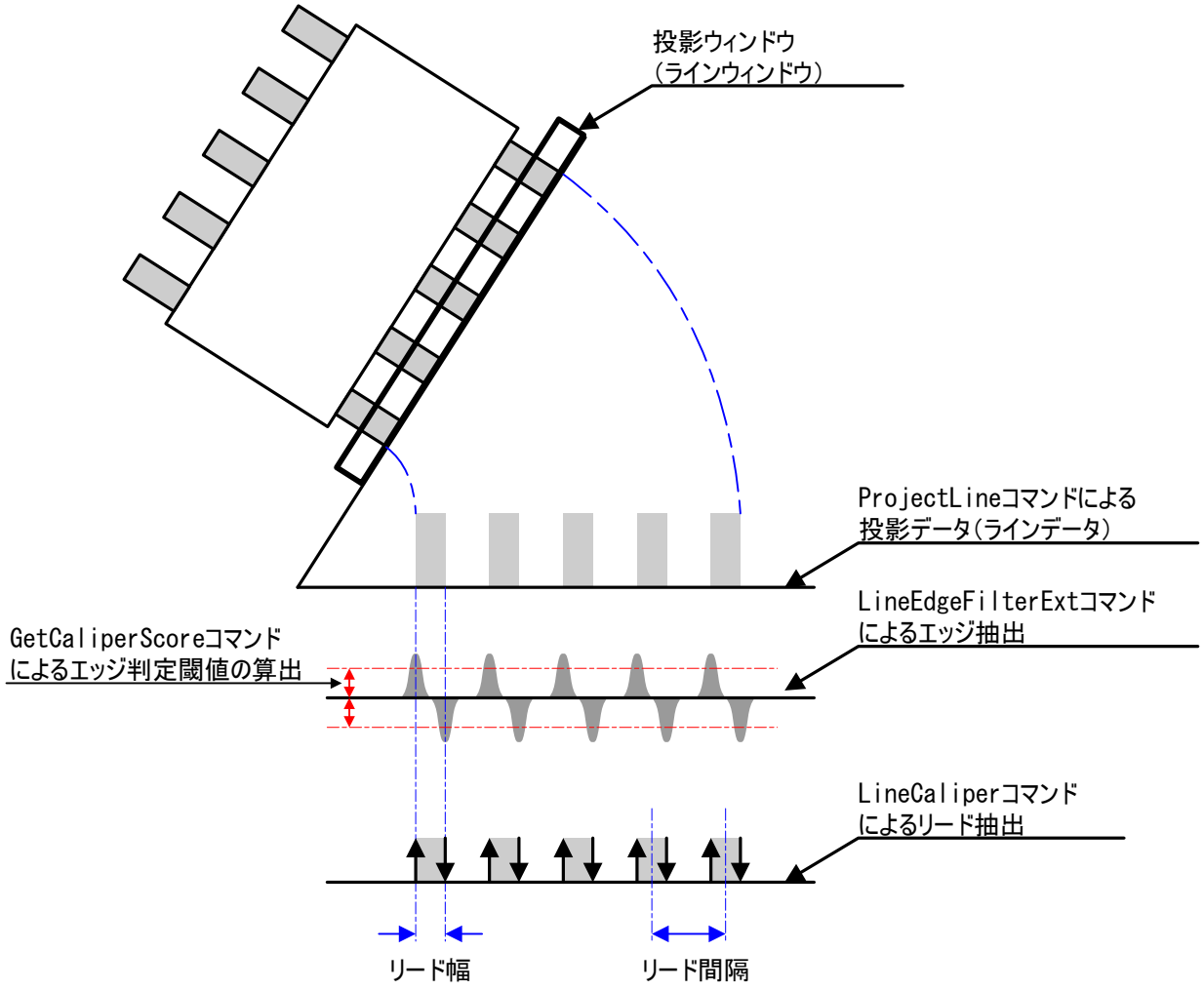


図9-47　寸法計測方法

9.22.2 ラインウィンドウについて

ProjectLine()コマンドで投影を行う場合やCaliperLPtoSP()コマンドによりラインデータを平面のデータに変換する場合、投影を行う領域を指定する必要があります。その場合の領域は、ラインウィンドウにより指定します。通常の画像処理コマンドのウィンドウは、傾斜した矩形領域を設定することはできませんが、ProjectLine()コマンドでは、ラインウィンドウにより傾斜した矩形領域を設定し、傾斜した領域を投影することができます。

以下に、ラインウィンドウの構造体を示します。

```
typedef struct {
    short     sx;       // 開始点Ｘ座標
    short     sy;       // 開始点Ｙ座標
    short     ex;       // 終了点Ｘ座標
    short     ey;       // 終了点Ｙ座標
    short   leng;       // 投影幅
    short    opt;       // オプション（未使用）
} LINEWINDOW;
```

次に、ラインウィンドウの例を挙げながら構造体の各メンバーについて説明します。

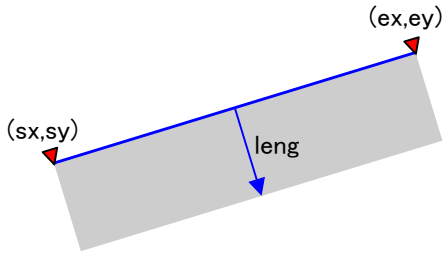


図9-48 ラインウィンドウ

ラインウィンドウでは、(sx,sy)と(ex,ey)で結ばれる直線で投影領域を指定し、lengで投影する画素数を指定します。(sx,sy)と(ex,ey)で結ばれる直線上のポイントが投影の開始ポイントになり、直線上のポイントに沿って(sx,sy)と(ex,ey)で結ばれる直線と垂直な方向にlengで指定される画素数分画素データの濃度累積が行われなす。
また、lengは(sx,sy)と(ex,ey)で結ばれる直線の状態と符号により濃度累積する方向が変わります。

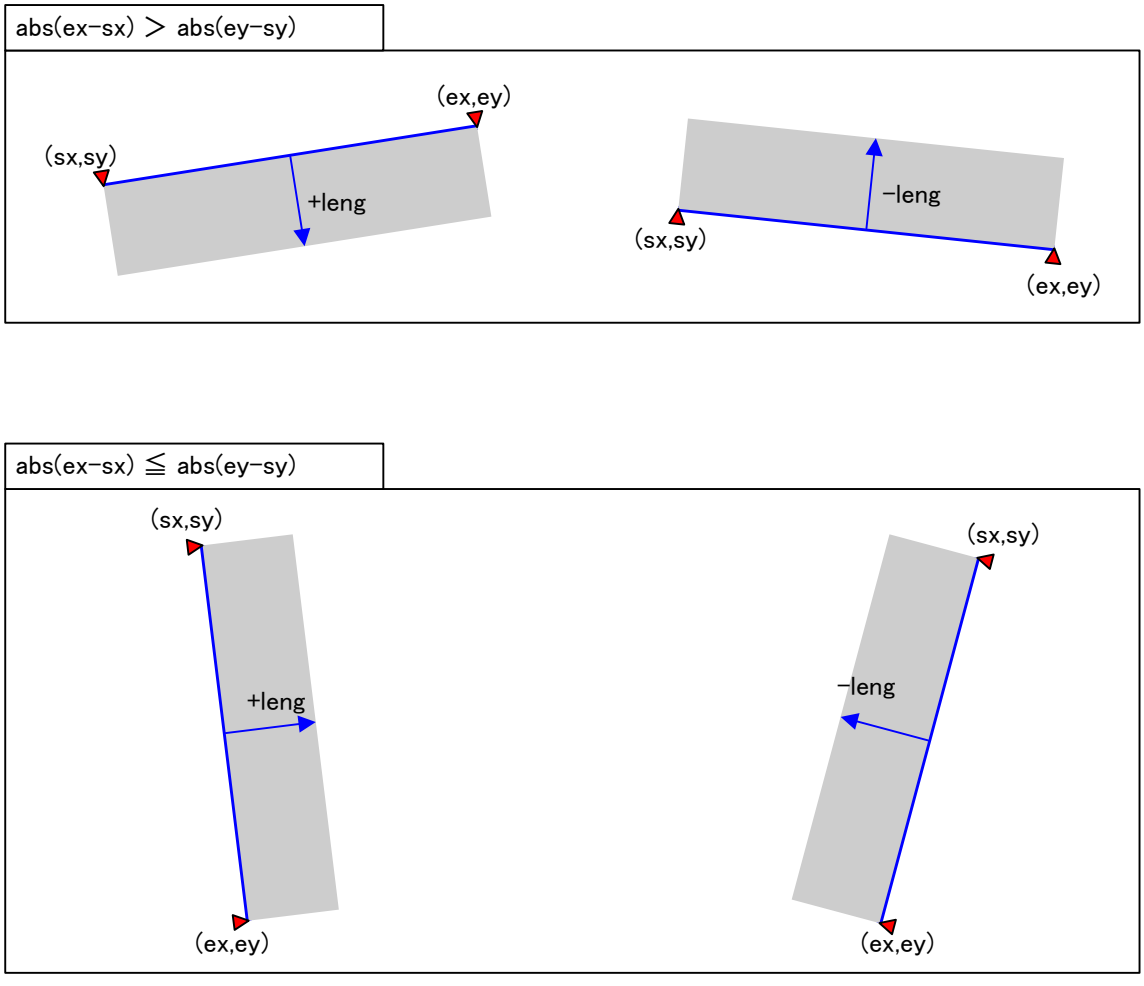


図9-49 lengの符号

9.23 エッジファインダ

画像処理コマンドではエッジファインダコマンドを用意しています。イメージキャリパでは画像処理により対象物のエッジを検出し、その対象物のエッジで平行なペアをサーチするため、エッジのペアがないパターンでは、エッジを抽出することはできません。そこで、エッジファインダーではその対象物に含まれるあらゆるエッジを解析し、位置、ポラリティ、レベルといった情報を抽出します。

9.23.1 ラインエッジファインダのエッジ抽出

以下にラインエッジファインダコマンドによるエッジ抽出のフローを示します。処理手順は基本的にイメージキャリパコマンドと同じですので、そちらも参照して下さい。

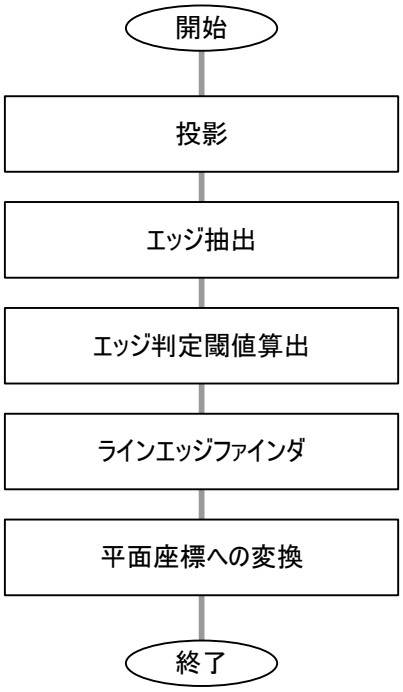


図9-50 エッジ抽出フロー

9.24 RGBLUT変換

RGBLUT変換は、24ビットのRGBカラー画像を濃度変換テーブル(LUT)により、8ビットの画像に変換する処理です。濃度変換のパターンを換えることにより、RGBカラー画像から特定カラーを抽出したり、色相、彩度、明度を抽出する等、様々な処理が可能です。

画像処理コマンドでは、RGBLUT変換情報をRGBLUTオブジェクトに格納し、RGBLUTハンドルで管理します。あらかじめ、RGBLUTオブジェクトにTGBLUT変換情報を設定し、コマンドを実行することによりRGBLUT変換を行います。RGBLUTオブジェクトの構成とRGBLUT変換の手順を以下に示します。

9.24.1 RGBLUT変換手順

RGBLUT変換するには、まず、**CreateRGBLUT()**コマンドでRGBLUTオブジェクトを生成します。次に、**OpenRGBLUT()**コマンドでLUTをアクセス可能にしてから、LUTに変換データを設定します。LUTへは、直接、変換データを書き込むか、あらかじめ用意されているマクロ(9.24.3 濃度変換テーブル)で設定します。LUTの設定が終了したら、**CloseRGBLUT()**コマンドでLUTのアクセスを終了します。RGBLUT変換は、**ConvertRGBLUT()**または**IP_ConvertRGBLUTEx()**コマンドを実行します。最後に**DeleteRGBLUT()**コマンドでRGBLUTオブジェクトを削除して処理を終了します。

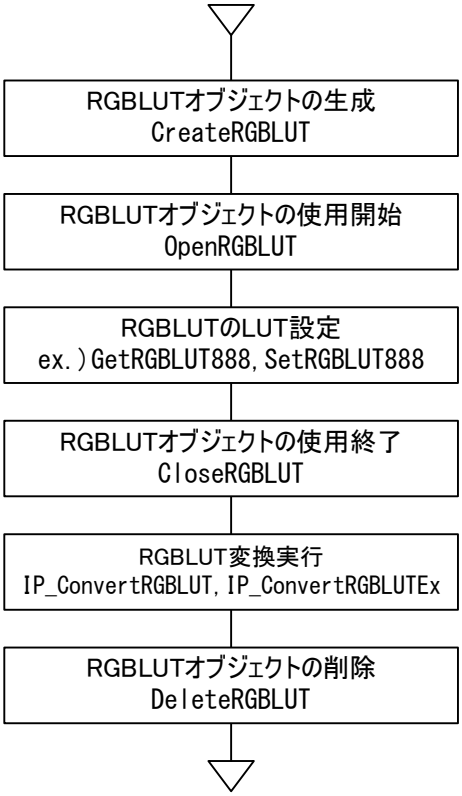


図9-51 RGBLUT変換手順

9.24.2 RGBLUTオブジェクト

RGBLUTオブジェクトは**CreateRGBLUT()**コマンドでNVP-Linux上に生成され、RGBLUTハンドルで管理します。RGBLUTオブジェクトの構成を以下に示します。

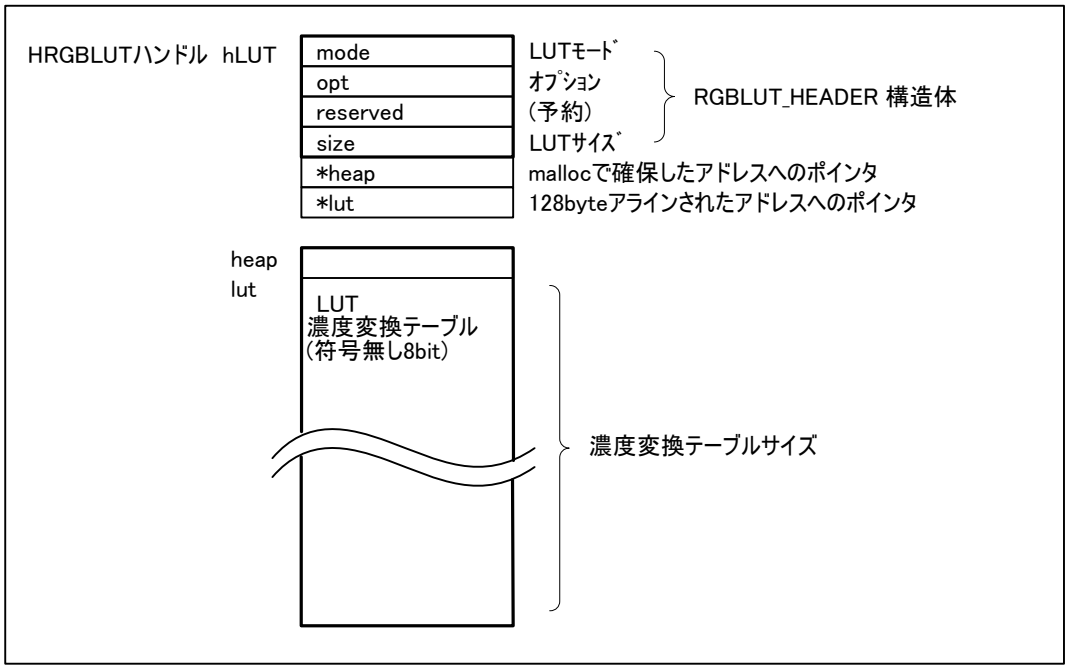


図9-52 RGBLUTオブジェクト構成

LUTモード(mode)は以下の通りで、濃度変換処理と濃度変換テーブル(LUT)の大きさを決定します。例えば、RGBLUT888は、Rデータが0～255(8bit)、Gデータが0～255(8bit)、Bデータが0～255(8bit)のRGBカラー画像を濃淡画像0～255(8bit)に変換するモードで、**CreateRGBLUT()**コマンド実行時に16,777,216byteの変換テーブルを確保します。

表9-11 LUTモード

LUTモード	定数値	内 容
RGBLUT888	0	R:8bit / G:8bit / B:8bit (16,777,216byte)
RGBLUT777	1	R:7bit / G:7bit / B:7bit (2,097,152byte)
RGBLUT666	2	R:6bit / G:6bit / B:6bit (262,144byte)
RGBLUT887	3	R:8bit / G:8bit / B:7bit (8,388,608byte)
RGBLUT878	4	R:8bit / G:7bit / B:8bit (8,388,608byte)
RGBLUT788	5	R:7bit / G:8bit / B:8bit (8,388,608byte)
RGBLUT787	6	R:7bit / G:8bit / B:7bit (4,194,304byte)
RGBLUT2CH88	7	R:8bit / G:8bit (65,536byte)
RGBLUT565	8	R:5bit / G:6bit / B:5bit (65,536byte)

9.24.3 濃度変換テーブル

濃度変換テーブル(LUT)は、**OpenRGBLUT()**コマンドを実行して、LUTサイズと先頭アドレスを取得し、直接アクセスすることが可能です。LUTへは、直接データを書き込むか、あらかじめ「ipxdef.h」に定義されている以下のマクロを用いて設定します。ここで、lutはLUTの先頭アドレス、r、g、bはそれぞれRGB画像のRデータ、Gデータ、Bデータ、dは変換データです。マクロ定義から分かるように、r、g、b値がLUTのアドレスを示すパラメータになります。

LUTの設定が終了したら、必ず、**CloseRGBLUT()**コマンドでLUTのアクセスを終了してください。

LUTへのデータ書き込みマクロ

```
#define SetRGBLUT888(lut,r,g,b,d) lut[r << 16 | g << 8 | b]= (d)
#define SetRGBLUT777(lut,r,g,b,d) lut[(r & 0xfe) << 13 | (g & 0xfe) << 6 | b >> 1]= (d)
#define SetRGBLUT666(lut,r,g,b,d) lut[(r & 0xfc) << 10 | (g & 0xfc) << 4 | b >> 2]= (d)
#define SetRGBLUT887(lut,r,g,b,d) lut[r << 15 | g << 7 | b >> 1]= (d)
#define SetRGBLUT878(lut,r,g,b,d) lut[r << 15 | (g & 0xfe) << 7 | b]= (d)
#define SetRGBLUT788(lut,r,g,b,d) lut[(r & 0xfe) << 15 | g << 8 | b]= (d)
#define SetRGBLUT787(lut,r,g,b,d) lut[(r & 0xfe) << 14 | g << 7 | b >> 1]= (d)
#define SetRGBLUT2CH88(lut,r,g,d) lut[r << 8 | g]= (d)
#define SetRGBLUT565(lut,r,g,b,d) lut[(r & 0xf8) << 8  | (g & 0xfc) << 5 | b >> 3]= (d)
```

LUTからのデータ読み出しマクロ

```
#define GetRGBLUT888(lut,r,g,b)   lut[r << 16 | g << 8 | b]
#define GetRGBLUT777(lut,r,g,b)   lut[(r & 0xfe) << 13 | (g & 0xfe) << 6 | b >> 1]
#define GetRGBLUT666(lut,r,g,b)   lut[(r & 0xfc) << 10 | (g & 0xfc) << 4 | b >> 2]
#define GetRGBLUT887(lut,r,g,b)   lut[r << 15 | g << 7 | b >> 1]
#define GetRGBLUT878(lut,r,g,b)   lut[r << 15 | (g & 0xfe) << 7 | b]
#define GetRGBLUT788(lut,r,g,b)   lut[(r & 0xfe) << 15 | g << 8 | b]
#define GetRGBLUT787(lut,r,g,b)   lut[(r & 0xfe) << 14 | g << 7 | b >> 1]
#define GetRGBLUT2CH88(lut,r,g)   lut[r << 8 | g]
#define GetRGBLUT565(lut,r,g,b)   lut[(r & 0xf8) <<  8 | (g & 0xfc) << 5 | b >> 3]
```

9.24.4 RGBLUT濃度変換

RGB画像のLUT変換は、**IP_ConvertRGBLUT()**コマンド、または、**IP_ConvertRGBLUTEx()**コマンドで実行します。以下にLUTモードに対する演算内容を示します。

ここで、ImgSrc(R)、ImgSrc(G)、ImgSrc(B)は、それぞれソース画像となるRGB画像のR、G、Bデータ、ImgDstはデスティネーション画像に設定されるデータです。

表9-12 LUTモードと演算内容

LUTモード	演算内容
RGBLUT888	ImgDst=LUT[ImgSrc(R) << 16 | ImgSrc(G) << 8 | ImgSrc(B)]
RGBLUT777	ImgDst=LUT[(ImgSrc(R) & 0xFE) << 13 | (ImgSrc(G) & 0xFE) << 6 | ImgSrc(B) >> 1]
RGBLUT666	ImgDst=LUT[(ImgSrc(R) & 0xFC) << 10 | (ImgSrc(G) & 0xFC) << 4 | ImgSrc(B) >> 2]
RGBLUT887	ImgDst=LUT[ImgSrc(R) << 15 | ImgSrc(G) << 7 | ImgSrc(B) >> 1]
RGBLUT878	ImgDst=LUT[ImgSrc(R) << 15 | (ImgSrc(G) & 0xFE) << 7 | ImgSrc(B)]
RGBLUT788	ImgDst=LUT[(ImgSrc(R) & 0xFE) << 15 | ImgSrc(G) << 8 | ImgSrc(B)]
RGBLUT787	ImgDst=LUT[(ImgSrc(R) & 0xFE) << 14 | ImgSrc(G) << 7 | ImgSrc(B) >> 1]
RGBLUT2CH88	ImgDst=LUT[ImgSrc(R) << 8 | ImgSrc(G)]
RGBLUT565	ImgDst=LUT[(ImgSrc(R) & 0xF8) << 8 | (ImgSrc(G) & 0xFC << 5) | (ImgSrc(B) << 3)]

9.24.5 RGBカラー抽出（色相・彩度・明度）

RGBLUT変換は、LUTのパターン設定により様々な変換をすることができます。
色の性質を色合い／色調(色相)、あざやかさ(彩度)、明るさ(明度)の3つの要素に分ける表現方法がありますが、RGB LUT変換を用いて、RGBカラー画像を色相(Hue)、彩度(Saturation)、明度(Intensity)の画像に変換することが可能です。以下に色相、彩度、明度の関係を示した図と画像変換の演算式を示します。

・色相（Hue）
　色を円周上に配列していると考え、色あい／色調を角度で表わしたものです。
　Hの範囲は 0～359 です。
・彩度（Saturation）
　色のあざやかさを示します。Sの範囲は 0 ～255 です。
・明度（Intensity）
　色の明るさを示します。Iの範囲は 0 ～255 です。

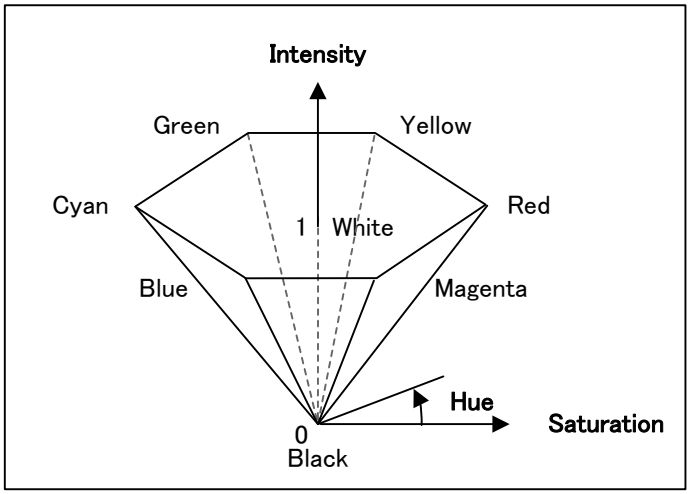


図9-53 色相、彩度、明度の関係

演算式	I= max(R,G,B) I= 0: S= 0: H= 0(不定) I≠ 0: i= min(R,G,B) S=(I- i)* 255/ I R= I: H=(G- B)* 60 /(I- i) G= I: H=(B- R)* 60 /(I- i)+ 120 B= I: H=(R- G)* 60 /(I- i)+ 240 (H < 0 の時は、H = H + 360 とする) H : Hue, S : Saturation, I:Intensity

画像メモリには、0 ～255 までしかデータを格納することができません。そこで、0～360度までのHueの値は0．71 倍するなどしてLUTに設定してください。

以下に、RGB画像から色相(Hue)画像へのRGBLUT変換例を示します。

```
/* 画面確保 */
ImgRGB = AllocRGBImg( IMG_FS_512H_512V );
ImgID  = AllocImg( IMG_FS_512H_512V );

.....................................

/* RGBLUTオブジェクト作成 */
mode = RGBLUT888;
hLUT = CreateRGBLUT( mode , 0 );

/*********************************************
  ＲＧＢ画像から色相(Hue)画像へ変換
*********************************************/
/* RGBLUTのLUTアクセス可能 */
mode = 0;
OpenRGBLUT( hLUT, &size, &lut, mode );

/* RGBLUTのLUT設定[色相(Hue)] */
for( r = 0; r < 256; r++ ){
   for( g = 0; g < 256; g++ ){
      for( b = 0; b < 256; b++ ){
         I = max( max( b, g ), max( g, r ) );
         i = min( min( b, g ), min( g, r ) );
         if( I == 0 ){
            H = 0;
         }else{
            val = 60.0f / ( I - i );
            if( I == r ){
               H = ( g - b ) * val;
            }else if( I== g ){
               H = ( b - r ) * val + 120;
            }else{
               H = ( r - g ) * val + 240;
            }
            if( H < 0 ) H += 360;
            H *= 0.71f;
         }
         SetRGBLUT888( lut, r, g, b, H );
      }
   }
}

/* RGBLUTのLUTアクセス終了 */
CloseRGBLUT( hLUT, lut );

/* RGBLUT変換 */
IP_ConvertRGBLUT( ImgRGB, ImgID , 0 , hLUT);

.....................................

/* RGBLUTオブジェクト削除 */
DeleteRGBLUT( hLUT );

/* 画面解放 */
FreeImg( ImgID );
FreeImg( ImgRGB );
```

r,g,b値(0～255)の組合せで
H(Hue)値が決定されます

r,g,b値のときのH(Hue)値を
LUTへ設定します

9.25 歪み補正

歪み補正とは、作成した歪み補正モデルにより、画像の変形を行う処理です。元の画像に対して、歪み補正を実行することにより、画像の傾きや歪みを解消することが可能です。
図9-54のような格子をソース画面(補正前画面)とデスティネーション画面(補正後画面)に作成することで、歪み補正を実行することができます。

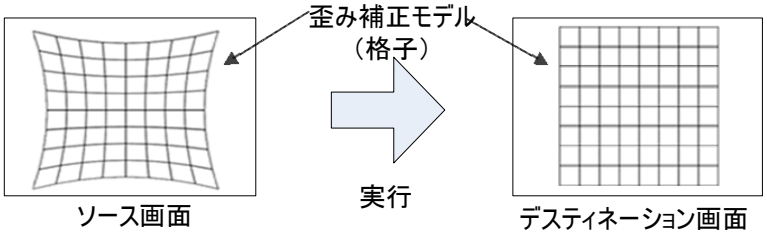


図9-54 歪み補正

9.25.1 歪み補正の実行手順

歪み補正実行の基本フローは以下のようになります。

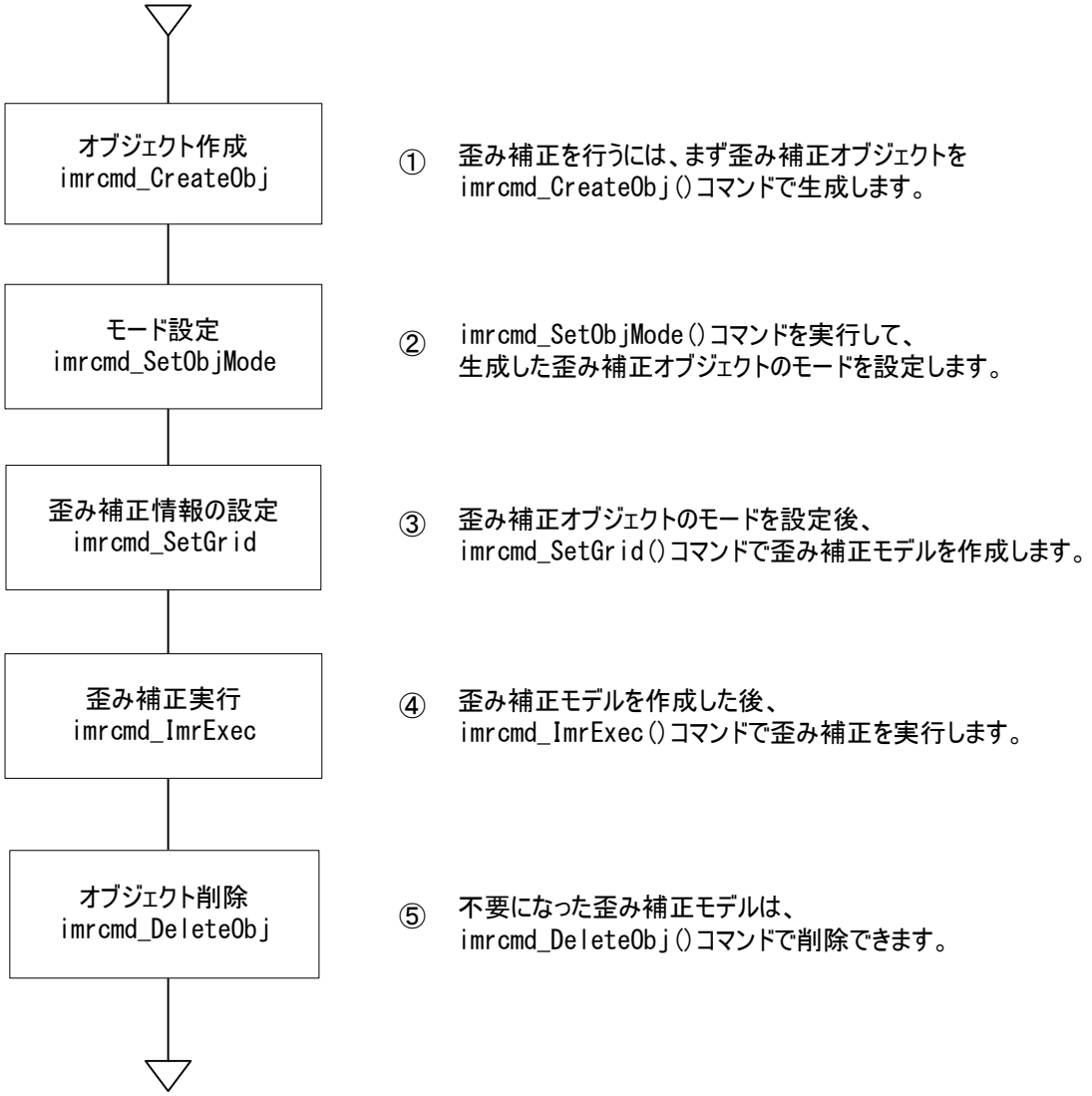


図9-55 歪み補正実行手順

9.25.2 歪み補正オブジェクト

歪み補正を行うためには、歪み補正モデルをあらかじめ作成します。
imrcmd_CreateObj()コマンドで歪み補正オブジェクトを生成し、歪み情報をセットすることで、歪み補正モデルが完成します。歪み補正モデルは複数作成することが可能です。

9.25.3 歪み補正のモード

歪み補正モードの設定は、**imrcmd_SetObjMode()**コマンドで行います。
歪み補正オブジェクトには、補間モードと描画モードの2つのモードがあります。

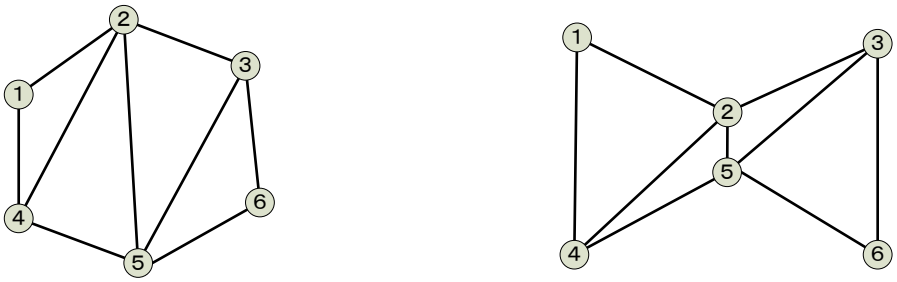
・補間モード　:バイリニア補間を実行することが可能です。
　　また、バイリニア補間のオンオフの切り替えが可能です。

・描画モード　:補正データをソース画面から参照せず、指定した画素値で単色描画されます。

9.25.4 歪み補正情報の設定

歪み補正を行う領域を、ソース画面(補正前画面)およびデスティネーション画面(補正後画面)の対応する頂点座標リストで指定します。**imrcmd_SetGrid()**コマンドは、指定された頂点情報をもとに歪み補正モデルを生成します。

図9-56(a)は補正前画面の歪み補正モデル、(b)は補正後画面の歪み補正モデルの例です。
補正前画面と補正後画面で、頂点①～⑥の頂点座標を座標リストに登録します。それから**imrcmd_SetGrid()**コマンドを実行することで、歪み補正モデルを生成されます。このモデルで**imrcmd_ImrExec()**コマンドを実行すると、補正前の画像を変形することができます。



(a) 補正前モデル　　(b)補正後モデル

図9-56　歪み補正モデル

9.25.5 頂点座標の設定について

頂点座標の描画基準について、頂点座標は、画素の左上が基準となります。
画素の座標値と頂点座標値は、図9-57のようになっているので、注意が必要です。
下図の例のように格子の頂点数4つのモデルを作成した場合、頂点座標値は次のようになっています。

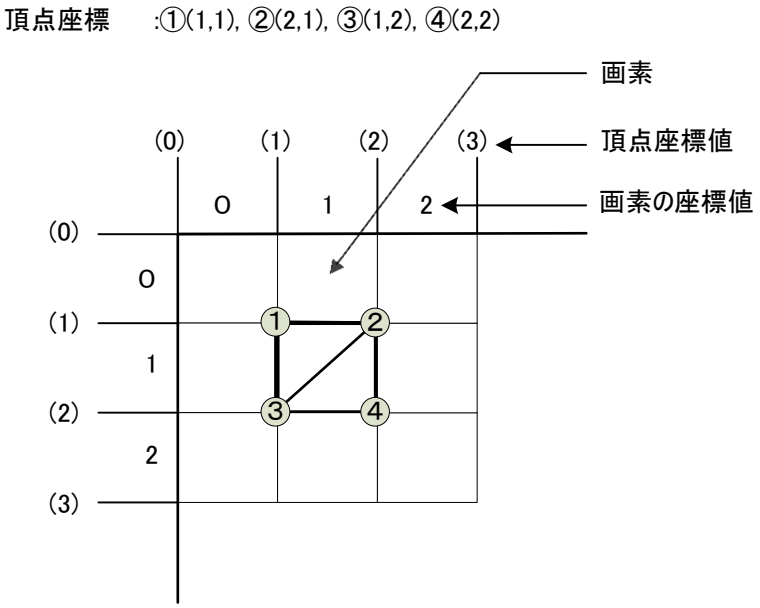


図9-57　歪み補正実行手順

9.25.6 処理領域

歪み補正の処理領域はIMRウィンドウ(IMR_WINDOW構造体)で指定します。処理領域とデスティネーション画面の関係を図9-57に示します。補正前(ソース)側と補正後(デスティネーション)側の処理領域の設定は次のようになっています。

・ソースウィンドウ
　　ウィンドウサイズはx座標が2048、y座標が2048で固定、描画始点の設定可能

・デスティネーションウィンドウ
　　ウィンドウサイズはx座標が最大1025, y座標最大1025で設定でき、描画始点の設定可能

に歪み補正の処理領域を示します。

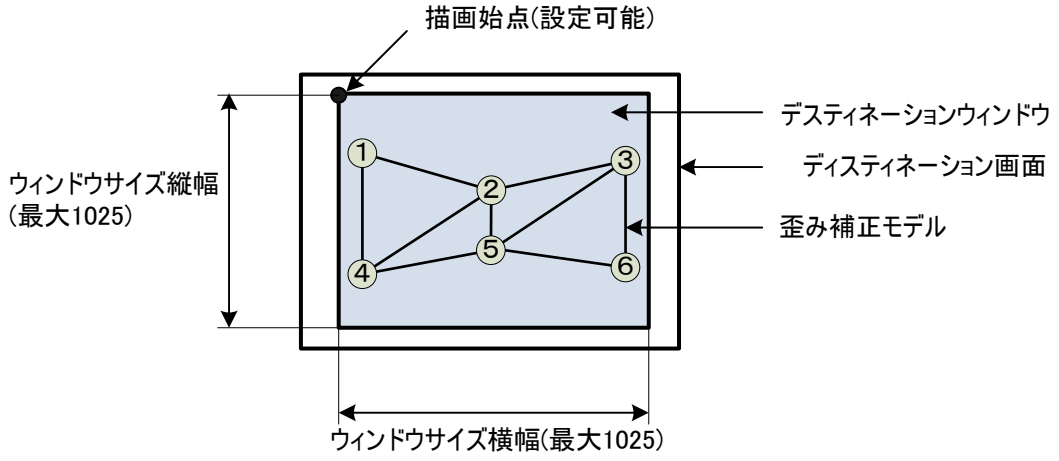


図9-58　歪み補正の処理領域(補正後側)

画像処理ユニット NVP-Ax430シリーズ　ソフトウェア開発キット

NVP-Ax430SDK

ユーザーズマニュアル（第3版）

（C）マクセルフロンティア株式会社

開発元

マクセルフロンティア株式会社

営業部　〒244-0801 神奈川県横浜市戸塚区品濃町549-2三宅ビル

技術サポート窓口　URL　www.frontier.maxell.co.jp
mail：vp-support@maxell.co.jp