修士学位論文

論文題目 Android OSとカメラによる

対象物追跡における処理の高速化

ふりがな
氏名 高木 佑介（たかぎ ゆうすけ）

専攻 工学研究科 機械工学専攻

指導教授 金丸 隆志 准教授

修了年月（西暦） 2012年3月

工学院大学大学院

目次

概要

近年 Android OS と画像処理アプリケーションへの注目が高まっている．本研究では，その双方に着目し，カメラを用いた対象物追跡を Android OS の利用法の 1 つとして提案する．本研究では，スマートフォンだけではなく，組み込みデバイスもターゲットとするものとする．

研究を進めるに当たり，まず画像処理の高速化に取り組んだ．JIT（Just-In-Time Compiler），JNI（Java Native Interface）など，様々な手法を用いて処理速度の高速化を図り，高速化に成功した．

その結果をふまえて，対象物追跡アプリケーションの作成に取り組んだ．ここでは，Android のシステム関数を利用した認識手法と，テンプレートマッチングをユーザー関数として実装した認識手法の 2 つを試した．システム関数を利用した認識処理よりも，認識部をユーザー関数化することで，さらなる処理の高速化につながった．

また，本研究で作成したアプリケーションの機能の拡張についても検証し，Bluetooth 端末による遠隔操作を実現した．

作成したアプリケーションを Android OS の利用法の一例として提案する．

第1章　序論

1.1　研究背景

Android は，Google 社が中心になって組織された Open Handset Alliance によって開発されたプラットフォームであり，スマートフォンやタブレット PC などの情報端末を主なターゲットとしている．2007 年の発表以来注目を集めており，2011 年 12 月現在，スマートフォン用の OS としては，日本とアメリカでトップシェアである．非常に汎用性が高く，様々なデバイスに移植可能であることが魅力の 1 つだが，最も注目すべきは，誰でも無償で利用することができるオープンソースな OS であるということであり，これが Android OS の普及における大きな要因の 1 つであると考えられる．

現在では主にスマートフォンの OS として注目を集めている Android であるが，我々はこの汎用性の高さに着目し，組み込みデバイス用の OS として利用することを考える．これにより，スマートフォンの OS としてだけではなく，Android の新たな利用の場が開拓できるのではないかと考える．

また，Android への注目が高まる中，それと同時に画像処理アプリケーションのニーズが高まってきている．スマートフォンの普及とともに一般的に触れる機会が多くなってきている画像処理アプリケーションとしては，カメラでとった画像を編集するアプリケーションが挙げられるのではないだろうか．これは写真撮影をする際に，トイカメラ，ポラロイド，魚眼やシンメトリなど，多いものでは 40 種類以上ある撮影方法を選択し，エフェクトのかけられた写真が撮れるというものである．カメラの初心者でも手軽に利用でき，様々な撮影が可能なことから，多くのスマートフォンユーザーが利用している．画像処理アプリケーションへの注目が高まっているのは，一般的なユーザーの視点だけではなく研究者のそれも同じである．セキュリティシステムや拡張現実への利用など，その用途，目指すところは様々であるが，画像処理アプリケーションは現在も進歩を続けている．このことから，画像処理アプリケーションの有用性は高く，まだまだ発展の余地があり，大きな可能性を秘めた分野であると言える．また，我々はこの分野に Android のさらなる活躍の場があるのではないかと考える．

1.2 目的

近年の Android OS と画像アプリケーションへの注目の高まりから，本研究ではこれら 2 つを取り上げ，カメラを用いた対象物追跡を Android OS の利用法の 1 つとして提案する．将来的には，監視カメラなどに技術的な転用が可能ではないかと考えている．

本研究における対象物の追跡手段として，画像認識とモータ制御の技術を利用する．先にも述べたが，我々は Android OS を利用するにあたり，メインターゲットをスマートフォンではなく，組み込みデバイスとする．その理由として，Android のスマートフォン以外への利用法を提案することだけでなく，USB カメラやモータといったハードウェアは，スマートフォンでは利用しにくいということも挙げられる．我々はこのようにモータを動かし，ロボットを制御するような用途に Android の新たな利用の可能性があるのではないかと考えている．

ここで，近年の Android OS について触れておく．Android3.0 から USB ホストモードが搭載され様々な USB デバイスが利用できるようになっているが, メーカーが対応したものに限られるという問題が残っており, 今回我々が用いる USB シリアルデバイスは通常ではスマートフォンでは有効になっていない．また，Android2.3.4 より ADK（Open Accessory Development Kit）ができたことで ADK の回路経由でモータを動かすことは可能となった．ただし，Android と ADK（回路）の両方でのプログラミングが必要であることや，回路用に別電源が必要であることなどやや手間がかかる．そのため，今回我々が用いるように組み込みシステム向けにソースからビルドした Android を用いることは，モータ制御を行う上では現実的である．

画像処理の観点からは，対象物認識のシステムとその処理速度の向上を目指す．画像処理が遅くては，その間にも動いている対象を追いきれず，実用性に欠けるため，処理速度の向上は必須であると言える．

1.3 本論文の構成

本論文の構成を以下に示す.

まず，第 1 章では本研究に至る背景とその目的について述べる．近年の Android OS や画像処理アプリケーションの可能性，本研究の方向性について示す.

第 2 章では本研究の準備段階として，Android OS の設定とターゲットの仕様について述べる．BeagleBoard や PandaBoard の詳細なスペックや，Android のビルド方法についてなどである.

第 3 章では画像処理における高速化について理論を述べる．考えられる手段を列挙し，それぞれについて詳細を示す.

第 4 章では本研究で使用したカメラの詳細と，画像処理の高速化の検証について述べる．製作したカメラの台座についてと，処理速度の測定とそれから得られた結果の考察を示す.

第 5 章では対象物追跡のアプリケーションについて述べる. 駆動部であるモータの制御や，画像認識についてである．また，認識方法の工夫により処理の高速化が見られるか検証し，その結果を考察する.

第 6 章では対応させた Bluetooth の機能を利用し，作成したアプリケーションの機能の拡張について述べる.

第 7 章では本論文で得られた成果を要約し，結論とする.

第2章　組み込みシステムと Android

2.1　Android OS

Android OS の構造は図 2.1 のようになっており，組み込み用の OS として利用するためには，ダウンロードした Android ソースをターゲット用に設定する必要がある．例をあげると，アプリケーション・フレームワークの libcameraservice を USB カメラに対応させるなどである．その際 sola 氏の android-development-environment という解説ページを参考とした[1]．その環境をベースに，Wifi，USB カメラ，Bluetooth などのデバイスを利用可能するため kernel や Android ソースを変更した．詳細は 2.5 章で触れるが，おおまかな変更手順は以下の通りである．

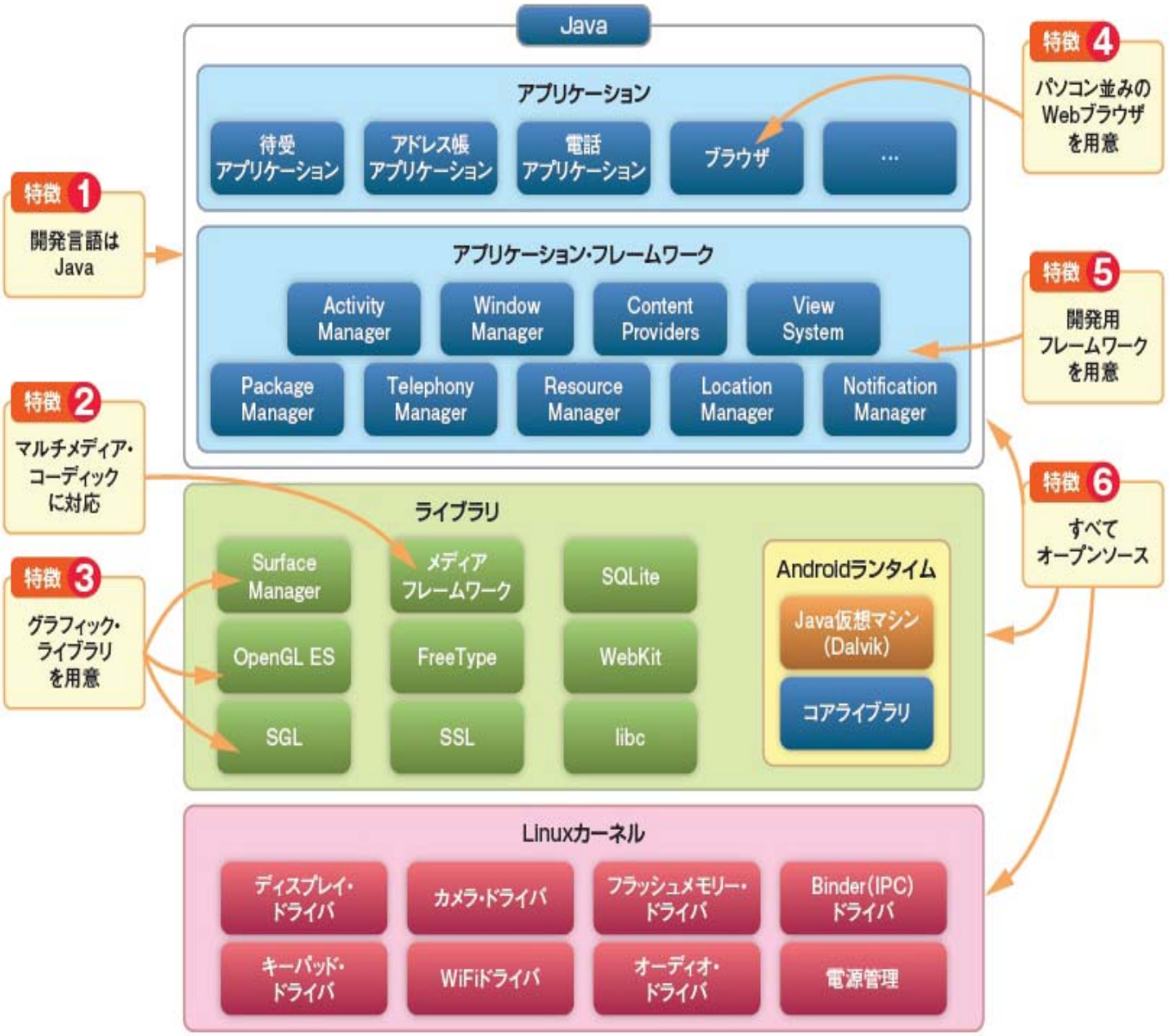
①kernel 変更…Wifi，USB カメラ，Bluetooth への対応

②Android ソース変更…kernel 変更への対応

③kernel と Android のビルド

また，この作業は Linux の端末上で行う．Linux は ubuntu10.04 を用いた．

変更作業が完了し，カメラなどのハードウェアが利用可能になった状態で，アプリケーションの作成にとりかかる．よって，本研究では，主に kernel と Android ソースの変更により Android を組み込み用の OS として利用可能にして様々なハードウェアに対応させる「開発準備段階」と，Java や C 言語を用いて対象物追跡を行うアプリケーションを製作する「開発段階」に分かれる．また，本研究ではテキサス・インスツルメンツ製の BeagleBoard と PandaBoard を組み込みデバイスのターゲットとする．両デバイスの詳細については次章で述べる．



●ライブラリ群

Surface Manager:表示サブシステムのアクセスを管理し,複数のアプリケーションの2D,3Dグラフィック・レイヤーを合成する

OpenGL ES:OpenGL ES 1.0のAPIに準拠した3Dライブラリ

SGL:2Dグラフィックエンジン

メディアフレームワーク:PacketVideo社のOpenCOREをベースにしたライブラリ。一般的なオーディオとビデオコーデック(MPEG4,H.264,MP3,AAC,AMR,JPGとPNGのような静止画を含む)の再生と録音をサポート,

FreeType:ビットマップとベクター・フォントのレンダリングを行うライブラリ

SSL:SSL(Secure Socket Layer)で通信するためのライブラリ

SQLite:コンパクトなデータベース管理システム

WebKit:オープンソースのHTMLレンダリング・エンジン

libc:C言語ライブラリ。Linuxでよく使われるglibcやuClibcではなく,Androidのために作られた独自のC言語ライブラリを使用している

図 2.1 Android の構造 [2]

2.2 BeagleBoard

BeagleBoard はテキサス・インスツルメンツ社の開発した約 8cm 四方の小型マザーボードである．高性能プロセッサ OMAP3，256MB のメモリーを搭載し，豊富な外部インターフェースを備えている．また，消費電力が小さく，小型でありながら Android OS を起動させるパワーがある点も魅力である．

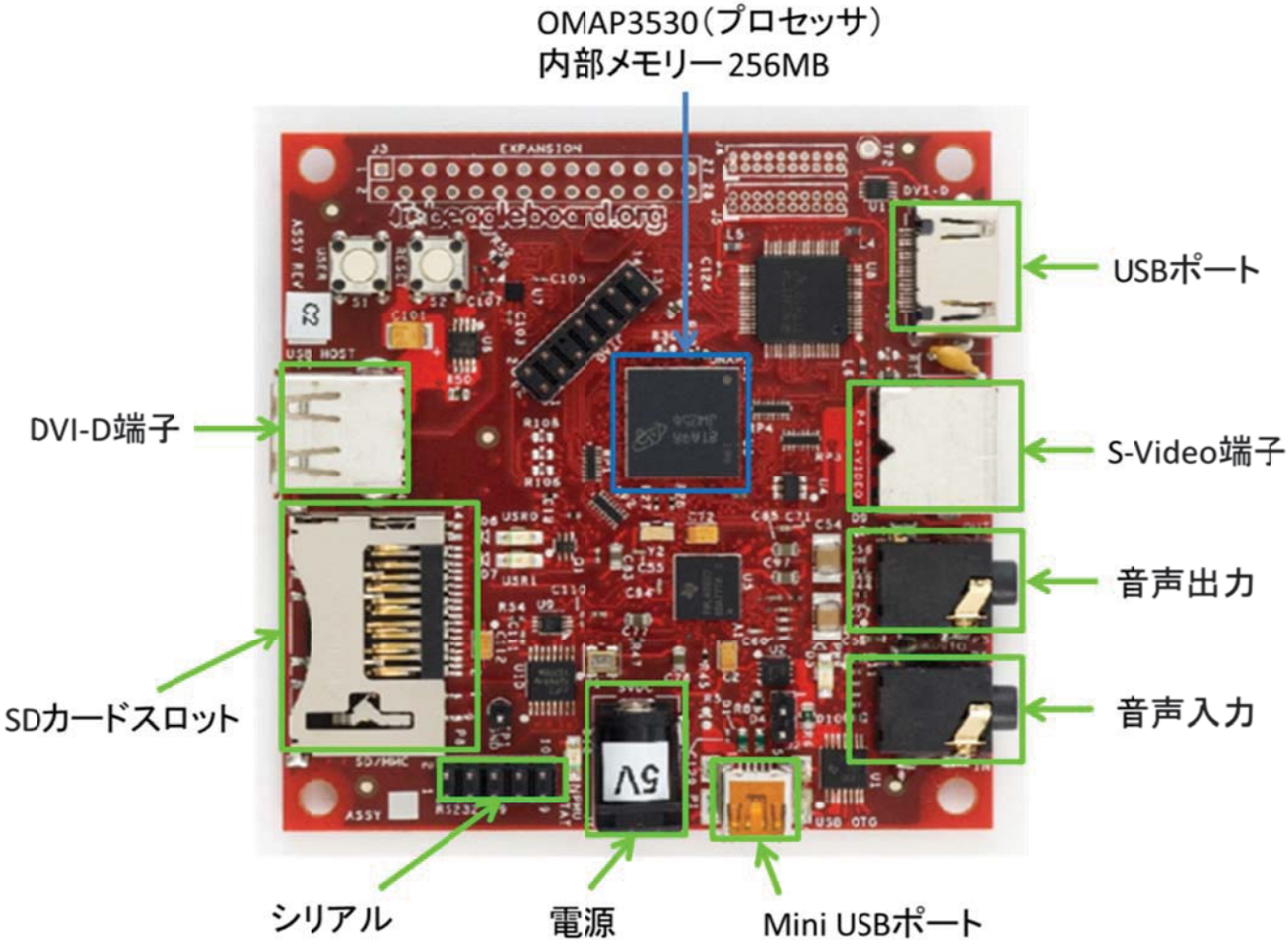


図 2.2 BeagleBoard

2.3 PandaBoard

PandaBoard は約 11cm 四方の小型マザーボードである．TI 製の OMAP4430(1.0GHz)および 1GB の DDR2RAM を搭載している．BeagleBoard と同じく外部インターフェースが豊富である．BeagleBoard よりも一回り大きいが，プロセッサの性能とメモリーの容量が上がり，インターフェースの数も増えている．研究開始当初は PandaBoard が発売されておらず（2010 年 12 月 9 日より発売開始），BeagleBoard のみをターゲットとしていたが，途中からより性能の高いものが発表されたため，PandaBoard も視野に入れて取り組んだ．また，本テーマに取り組むに当たり，BeagleBoard と PandaBoard の性能比較も行った．

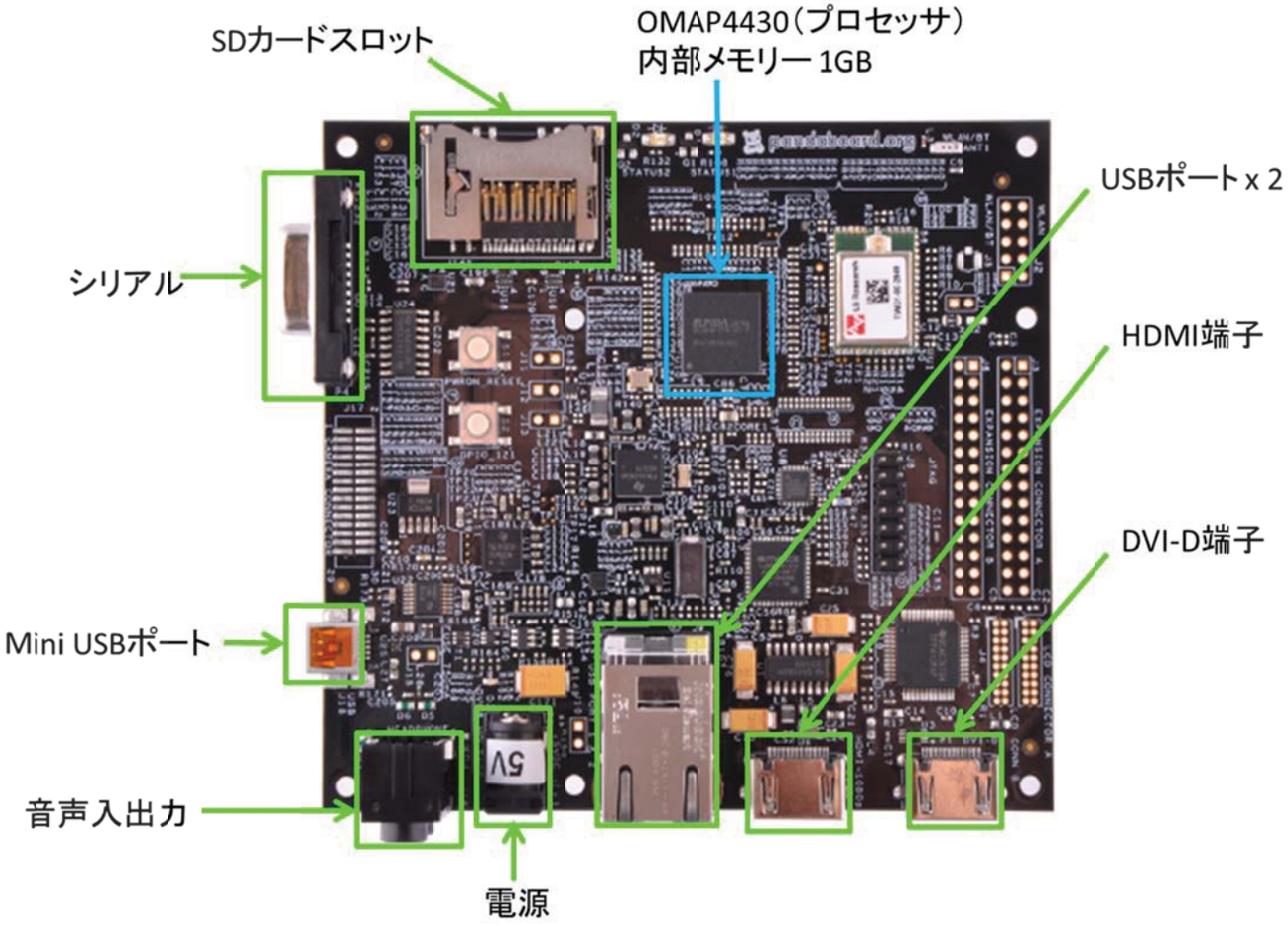


図 2.3 PandaBoard

2.4 BeagleBoard と PandaBoard の仕様比較

以下の表 2.1 に BeagleBoard と PandaBoard の仕様の詳細をまとめる．BeagleBoard と PandaBoard を比較すると CPU がシングルコア（720MHz）からデュアルコア（1GHz）になり，内部メモリーが 256MB から 1GB に増え，価格が同程度（ BeagleBoard:$149，PandaBoard:$174）にも関わらず性能が飛躍的に伸びている．このようなハードの性能向上は Android 端末の需要が急速に増えていることが要因の 1 つであると考えられる．

表 2.1 BeagleBoard と PandaBoard 仕様比較

	BeagleBoard	PandaBoard
プロセッサ	OMAP3530	OMAP4430
CPU	720 MHz ARM Cortex-A8	1GHz デュアルコア ARM Cortex-A9
命令セット	ARMv7	ARMv7
内部メモリー	256[MB] DDR SDRAM	1[GB] DDR2 RAM
寸法	78.74 x 76.2 [mm]	114.3 x 101.6 [mm]
重量	37[g]	74[g]
USB 2.0	シングル USB ポート	USB ポート x 2
	Mini-AB USB コネクタ	Mini-AB USB コネクタ
シリアル通信	RS-232	RS-232
映像	DVI-D, S-Video	DVI-D, HDMI
音声	35mm L+R 出力	35mm L+R 出力
	35mm L+R ステレオ入力	35mm L+R ステレオ入力
SD カードコネクタ	6 in 1 SD/MMC/SDIO	6 in 1 SD/MMC/SDIO
電源	5[V]	5[V]

2.5 Android のソースコード

本章では，Android OS のソースコードについて述べる．ソフトウェアの規模を表す指標の一つであるファイル数，ファイル総行数について，以下の表に各 Android のバージョンについて示す．ファイル数とファイル総行数は Linux 端末上で検出するのだが，その方法については以下のとおりである．

まず，ファイル数の検出方法であるが，各言語ファイルについて以下のコマンドでファイル数が得られる．

(a) C 言語ファイル	find .	grep "¥.c$"	wc -l
(b) C++言語ファイル	find .	grep "¥.cpp$"	wc -l
(c) Java 言語ファイル	find .	grep "¥.java$"	wc -l
(d) アセンブリ言語ファイル	find .	grep "¥.[sS]$"	wc -l

「find .」コマンドは，「今いるディレクトリより下に存在するファイルと「ディレクトリを列挙する」の意味である．ファイルの検出を途中でとめるには「Ctrl + c」である．また，例えば(a)C 言語ファイルの「grep "¥.c$"」は，「ファイル名の末尾が .c であるファイルのみ抽出」の意味である．最後に，「wc –l」は「行数を数える」の意味である。その結果，「末尾が .c のファイル名のリストの行数を数える」，すなわち「ファイルの個数」が得られる．

つぎに，ファイル総行数の各言語ファイルの検出方法である．先述した「wc –l」コマンドであるが，使い方によって「ファイルの行数を調べる」ことができる．これを利用して各言語ファイルの行数を調べる．

(a) C 言語ファイル	wc -l `find .	grep "¥.c$"`
(b) C++言語ファイル	wc -l `find .	grep "¥.cpp$"`
(c) Java 言語ファイル①	wc -l `find .	grep "[0-9]¥.java$"`
(d) Java 言語ファイル②	wc -l `find .	grep "[a-zA-Z_]¥.java$"`
(e) アセンブリ言語ファイル	wc -l `find .	grep "¥.[sS]$"`

ファイル数が多いため, Java 言語ファイルのみ 2 つにわけた. よって, (c)と(d)の合計が Java 言語ファイルの総行数である．

各バージョンを比較してみると，バージョン 1.6 のファイル総行数は約 800 万行，バージョン 2.2.2 では約 1120 万行，バージョン 2.3.4 では約 1170 万行，バージョン 4.0.1 では約 1540 万行であった. バージョンが新しくなる度にその規模が大きくなっているのが確認できる．また，Windows XP のファイル総行数は約 4000 万行と言われており，Android が複雑化し，PC の OS に近づいてきたことがうかがえる．

表 2.2　ソースファイルのファイル数・行数（Android version1.6）

言語ファイル	ファイル数	ファイル行数
C 言語	6,296	3,267,190
C++言語	4,569	1,994,022
Java 言語	11,090	2,687,179
アセンブリ言語	1,120	90,583
総　数	23,075	8,038,974

表 2.3　ソースファイルのファイル数・行数（Android version2.2.2）

言語ファイル	ファイル数	ファイル行数
C 言語	7,972	4,505,470
C++言語	6,462	2,671,115
Java 言語	17,645	3,824,297
アセンブリ言語	1,836	195,741
総　数	33,915	11,196,623

表 2.4　ソースファイルのファイル数・行数（Android version2.3.4）

言語ファイル	ファイル数	ファイル行数
C 言語	9,067	5,107,191
C++言語	6,297	2,367,119
Java 言語	18,278	3,863,672
アセンブリ言語	1,902	382,974
総　数	204,544	11,720,956

表 2.5　ソースファイルのファイル数・行数（Android version4.0.1）

言語ファイル	ファイル数	ファイル行数
C 言語	11,362	5,590,921
C++言語	12,265	4,159,559
Java 言語	24,888	5,218,219
アセンブリ言語	3,207	401,297
総　数	51,722	15,369,996

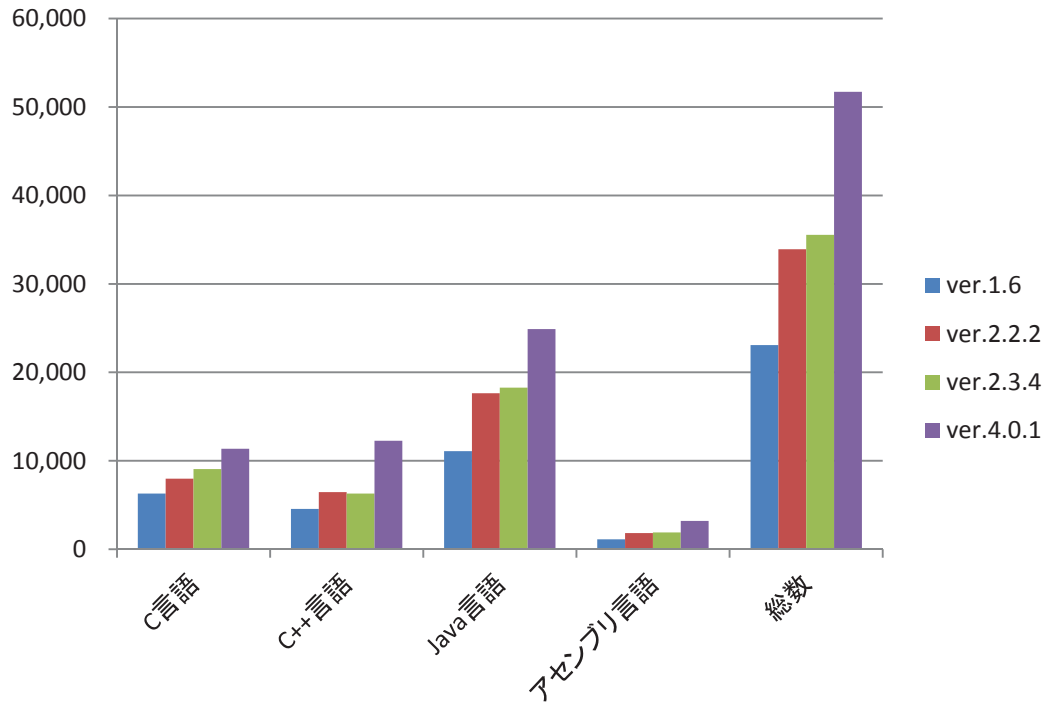


図 2.4　ソースファイルのファイル数比較

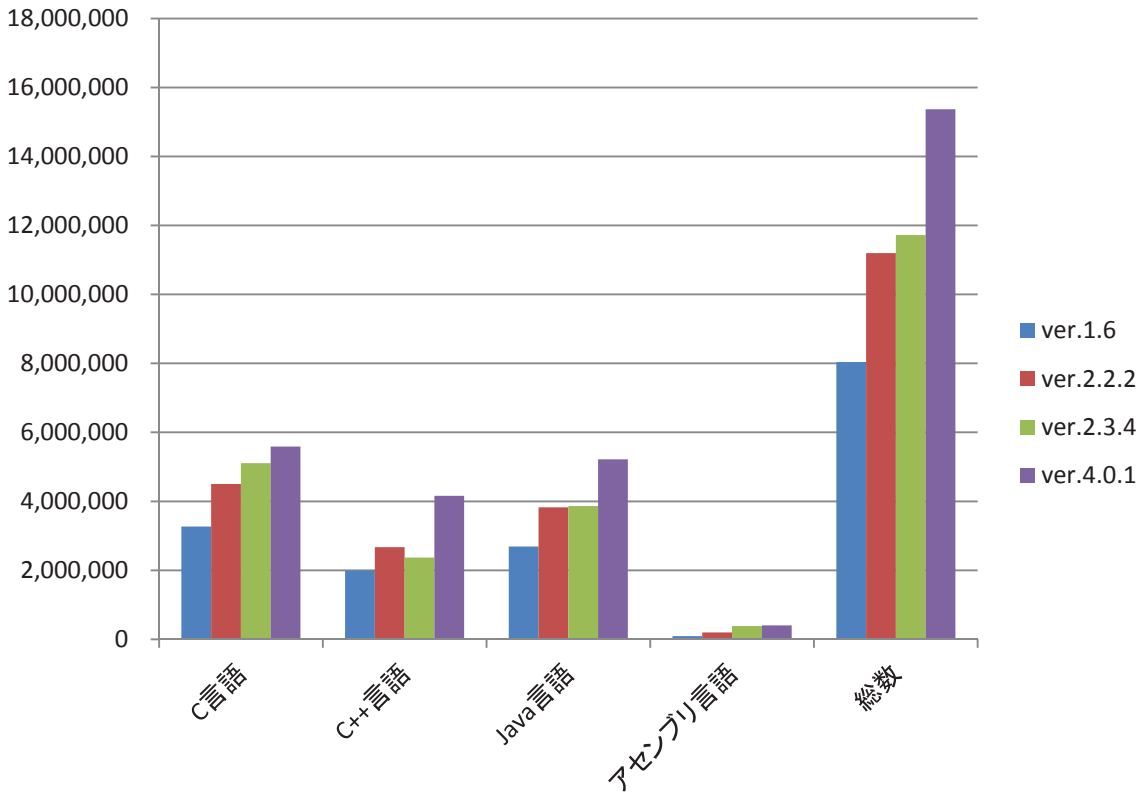


図 2.5　ソースファイルのファイル行数比較

2.6 Android の設定

先に述べたように，Android を組み込み用の OS として利用するためには，ソースの変更など設定が必要である．本節では，開発準備段階として，Android2.2.2(Froyo)の導入から利用可能に至るまでを述べる．

2.6.1 導入の流れ

Linux 端末上で，Android のビルド環境を整え，Web から Android ソースをダウンロードする．端末でダウンロードしたソースをビルドし，パーティションを切って HDD にみたてた SD カードにビルドした Android を保存する．Android が保存された SD カードを BeagleBoard(PandaBoard)に接続し，Windows 上でボード自体の設定を変更する．

図 2.6 Android の導入の流れ

2.6.2 ビルド環境設定

Ubuntu10.04 を搭載した Linux の端末を操作し，Android ソースのビルドができる環境を構築する．主な作業は以下の通りである．

◯Java5 のインストール（Android version2.2）
◯Java6 のインストール（Android version2.3 以降）
◯git/repo の設定
◯パスの追加
◯USB デバイスの設定
◯mkImage コマンドの入手

ここで，環境設定の詳細について記す．まず以下 4 つのコマンドを入力し，Android 開発に必要なソフトをインストールする．

```
$ sudo apt-get install git-core gnupg flex bison gperf libsdl-dev
libesd0-dev libwxgtk2.6-dev build-essential zip curl libncurses5-dev
zlib1g-dev
$ sudo apt-get install valgrind
$ sudo apt-get install libreadline5-dev
$ sudo apt-get install uboot mkImage
```

続いて，Java5 のインストールを行う．次のコマンドで sources.list を開き，編集する[3].

```
$ sudo gedit /etc/apt/sources.list
以下を追記
deb http://us.archive.ubuntu.com/ubuntu/ hardy multiverse
deb http://us.archive.ubuntu.com/ubuntu/ hardy-updates multiverse
```

その後，以下のコマンドを入力し，インストール完了である．

```
$ sudo apt-get update
$ sudo apt-get install sun-java5-jdk
```

インストール完了後，追記した内容を戻しておく．また，別のバージョンの Java を入れている場合、以下コマンドで変更する．

```
$ sudo update-alternatives --config java
$ sudo update-alternatives --config javac
```

Java6 のインストールを行う．コマンドは以下の通りである．

```
$ sudo add-apt-repository "deb http://archive.canonical.com/ lucid
partner"
$sudo apt-get update
$sudo apt-get install sun-java6-jdk
```

git/repo の設定を以下に記す．

```
$mkdir ~/bin
$ curl http://android.git.kernel.org/repo >~/bin/repo
$ chmod a+x ~/bin/repo
```

```
※export PATH=$PATH:~/bin を .bashrc へ追記
$ git config --global user.email "メールアドレス"
$ git config --global user.name "ユーザー名"
```

パスの追加方法は以下の通りである.

```
$ gedit ~/.bashrc

#.bashrc
# User specific aliases and functions
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then
. /etc/bashrc
Fi

PATH="$PATH":~/bin    ←追加
```

また，$source ~/.bashrc，$set|grep PATH と端末上で記述すれば PATH の確認が可能である.

2.6.3 Android のビルド

2.5.2 章のようにビルド環境を整えた Linux 端末で，repo sync コマンドを用いて Android ソースをダウンロードし，kernel，Android の順でビルドする.

まず，Android ソースのダウンロード方法について述べる．初めにダウンロード先フォルダを作成する．ここではダウンロード先を環境変数$ANDROID とセットして以下の作業を行う.

```
$ export ANDROID=(ソースコードのダウンロード先)
$ mkdir -p $ANDROID
$ cd $ANDROID
$ repo init -u git://android.git.kernel.org/platform/manifest.git -b
android-2.2.2_r1.1
```

local_manifest.xml を変更し，ALSA 関連のファイルを追加する．

```
$gedit .repo/local_manifest.xml
```

上記コマンドで編集画面を開き，以下のように local_manifest.xml の内容を変更する．

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
    <project   path="external/alsa-lib"   name="platform/external/alsa-lib"
revision="froyo"/>
    <project path="external/alsa-utils" name="platform/external/alsa-utils"
revision="froyo"/>
    <project path="hardware/alsa_sound" name="platform/hardware/alsa_sound"
revision="froyo"/>
</manifest>
```

local_manifest.xml を保存後，以下を実行する．

```
$ repo sync
```

これで Android ソースのダウンロードとは完了である．次に，kernel をダウンロードする．

```
$ cd $ANDROID
$ wget http://sola-dolphin-1.net/data/android/BeagleBoard/kernel
_beagle.0xlab.sgx.tar.bz2
$ tar jxvf kernel_beagle.0xlab.sgx.tar.bz2
```

続いて，以下の操作により sola 氏によるパッチのダウンロードと適用を行う．また，それと同時に筆者の所属する研究室のウェブサイト [4] で公開されている vendor_tk-beagle-froyo2.2.x-20110725.tar.gz というファイルを以下の操作によりダウンロードし，パッチを適用する．

```
$ cd $ANDROID
$ wget http://android-development-environment.googlecode.com/files/
vendor_sola-omap3-froyo.tar.gz
$ mkdir vendor
$ tar zxvf vendor_sola-omap3-froyo.tar.gz -C $ANDROID/vendor/
```

```
$ $ANDROID/vendor/sola/omap3/patch/omap3-patch.sh
$ wget http://brain.cc.kogakuin.ac.jp/research/files/vendor_tk-beagle
-froyo2.2.x-20110725.tar.gz
$ tar zxvf vendor_tk-beagle-froyo2.2.x-20110725.tar.gz -C $ANDROID/
vendor/
$ $ANDROID/vendor/tk/patch/tk_patch.sh
```

これで Android ソースのビルドの準備が完了した.

以下より，Android のビルドについてである．前述のように，sola 氏による android-development-environment という解説ページを参考[1]にし，まずは端末を操作して kernel をビルドする．ダウンロードした Android のソースを展開し，このディレクトリを ANDROID という環境変数に設定する．その後，以下の操作を行う.

```
$ cd $ANDROID/kernel-beagleboard
$ make ARCH=arm
CROSS_COMPILE=../prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin/arm
-eabi- sola_omap3_beagle_android_defconfig
$ make ARCH=arm
CROSS_COMPILE=../prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin/arm
-eabi- uImage modules
```

続いて端末から Android のビルドを行う.

```
$ cd $ANDROID
$ source build/envsetup.sh
$ lunch beagleboard-eng
$ make -j8
```

ビルドが完了するまでにかなりの時間を必要とする．また，Froyo（Android version2.2）は Java バージョン 5 でコンパイルを行い，Gingerbread（Android version2.3）以降はバージョン 6 でビルドを行う．終了後，イメージの作成を行う．イメージの作成については以下の通りである.

```
$ cd $ANDROID
$ $ANDROID/vendor/sola/omap3/image/beagleboard-image.sh
```

この操作で，$ANDROID/vendor/sola/omap3/image/beagleboard/android が作成されるが，下の TI's Android SGX SDK を取り込むまで SD カードにはコピーしない．TI's Android SGX SDK を取り込む．

```
$ cd $ANDROID
$ git clone git://gitorious.org/rowboat/ti_android_sgx_sdk.
git TI_Android_SGX_SDK
$ cd $ANDROID/TI_Android_SGX_SDK
$ ./OMAP35x_Android_Graphics_SDK_setuplinux_3_01_00_03.bin
```

Destination Folder を $ANDROID/TI_Android_SGX_SDK にセットしてインストールする．$ANDROID/TI_Android_SGX_SDK/Rules.make を編集する．編集は次のようにする．

```
HOME=$(ANDROID)
GRAPHICS_INSTALL_DIR=$(ANDROID)/TI_Android_SGX_SDK
ANDROID_ROOT=$(ANDROID)/vendor/sola/omap3/image/beagleboard/android
CSTOOL_DIR=$(ANDROID)/prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/
KERNEL_INSTALL_DIR=$(ANDROID)/kernel-beagleboard
```

編集が完了した後，以下の通りビルドを行う．

```
$ cd $ANDROID/TI_Android_SGX_SDK
$ make
$ make install OMAPES=3.x
```

ここで，パーティションを分けた SD カードを用意し，分けたパーティションをそれぞれ DISK1，DISK2，DISK3 とする．設定は以下の通りである．

DISK1：ファイルシステム fat32，容量 64MB．kernel の格納用に用いる．

DISK2：ファイルシステム fat32，容量は DISK1 と DISK3 に使用した残りとする．Android から SD カードとして見える．

DISK3：ファイルシステム ext3，容量 1GB～2GB 程度．Android のシステム領域として用いる．

kernel のビルド完了後，$ANDROID/kernel-beagleboard/arch/arm/boot/uImage を DISK1 にコピーする．

また，上で作成された$ANDROID/vendor/sola/omap3/image/beagleboard/android の中身を DISK3 にコピーする．コピー後に以下を実行しアクセス権限を変更する．

```
sudo chmod -R 777 /media/DISK3
```

Android のビルドはこれで完了である．今後カメラなどの各種デバイスを利用するために，kernel や Android ソースを変更していく．変更した場合，再びビルドし，kernel を DISK1 に格納，システムを DISK3 に保存しなおす．

2.6.4 ボード構成の設定

SD カードを 3 つのパーティションに区切り，DISK1 を kernel 領域， DISK2 を保存領域，DISK3 をシステム領域に設定したが，BeagleBoard のデフォルトの設定ではそのような構成を前提としていないため，事前に Windows 端末上で設定変更しておく必要がある．

Putty というターミナルソフトウェアを用いて BeagleBoard の設定を変更する．Putty で BeagleBoard へシリアル接続するために，以下の 2 点の変更を行う．この変更により，BeagleBoard に接続可能となる．

```
通信速度: 115200
フロー制御: None
```

Windows 端末に Putty で BeagleBoard をシリアル接続した状態で BeagleBoard を起動すると, Hit Any Key と表示されるので, 何かキーを叩く. コマンドで printenv を実行すると様々な環境変数をみることができる．ここでは mmcroot という環境変数を変更する．

```
setenv mmcroot "/dev/mmcblk0p3 rw init=/init"
```

さらに以下を入力して変更した設定を保存する．

```
saveenv
```

入力完了後リブートすれば，android が起動する．

2.6.5 Android ソースの変更

2.6.3 章でダウンロードした vendor_tk-beagle-froyo2.2.x-20110725.tar.gz（以下 vendor_tk）の適応により変更される内容は以下の通りである．

○日本語フォントがデフォルトでインストールされるようにする．
○常時点灯に設定する．
○機内モードを削除する．
○各種デバイスに対応させる（USB カメラ，Wifi，Bluetooth，Audio，USB シリアル，タッチスクリーン）．

詳しい設定方法については巻末の付録Ⅰ「Android の環境設定」と付録Ⅱ「各種デバイスへの対応」にまとめ，下記では本論文で使用するデバイス用の変更についてまとめる．

(a) USB カメラの利用

USB カメラを利用するために，vender_tk により以下のような変更がされている．まず，kernel の変更についてであるが，下記 kenel の設定ファイル（sola_omap3_beagle_android_defconfig）に以下が追加される．

```
# Camera 関連
CONFIG_VIDEO_DEV=y
CONFIG_VIDEO_V4L2_COMMON=y
CONFIG_VIDEO_V4L1_COMPAT=y
CONFIG_VIDEO_MEDIA=y
CONFIG_VIDEO_V4L2=y
CONFIG_VIDEO_CAPTURE_DRIVERS=y
```

続いて，Android で USB カメラを利用するためにソースを変更するわけだが，既にソースは各種デバイスに対応した vendor_tk により変更されている．具体的には USB カメラへの対応には libcameraservice の変更により行っているが，ファイルの変更内容の詳細については，文献[5]を参照とする．

$ANDROID/vendor/sola/beagleboard/BoardConfig.mk にて，下記の行の先頭に「#」をつけてコメントアウトされている．

```
USE_CAMERA_STUB := true
          ↓
# USE_CAMERA_STUB := true
```

そして，$ANDROID/system/core/init/devices.c の以下の部分を 666 設定（誰でも読み書き可能）に変更することでユーザー利用可にしている．なお，この設定は android2.3 以降では，/system/etc/uevent.rc で行う．

```
{ "/dev/ttyUSB0",      0666,   AID_ROOT,      AID_ROOT,      0 },
{ "/dev/ttyUSB1",      0666,   AID_ROOT,      AID_ROOT,      0 },
```

これにより，自作動画処理アプリケーションを利用できるようになる．

(b) USB シリアル変換デバイスの利用

次に, USB シリアル変換デバイスについて述べる. 特に本研究で利用した Elecom UC-SGT や双葉工業製モータ制御用デバイス RSC-U485 を使うための方法を記す．以下では，はじめに kernel について述べ，その後 Android 側の変更について述べる．以下 kernel についてである．まず，下記 kernel の設定ファイル（sola_omap3_beagle_android_defconfig）にて，下記の 3 つの設定が有効にされている．

```
CONFIG_USB_SERIAL=y
CONFIG_USB_SERIAL_FTDI_SIO=y
CONFIG_USB_SERIAL_PL2303=y
```

なお，FTDI_SIO は RSC-U485 を使うためのドライバ，PL2303 は Elecom UC-SGT 等を用いるためのドライバである．PL2303 利用の Elecom UC-SGT はこれだけで使えるようになるが，FTDI_SIO 利用の RSC-U485 は使うためにもうひと手間必要である．変更が必要なのは下記の 2 ファイルである．

$ANDROID/kernel-beagleboard/drivers/usb/serial/ftdi_sio.h

$ANDROID/kernel-beagleboard/drivers/usb/serial/ftdi_sio.c

このファイルには，ftdi_sio ドライバを用いるデバイスの vendor ID と product ID （0x1115 と 0x0008）を登録する必要がある（多くのデバイスは既に書かれているのだが，RSC-U485 は書かれていないため). なお, vendor ID と product ID を調べるには，Ubuntu などに USB デバイスを挿した状態で，lsusb というコマンドを実行すればよい． ftdi_sio.h には，下記のように RATOC のデバイスの後ろに 2 行挿入されている．

```
#define RATOC_VENDOR_ID        0x0584
#define RATOC_PRODUCT_ID_USB60F 0xb020

#define FUTABA_VID                    0x1115
#define FUTABA_RSCU485_PID     0x0008
```

ftdi_sio.c には，下記の位置に 1 行挿入する．

```
  { USB_DEVICE(FTDI_VID, FTDI_DOMINTELL_DGQG_PID) },
  { USB_DEVICE(FTDI_VID, FTDI_DOMINTELL_DUSB_PID) },
  { USB_DEVICE(FUTABA_VID, FUTABA_RSCU485_PID) },
  { },                                  /* Optional parameter entry */
```

以上が kernel の変更点である．

続いて Android 側の変更についてであるが，USB シリアルデバイスを使うためには $ANDROID/system/core/init/devices.c にて，下記のように USB シリアル変換デバイスが 666 設定(誰でも読み書き可)になっている必要がある． なお，Android2.3 以降ではこの設定は/system/etc/uevent.rc にて行う．

```
{ "/dev/ttyUSB0",      0666,   AID_ROOT,      AID_ROOT,      0 },
{ "/dev/ttyUSB1",      0666,   AID_ROOT,      AID_ROOT,      0 },
```

第3章　画像処理の高速化

対象物追跡を行うためには画像処理の高速化は必要不可欠となる．これは，処理が重いと動作にラグができてしまい，実用性に欠けるためである．

処理を高速化するために，プログラムを書き換え，最も処理の速い方式を採用して対象物追跡のアプリケーションを作成する．

3.1　処理の高速化の手法

3.1.1　JIT（Just-In-Time Compiler）

JIT(Just-In-Time Compiler)は，Java で用いられる実行時コンパイル機能の名称であり，中間コードから機械語への変換処理を実行直前にまとめて行う機能である．開発時にコンパイラで機械語に変換する場合とほとんど変わらない実行速度で実行できると言われている．Android ではバージョン 2.2 より搭載された．

ここで，C 言語と Java を比較してみよう．C 言語はビルド（コンパイル）によって実行ファイルを各々のプロセッサ用に最適化することができる（図 3.1）が，Java はコンパイルにより実行ファイルを中間ファイルにし，JVM（Java Virtual Machine：仮想マシン）上で動作させるため，最適化することができない（図 3.2）．Java は JVM の環境が整っていればどのプロセッサでも動作するという利点があるが，プロセッサの最適化ができないことにより速度面で不利である．

そこで登場したのが，実行時コンパイル機能 JIT(Just-In-Time Compiler)である．最適化ができないことによる速度面での不利を，ファイルの実行時に最適化を行うことで補い，開発時にコンパイラで最適化したものとほとんど変わらない実行速度で実行することを可能だと言われている．

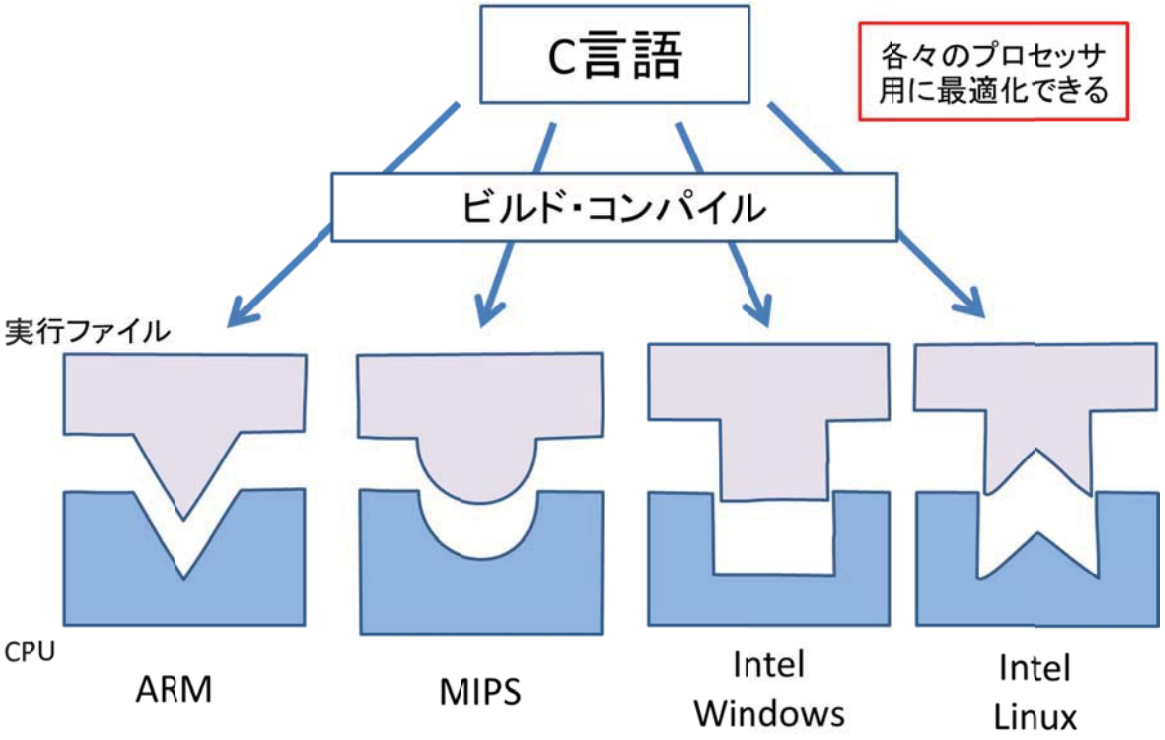


図 3.1　C 言語コンパイル簡易図

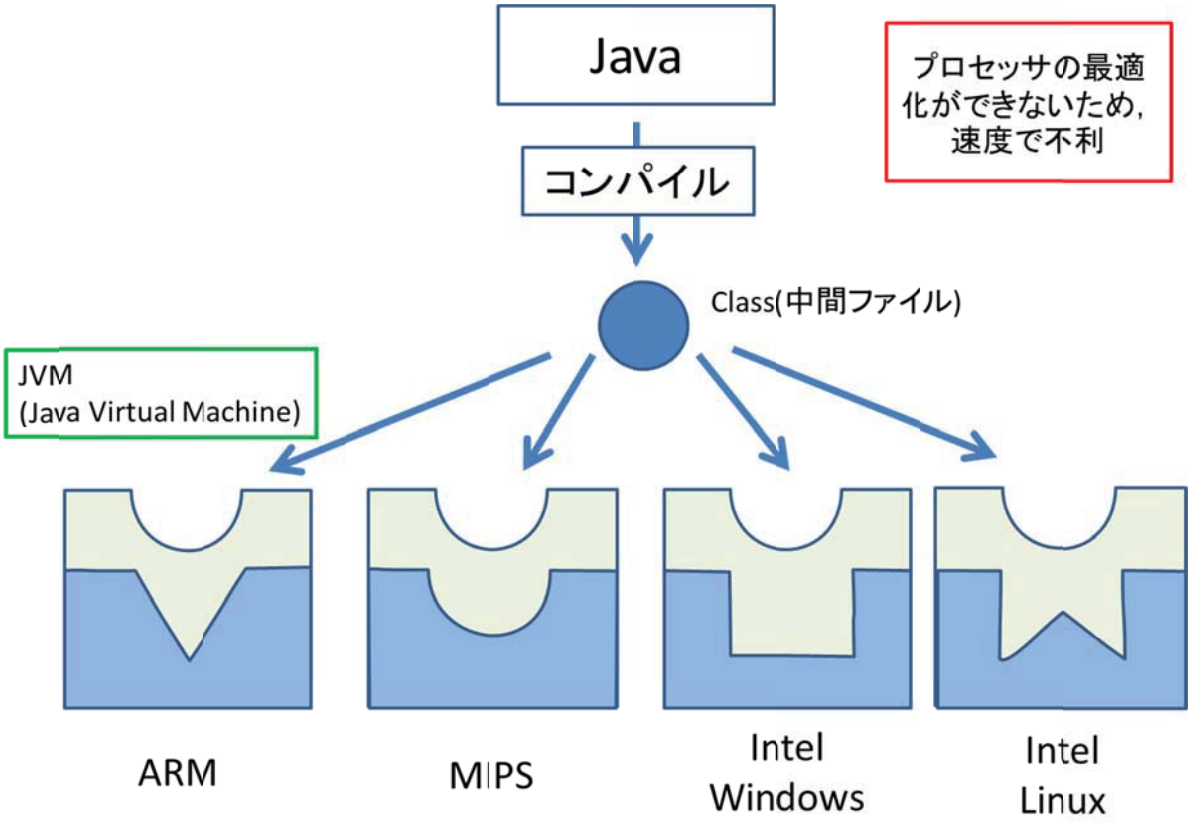


図 3.2　Java コンパイル簡易図

3.1.2　JNI（Java Native Interface）

JNI(Java Native Interface)は，Java で記述されたプログラムと他の言語（C，C++）で書かれたコードを連携するためのインターフェースである．メリットとして既存のライブラリの活用ができることや，パフォーマンスの向上が挙げられる．これは，C 言語で各プロセッサに最適化したライブラリを Java から利用できるからである．高速化のためには，理論的には JNI を用いるよりも 3.1.1 章で触れた JIT を利用したほうがよい．それは，1 つの中間コードで複数のプロセッサ（ARM，MIPS，Intel など）に対応できるためである．しかし，今現在高速処理が必要な Android アプリケーションでは，多くの場合 JNI により ARM 用に最適化している．その理由として，Android2.1 以前では JIT が使えなかったことや，JIT による最適化がそれほど速くないことが挙げられる．このことは本研究でも後に確認する．しかしながら，JNI で ARM 用に最適化された Android アプリケーションは，今後「Intel 用 Android」や「MIPS 用 Android」が出てきた際に，そのままでは動かない（再ビルドを要する）という問題を孕んでいる．

JNI の利用法として，計算量の多いプログラムを部分的にネイティブコードに置き換えて高速化する場合の他に，ハードウェアアクセスする場合（シリアル通信など）がある．特に，Java で作成したユーザーアプリケーションからでは直接ハードウェアにアクセスすることができないため，ハードウェアを利用するには JNI の利用は必須である．図 3.3 に JNI を用いたハードウェアアクセスの流れを示す．本研究では処理の高速化のためだけではなく，第 5 章の駆動部においてモータへのアクセスに利用する．

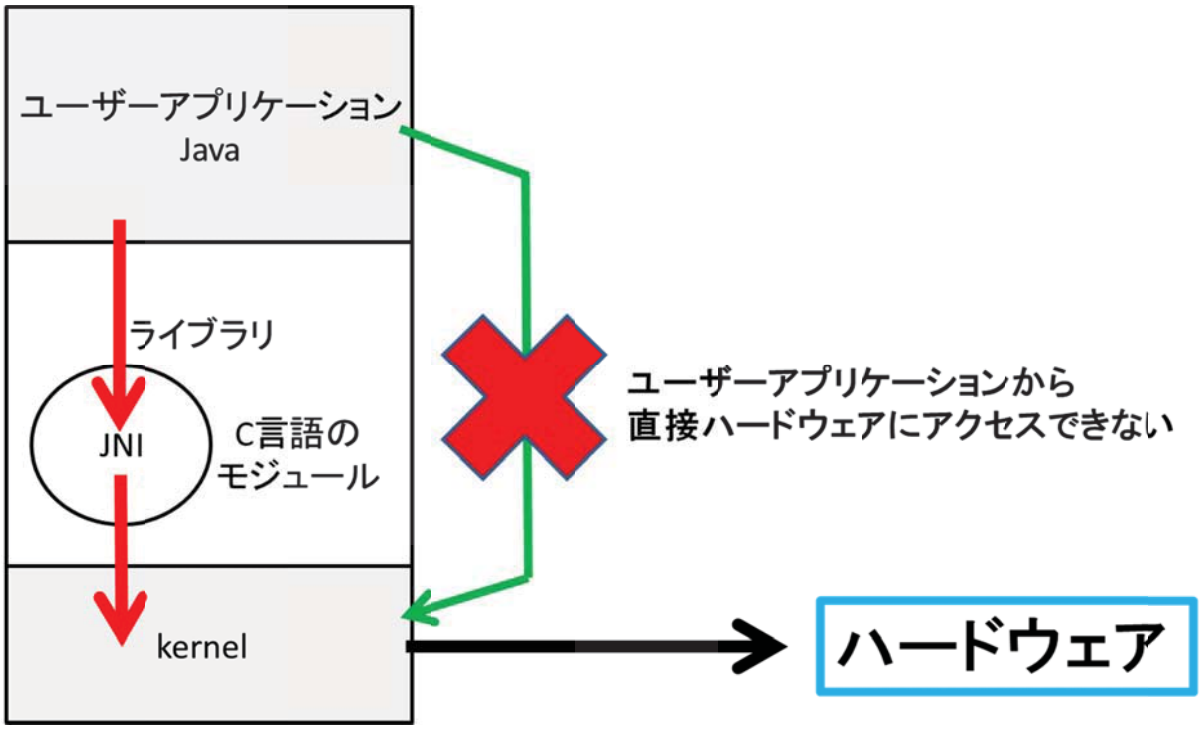


図 3.3　JNI を用いたハードウェアアクセスの流れ

3.1.3　Neon 命令

Neon 命令は ARM の SIMD（Single Instruction Multiple Data）命令セットで，1 つの命令でレジスターにセットされた複数のデータに対して一度に演算できる． ARMv6 で採用された SIMD 命令の拡張版で，新しい“ARMv7”アーキテクチャで使われる．ただし，ARMv7 でも Neon がサポートされていないものもあるので，利用の際は注意が必要である．例を挙げれば，タブレットでよく扱われる Tegra2 は ARMv7 であるが，Neon 非対応である．本テーマのターゲットである BeagleBoard，PandaBoard の CPU は ARMv7 以降で，Neon 対応のコアを採用しているため，この NEON テクノロジーを利用することが可能である．主な特徴は以下のとおりである．

●8，16，32，64bit 整数，単精度浮動小数点データの SIMD 演算をサポートしている．

●256byte の専用レジスターファイルを備え，32 個の 64bit レジスターや 16 個の 128bit レジスターとして使用可能な柔軟性を持つ．本研究では図 3.4 のように 128bit レジスターを整数型 4 つ分として用いる．

●C 言語でプログラミング可能である．

なお，本研究では Neon 命令を C 言語で記述するので，3.1.2 章で述べた JNI が必須である．

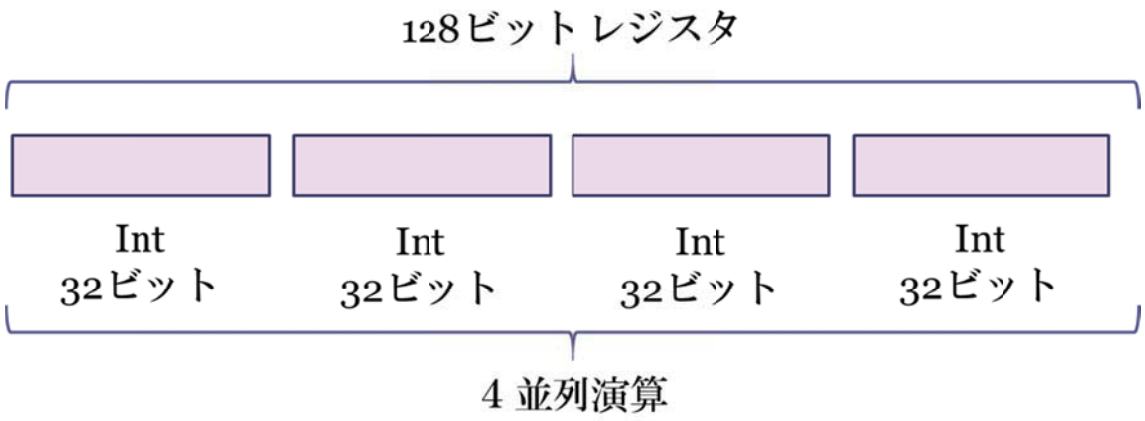


図 3.4　Neon 命令を用いた 4 並列演算の例

3.2 仮説

ここまで述べた技術を実際に画像処理に適用し，処理の高速化が図れるか検証する．本研究では次の 5 つの方法でアプリケーションを記述し，処理速度の比較を行う．検証を行うのは，BeagleBoard，PandaBoard，およびスマートフォンの Xperia（SO-01B）とする．

①Java のみ（JIT なし）
②Java のみ（JIT あり）
③JNI の利用：ユーザー処理部の一部を C 言語化
④JNI+Neon の利用：ユーザー処理部に ARM の Neon 命令を利用
⑤システム部でも Neon を利用

上記の 5 つの方式で書かれたプログラムでは，①から⑤になるにつれて，だんだんと処理速度は速くなり，⑤のシステム部でも Neon を利用したものが最も速くなるだろうと推測される．この仮説のもと，処理時間を測定し，比較検証を行う．

第4章　カメラと高速化

本章では，対象物追跡におけるカメラの概要と，カメラアプリケーションの画像処理における高速化の検証について触れる．

4.1　カメラアプリケーション

カメラを利用し，速度比較を行うためのアプリケーションを作成した．カメラで取得した画像を表示させるプログラムであり，静止画，動画を撮影する機能はつけていない．なお，BeagleBoard と PandaBoard では対応させた USB カメラを利用し，Xperia では内蔵カメラを利用する．以下の表 4.1 にデバイスごとのフレームレートと利用するカメラの解像度をまとめる．ここで，フレームレートの単位は fps = frame per second（1 秒あたりのフレーム数）である．

表 4.1　各デバイスのフレームレートと解像度

	フレームレート[fps]	解像度(横x縦)
BeagleBoard	15	352x288
PandaBoard	30	352x288
Xperia	30	854x480

本アプリケーションの主たる機能は，タッチパネルに触れると，エッジを緑色で表示する画像処理を行うモードに切り替わるというものである．図 4.1 にこの処理結果のサンプルを示す．

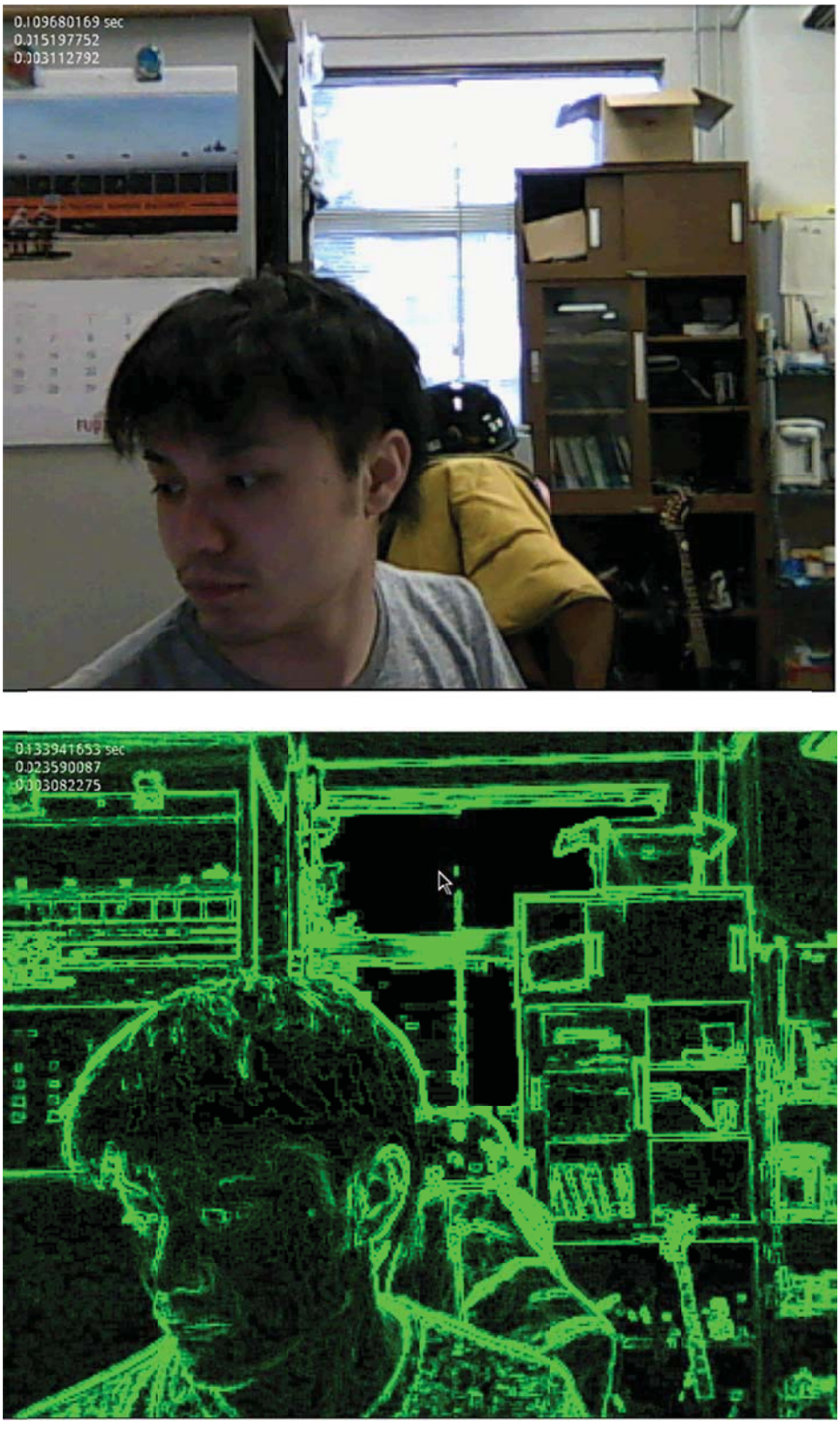


図 4.1　カメラアプリケーション（上：カメラからの画像　下：エッジ処理後）

この画像処理プログラムを第 3 章で示した①～⑤の方式（以下に再掲）で書き換え，処理速度の高速化を検証する．以下それぞれについて解説する．

①Java のみ（JIT なし）
②Java のみ（JIT あり）
③JNI の利用：ユーザー処理部の一部を C 言語化
④JNI+Neon の利用：ユーザー処理部に ARM の Neon 命令を利用
⑤システム部でも Neon を利用

まず①Java のみ（JIT なし）と②Java のみ（JIT あり）の 2 つであるが，すべて Java を用いて書いた同じプログラムを使用する．①の場合は JIT 非対応でビルドした Android version2.2 を，②の場合は JIT 対応でビルドした Android version2.2 を用いて処理速度を測定し，比較する．これらは Android ビルド時のオプション WITH_JIT := true/false によって切り替えられる．BeagleBoard と PandaBoard は自分で OS をビルドするので，このような変更が可能であるが，Xperia ではこのような変更はできないので，公式 2.1（JIT 非対応）と公式 2.3.3（JIT 対応）を用いて処理速度を測定し，比較した．

次に③JNI を利用のプログラムは，画像処理のプログラムの一部を C 言語モジュール化して処理速度の向上を図るものである．以下より変更の一部を示し，解説する．まず，Java のみで書かれた画像処理の一部である．こういった画像処理部を C 言語のモジュールを利用することで処理速度の向上を狙う．

```
public void rgb16toABGRY(byte[] data, int[] rgbbuf, int[] ybuf, int width,
int height){
       int y;
       int pos;
       int frames =  (width * height) * 2;

       for (y = 0, pos=0; y <frames; y+=2, pos+=1) {

              int tmp1 = (0xff & ((int) data[y + 0]));
              int tmp2 = (0xff & ((int) data[y + 1]));
              int rgb16 = tmp2<<8 | tmp1;

              int r = (rgb16 & 0xf800)>>8;
              int g = (rgb16 & 0x07e0)>>3;
              int b = (rgb16 & 0x1f)<<3 ;

              ybuf[pos]= (306*r + 601*g + 117*b)>>10;
              rgbbuf[pos] = 0xff000000| r<<16 | g<<8 | b;

       }
}

public void calcEdgePixels(int[] ybuf, int[] outbuf, int width, int height){

       int i,j, pos;

       for(j=0, pos=0; j<height-1 ; j++){
              for(i=0 ; i<width-1 ; i++, pos++){
```

```
                    int edgeval = 0;

                    int val = ybuf[pos] - ybuf[pos+1] + ybuf[pos+width] -
                    ybuf[pos+width+1];
                    edgeval += Math.abs(val);
                    val = ybuf[pos] - ybuf[pos+1] - ybuf[pos+width] +
                    ybuf[pos+width+1];
                    edgeval += Math.abs(val);
                    val = ybuf[pos] + ybuf[pos+1] - ybuf[pos+width] -
                    ybuf[pos+width+1];
                    edgeval += Math.abs(val);

                    edgeval <<= 2;

                    edgeval = edgeval>255 ? 255 : edgeval;

                    outbuf[pos] = 0xff000000 | edgeval<<8;
            }

            outbuf[pos++] = 0xff000000;
        }
        for(i=0 ; i<width ; i++){
            outbuf[pos++] = 0xff000000;
        }
}
```

このソースを JNI に対応させたコードについて解説する．Java のみで書かれていたプログラムに，C 言語のモジュールを呼び出す宣言文を挿入し，C 言語で書かれた画像処理部のプログラムを利用する．以下モジュールを呼び出す宣言文の一例である．

```
public native void rgb16toABGRY(byte[] data, int[] rgb, int[] y,int width,
int height);
public native void calcEdgePixels(int[] ybuf, int[] outbuf, int width, int
height);
```

これにより呼び出される C 言語モジュールが以下のようになっている．呼び出されたモジュールは先述の Java のみのプログラムと同じ画像処理を行う．

```
void
Java_com_tk_camerajni_CameraPreview_rgb16toABGRY( JNIEnv* env,
                                                  jobject thiz,
byteArray data, jintArray rgbbuf, jintArray ybuf, jint width, jintheight){

    jboolean bo;

    jbyte *jdata = (*env)->GetByteArrayElements(env,data,&bo);
    jint *jrgbbuf = (*env)->GetIntArrayElements(env,rgbbuf,&bo);
    jint *jybuf = (*env)->GetIntArrayElements(env,ybuf,&bo);

    int i, y;
    int pos;
    int frames = width * height;
```

```
     if(rgb_tbl_ready==0){
       for(i=0 ; i<256 ; i++){
              r306_tbl[i] = 306*i;
              g601_tbl[i] = 601*i;
              b117_tbl[i] = 117*i;
       }

       rgb_tbl_ready=1;
     }
       short* sdata;
       sdata = (short*)jdata;
       for (pos=0; pos<frames; pos+=1) {

              int rgb16 = sdata[pos];

              int r = (rgb16 & 0xf800)>>8;
              int g = (rgb16 & 0x07e0)>>3;
              int b = (rgb16 & 0x1f)<<3 ;

       //jybuf[pos]= (r306_tbl[r] + g601_tbl[g] + b117_tbl[b])>>10;
              jybuf[pos]= (306*r + 601*g + 117*b)>>10;
              jrgbbuf[pos] = 0xff000000| b<<16 | g<<8 | r;
       }
       (*env)->ReleaseByteArrayElements(env, data, jdata, 0);
       (*env)->ReleaseIntArrayElements(env, rgbbuf, jrgbbuf, 0);
       (*env)->ReleaseIntArrayElements(env, ybuf, jybuf, 0);
}
void
Java_com_tk_camerajni_CameraPreview_calcEdgePixels( JNIEnv* env,
                                                    jobject thiz,
              jintArray ybuf, jintArray outbuf, jint width, jint height){

       jboolean bo;

       jint *jybuf = (*env)->GetIntArrayElements(env,ybuf,&bo);
       jint *joutbuf = (*env)->GetIntArrayElements(env,outbuf,&bo);

       int i,j, pos;

       for(j=0, pos=0; j<height-1 ; j++){
              for(i=0 ; i<width-1 ; i++, pos++){
                     int edgeval = 0;

                     int val = jybuf[pos] - jybuf[pos+1] + jybuf[pos+width]
                     - jybuf[pos+width+1];
                     edgeval += abs(val);
                     val = jybuf[pos] - jybuf[pos+1] - jybuf[pos+width] +
                     jybuf[pos+width+1];
                     edgeval += abs(val);
                     val = jybuf[pos] + jybuf[pos+1] - jybuf[pos+width] -
                     jybuf[pos+width+1];
                     edgeval += abs(val);

                     edgeval <<= 2;
```

```
                        edgeval = edgeval>255 ? 255 : edgeval;

                        joutbuf[pos] = 0xff000000 | edgeval<<8;
                }

                joutbuf[pos++] = 0xff000000;
        }
        for(i=0 ; i<width ; i++){
                joutbuf[pos++] = 0xff000000;
        }


        (*env)->ReleaseIntArrayElements(env, ybuf, jybuf, 0);
        (*env)->ReleaseIntArrayElements(env, outbuf, joutbuf, 0);
}
```

変更したプログラムの流れの簡易図を図 4.2 に示す，また， JNI を利用するには環境設定などが必要であるが, ここでは詳細について触れず, JNIの詳細な利用方法は巻末の付録Ⅲ「JNIの利用における環境設定と利用法」にまとめる.

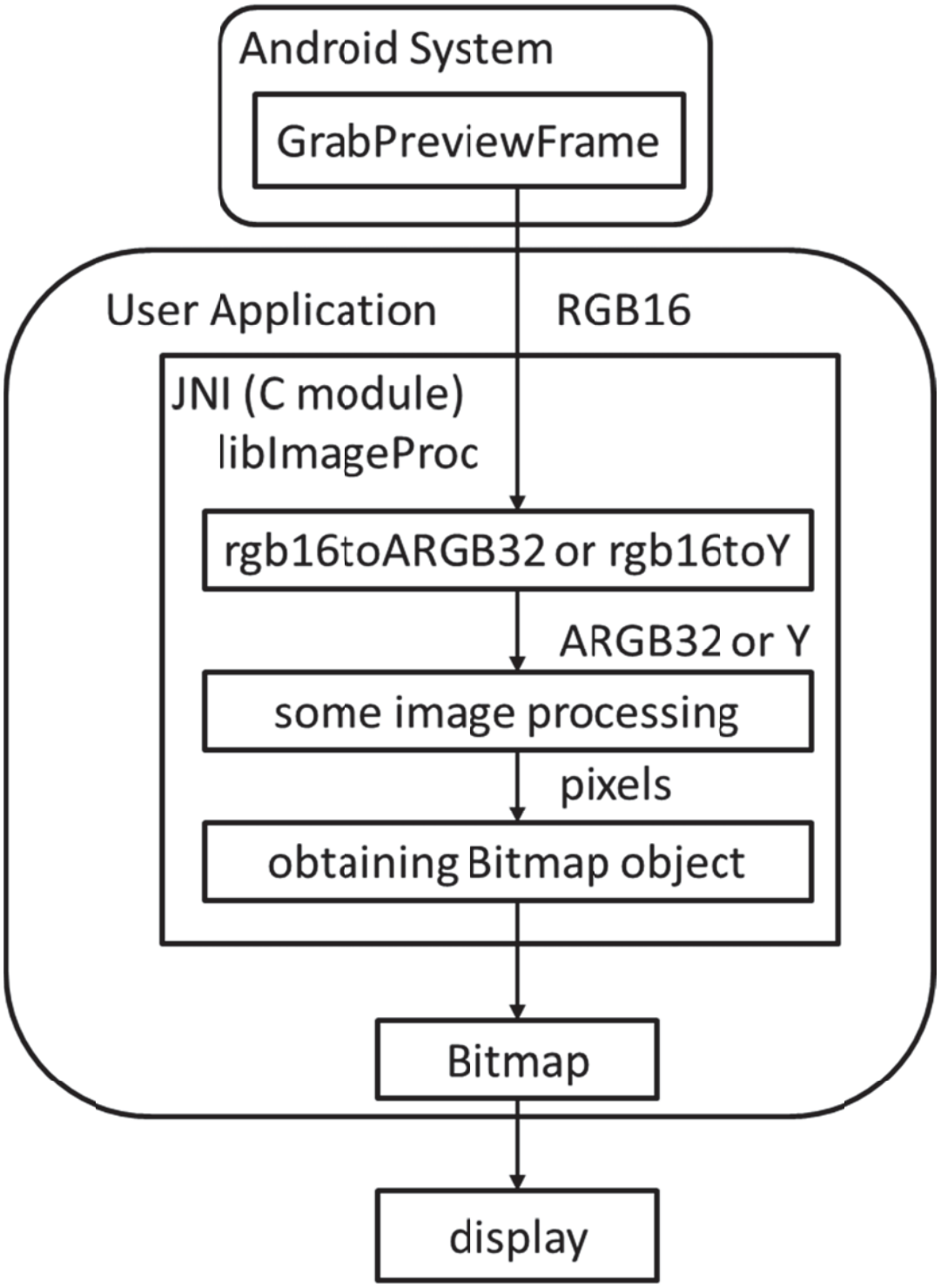


図 4.2　JNI を用いたカメラアプリケーションの処理の流れ

④Neon 命令の利用であるが, C 言語にて並列演算をする Neon 命令を実行するモジュールを作成し，JNI を用いて利用する．以下に，Neon 命令を用いたプログラムの一部を示す．このプログラムにより，Neon 命令を利用して先述のプログラムと同じ画像処理を行う．

```
void
Java_com_tk_cameraneon_CameraPreview_rgb16toABGRY( JNIEnv* env,
                                                   jobject thiz,
               jbyteArray data, jintArray rgbbuf, jintArray ybuf, jint width,
jint height){

       jboolean bo;

       jbyte *jdata = (*env)->GetByteArrayElements(env,data,&bo);
       jint *jrgbbuf = (*env)->GetIntArrayElements(env,rgbbuf,&bo);
       jint *jybuf = (*env)->GetIntArrayElements(env,ybuf,&bo);

       int i, y;
       int pos;
       int frames = width * height;

       int32x4_t getr = vdupq_n_s32(0xf800);
       int32x4_t getg = vdupq_n_s32(0x07e0);
       int32x4_t getb = vdupq_n_s32(0x1f);
       int32x4_t rgb_trans = vdupq_n_s32(0xff000000);
       int32x4_t rgb16, r, g, b, r306, g601, b117, out;

       int rgb16_org[4];

       short* sdata;
       sdata = (short*)jdata;

       for (pos=0; pos<frames; pos+=4) {

              for(i=0; i<4 ; i++) {
                     rgb16_org[i] = (int)sdata[pos+i];
              }

              rgb16 = vld1q_s32(rgb16_org);

              r  = vshrq_n_s32(vandq_s32(rgb16, getr), 8);
              g  = vshrq_n_s32(vandq_s32(rgb16, getg), 3);
              b  = vshlq_n_s32(vandq_s32(rgb16, getb), 3);

              out = vaddq_s32(vmulq_n_s32(r, 306),
                                                 vmulq_n_s32(g, 601));
              //out = vaddq_s32(out,vmulq_n_s32(b, 117));
              out = vmlaq_n_s32(out, b, 117);
              out = vshrq_n_s32(out, 10);

              vst1q_s32(&jybuf[pos], out);

              //out = vdupq_n_s32(0xff000000);
              //out = vorrq_s32(out, vshlq_n_s32(b, 16));
              out = vorrq_s32(rgb_trans, vshlq_n_s32(b, 16));
              out = vorrq_s32(out, vshlq_n_s32(g, 8));
```

```
                out = vorrq_s32(out, r);

                vst1q_s32(&jrgbbuf[pos], out);

        }

        (*env)->ReleaseByteArrayElements(env, data, jdata, 0);
        (*env)->ReleaseIntArrayElements(env, rgbbuf, jrgbbuf, 0);
        (*env)->ReleaseIntArrayElements(env, ybuf, jybuf, 0);
}
void
Java_com_tk_cameraneon_CameraPreview_calcEdgePixels( JNIEnv* env,
                                                jobject thiz,
                jintArray ybuf, jintArray outbuf, jint width, jint height){

        jboolean bo;

        jint *jybuf = (*env)->GetIntArrayElements(env,ybuf,&bo);
        jint *joutbuf = (*env)->GetIntArrayElements(env,outbuf,&bo);

        int i,j, pos;

        int32x4_t p0, px, py, pxy;
        int32x4_t val0, val, val2, val255, edgeval;
        val0 = vdupq_n_s32(0xff000000);
        val255 = vdupq_n_s32(255);

        int w4 = (width>>2)<<2 ;  // widthより小さい最大の4の倍数
        int wmod4 = width%4;
        for(j=0, pos=0; j<height-1 ; j++){
                for(i=0 ; i<w4 ; i+=4, pos+=4){//width が4でない場合の対策


                        p0  = vld1q_s32(&jybuf[pos]);
                        px  = vld1q_s32(&jybuf[pos+1]);
                        py  = vld1q_s32(&jybuf[pos+width]);
                        pxy  = vld1q_s32(&jybuf[pos+width+1]);

                        val = vaddq_s32(p0, py);
                        val2 = vaddq_s32(px, pxy);
                        edgeval = vabdq_s32(val, val2);

                        val = vaddq_s32(p0, pxy);
                        val2 = vaddq_s32(px, py);
                        edgeval = vabaq_s32(edgeval, val, val2);

                        val = vaddq_s32(p0, px);
                        val2 = vaddq_s32(py, pxy);
                        edgeval = vabaq_s32(edgeval, val, val2);

                        edgeval=vminq_s32(vshlq_n_s32(edgeval, 2), val255);

                        edgeval = vorrq_s32(val0, vshlq_n_s32(edgeval, 8));

                        vst1q_s32(&joutbuf[pos], edgeval);
```

```
                }
                pos += wmod4;

        }

        // 最下段を黒に
        for(i=0 ; i<w4 ; i+=4, pos+=4){
                vst1q_s32(&joutbuf[pos], val0);
        }

        // 横方向4の倍数の余り、および再右列を黒に
        int ps;
        if(wmod4==0){
                ps = width-1;
        }else{
                ps = w4;
        }
        pos = ps;
        for(j=0 ; j<height ; j++){
                for(i=ps ; i<width ; i++){
                        joutbuf[pos++]=0xff000000;
                }
                pos+=ps;
        }

        (*env)->ReleaseIntArrayElements(env, ybuf, jybuf, 0);
        (*env)->ReleaseIntArrayElements(env, outbuf, joutbuf, 0);
}
```

⑤システム部でも Neon を利用する方法では，Android のシステムライブラリの libcamera.so と libcameraservice.so を変更することで，システムのカメラ部でも Neon を使うようにした．この変更を行うための具体的な手順を以下に示す．

Android ソースの framework/base/service/camera/libcameraservice/というフォルダ内にある CameraService.cpp と V4L2Camera.cpp の 2 つのファイルを Neon 化対応に変更しビルドすると Neon 化対応に変更された libcamera.so と libcameraservice.so ができる．変更点を以下に示す．

まず， CameraService.cpp を変更する．画像のコピーを Neon 命令で行っている．

```
 #include <arm_neon.h>  追記

// memcpy(previewBuffer->base(), (uint8_t *)heap->base() + offset, size);
コメントアウト
for(int i=0 ; i<size ; i+=16){  以下追記
       vst1q_u8((unsigned char*)previewBuffer->base()+i, vld1q_u8
       ((unsigned char*)heap->base() + offset + i));
   }
```

次に，V4L2Cameraservice.so を変更する．カメラから取得した YUYV 形式の画像を RGB 形式に変換する関数を Neon 化に対応するように変更した． Neon 化対応したソースコードを以下に記す．

```
void V4L2Camera::convert(unsigned char *buf, unsigned char *rgb, int width,
int height)
{
    int x,y,z=0;
    int blocks;

    blocks = (width * height) * 2;

    int Y1align[4], Ualign[4], Y2align[4], Valign[4];
    int RGB1[4], RGB2[4];


    int32x4_t Y1_4, U_4, Y2_4, V_4;
    int32x4_t Y1_1192, Y2_1192, V128, U128;
    int32x4_t val16, val128, val0, val255;

    int32x4_t Rbase, Gbase, Bbase;
    int32x4_t R1_4, G1_4, B1_4;
    int32x4_t R2_4, G2_4, B2_4;
    int32x4_t RGB1_4, RGB2_4;

    val16 = vdupq_n_s32(16);
    val128 = vdupq_n_s32(128);
    val255 = vdupq_n_s32(255);
    val0 = vdupq_n_s32(0);

    for (y = 0; y < blocks ; y+=16) {

      for(int i=0 ; i<4 ; i++){
        int i4 = i<<2;
        Y1align[i] = buf[y+i4];
        Ualign[i] = buf[y+i4+1];
        Y2align[i] = buf[y+i4+2];
        Valign[i] = buf[y+i4+3];
      }

      Y1_4 = vld1q_s32(Y1align);
      U_4 = vld1q_s32(Ualign);
      Y2_4 = vld1q_s32(Y2align);
      V_4 = vld1q_s32(Valign);

      Y1_1192 = vmulq_n_s32(vsubq_s32(Y1_4 , val16), 1192);
      Y2_1192 = vmulq_n_s32(vsubq_s32(Y2_4 , val16), 1192);

      U128 = vsubq_s32(U_4 , val128);
      V128 = vsubq_s32(V_4 , val128);

      Rbase = vmulq_n_s32(V128, 1634);
      Gbase = vmulq_n_s32(V128, -833);
      Gbase = vmlaq_n_s32(Gbase, U128, -400);
```

```
    Bbase = vmulq_n_s32(U128, 2066);

    R1_4 = vshrq_n_s32(vaddq_s32(Y1_1192 , Rbase), 10);
    G1_4 = vshrq_n_s32(vaddq_s32(Y1_1192 , Gbase), 10);
    B1_4 = vshrq_n_s32(vaddq_s32(Y1_1192 , Bbase), 10);

    R1_4 = vminq_s32(vmaxq_s32(R1_4, val0), val255);
    G1_4 = vminq_s32(vmaxq_s32(G1_4, val0), val255);
    B1_4 = vminq_s32(vmaxq_s32(B1_4, val0), val255);

    R2_4 = vshrq_n_s32(vaddq_s32(Y2_1192 , Rbase), 10);
    G2_4 = vshrq_n_s32(vaddq_s32(Y2_1192 , Gbase), 10);
    B2_4 = vshrq_n_s32(vaddq_s32(Y2_1192 , Bbase), 10);

    R2_4 = vminq_s32(vmaxq_s32(R2_4, val0), val255);
    G2_4 = vminq_s32(vmaxq_s32(G2_4, val0), val255);
    B2_4 = vminq_s32(vmaxq_s32(B2_4, val0), val255);
    R1_4 = vshlq_n_s32(vshrq_n_s32(R1_4, 3), 11);
    G1_4 = vshlq_n_s32(vshrq_n_s32(G1_4, 2), 5);
    B1_4 =           vshrq_n_s32(B1_4, 3);

    RGB1_4 = vorrq_s32(R1_4, G1_4);
    RGB1_4 = vorrq_s32(RGB1_4, B1_4);

    R2_4 = vshlq_n_s32(vshrq_n_s32(R2_4, 3), 11);
    G2_4 = vshlq_n_s32(vshrq_n_s32(G2_4, 2), 5);
    B2_4 =           vshrq_n_s32(B2_4, 3);

    RGB2_4 = vorrq_s32(R2_4, G2_4);
    RGB2_4 = vorrq_s32(RGB2_4, B2_4);

    vst1q_s32(RGB1, RGB1_4);
    vst1q_s32(RGB2, RGB2_4);

    short* rgbs = (short *)(&rgb[y]);

    for(int i=0 ; i<4 ; i++){
      int i2 = i<<1;
      rgbs[i2] = RGB1[i];
      rgbs[i2+1] = RGB2[i];
    }

  }

}

}; // namespace android
```

4.2　処理時間の測定

4.2.1　測定理論

エッジ処理のプログラムの処理の流れは図 4.3 のようになっている．1 フレームでこの処理が 1 回行われる．ここで，フレームとは画像処理の 1 サイクルのことを指す．まず，カメラで画像を取得し，表示までの 1 サイクルにかかる時間を測定し，LogCat に Log として出力させることで，プログラムの処理にかかる時間を測定することができる．今回は以下の測定方法に示すように，20 回の処理の平均時間を表示するようにした．それを 10 回記録し，1 フレームの処理時間を算出する．1 回の処理ごとに表示させず，20 回の処理の平均時間で表示させるのは，1 フレームの処理時間が非常に短く，測定誤差が大きくなり，数字の大幅なばらつきが出てしまうのを防ぐためである．また、図 4.3 に示してあるように，以下ではハードウェアの依存の高い処理部分をシステム部，それ以外の処理部分をユーザー処理部とする．

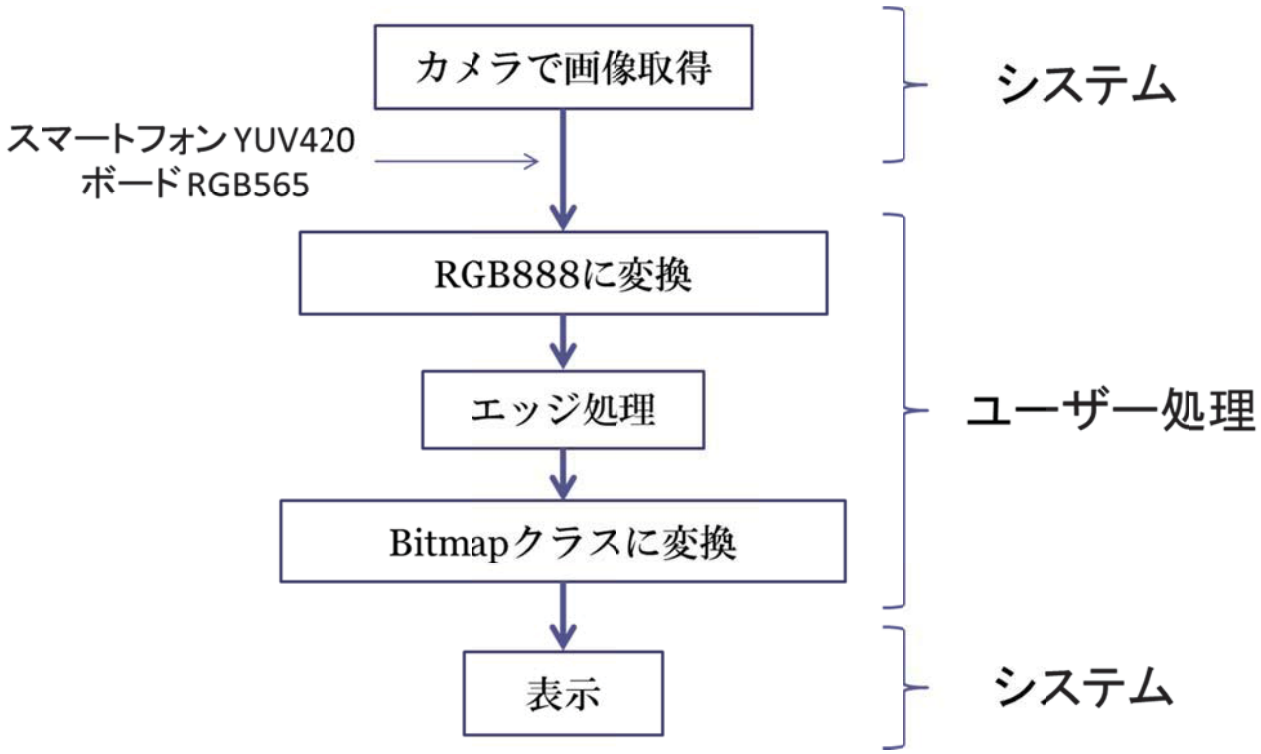


図 4.3　プログラムの処理の流れとシステム部・ユーザー処理部の定義

4.2.2　測定方法

処理にかかる時間の確認はプログラム内から LogCat に Log を出力することで測定する．20 フレームの平均時間をさらに 10 回平均することで，1 フレームの処理にかかる時間を算出する．図 4.4 にて図解する．

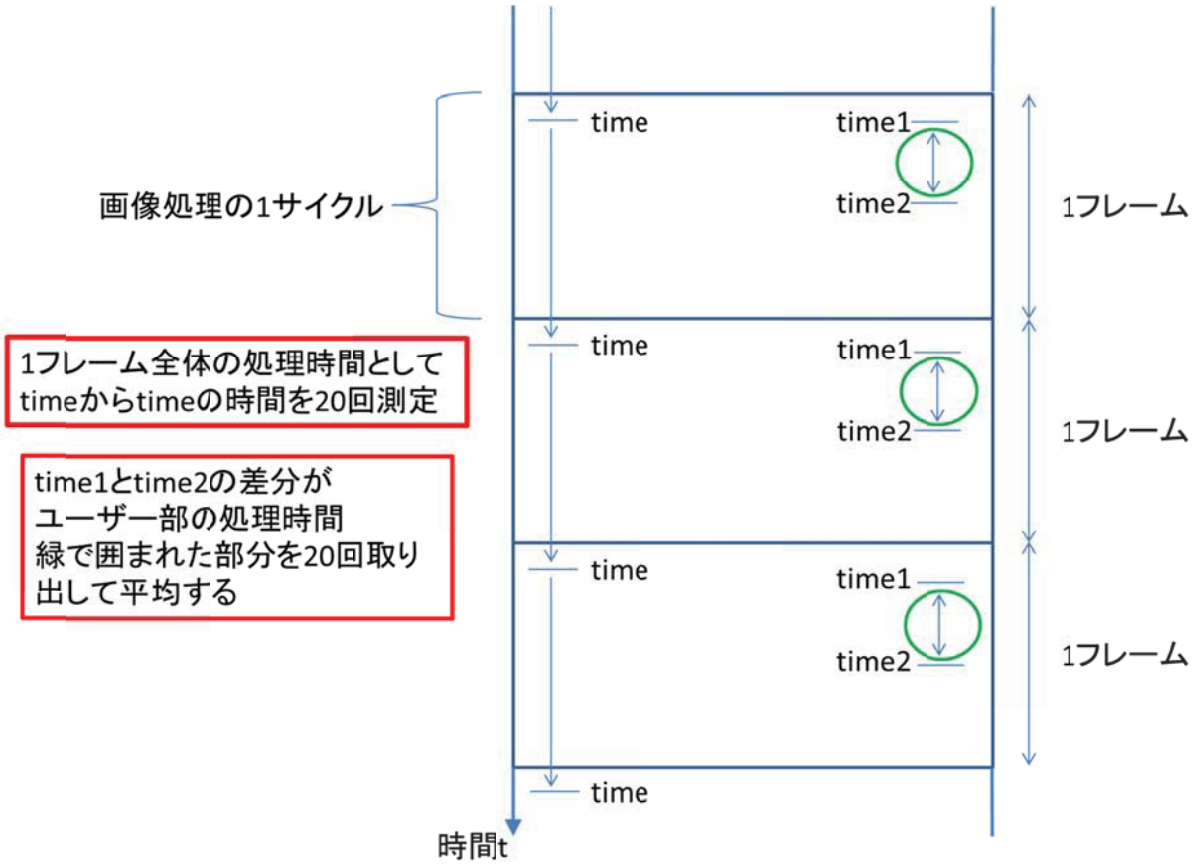


図 4.4　1 処理時間の取り出し方

また，Log の出力方法の一例が以下のコードである．

```
Log.v("SYSTEM2","SystemTime=" + ((System.nanoTime() - time) / 1000000000D)/20);
```

SYSTEM2 の部分は任意に設定できるタグであり，SYSTEM2 ではシステム全体にかかる処理時間の 20 フレームの平均を，SYSTEM3 でユーザー処理部にかかる処理時間の 20 フレームの平均を表示する．全体にかかる処理時間からユーザー処理部の処理時間を除いた時間がシステム部の処理時間である．また，20 フレームの算出方法として，プログラムのところどころに以下のようなコードを書く．

```
time = System.nanoTime();
```

これはシステム内の時間を得る関数であり，これによって得た処理時間 20 回分を合計する．そして，Log を出力する際に処理時間を 20 で割ることで，処理時間の平均を表示させる．以下に Log 出力部のコードをサンプルとして載せる．

```
if (n==20){
    if(time!=0){
        str = "" + (((System.nanoTime() - time) / 1000000000D)/20) + " sec";
        Log.v("SYSTEM2","SystemTime=" + ((System.nanoTime() - time) /
        1000000000D)/20);
    }
n = 0;
time = System.nanoTime();
}

n++;
```

これを利用して測定した処理時間をプログラムごとにグラフ化し，その際にシステム部とユーザー処理部に分けて可視化する．これにより，プログラム内のどの部分で時間がかかっているのかを視覚的にわかるようにする．また，性能比較するデバイスごとに測定した数値をまとめる．

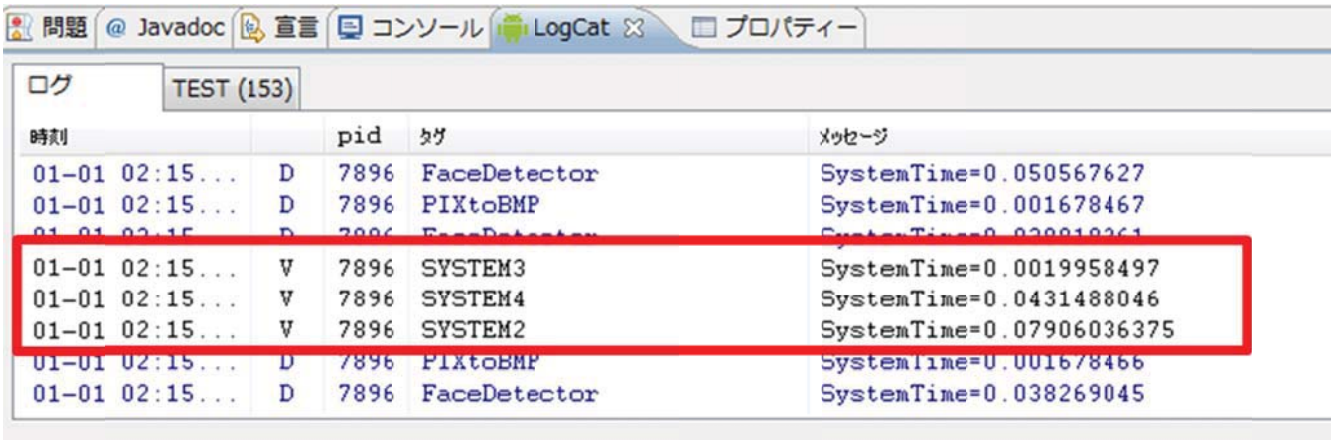


図 4.5　LogCat による Log 出力の様子

4.2.3 測定結果

測定結果は次のようになった．BeagleBoard，PandaBoard，Xperia の順でまとめる．

表 4.2 BeagleBoard 処理時間測定結果

	ユーザー[秒]	システム[秒]	全体[秒]	フレームレート[fps]
①Java のみ JIT 無	0.199	0.075	0.274	3.648
②Java のみ JIT 有	0.030	0.106	0.136	7.344
③Java+JNI	0.012	0.099	0.111	9.000
④JNI+Neon（ユーザー部のみ）	0.009	0.099	0.107	9.310
⑤JNI+Neon（システム部含む）	0.011	0.092	0.102	9.779

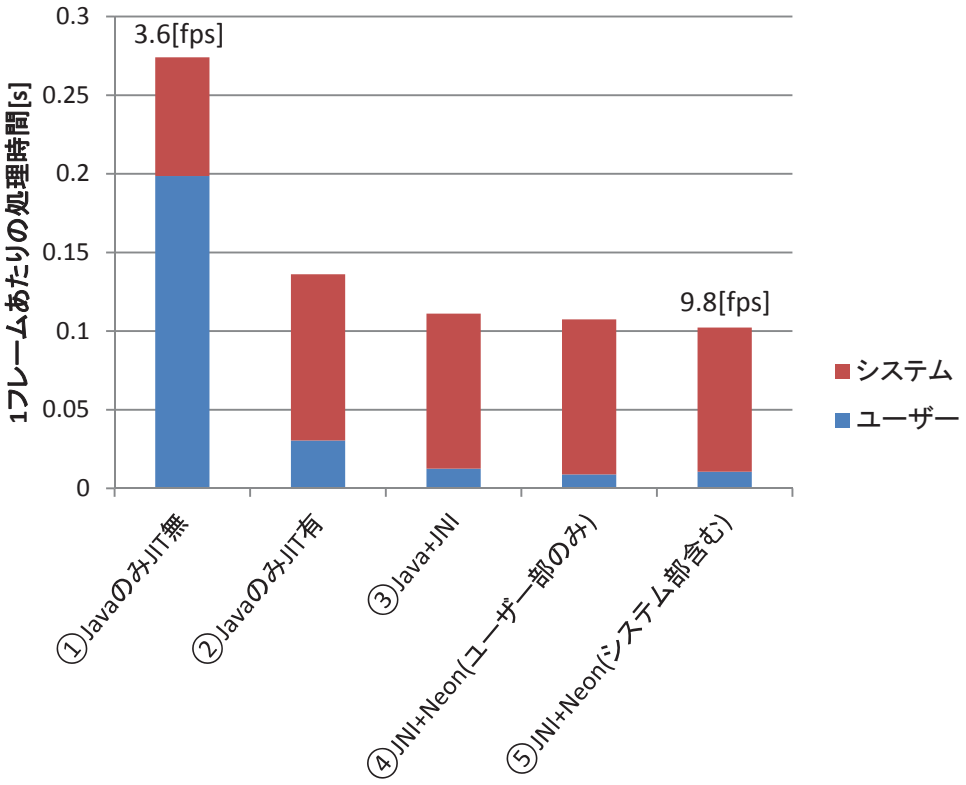


図 4.6 BeagleBoard 処理時間グラフ

表 4.3　PandaBoard 処理時間測定結果

	ユーザー[秒]	システム[秒]	全体[秒]	フレームレート[fps]
①Java のみ JIT 無	0.063	0.036	0.099	10.130
②Java のみ JIT 有	0.009	0.033	0.042	23.999
③Java+JNI	0.004	0.036	0.040	25.238
④JNI+Neon（ユーザー部のみ）	0.003	0.034	0.037	27.015
⑤JNI+Neon（システム部含む）	0.003	0.031	0.034	29.558

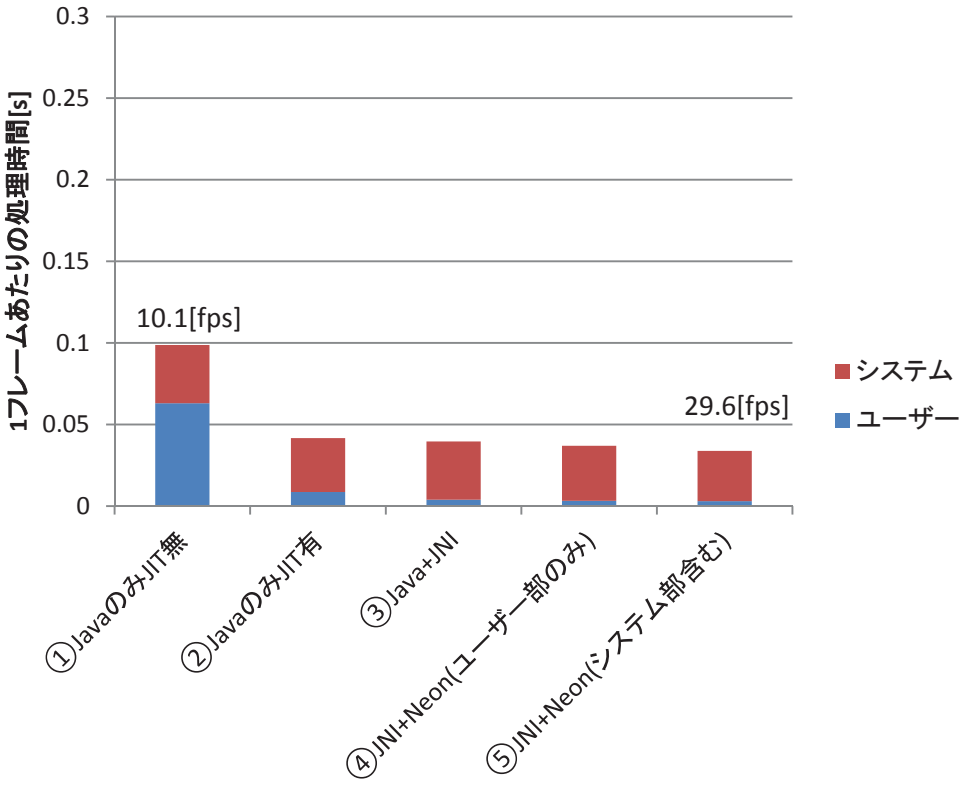


図 4.7　PandaBoard 処理時間グラフ

表 4.4　Xperia 処理時間測定結果

	ユーザー[秒]	システム[秒]	全体[秒]	フレームレート[fps]
①Java のみ JIT 無（Android2.1）	0.900	0.051	0.951	1.051
②Java のみ JIT 有（Android2.3.3）	0.134	0.047	0.181	5.517
③Java+JNI	0.024	0.044	0.068	14.775
④JNI+Neon（ユーザー部のみ）	0.016	0.052	0.067	14.893
⑤JNI+Neon（システム部含む）				

※⑤に関してはシステム部の変更ができないため実験不可能

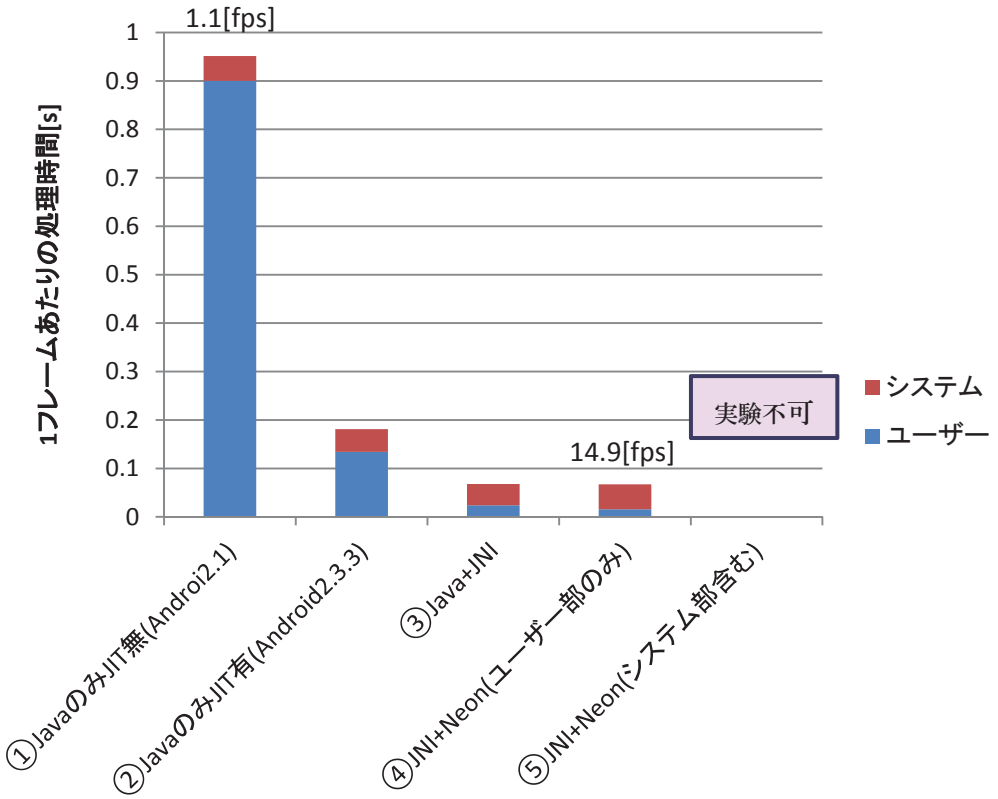


図 4.8　Xperia 処理時間グラフ

4.3 考察

仮説で考えていた通り，①～⑤のプログラムでは，どの端末でも⑤に近づくにつれて処理速度が速くなっている. Xperia では⑤の検証ができなかったが, 他 2 つのデバイスと同様に，だんだんと速くなっていることを確認できることから，システム部を Neon 化できていれば Xperia でも同じ結果を得られる可能性が高いと考えられる．よって，もっとも速いプログラムは⑤の JNI+Neon（システム部含む）であると言える．このことから，対象物追跡アプリケーションはこの方式を採用して作成することとした.

また，各デバイスについて処理速度を検証していく．まず，表 4.2，図 4.6 の BeagleBoard であるが，最も遅い①Java のみ JIT なしのフレームレートは約 3.6[fps]，最も速い⑤ JNI+Neon（システム部含む）のフレームレートは約 9.8[fps]である．BeagleBoard の画像取得の最大フレームレートは libcameraservice により 15[fps]と設定した．このことから，処理できるフレーム数は，①のプログラムでは最大フレームレートの 24%，⑤のプログラムでは 65.3%となった．処理速度は，約 2.7 倍に向上している．しかし，それでも最大の 6 割程度しか処理しきれていない．また，動作はしたものの画像処理の観点から見ると，フレームレートが 10[fps]を切ると体感的にも処理が重たく，実用性は低いと考える.

次に表 4.3, 図 4.7 の PandaBoard についてみてみると, ①のプログラムでは約 10.1[fps]，⑤のプログラムでは約 29.6[fps]である．PandaBoard の画像取得の最大フレームレートは 30[fps]と設定したことから，処理できるフレーム数は，①のプログラムでは最大フレームの 33.7%，⑤のプログラムでは 98.7%となった．処理速度は，約 2.9 倍に向上している．また、⑤のプログラムでは，最大フレームレートとほぼ変わらず，デバイスの性能を十分に発揮できていると言える．少々BeagleBoard より寸法が大きいが，その性能差は歴然で，測定結果からはっきりとわかる.

組み込みデバイスと比較するためにスマートフォンの Xperia でも測定を行い, その結果を検証したのが，表 4.4，図 4.8 である．①のプログラムでは約 1.1[fps]，Xperia では最速である④のプログラムでは約 14.9[fps]である. 画像取得の最大フレームレートは 30[fps]であることから，処理できるフレーム数は，①のプログラムでは 3.7%，④のプログラムでは 49.6%となった．処理速度は，約 13.5 倍に向上している．最大フレームレートの 5 割程度しか処理できていないが，BeagleBoard よりは多くのフレームを処理していることがわかる．ただし，これは Xperia は BeagleBoard や PandaBoard より処理するピクセル数が多いためであり，単純に性能を比較することができない．そこで，各デバイスの解像度に注目し，1 ピクセルあたりの計算時間を算出することで計算能力を比較する.

（1 ピクセルあたりの計算時間）＝（処理時間）/（解像度）

BeagleBoard と PandaBoard で用いた USB カメラの解像度は 352x288，Xperia は 854x480 である．上記の式の通り 1 ピクセルあたりのユーザー処理部の計算時間を算出すると，BeagleBoard は 0.104[μs]，PandaBoard は 0.031[μs]，Xperia は 0.038[μs]であった．この

ことから，PandaBoard と Xperia の計算能力は同程度であり，今回の測定結果から見てみると最も計算能力の高いデバイスは PandaBoard であることがわかった．

また，図 4.8 の Xperia のグラフをよく見ると，2 つのボードのグラフと違い，システム部の処理時間が短くなっている．これは，2 つのボードは USB カメラを使用しているのに対し，Xperia は内蔵されているカメラを使っている違いから生じているものと推測される．つまり，内蔵カメラと USB カメラでは，USB での接続が無くデバイスに直接つながっている内蔵カメラのほうがデータ転送にかかる時間が短いためだと考えられる．

こうして比較してみると，グラフのバランスに違いはあるものの，処理の高速化について工夫したプログラムは，先に述べたように，すべてのデバイスで仮定した通りの結果を示したことを確認できた．

第5章　対象物追跡

本章では，第 4 章の検証結果に基づいた対象物追跡アプリケーションについて述べる．また，作成したアプリケーションの処理速度について検証し，考察する．

5.1　カメラと実験環境

先述の通り，本研究では Android OS 利用法の一例として，カメラを利用した対象物追跡手法を提案する．Android は組込みデバイスにインストールし，そのターゲットは，BeagleBoard, PandaBoard である．また比較対象としてスマートフォンの Xperia（SO-01B）を利用する．本研究で組み込みデバイスをターゲットとしたのは，研究当初スマートフォンでは USB カメラやモータを利用することができなかったことが挙げられる．しかし，2012 年 1 月現在 ADK（Google I/O 2011 で発表された Open Accessory Development Kit）があるため，モータはスマートフォンでも利用することが可能となったが，USB カメラについては現在も公式にサポートされていないことが多い．そのため，今回我々が用いるように組み込みシステム向けにソースからビルドした Android を用いることはモ対象物追跡などを行う上では現実的である．

対象を追跡するには，画像処理により対象物の座標を取得し，その値によりカメラを取り付けた台座のモータを制御すればよい．その際，画像処理の計算手段を改善し，処理速度の向上を図る．これは，対象物追跡に必要とする画像認識において，動く対象物をリアルタイムで追跡するには，処理速度の向上が必要不可欠なためである．将来的な展望として，車載カメラや防犯カメラ等への技術的な転用の可能性があると考える．図 5.1 に構想図を示す．

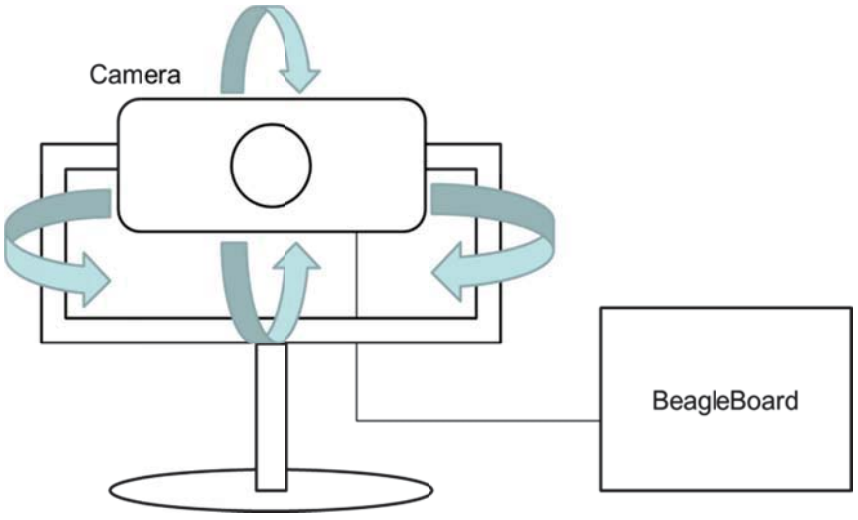


図 5.1　カメラの台座構想図

本研究で用いた実験機材のリストは以下の通りである.

（1）Android Version … Android 2.2.2（Froyo）：BeagleBoard 用
Android2.3.4（Gingerbread）：PandaBoard 用
Android4.0.1（IceCream Sandwich）：PandaBoard 用

（2）Java 統合開発環境 … Eclipse 3.6

（3）組み込みデバイス … BeagleBoard Rev.C4，PandaBoard Rev.A2

（4）サーボモータ …双葉電子工業製 RS304MD-FF （2 個）

（5）シリアル USB 変換器 …双葉電子工業製　RSC-U485

（6）カメラ … Logicool Webcam C210

（7）7 インチタッチパネルモニタ … Hanwha 製 HM-TL7T

（8）SD カード … Transcend（8GB）

（9）LAN アダプター … corega CG-FEUSBTXCW

（10）USB wifi デバイス … Planex 製 GW-US54Mini2

（11）USB Bluetooth … Buffalo 製 BSHSBD02

（12）カメラ台座 … 5.2 章台座の製作参照

5.2　台座の製作

本研究で製作したカメラの台座について詳細を示す．完成した台座は図 5.2 のようになっている．

図 5.2　カメラ台座完成図

台座の製作に使用した部品を表 5.1 に示す．製作を説明するうえで，部品の呼称を部品名とし，実際に使用したものを製品名とした．完成した台座の正面と側面から見た見取り図を図 5.3，図 5.4 に示す．また，台座の製作の手順については巻末の付録Ⅳ「台座の製作手順」にまとめる．

表 5.1　カメラ台座部品リスト

部品名	製品名
USB カメラ	Logicool Webcam C210
モータ	双葉電子工業製 RS304MD-FF （2 個）
サーボホーン	G-ROBOTS サーボホーン（5 個入）　（うち 2 個使用）
ワッシャー	POM ワッシャー 2x12x1mm（10 個入）　（うち 2 個使用）
	ワッシャー M2x6mm（10 個入）　（うち 2 個使用）
ラバーブッシュ	G-ROBOTS ラバーブッシュ（5 個入）（うち 1 個使用）
プラブッシュ	G-ROBOTS プラブッシュ（5 個入）（うち 1 個使用）
モータケース	G-ROBOTS GR-001 股関節サーボホルダー（L/R）セット （股関節サーボホルダーR，股関節サーボホルダー1 個のみ使用）
ロボットパーツ①	G-ROBOTS GR-001 肩（L/R）セット　（肩 L のみ使用）
ロボットパーツ②	G-ROBOTS GR-001 すね（L/R）セット　（すね L のみ使用）
アクリル接続部	アクリル角材 24x30x30mm
	アクリルパイプ φ8
	アクリルパイプ φ10
三脚	ケンコー ケータイ&デジカメポッド スーパーミニ[コーラル]
おもり	アルミ台形（3 個）

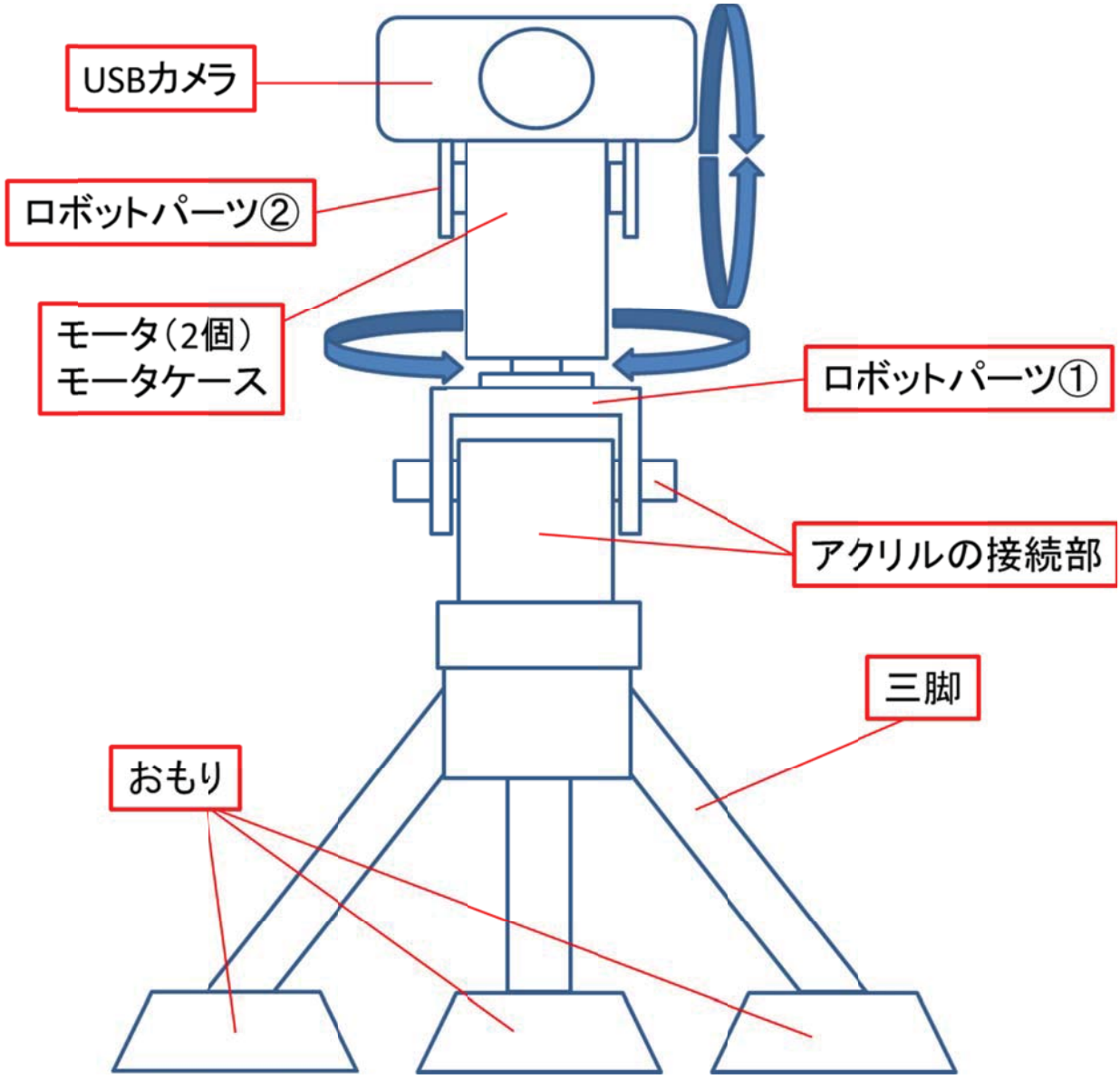


図 5.3　カメラ台座見取り図（正面）

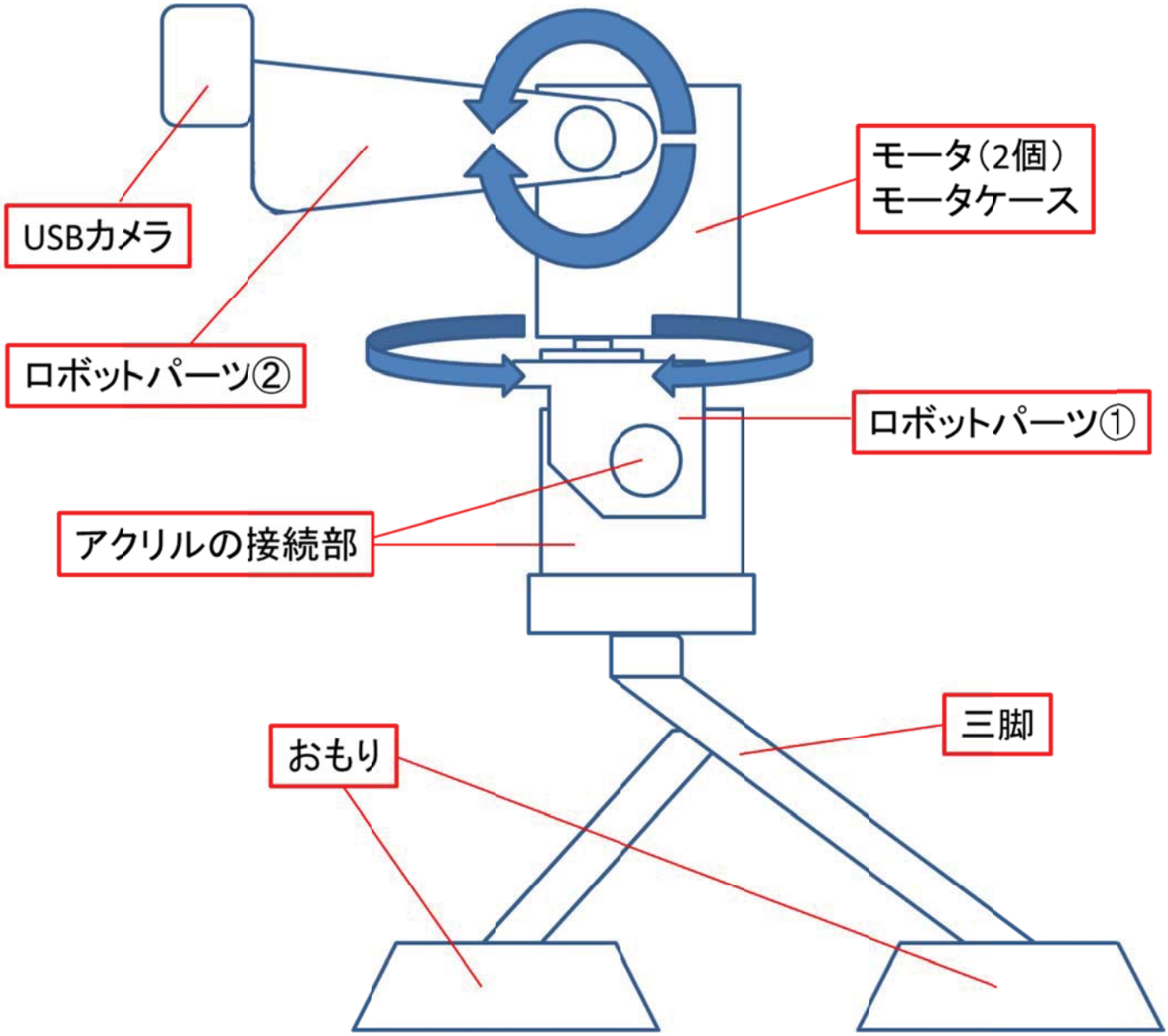


図 5.4　カメラ台座見取り図（側面）

5.3　駆動部におけるモータの詳細

本研究で使用するサーボモータは双葉電子工業製 RS304MD-FF（図 5.5）である．以下に，その詳細を記す．

図 5.5　RS304MD-FF

表 5.2　RS304MD-FF 仕様詳細

寸法（LxWxH）	35.8x19.6x25.0[mm]
重量	21[g]
出力トルク	5.0[kg･cm]
動作スピード	0.16[sec/60°]
使用電圧範囲	4.8～7.4[V]
可動範囲	-150～150[°]

まず，本研究でこのモータを利用するにあたり，双葉電子工業が提供しているサンプルプログラム[6]を用いてサーボIDを変更する．サーボIDとは，シリアル通信で複数のモータに指示を与える際，モータを区別するために用いられるIDである．サンプルプログラムの概要は，Windows APIを使用してコマンド式RSシリーズサーボとシリアル通信するものである．納品時のモータのサーボIDを1と仮定してあるが，そうで無い場合には修正が必要である．以下サーボIDの変更と保存を行うために変更したサンプルプログラムの内容の詳細である．main関数内に以下を記述する．

```
IDCh(hComm,1,3); // ID 1 を 3 に変更
Sleep(100);

FLASHW(hComm, 3); // ID 3を保存
Sleep(1000);
```

ここで呼び出す関数IDCh，FLASHWは自作関数である．以下でそれぞれについてを示す．

まず IDCh についてであるが，これはサーボ ID を変更するための関数である．内容は以下の通りで， IDCh(hComm，今のサーボ ID，新しく設定するサーボ ID) で呼び出す．

```
int IDCh( HANDLE hComm, unsigned char fromID, unsigned char toID )
{
    unsigned char sendbuf[28];
    unsigned char sum;
    int                         i;
    int                         ret;
    unsigned long len;

    // ハンドルチェック
    if( !hComm ){
            return -1;
    }

    // バッファクリア
    memset( sendbuf, 0x00, sizeof( sendbuf ));

    //ID 変更
    sendbuf[0]  = (unsigned char)0xFA;              // ヘッダー1
    sendbuf[1]  = (unsigned char)0xAF;               // ヘッダー2
    sendbuf[2]  = (unsigned char)fromID;            // 古い ID
    sendbuf[3]  = (unsigned char)0x00;              // フラグ
    sendbuf[4]  = (unsigned char)0x04;              // アドレス(0x1E=30)
    sendbuf[5]  = (unsigned char)0x01;               // 長さ(4byte)
    sendbuf[6]  = (unsigned char)0x01;               // 個数
    sendbuf[7]  = (unsigned char)toID;              // 新しい ID

    // チェックサムの計算
    sum = sendbuf[2];
    for( i = 3; i < 8; i++ ){
            sum = (unsigned char)(sum ^ sendbuf[i]);
    }
    sendbuf[8] = sum;                               // チェックサム
    // 通信バッファクリア
    PurgeComm( hComm, PURGE_RXCLEAR );

    // 送信
    ret = WriteFile( hComm, &sendbuf, 9, &len, NULL );

    return ret;
}
```

しかし，IDCh だけではサーボ ID の保存ができない．変更した ID をフラッシュ RAM に書き込んで保存するための関数として FLASHW を作成した．内容は以下の通りで，FLASHW(hComm，保存するサーボ ID)で呼び出す．

```
int FLASHW( HANDLE hComm, unsigned char ID )
{
    unsigned char sendbuf[28];
    unsigned char sum;
    int                         i;
    int                         ret;
    unsigned long len;

    // ハンドルチェック
    if( !hComm ){
          return -1;
    }

    // バッファクリア
    memset( sendbuf, 0x00, sizeof( sendbuf ));

    // パケット作成
    sendbuf[0]  = (unsigned char)0xFA;              // ヘッダー1
    sendbuf[1]  = (unsigned char)0xAF;              // ヘッダー2
    sendbuf[2]  = (unsigned char)ID;                // サーボ ID
    sendbuf[3]  = (unsigned char)0x40;              // フラグ
    sendbuf[4]  = (unsigned char)0xFF;              // アドレス(0x24=36)
    sendbuf[5]  = (unsigned char)0x00;              // 長さ(4byte)
    sendbuf[6]  = (unsigned char)0x00;              // 個数
    // チェックサムの計算
    sum = sendbuf[2];
    for( i = 3; i < 7; i++ ){
          sum = (unsigned char)(sum ^ sendbuf[i]);
    }
    sendbuf[7] = sum;                   // チェックサム

    // 通信バッファクリア
    PurgeComm( hComm, PURGE_RXCLEAR );

    // 送信
    ret = WriteFile( hComm, &sendbuf, 8, &len, NULL );

    return ret;
}
```

以上 2 つの関数を用いることで，サーボ ID の変更と保存が可能となった.

5.4 Windows 用モータ制御アプリケーションの試作

本研究で使用するサーボモータの動作確認とモータのために，Java 言語を用いて Windows 上で動作するモータ制御のアプリケーションを作成した．作成したアプリケーションは 2 つの slider で 2 つのモータの位置（角度）を制御するもので，命令は slider のイベントハンドラから呼び出し，シリアル通信によってモータに与える．それぞれの slider がモータ 1 つと対応しており，slider の移動量に合わせてモータが動作する．図 5.6 にアプリケーションを示す．サーボモータの可動域に対応させるため，slider の値が-1500～1500 に設定されている．

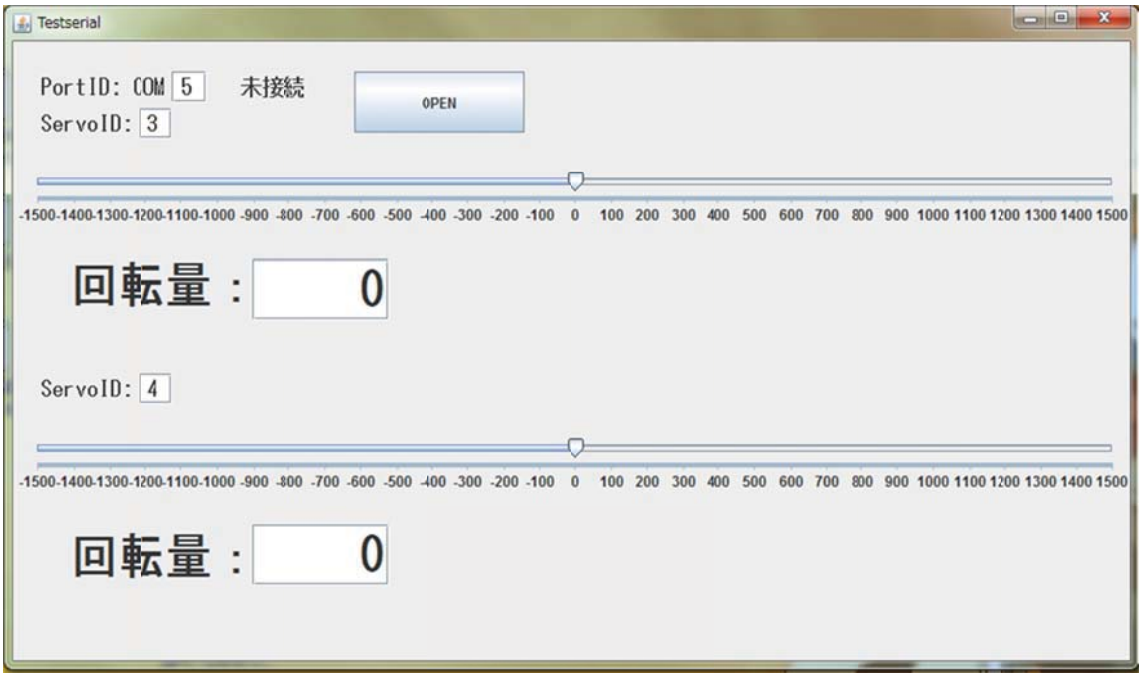


図 5.6 Windows 用モータ制御アプリケーション

なお，Java では標準でシリアル通信を使うためのクラスが含まれていないため，シリアル通信を行うには外部ライブラリを追加する必要がある．本研究では RXTX ライブラリ(Using RXTX のウェブページ[7]よりダウンロード)を利用したが、そのインストール方法については巻末の付録Ⅳ「RXTX ライブラリの利用方法」にまとめる．

設定が完了した後，Eclipse 上で確認すると，JRE システム・ライブラリー内に先述の手順でインストールした jar ファイルが確認できる（図 5.7）．

図 5.7　RXTX ライブラリの設定完了後(Eclipse)

5.5　Android 用モータ制御アプリケーション

試作したモータ制御のアプリケーションを参考にしながら，Java 言語を用いて Android 用のアプリケーションを作成した．上下 2 つのシークバーがそれぞれ一つずつモータに対応している．シークバーを左右に動かすことで，対応したモータがシークバーの移動量に合わせて動く．図 5.8 にアプリケーションのサンプルを示す．

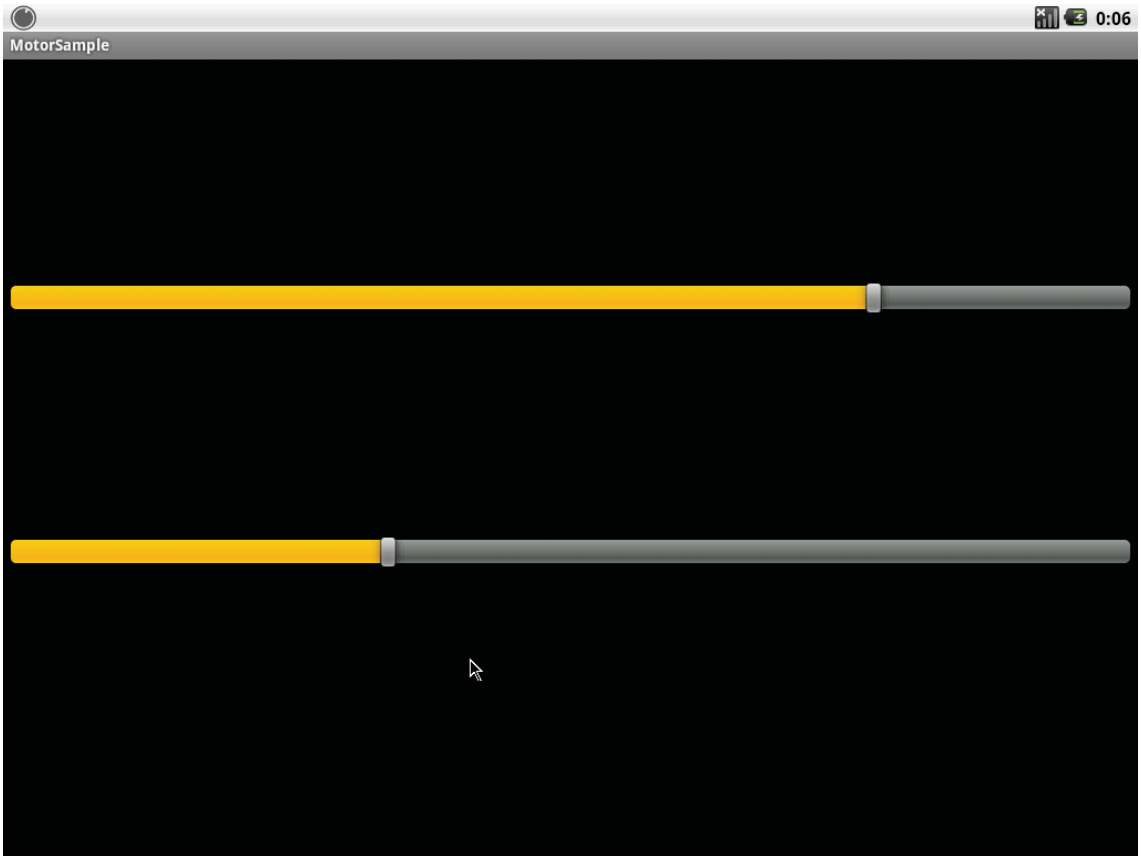


図 5.8　Android 用のモータ制御アプリケーション

なお, Android でシリアル通信を実現するには第 3 章で触れた JNI を利用する必要がある．先述のように Java 言語で作成されたユーザーアプリからでは直接ハードウェアにアクセスすることができないためである．5.4 章の Java アプリケーションでは，外部ライブラリである RXTX が JNI を利用していたことになる．本アプリケーションでは，図 5.9 のように JNI を利用し，シリアル通信を可能にすることでモータにアクセスする．

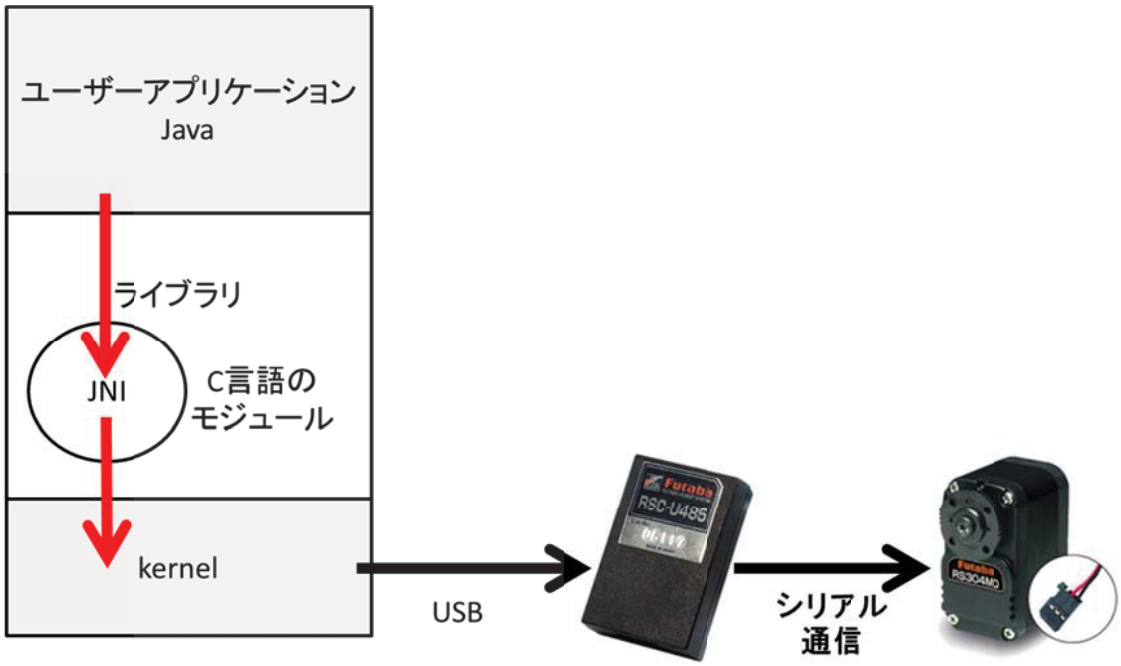


図 5.9　JNI を用いたモータへのアクセス方法

JNI を Windows 上で利用するための環境設定法については巻末の付録Ⅲ「JNI の利用における環境設定と利用法」にまとめる.

5.6 対象物追跡アプリケーション

第 4 章の検証結果から得た，最も処理速度の速かったプログラムを利用して対象物追跡のアプリケーションを作成した．モータにより上下，左右に動く台座にカメラを取り付け，カメラで取得した画像から対象を認識し，その座標を取得する．取得した座標が画面の中央に移動するようにモータを動かし，対象を追跡する．作成にあたり対象物を認識する手段として 2 つの方法をとった．Android に備わっているシステムを利用した顔認識を行う方法（FaceDetector）と，画像認識部を C 言語によるユーザー関数化する方法の 2 つである．以下にそれぞれについて述べる．これらのアプリケーションは BeagleBoard と PandaBoard のどちらでも動作するので，2 つの環境において処理速度の比較検証をする．

5.6.1 FaceDetector

Android のフレームワークには顔認識を行う FaceDetector というクラスが標準で存在する．このクラスを利用することによって，画像の中の顔の数やその位置を簡単に得ることが可能である．本テーマでは認識した顔を追跡するように，認識した顔の両目の中心座標を取得する getMidPoint 関数を利用して両目の中心座標が画像の中心に近づくようにモータを動かすことにした．また，顔の最大認識数を調整することで一度に複数の顔を認識することができるため，目と目の間隔である eyesDistance が最も大きいものをカメラから一番近い顔として判別するようにした．これにより複数の顔を認識しながら，最もカメラに近い対象物を追跡するアプリケーションを作成した．認識結果を表示しているのが図 5.10 である．

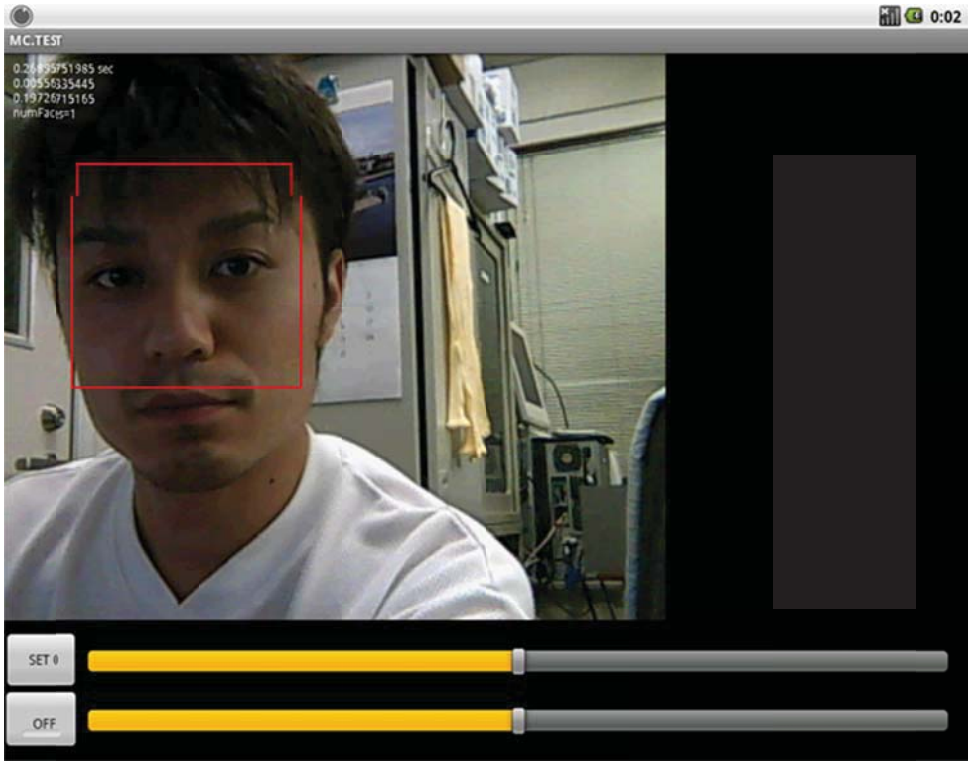


図 5.10 FaceDetector 認識結果

5.6.2　テンプレートマッチング

FaceDetector を利用せずに，画像の認識部分をユーザー関数化する方法としてテンプレートマッチングを用いた．テンプレートマッチングとは，ある元になる画像（テンプレート）があり，それに近いもの（部分）を，対象の画像から探すという方法である（図 5.11）．これを用いて対象物を認識させるアプリケーションを作成した．しかし，FaceDetector を利用したアプリケーションとは違い, 複数の対象物を認識させることは行わなかった. FaceDetector との違いでもう一つあげられることは，テンプレートマッチングの場合，対象画像から検出するものは正確にはテンプレートに似た部分であるということである．つまり，人間の顔を認識する FaceDetector に対し，テンプレートを認識するテンプレートマッチングは，認識する対象が人間の顔でなくてもよい．テンプレートを自由に設定することで，人間以外のものを対象物として追跡することが可能である．

今回作成したアプリケーションは，取得した画像をタッチパネルに表示し，触れたところをテンプレートとして切り出す．切り出したテンプレートをカメラで取得した画像から検出し，対象物を認識する．画像の中心座標と，検出したテンプレートの中心座標の差から，テンプレートを画像中央に近づくようにモータが動くようになっている．また，もう一度触れることでテンプレートをリセットし, 追跡を解除できる. 認識結果を表示しているのが図 5.12 である．

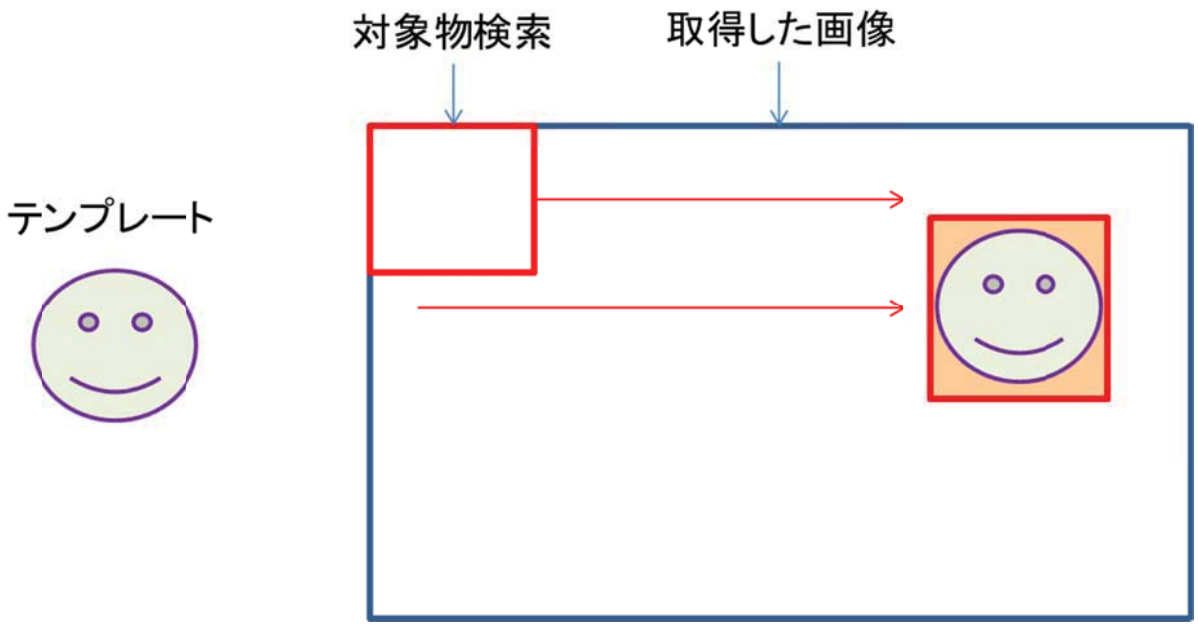


図 5.11　テンプレートマッチング

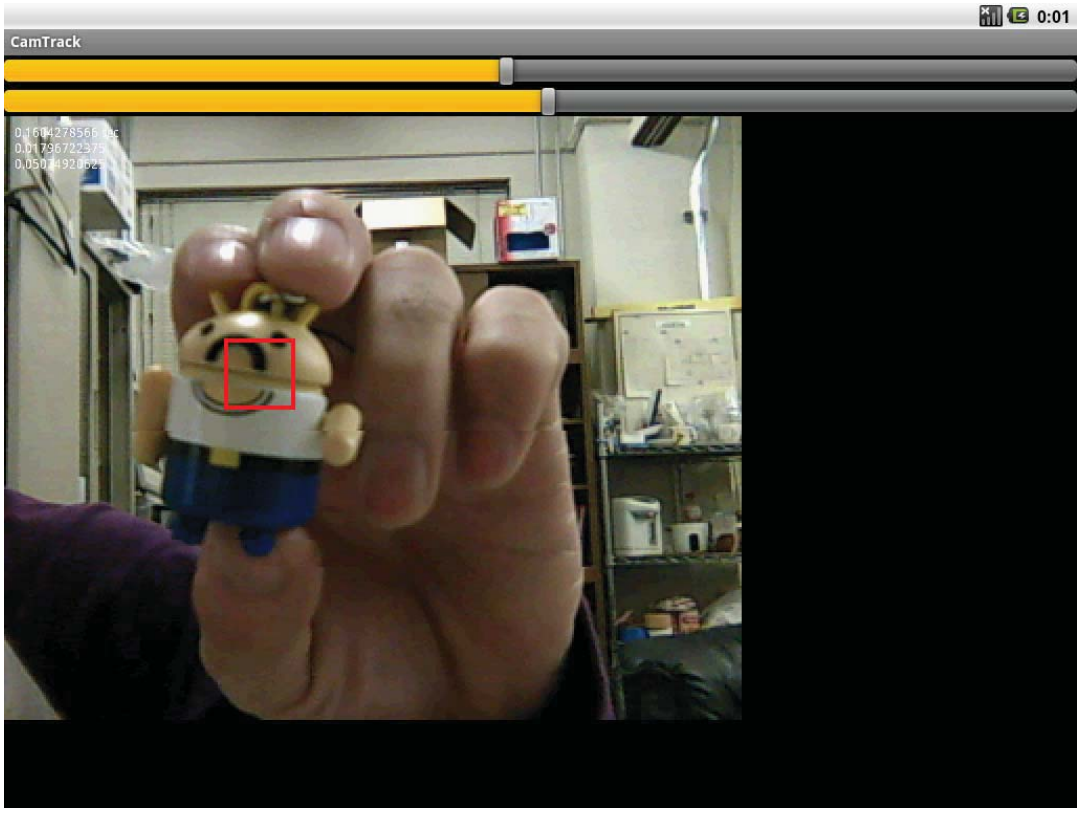


図 5.12　テンプレートマッチング認識結果

5.7 処理時間の測定

本章で，処理時間測定の方法と結果を示す．最も処理時間がかかると予測される対象物認識部において Android のシステムを利用した FaceDetector アプリケーションに対し，認識部分をユーザー関数化することで処理速度の向上を図る．また，その際に BeagleBoard と PandaBoard についての性能比較についても行う．

5.7.1 測定理論

第 4 章 4.2.2 に記したように，各処理にかかる時間を 20 回の処理の平均時間をさらに 10 回平均して，1 フレームの処理時間を算出する．また，図 5.13 にプログラムの流れを示す．ハードウェアの依存の高い処理部分をシステム部，それ以外の処理部分をユーザー処理部とする．さらに，システム部内の対象物認識部分を認識部とする．

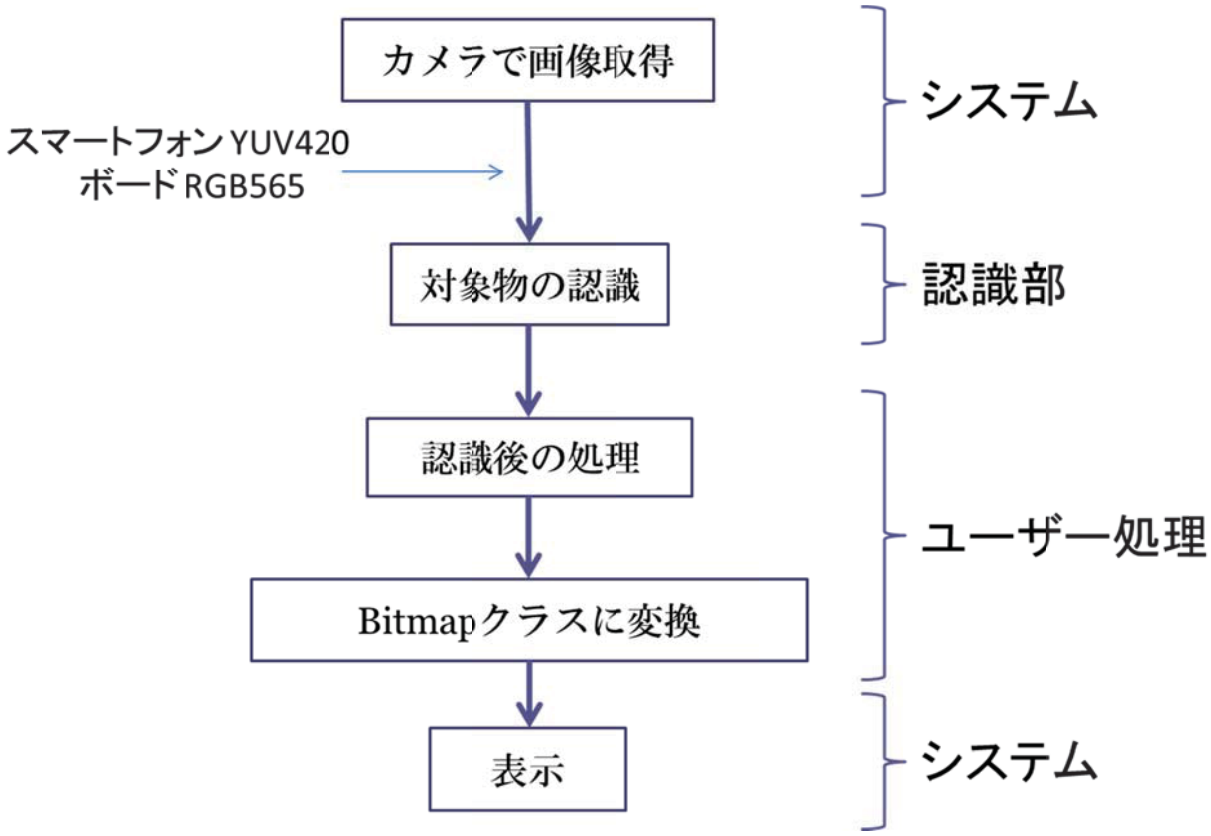


図 5.13 対象物追跡アプリケーション処理の流れ

5.7.2 測定方法

第 4 章 4.2.2 に記した方法と同様に，プログラム内から Log を出力して測定する方法を用いる．20 フレームの平均時間を 10 回測定し，処理にかかる時間を算出する．また，今回は画像処理部における Android のシステムを利用した FaceDetector と，処理部を User 関数化したテンプレートマッチングの方式（以下 User）の処理速度の違いが分かるように，画像処

理部の Log も出力した．プログラムごとに測定時間をグラフ化し，その際にシステム部とユーザー処理部，さらにシステム内の対象物認識部分を認識部として分けて可視化する．これにより，プログラム内のどの部分で時間がかかっているのかを視覚的にわかるようにする．また，性能比較するデバイスごとに測定した数値をまとめる．

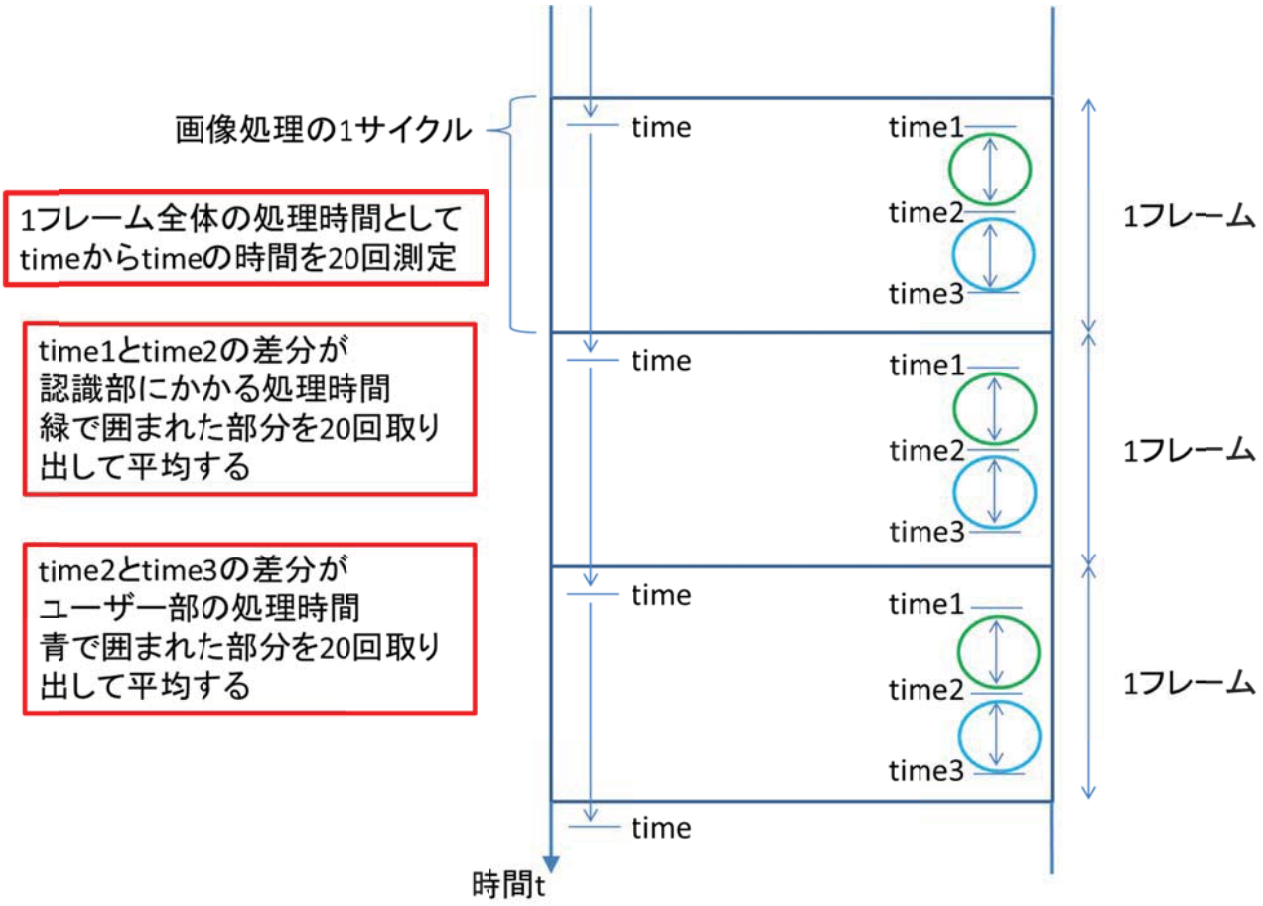


図 5.14　処理時間の取り出し方

5.7.3 測定結果

測定結果は次のようになった． BeagleBoard，PandaBoard の順にまとめる．

表 5.3 BeagleBoard 処理時間測定結果

	ユーザー[秒]	認識部[秒]	システム[秒]	全体[秒]	フレームレート[fps]
FaceDetector	0.008	0.240	0.110	0.358	2.795
User	0.036	0.016	0.121	0.173	5.788

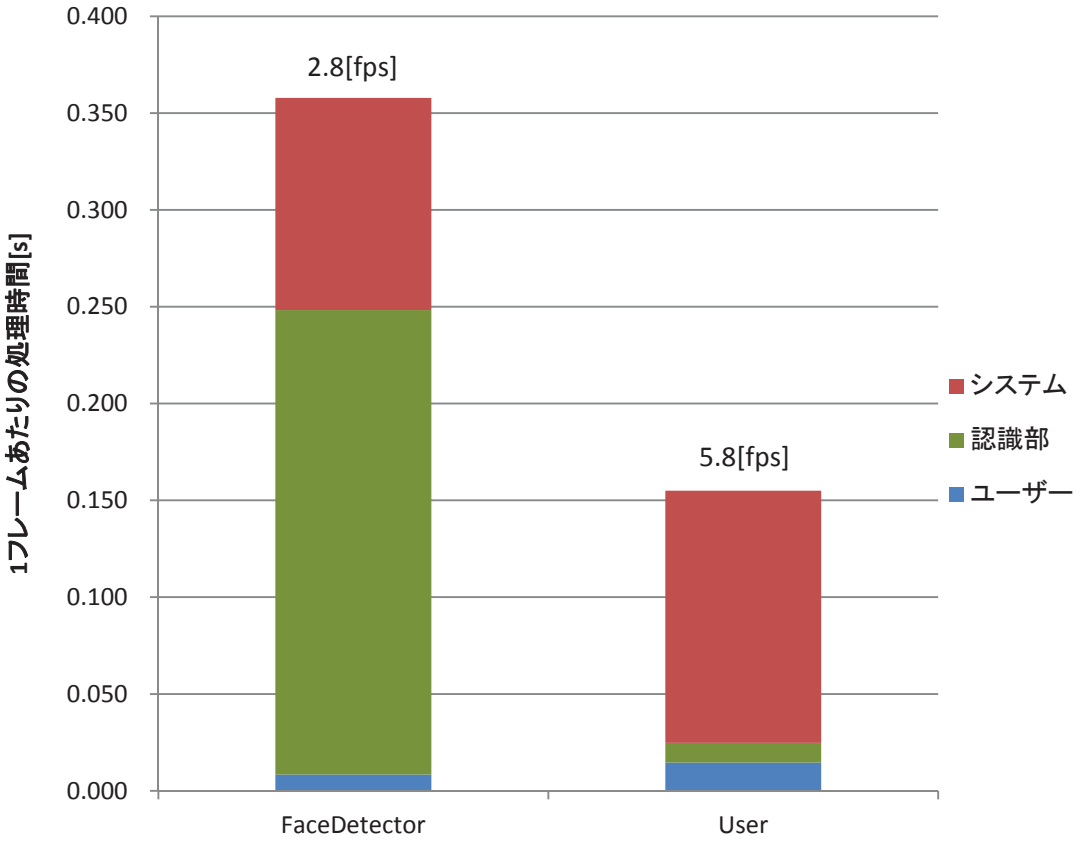


図 5.15 BeagleBoard 処理時間グラフ

表 5.4　PandaBoard 処理時間測定結果

	ユーザー[秒]	認識部[秒]	システム[秒]	全体[秒]	フレームレート[fps]
FaceDetector	0.002	0.045	0.034	0.081	12.421
User	0.006	0.002	0.036	0.043	23.298

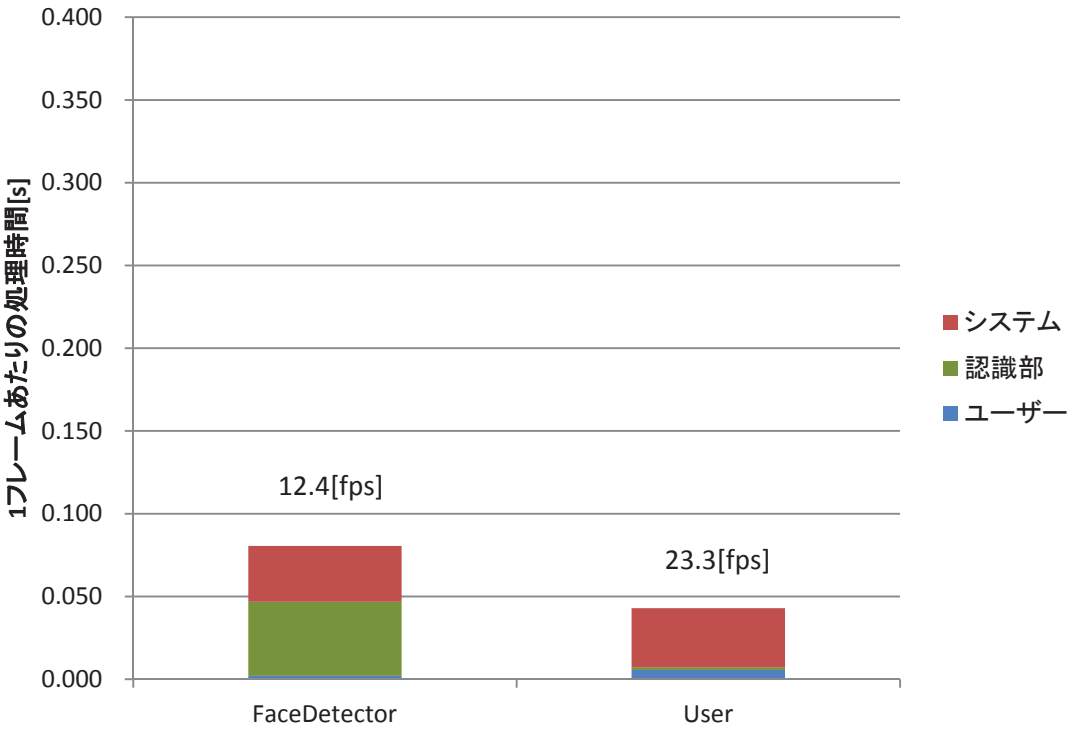


図 5.16　PandaBoard 処理時間グラフ

5.8 考察

測定結果から，想定していた通り処理にかかる時間で最も時間がかかっていたのは対象物認識部分であった．また，BeagleBoard と PandaBoard のどちらも認識部分で Android のシステムを利用する FaceDetector アプリケーション（以下 FaceDetector）よりも，対象物認識をユーザー関数化したアプリケーション（以下 User）のほうが意図していた通り速くなっていることがわかる．よって，認識部の処理速度向上に成功したと言える．

続いて，各デバイスの処理速度を検証していく．まず表 5.3 と図 5.15 の BeagleBoard についてであるが，FaceDetector のフレームレートは約 2.8[fps]， User は約 5.8[fps]であった．BeagleBoard の画像取得の最大フレームレートは 15[fps]に設定してあることから，処理できるフレーム数は，FaceDetector では最大フレームレートの 18.7%，User では 38.7%となった．処理速度は，約 2.1 倍に向上している．また，対象物認識部にかかる処理時間に注目してみると，FaceDetector では 0.24 秒，User では 0.016 秒であり，処理時間はおよそ 15 分の 1 に短縮されている．ユーザー関数化することで対象物認識部の処理速度の向上が見られたものの，処理できるフレーム数はデバイスの最大フレームレートの 5 割にも満たなかった．アプリケーションを動作させてみると，対象物追跡は可能であるが，BeagleBoard の計算能力ではラグが大きく，第 4 章の結果と合わせて実用性が低いと判断できる．

次に表 5.4 と図 5.16 の PandaBoard についてみてみると，FaceDetector では約 12.4[fps]，User では約 23.3[fps]である．PandaBoard の画像取得の最大フレームレートは 30[fps]に設定してあることから，処理できるフレーム数は，FaceDetector では最大フレームの 41.3%，User では 77.7%となった．処理速度は，約 1.7 倍に向上している．また，対象物認識部にかかる処理時間に注目してみると，FaceDetector では 0.045 秒，User では 0.002 秒であり，処理時間はおよそ 23 分の 1 に短縮されている．BeagleBoard 同様に処理速度の向上が見られ，処理できるフレーム数は約 23.3[fps]と BeagleBoard で起動した場合の約 3.2 倍であり，BeagleBoard の最大フレームレートを上回る．これらのことから，第 4 章に続き BeagleBoard と PandaBoard の性能差をはっきりと見て取ることができる．さらに，User ではユーザー処理部と認識部の処理時間が短縮され，0 秒に近づいていることがわかる．しかしながら，最大フレームレートには届かなかった．ここでシステム部を注目してみると，FaceDetector では 0.034 秒，User では 0.036 秒である．仮に画像認識を 0 秒とした場合（何も処理をしない場合），PandaBoard の最大フレームレートから考えるとシステム部での処理時間は 0.033 秒になるが，両アプリケーションにおいてそれよりも時間がかかっている．これは，画像取得の際の明るさなどの影響により，システム部に負荷がかかっているもの考えられる．

また，検証した両デバイスにおいて User でユーザー処理部にかかる時間が増加している．これは認識を行う際に，処理時間短縮のため取得した画像を一度小さくし，表示する際にもとに戻す工程があるためだと考える．

先述の通りどちらのデバイスでも認識部での大幅な処理速度の向上がみられたことから，処理の高速化に成功したと言える．また，作成した対象物追跡アプリケーションを Android OS の利用法の 1 つとして提案する．

第6章　機能の拡張

作成した対象物追跡アプリケーションの機能を拡張し，Bluetooth デバイスによるカメラの遠隔操作の実現を目指す． Bluetooth デバイスとして Zeemete 製 JS1 H 利用する．

またそれと並行して，ローパスフィルタを用いて対象物追跡におけるカメラの移動を滑らかにし，アプリケーションの追随性の向上を図る．

6.1　Zeemote JS1 H の利用

6.1.1　JS1 H

Zeemote 製の JS1 H（図 6.1）は 4 つのボタンとジョイスティックがついたスマートフォン用コントローラである．手の平にフィットするコンパクトなサイズで，重さは 25.5g（使用する乾電池[単 4：2 本]除く）である．端末と Bluetooth で接続するため，接続した端末の遠隔操作が可能である．

図 6.1　Zeemote JS1 H

6.1.2 利用方法

JS1 H を利用するには，まず Zeemote のウェブサイト [8]から Zeemote Android SDK をダウンロードする．Eclipse の Android プロジェクトでライブラリを使うためには以下の手順をとる必要がある（Zeemote Android Programmer's Guide 参照）．

○プロジェクトの下（フォルダ内）に libs ディレクトリをコピーする．

○AndroidManifest.xml ファイルに以下の行を追記する．

android.permission.BLUETOOTH

android.permission.BLUETOOTH_ADMIN

○Zeemote Android SDK で提供されるコントローラ UI を使うならば，さらに AndroidManifest.xml ファイルに以下の行を追記する．

Com.zeemote.zc.ui.android.ControllerAndroidUi$Activity

この操作を実行することで，プロジェクトはライブラリを使う準備が整う．操作の詳細については巻末資料として付録Ⅵ「Bluetooth コントローラ JS1 H の利用法」にまとめる．操作を完了すると，図 6.2 のようにプロジェクト内に外部ライブラリが確認できる．

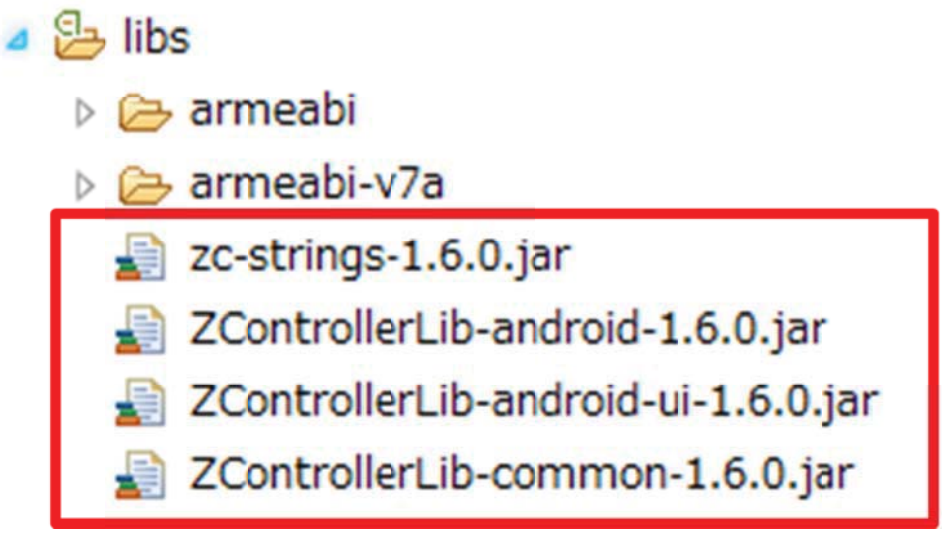


図 6.2 Zeemote Android SDK の外部ライブラリの利用

6.1.3 拡張機能の構想

作成した対象物追跡アプリケーションの機能を拡張し，Bluetooth で接続した JS1 H により，台座に取り付けたモータを遠隔で任意で操作できるようにする．また，作成したアプリケーションには，初期位置に戻る Set 0，対象物認識の ON/OFF の切り替えの機能があるので，それらをボタンで操作できるようにする．さらに，アプリケーションの終了もマウスを使わずコントローラでできるようにする．これにより，今まではマウスやタッチパネルで行っていた操作が，ワイヤレスで行えるようになる．

6.2　機能の拡張

第 5 章で作成した対象物追跡のアプリケーションに変更を加え，JS1 H を対応させた．各ボタンの機能は図 6.3 に示す．これにより，作成した対象物追跡のアプリケーションを遠隔操作することが可能となった．

図 6.3　JS1 H のボタンを対応

アプリケーションの大きな変更要素として挙げられるのは以下の 2 点である．

○ジョイスティックの対応

○ボタンの割り当て

この 2 点についてプログラムの一部を参照しながら詳細を示す．まず，ジョイスティックへの対応について示す．以下の joystickMoved により，ジョイスティックを動かすと横方向 x と縦方向 y の値を返す．

```
@Override
    public void joystickMoved(JoystickEvent e) {
            // A joystick moved. Scale the values between -50 and 50
            x = e.getScaledX(-50, 50);
            y = e.getScaledY(-50, 50);

            setProgress3();
            setProgress4();

            traceLog("X=" + x + ",Y=" + y);
    }
```

またこの関数により，以下に示す setProgress3 と setProgress4 が呼ばれ，それぞれ横方向と縦方向にモータを動かせるよう対応させた．

```
   //コントローラのスティック横方向
void setProgress3(){
       xx = ((xx + x)/5)*5;

       if(xx>1400) xx = 1400;
       if(xx<-1400) xx = -1400;

       seek1.setProgress(1400 + xx);
}

//コントローラのスティック縦方向
void setProgress4(){
       yy = ((yy + y)*5)/5;

       if(yy>1400) yy = 1400;
       if(yy<-1400) yy = -1400;

       seek2.setProgress(1400 - yy);
}
```

続いてボタンの割り当てである．以下のようにボタンが押された際に，それぞれのボタンに割り当てられたボタン ID を取得し，if 文を用いて ID ごとに呼び出すメソッドを変更する．

ID = 0 は A ボタンで，呼び出すメソッドは settoggle2 である．これはトグルボタンを OFF にするというもので，本アプリケーションではトグルボタンが OFF の時，対象物認識をしない．よって，A ボタンを押すことで対象物認識を OFF とする．

ID = 1 は B ボタンで，呼び出すメソッドは settoggle3 である．これはトグルボタンを ON にするというもので，トグルボタンが ON の時対象物認識を行うため，B ボタンを押すことで対象物認識を ON にする．

ID = 2 は C ボタンで，呼び出すメソッドは SETzero である．これは移動していたサーボモータを初期位置に戻すものである．よって，C ボタンを押すことで初期位置をとる．

ID = 3 は D ボタンで，呼び出すメソッドは finish である．これはアプリケーションを終了させる際に呼ばれるものである．よって，D ボタンを押すことでアプリケーションを終了する．

```
       protected String getButtonName(ButtonEvent event) {
       int b = event.getButtonID();
       int gameAction = event.getButtonGameAction();
       String label = event.getController().getConfiguration()
                    .getButtonLabel(b);

       if (b == 0){
              settoggle2();        //対象物認識オフ
       }
```

```
        if (b == 1){
               settoggle1();          //対象物認識オン
        }

        if (b == 2){
               SETzero();             //モータ位置リセット
        }

        if (b == 3){
               finish();              //終了
        }
```

6.2　操作性の向上

6.2.1　ローパスフィルタ

ローパスフィルタは低周波の信号をよく通し，ある遮断周波数より高い周波数帯の信号を通さない（減衰させる）フィルタである．信号に不要な高周波成分を取り除くのに有効なフィルタであり，遮断周波数は任意に選ぶことができる．この理論について，最も簡単なローパスフィルタとされるコンデンサ C と抵抗 R を用いた回路（図 6.4）を例に述べる（文献[9]参照）．

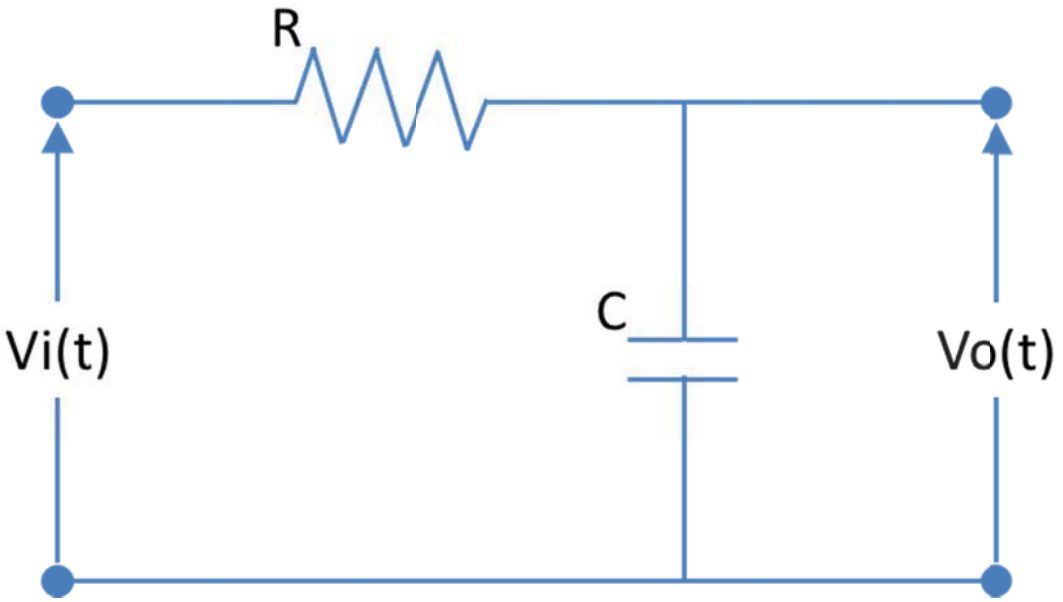


図 6.4　ローパスフィルタの回路

回路への入力電圧を$V_i(t)$とし，出力電圧を$V_o(t)$とすると，この回路の時間発展を示す微分方程式は以下のようになる．

$$\frac{dVo}{dt} = -\frac{1}{RC}V_0 + \frac{1}{RC}V_i(t) \quad \cdots \quad (6.1)$$

そして，$t = t_0$の時の初期状態$V_o(t_0)$を満たす解は以下のようになる．

$$V_o(t) = V_o(t_0)e^{-\frac{t-t_0}{RC}} + \frac{1}{RC}\int_{t_0}^{t} V_i(t')e^{-\frac{t-t'}{RC}}dt' \quad \cdots \quad (6.2)$$

ここで，$t_0 \to -\infty$の極限をとると次のようになる．

$$V_o(t) = \frac{1}{RC}\int_{-\infty}^{t} V_i(t')e^{-\frac{t-t'}{RC}}dt' \quad \cdots \quad (6.3)$$

さらに，ここに$t - t' = t''$の変数変換を行うと以下のように整理できる．

$$V_o(t) = -\frac{1}{RC}\int_{\infty}^{0} V_i(t-t'')e^{-\frac{t''}{RC}}dt''$$

$$= \frac{1}{RC}\int_{0}^{\infty} V_i(t-t'')e^{-\frac{t''}{RC}}dt'' \quad \cdots \quad (6.4)$$

ここでインパルス応答として以下を定義する．

$$\mathrm{H}(t) = \begin{cases} \dfrac{1}{RC}e^{-\frac{t}{RC}} & (t \geqq 0) \\ 0 & (\text{それ以外}) \end{cases} \quad \cdots \quad (6.5)$$

これを用いると，(6.4)式は次のようになる．

$$V_0(t) = \int_{-\infty}^{\infty} H(t')V_i(t-t')dt' = H(t) \times V_i(t) \quad \cdots \quad (6.6)$$

よって，出力$V_o(t)$がインパルス応答$H(t)$と入力$V_i(t)$の畳み込み積分で表せることがわかる．

次に出力と入力の様子について例を図示すると（図 6.5），出力はコンデンサに充電された電荷の量に比例した量であることから，電荷の充電と放電は入力をゆっくり追いかける形になる．

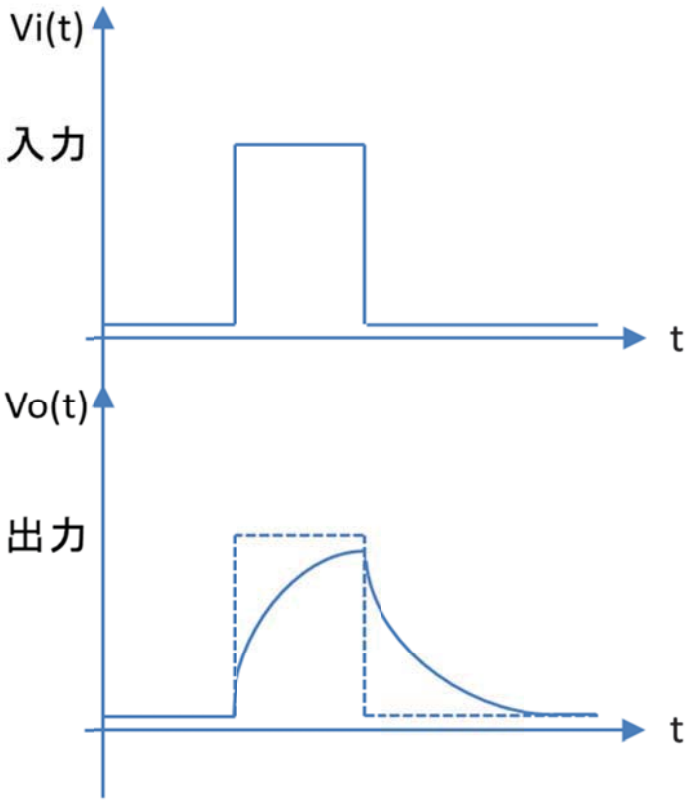


図 6.5　出力と入力の様子（コンデンサの充電/放電）

ここで，出力$V_o(t)$，インパルス応答$H(t)$，入力$V_i(t)$のフーリエ変換を$V_o(f)$，$H(f)$，$V_i(f)$とすると畳み込み積分のフーリエ変換の性質から次のように表せる．

$$V_o(f) = H(f)V_i(f) \quad \cdots \quad (6.7)$$

また，$H(f)$は以下のようになる．

$$H(f) = \frac{f_L^2}{f^2 + f_L^2} - i\frac{f_L f}{f^2 + f_L^2} \quad \cdots \quad (6.8)$$

ただし，(6.8)式は以下の(6.9)式を用いて整理したものである．また(6.9)式で表されるf_Lをローパスフィルタのカットオフ周波数と呼ぶ．

$$f_L = \frac{1}{2\pi RC} \quad \cdots \quad (6.9)$$

(6.8)式の振幅スペクトルの概形は高周波成分が減衰するもの（図 6.6）であるため，(6.7)式のように$V_i(f)$に$H(f)$を掛けるということは，$V_i(f)$の高周波成分を小さくし，低周波成分を通すということになる．

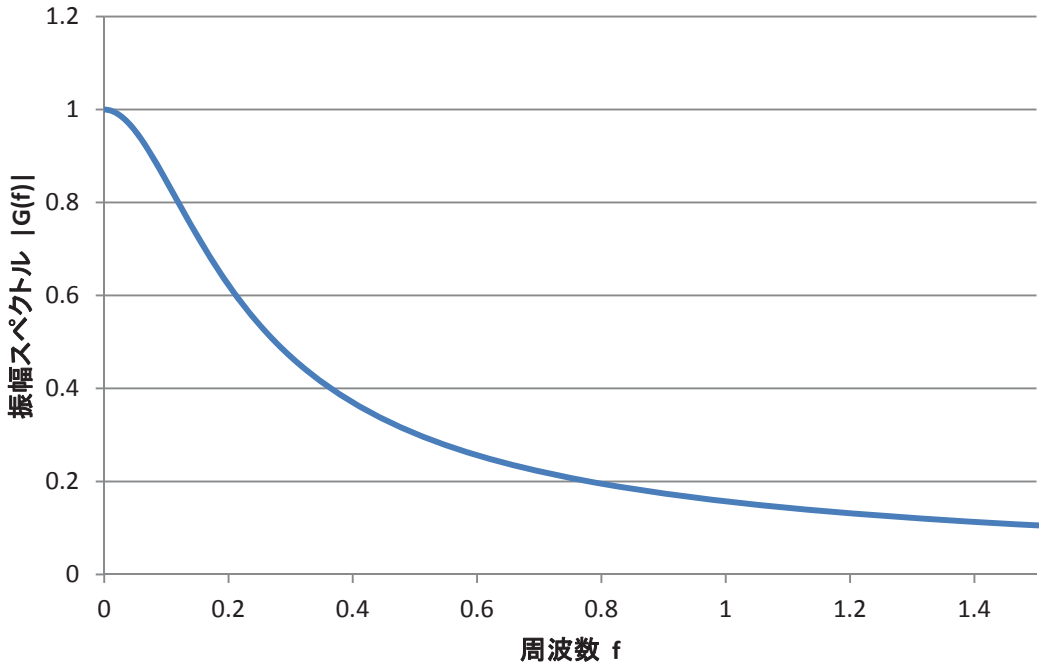


図 6.6　振幅スペクトルの概形

6.2.2　実装

ローパスフィルタを利用して操作性の向上を図る．カメラから取得した画像から，目標角xと１つ前の目標角x_0を用いて，修正した目標角x_1をモータに送る．

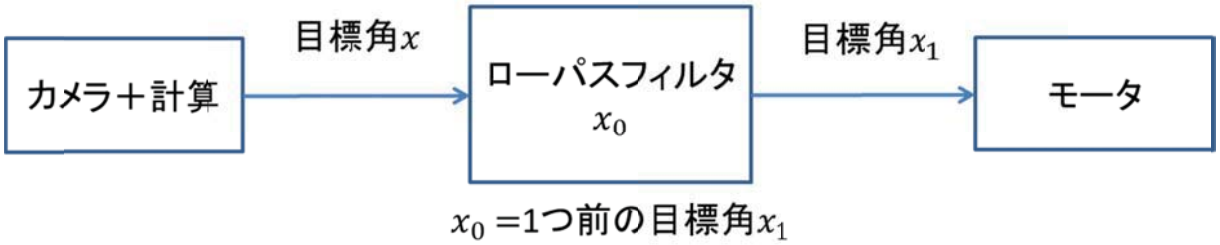


図 6.7　ローパスフィルタの実装

以下の計算により，x_1を求める．

$$x_1 = (1-\alpha)x + \alpha x_0$$

ただし$0 \leqq \alpha \leqq 1$である．ここでこれにより起こる結果について例を挙げてみてみる．

$\alpha = 0$　$x_1 = x$　(xそのもの)

$\alpha = 0.5$　$x_1 = \frac{x+x_0}{2}$　(過去と今の平均)

$\alpha = 1$　$x_1 = x_0$　(過去から変化しない)

このようにαを大きくするほど高周波成分をカットする．本研究ではαを 0.25 と設定して，このローパスフィルタを利用した．結果，対象物追跡時のカメラのブレを軽減することに成功した．

第7章　結論

カメラを利用した対象物追跡アプリケーションを作成した．また，その際に画像処理における処理の高速化に成功した．以下，それについて具体的に示す．

まず，画像処理の高速化についてであるが，本研究では 5 つの手法でカメラアプリケーションを書き換え，処理速度について検証した．

①Java のみ（JIT なし）
②Java のみ（JIT あり）
③JNI の利用：ユーザー処理部の一部を C 言語化
④JNI+Neon の利用：ユーザー処理部に ARM の Neon 命令を利用
⑤システム部でも Neon を利用

その結果，⑤のシステム部でも Neon を利用したプログラムが最も速いことがわかった．また，検証した各デバイスにおける処理の高速化に成功した．それぞれで最も処理が遅かったプログラムと比較すると，処理速度は BeagleBoard において約 2.7 倍，PandaBoard においては約 2.9 倍，Xperia では約 13.5 倍に向上している．

この結果に基づき，対象物追跡アプリケーションを作成した．それぞれ Android のシステムを利用した認識の手法と，テンプレートマッチングを利用した認識の手法を利用した 2 つのアプリケーションを作成した．システム関数を利用した認識処理よりも，認識部をユーザー関数化したものの方が, 処理が高速に行えた. 処理速度を検証したデバイスは BeagleBoard と PandaBoard であり，認識部をユーザー関数化することでそれぞれ処理速度は約 2.3 倍，約 3.2 倍に向上した．

また，作成したアプリケーションにおける機能の拡張についても検証した． Bluetooth デバイスによる遠隔操作と，ローパスフィルタによる追随性の向上に成功した．これにより，Android アプリケーションの自由度の高さと，その拡張性について確認できた．このことから Android OS だけでなく，利用できる外部ハードウェアの性能や利便性の向上で，アプリケーションの幅が広がることがわかる．

最後に，処理速度向上のためのさらなるプログラムの改善を今後の課題とし，Android OS の利用の場を広げるためにもハードウェアの発展にも期待したいと考える．

謝辞

本研究を行うに当たり，終始懇切なるご指導を賜りました金丸隆志准教授に深く感謝し，本論文の執筆に当たり，ご指導を賜りました疋田光孝教授，濱根洋人准教授に心より御礼申し上げます．

私はとりわけ秀でた学生でもなく，勉学に対し生真面目というわけでもありませんでした．一言で表すなら手のかかる学生だったと思います．学部の専攻とは違った情報系の勉強がしたいと考えて進学した私でしたが，研究室に入ったころは情報系の知識やスキルはほぼ皆無で，何もわからず指示されたことをする，時にはそれすら満足にできないことすらありました．そんな私の事も考慮して下さり，作業とは別に毎週勉強会を行ったり，少しでも勉強になるからと専門雑誌を研究室に買ってきてくださったりと環境にまで手を尽くして下さいました．また，研究の方針が決まってからも，作業がなかなか進まず悪戦苦闘する私に対し，多大な助力をいただきました．まだまだ至らないところだらけの私ですが，研究室での 2 年間で成長できたと思います．改めて，金丸准教授に心より感謝の意を表します．

そして，いつも私を支えてくれた研究室の同輩，後輩たちに感謝します．中でも，机を並べて共に 2 年間学び，研究だけでなく私生活においても良き友人である齋藤翼氏にこの場で感謝の意を表します．

最後に，遠方からいつも私を支え，応援して下さった両親に感謝いたします．

付録

I　Android の環境設定

(a)　デフォルトでの日本語フォントインストール

まず，デフォルトで日本語フォントがインストールされるように設定する．設定方法は，Linux 端末上で，$ANDROID/frameworks/base/data/fonts/Android.mk を開き，以下のように DroidSansJapanese.ttf を追加する．ここで，$ANDROID は Android ソースファイルの場所を指す．

```
    DroidSans-Bold.ttf      ¥
    DroidSansJapanese.ttf   ¥
    DroidSerif-Regular.ttf  ¥
```

追記する場所（順番）はそれほど重要ではないが，行末の「¥」は「改行を無視する」という意味があるので重要である．

(b)　常時点灯

次に，常時点灯の設定に移る．$ANDROID/packages/apps/Settings/res/values/arrays.xml（英語環境用）において screen_timeout_entries の項目を以下のように末尾に「Never timeout」を追記する．

```
<string-array name="screen_timeout_entries">
    (中略)
    <item>10 minutes</item>
    <item30 minutes</item>
    <item>Never timeout</item>
</string-array>
```

続いて，screen_timeout_values の項目を以下のように末尾に値 -1 を追記する．

```
<string-array name="screen_timeout_values" translatable="false">
    (中略)
    <!-- Do not translate. -->
    <item>600000</item>
    <!-- Do not translate. -->
    <item>1800000</item>
    <!-- Do not translate. -->
    <item>-1</item>
```

```
</string-array>
```

すると，端末の設定でタイムアウト時間の設定に「Never timeout」という項目が現れ，常時点灯が実現できる．ただし，上記は言語が英語の時のみなので，日本語環境では $ANDROID/packages/apps/Settings/res/values-ja/arrays.xml（日本語環境用）において，英語環境用のように screen_timeout_entries の項目の 30 分の下に以下の追記が必要となる．

```
<item msgid="1781492122915870416">"常時点灯"</item>
```

ちなみに，values/arrays.xml は英語環境，values-ja/arrays.xml 日本語環境でそれぞれ用いられる．本来ならば values-?? の全ての言語のファイルを変更すべきであるが，本研究では英語と日本語のみにとどめた．

(c) 機内モード削除

次に，機内モードの削除を行った．誤って機内モードにしてしまった場合に復帰できなくなることがあるので，無効にした方が良いと判断したためである．その変更法は Android version2.2 を例とすると，以下の Java ファイルを開き，airplane に関連する項目をすべてコメントアウトするというものである．

```
$ANDROID/packages/apps/Settings/src/com/android/settings/WirelessSettings.java
```

コメントアウト終了後も設定に「機内モード」の項目は残るが，クリックしても何も起こらない．

II　各種デバイスへの対応

(a)　kernel の変更

まず，kernel を「無線 LAN」,「Bluetooth」,「タッチパネル」に対応させる．なお,「USB カメラ」と「USB シリアル変換デバイス」については 2.6.5 章で解説した．

```
$ANDROID/kernel-beagleboard/arch/arm/configs/sola_omap3_beagle_andro
id_defconfig
```

上記のファイルが kernel の設定ファイルなので，これを変更する．下記の内容を反映させる．

```
# Wireless LAN
CONFIG_CFG80211=y
CONFIG_NL80211=y
CONFIG_WIRELESS_EXT=y
CONFIG_WIRELESS_EXT_SYSFS=y
CONFIG_MAC80211=y

CONFIG_MAC80211_RC_PID=y
CONFIG_MAC80211_RC_DEFAULT_PID=y
CONFIG_MAC80211_RC_DEFAULT="pid"
CONFIG_MAC80211_MESH=y
CONFIG_MAC80211_LEDS=y
CONFIG_IEEE80211=y
CONFIG_IEEE80211_CRYPT_WEP=y
CONFIG_IEEE80211_CRYPT_CCMP=y
CONFIG_IEEE80211_CRYPT_TKIP=y

CONFIG_WLAN_80211=y
CONFIG_HOSTAP=y
CONFIG_HOSTAP_FIRMWARE=y
CONFIG_HOSTAP_FIRMWARE_NVRAM=y
CONFIG_RT2X00=m
CONFIG_RT2X00_LIB=m
CONFIG_RT2X00_LIB_USB=m
CONFIG_RT2X00_LIB_FIRMWARE=y
CONFIG_RT2X00_LIB_LEDS=y
```

```
CONFIG_RT73USB=m
CONFIG_RT73USB_LEDS=y
CONFIG_CRC_ITU_T=y

# Bluetooth 関連
CONFIG_BT=y
CONFIG_BT_L2CAP=y
CONFIG_BT_SCO=y
CONFIG_BT_RFCOMM=y
CONFIG_BT_RFCOMM_TTY=y
CONFIG_BT_BNEP=y
CONFIG_BT_HIDP=y
CONFIG_BT_HCIBTUSB=y
CONFIG_USB_ARCH_HAS_OHCI=y
CONFIG_INPUT_UINPUT=y

# タッチパネル関係
# CONFIG_TOUCHSCREEN_USB_EGALAX is not set   // 有効だったものを無効にする
CONFIG_TOUCHSCREEN_USB_GENERAL_TOUCH=y
```

以上の反映を全て施したら，kernel をビルドする．kernel のビルドは以下のコマンドで実行可能である．

```
$ cd $ANDROID/kernel-beagleboard
$ make ARCH=arm CROSS_COMPILE=../prebuilt/linux-x86/toolchain/
  arm-eabi-4.4.0/bin/arm-eabi-sola_omap3_beagle_android_defconfig
$ make ARCH=arm CROSS_COMPILE=../prebuilt/linux-x86/toolchain/
  arm-eabi-4.4.0/bin/arm-eabi- uImage modules
```

ビルドが終ったら，uImage を SD カードの DISK1 にコピーする． さらに，Android ビルド後に rt73usb.ko，rt2x00lib.ko，rt2x200usb.ko の 3 ファイルを/system/lib/modules にコピーする．続いて，rt73.bin を/system/etc/firmware にコピーする．

(b) 無線 LAN の利用

次に，Android のソースを変更して，無線 LAN を利用可能にする．$ANDROID/vendor/sola/beagleboard/BoardConfig.mk に下記の 2 行を追加する．

```
WPA_BUILD_SUPPLICANT := true
BOARD_WPA_SUPPLICANT_DRIVER := WEXT
```

$ANDROID/vendor/sola/omap3/image/beagleboard/ に下記の内容の wpa_supplicant.conf を作成する.

```
Update config=1
Ctrl_interface=wlan0
Eapol_version=1
Ap_scan=1
Fast_reauth=1
```

$ANDROID/vendor/sola/omap3/image/beagleboard-image.sh のファイルを編集用に開く. 「cp $ANDROID/vendor/sola/omap3/image/beagleboard/rt73.bin ... (略)」の次に下記の 2 行を追加する.

```
mkdir -p $ANDROID/vendor/sola/omap3/image/beagleboard/android/system/
etc/wifi
cp $ANDROID/vendor/sola/omap3/image/beagleboard/wpa_supplicant.conf
$ANDROID/vendor/sola/omap3/image/beagleboard/android/system/etc/wifi
```

次に $ANDROID/vendor/sola/beagleboard/init.omap3.rc を修正するために開く. on boot の次の行に下記の行を追加する.

```
setprop wifi.interface "wlan0"
setprop wlan.driver.status "ok"
setprop dalvik.vm.heapsize "32m"
```

同じく $ANDROID/vendor/sola/beagleboard/init.omap3.rc の末尾に下記を追加する.

```
#WIFI
service ifcfg_ralink /system/bin/ifconfig wlan0 up
#  disabled
#  oneshot

# service wpa_supplicant /system/bin/logwrapper /system/bin/wpa_
  supplicant -Dwext -iwlan0 -c /system/etc/wifi/wpa_supplicant.conf -dd
#    disabled
#    group wifi
```

```
# for Android private socket
  service wpa_supplicant /system/bin/wpa_supplicant -dd -Dwext -iwlan0
  -c /system/etc/wifi/wpa_supplicant.conf
  socket wpa_wlan0 dgram 660 wifi wifi
  group system wifi inet
  disabled
  oneshot
service dhcpcd /system/bin/logwrapper /system/bin/dhcpcd -d wlan0
    disabled
    oneshot
#group system dhcp

on property:init.svc.wpa_supplicant=stopped
    stop dhcpcd
```

次に、$ANDROID/vendor/sola/beagleboard/system.prop を編集用に開き，末尾に下記の 1 行を追加する.

```
wifi.interface = wlan0
```

続いて$ANDROID/vendor/sola/beagleboard/init.rc を編集用に開く. 既存の下記の 3 行をコメントアウトする.

```
#    mkdir /data/misc/wifi 0770 wifi wifi
#    chmod 0770 /data/misc/wifi
#    chmod 0660 /data/misc/wifi/wpa_supplicant.conf
```

そして，その位置にかわりに以下を挿入する.

```
chmod 755 /system/etc/dhcpcd
chmod 755 /system/etc/dhcpcd /dhcpcd-run-hooks
chmod 755 /system/etc/dhcpcd /dhcpcd-hooks
chmod 755 /system/etc/dhcpcd /dhcpcd-hooks/*

chown 1010 1010 /system/etc/wifi
chmod 770 /system/etc/wifi
```

```
chown 1010 1010 /system/etc/wifi/wpa_supplicant.conf
chmod 660 /system/etc/wifi/wpa_supplicant.conf

mkdir /data/misc/wifi
chown 1010 1010 /data/misc/wifi
chmod 770 /data/misc/wifi
mkdir /data/misc/wifi/sockets
chown 1010 1010 /data/misc/wifi/sockets
chmod 770 /data/misc/wifi/sockets
chown 1010 1010 /data/misc/wifi/wpa_supplicant.conf
chmod 660 /data/misc/wifi/wpa_supplicant.conf

mkdir /data/misc/dhcp
chown 1014 1014 /data/misc/dhcp
chmod 0770 /data/misc/dhcp
```

次に，$ANDROID/frameworks/base/wifi/java/android/net/wifi/WifiStateTracker.java を編集用に開く．下記のように，tiwlan0 を wlan0 に変更する．

```
// mInterfaceName = SystemProperties.get("wifi.interface", "tiwlan0");
    mInterfaceName = SystemProperties.get("wifi.interface", "wlan0");
```

続いて，driver_wext.h，driver_wext.c，wifi.c を 2.6.3 章でダウンロードした vendor_tk-beagle-froyo2.2.x-20110725（以下 vendor_tk）に含まれているもので置き換える．以上の変更を施した後，Android を再ビルドし，DISK3 に載せれば無線 LAN が使える．

(c) Bluetooth の利用

次に，Bluetooth の利用についてである．まず bluetooth.c を vendor_tk に含まれているものに置き換える．

続いて，$ANDROID/system/bluetooth/bluedroid/Android.mk を編集用に開き，include $(CLEAR_VARS) の次の行に以下の三行を挿入する．

```
ifeq ($(HAVE_NO_RFKILL_SWITCH),true)
    LOCAL_CFLAGS += -DNO_RFKILL_SWITCH
endif
```

次に，$ANDROID/external/bluetooth/bluez/src/adapter.c を編集用に開いて add_rfcomm

_service_record 関数内で uint8_t channel; を以下に変更する.

```
//       uint8_t channel;
   uint16_t channel;
```

そして，$ANDROID/vendor/sola/beagleboard/BoardConfig.mk を編集用に開き，末尾に以下の 2 行を追加する.

```
BOARD_HAVE_BLUETOOTH := true
HAVE_NO_RFKILL_SWITCH := true
```

$ANDROID/vendor/sola/beagleboard/init.rc を編集用に開き, コメントアウトされた下記の部分を見付け，行頭の # を全て削除して有効にする．ただし，以下には本当のコメントの２行が含まれているので，その# は削除しないことに注意する(init.rc does not yet... の部分).

```
#service dbus /system/bin/dbus-daemon --system --nofork
#    socket dbus stream 660 bluetooth bluetooth
#    user bluetooth
#    group bluetooth net_bt_admin

#service bluetoothd /system/bin/bluetoothd -n
#    socket bluetooth stream 660 bluetooth bluetooth
#    socket dbus_bluetooth stream 660 bluetooth bluetooth
# init.rc does not yet support applying capabilities, so run as root and
# let bluetoothd drop uid to bluetooth with the right linux capabilities
#    group bluetooth net_bt_admin misc
#    disabled
(中略)
#service opush /system/bin/sdptool add --channel=12 OPUSH
#    user bluetooth
#    group bluetooth net_bt_admin
#    disabled
#    oneshot

#service pbap /system/bin/sdptool add --channel=19 PBAP
#    user bluetooth
#    group bluetooth net_bt_admin
```

```
#    disabled
#    oneshot
```

以上の変更を施した後, Android を再ビルドする. その際, 一度 make clean した方がよい. ビルドが完了し，新しく作ったイメージを DISK3 に載せれば Bluetooth が動く.

(d) タッチスクリーンの利用

次に，タッチスクリーンの利用について示す．タッチスクリーンのタッチ機能を有効にするには，h_kojima 氏のページ[10]を参考にした．kernel の更新だけで良い.
下記 kernel の設定ファイルで EGALAX を無効化し，GENERAL を有効化する必要がある.
$ANDROID/kernel-beagleboard/arch/arm/configs/sola_omap3_beagle_android_defconfig
以下その変更点である.

```
# タッチパネル関係
# CONFIG_TOUCHSCREEN_USB_EGALAX is not set   // 有効だったものを無効にする
CONFIG_TOUCHSCREEN_USB_GENERAL_TOUCH=y
```

さらに，$ANDROID/kernel-beagleboard/drivers/input/touchscreen/usbtouchscreen.c を，vendor_tk に含まれているものに変更してみる．ただし，kernel 2.6.32.40 のファイルに差し替えた上で変更を行わないとうまく動作しないことに注意する．以上で，タッチパネル利用のための kernel 変更は完了である．あとは，kernel をビルドし，kernel のみを差し替えれば良い．しかし，ここまでの変更だけではスリープ解除時にキーボードが必要である．以下，タッチによるスリープの解除に対応方法である.
$ANDROID/frameworks/policies/base/phone/com/android/internal/policy/impl/PhoneWindowManager.java を変更する．ファイルを探すと，以下のような場所が一箇所だけある.

```
        }
    } else if (!screenIsOn) {
        // If we are in-call with screen off and keyguard is not showing,
        // then handle the volume key ourselves.
        // This is necessary because the phone app will disable the keyguard
```

ここを下記のように変更する．変更の際，位置に気をつける．1 行目の Log.d はデバッグ用なので必須ではないが、念のため入れておく.

```
                Log.d("TK", "event.type="+event.type+", event.keycode="+event.keycode);
            if(event.type==1 && event.keycode==0){
```

```
mKeyguardMediator.onWakeKeyWhenKeyguardShowingTq(event.keycode);
                }


        }
    } else if (!screenIsOn) {
        // If we are in-call with screen off and keyguard is not showing,
        // then handle the volume key ourselves.
        // This is necessary because the phone app will disable the keyguard
```

この変更を施した後, Android 全体を更新しなければならない. kernel の更新は不要である.
これにより，タッチでスリープを解除できるようになった.

以上でデバイスへの対応は完了である.

III　JNI の利用における環境設定と利用方法

本章では，JNI を利用するための環境設定と，設定後の JNI の利用法についてまとめる．まず，Cygwin のウェブサイト[11]から setup.exe をダウンロードする（Figure III-i）．

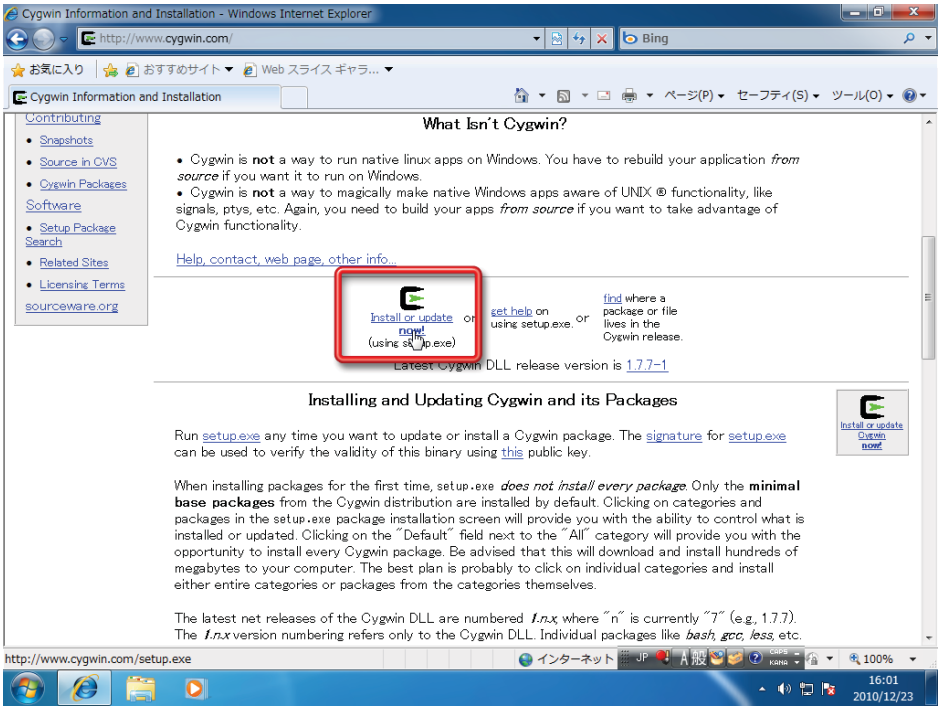


Figure III-i　Cygwin のダウンロード

ダウンロードした Setup.exe を実行する．起動したインストーラに従ってインストールを進めると，ダウンロードパッケージの選択画面が開く．「Search」欄に「make」と入力して，「Devel」にある「make: The GNU version of the 'make' utility」の項目で「Skip」をクリックして「Skip」がバージョン表記（ここでは「3.81-2」）になるようにする（Figure III-ii）．

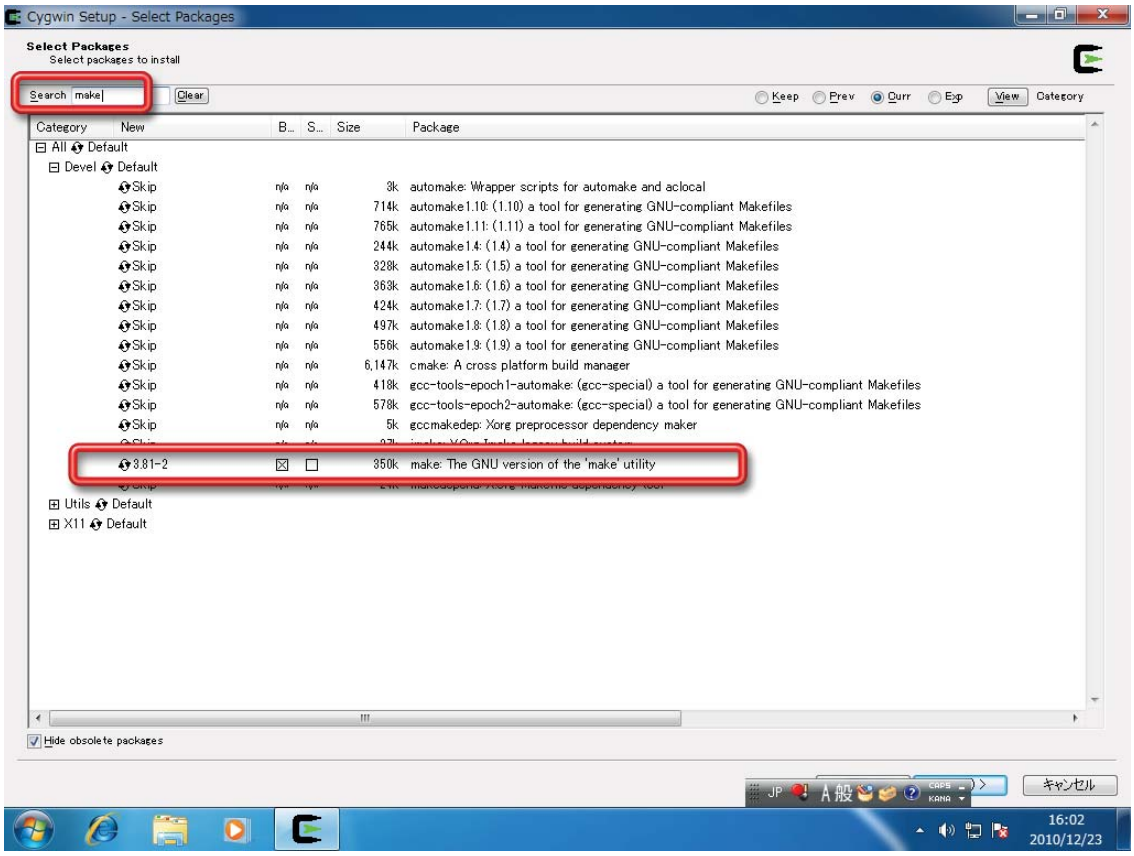


Figure III-ii　ダウンロードパッケージの選択

次に「Search」欄に「gcc4」と入力して，「Devel」にある「gcc4: GCC Release series 4 compiler (C & C++ install helper」の「Skip」をクリックして同じ様にバージョン表記になるようにする．このとき複数回クリックするとインストールバージョンを選ぶことができる（Figure III-iii）．ここまで設定が済んだら「次へ」ボタンを押す．すると，自動的に依存関係が調べられ，必要となるほかのパッケージが自動選択される．そして，インターネットからダウンロード，インストールされる．これで Cygwin のインストールは完了である．

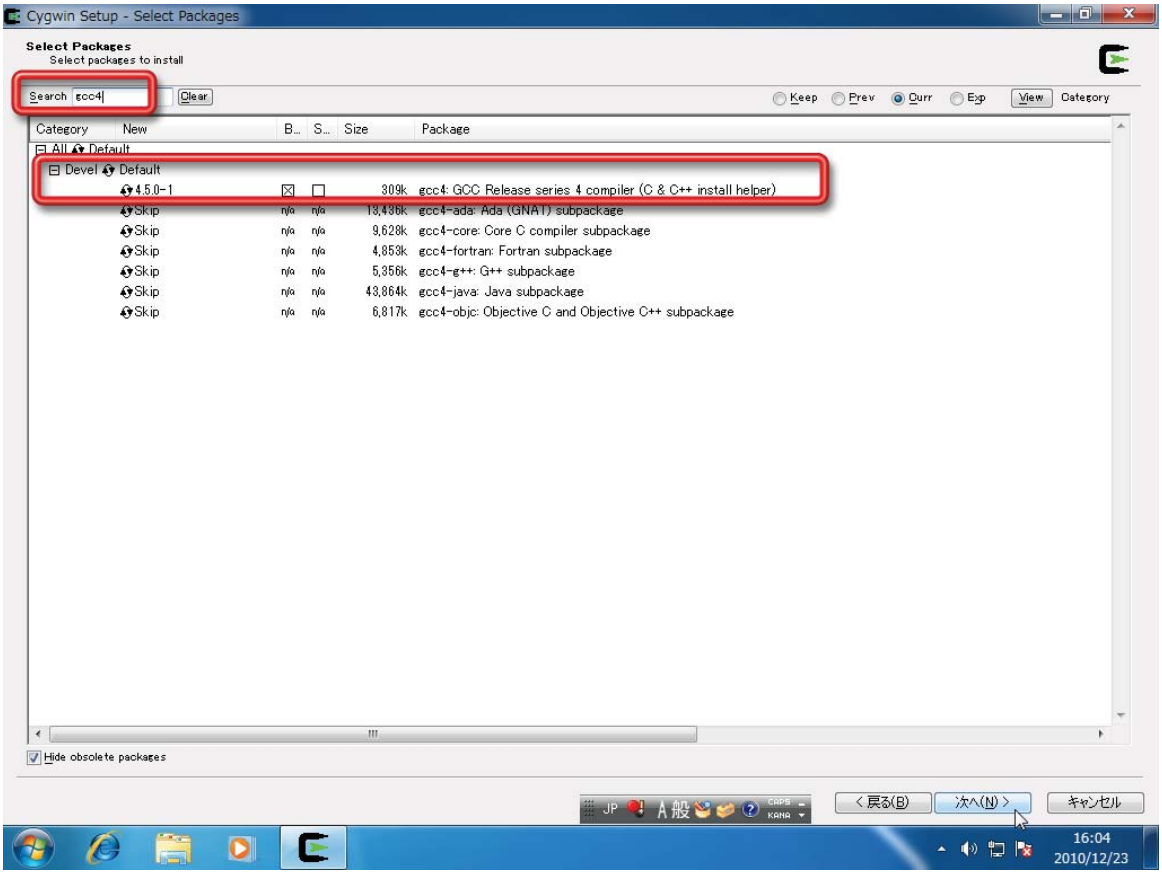


Figure III-iii　インストールバージョンの選択

続いて，Android NDK の設定についてである．まず，Android NDK をダウンロードする．ダウンロード元は android developers のダウンロードページ[12]である．本研究でダウンロードしたのは android-ndk-r5-windows.zip である（Figure III-iv）が，2012 年 1 月現在最新版として android-ndk-r7-windows.zip がダウンロードできるようになっている．Android NDK のインストールは解凍するだけなので，ダウンロードしたファイルを解凍してインストールは完了である．その後，C ドライブ直下にフォルダを移動し，フォルダ名を android-ndk とした．次に一度 Cygwin を起動する．初回起動時にはユーザー用のスクリプトファイルが自動生成される．確認したら Cygwin を閉じる．

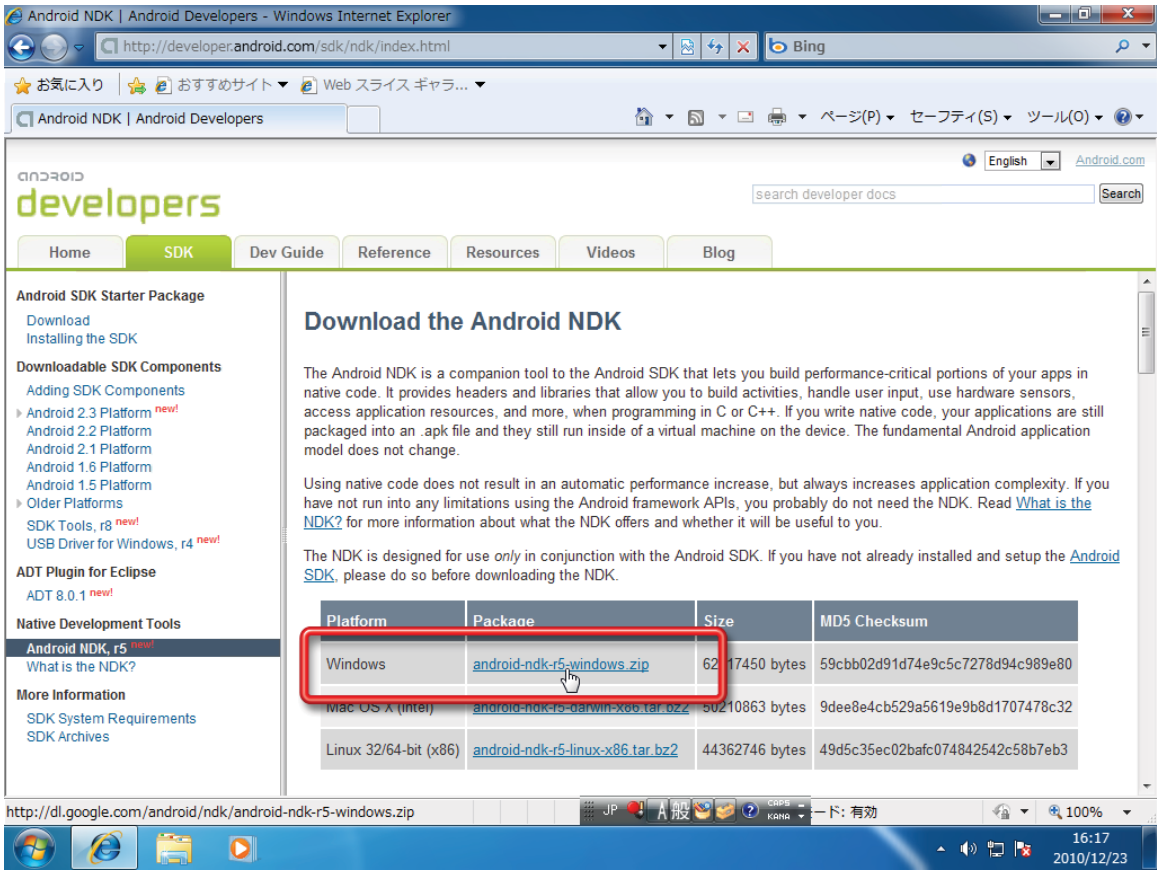


Figure III-iv　Android NDK ダウンロードファイル選択

次に Eclipse を起動し，「ファイル」メニューの「ファイルを開く」を選択する．そして先ほど Cygwin で自動生成させたスクリプトのうち「.bashrc」を開く（Figure III-v）．ファイルの場所は「c:¥cygwin¥home¥ユーザー名¥」である．.bashrc が開いたら，その末尾に以下の 2 行を追加して保存する（Figure III-vi）．

```
export ANDROID_NDK_ROOT=/cygdrive/c/android-ndk
export PATH=$PATH:$ANDROID_NDK_ROOT
```

次に「ヘルプ」メニューの「新規ソフトウェアのインストール」を選択する．そして「作業対象」のリストボックスから「Helios - http://download.eclipse.org/release/helios」を選択する．そして「プログラム言語」の中から「C/C++ Development Tools」と「C/C++ Library API Documentation Hover Help」にチェックを入れて「次へ」ボタンを押す．ライセンスに承諾すると，インストールが開始される．

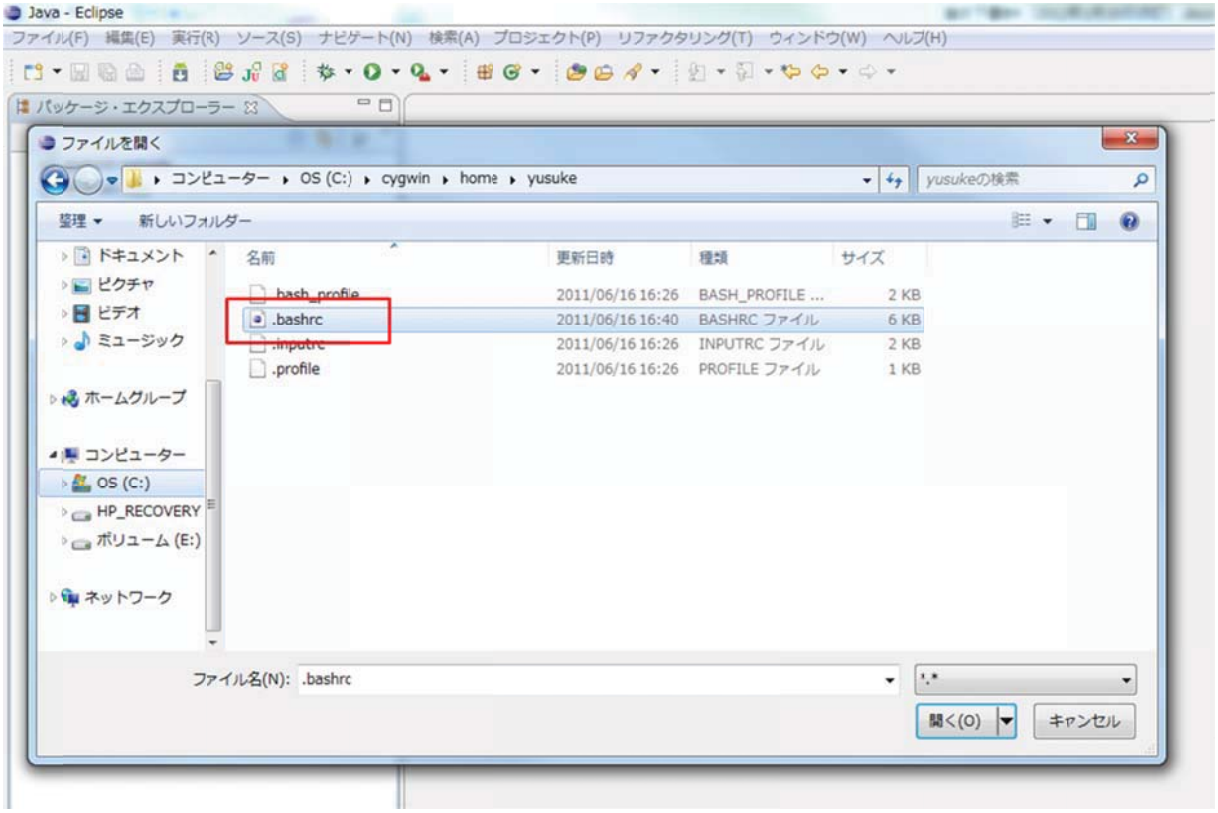


Figure III-v　.bashrc ファイルの選択

```
#     x2=$(dirs +${cnt} 2>/dev/null)
#     [[ $? -ne 0 ]] && return 0
#     [[ ${x2:0:1} == '~' ]] && x2="${HOME}${x2:1}"
#     if [[ "${x2}" == "${the_new_dir}" ]]; then
#       popd -n +$cnt 2>/dev/null 1>/dev/null
#       cnt=cnt-1
#     fi
#   done
#
#   return 0
# }
#
# alias cd=cd_func

export ANDROID_NDK_ROOT=/cygdrive/c/android-ndk
export PATH=$PATH:$ANDROID_NDK_ROOT]
```

Figure III-vi　末尾への追記

インストール完了後，Eclipse の再起動をする．次に「ウィンドウ」メニューの「設定」を選択する．そして「実行/デバッグ」の「起動」「デフォルトランチャー」にある「C/C++ Application」の「Debug」の「Standard Create Process Launcher」にチェックを入れる．そして，「一般」「ワークスペース」にある「自動的にリフレッシュ」にチェックを入れる．これを入れ忘れると C のソースコード変更ごとに手動でリフレッシュ作業が必要になるので，

忘れないようにする．

これで Android NDK のインストールから Eclipse の設定が完了し，JNI を利用するための環境が整った．なお，本手順は UsefullCode.net[13][14][15]を参考とし，1 部の図（Figure III-i～Figure III-iv）はここからの引用である．

続いて，以下に本アプリケーションの仕組みと照らし合わせながら，JNI の利用方法について述べる．まず，Eclipse を起動し，Android プロジェクトを作成する．次に C 言語ソースファイルを保存するフォルダをプロジェクト内に作成する．プロジェクトを右クリックしてあらわれる「新規」にある「フォルダ」を選択することで Android プロジェクト内に新しいフォルダを作ることができる．また，フォルダ名は jni とした．次に作成した jni フォルダ内に Makefaile を作成する．ファイル名は「Android.mk」とする．これは常に固定のファイル名で，大文字小文字を区別するので要注意である．続いて，Android.mk ファイルの中身を編集する．赤い枠で囲まれた部分を記述し，本アプリケーションでは，C 言語のモジュール名を Serial.JNI とした（Figure III-vii）．

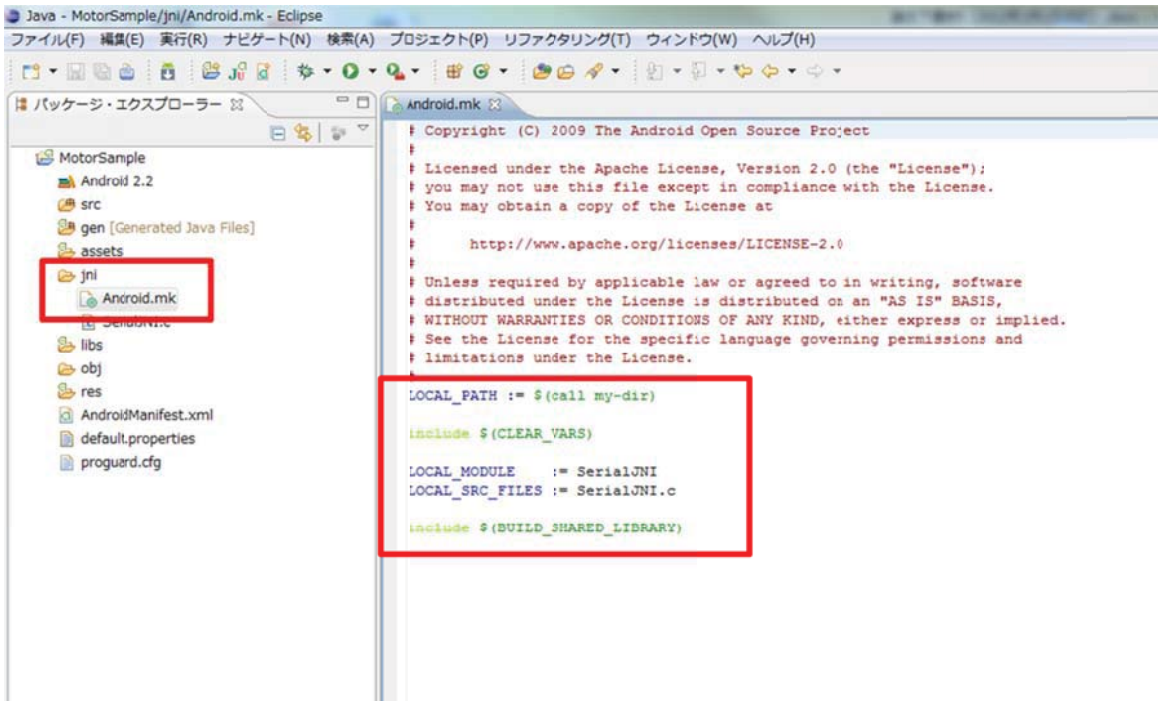


Figure III-vii　Android.mk ファイルの編集

次に jni フォルダ内に C 言語のソースファイルを作成する．ファイル名は任意でつけられるが，Android.mk のモジュール名と同じにすることに気をつける．つまり，本アプリケーションの場合は，モジュール名を Serial.JNI とする． Java へエクスポートする関数名は，「Java_プロジェクト名_アクティビティ名_関数名」と設定する．一例として本アプリケーションの一部を記すと Java_jp_MotorSample_MotorSampleActivity_openSerial のようになっている（Figure III-viii）．

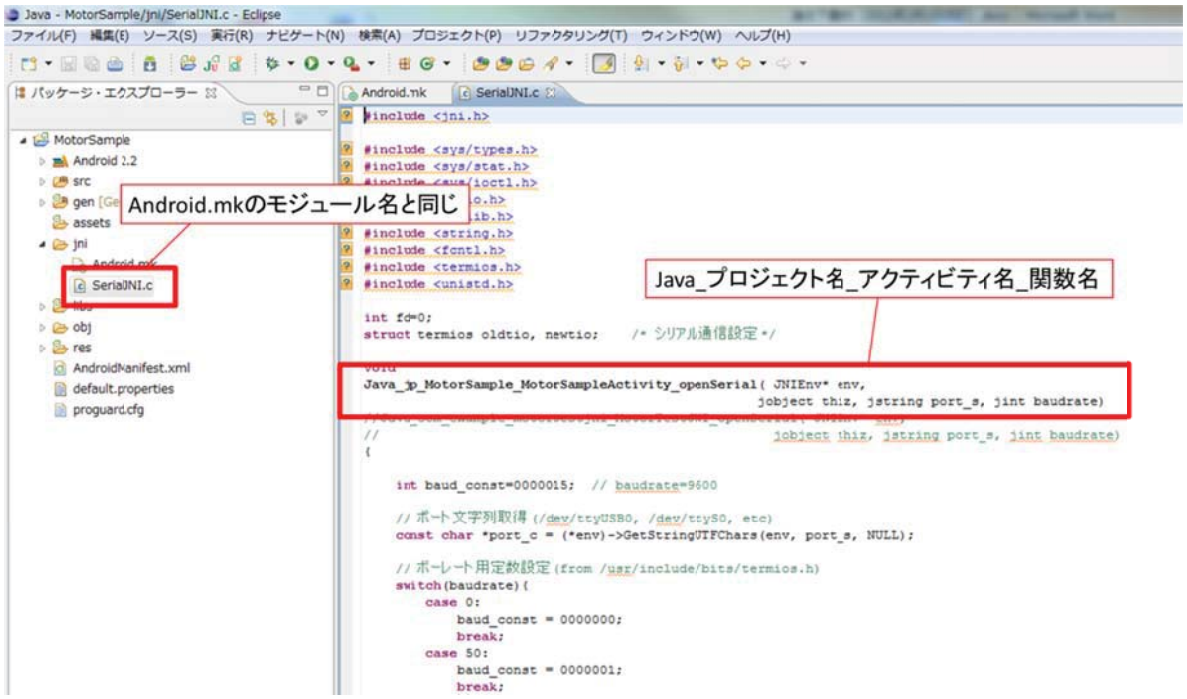


Figure III-viii　C 言語ソースファイルの実装

続いて，Java のソースファイルを編集する．ここで特に重要なのが Figure III-ix の赤枠にある JNI ライブラリの宣言部分である. これにより, 先に編集した C 言語ファイル Serial.JNI を呼び出し，C 言語のモジュールを利用することが可能となる．本アプリケーションはこれにより，Java のソースファイルからシリアル通信ができるようにした.

しかし，この時点では Android のプロジェクトをビルドできない．C モジュールへのアクセスをするには，先述した Cygwin で Android NDK のツールの一つである ndk-build をしなければならないためである．Cygwin を起動し，以下のコマンドを実行する.

```
$cd /cygdrive/c/（JNI のプロジェクトのある場所）
$ndk-build
```

本アプリケーションを例にすると，$cd /cygdrive/c/Users/yusuke/workspace/MotorSample と入力した後，ndk-build となる（Figure III-x）.

Cygwin でビルドが完了したら，Eclipse で Android のパッケージのビルドを行う．これによって，JNI を利用したアプリケーションが完成する.

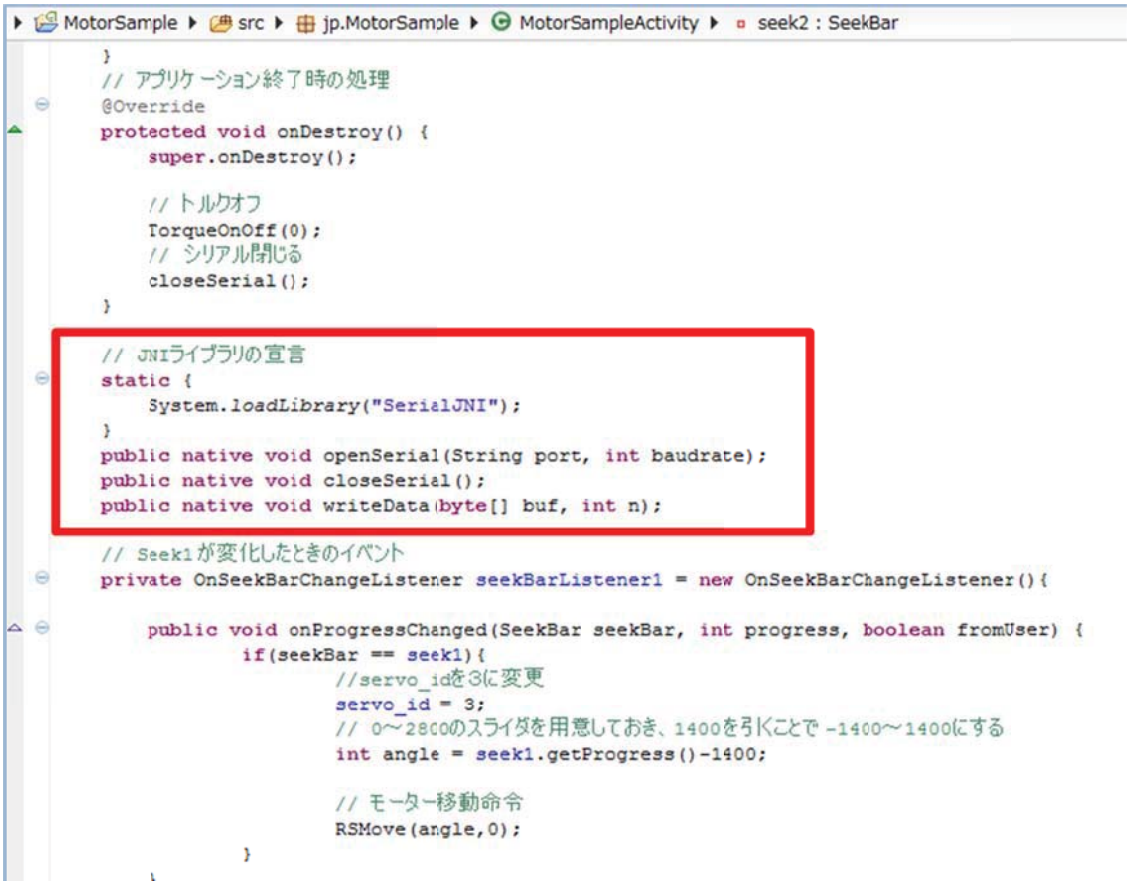


Figure III-ix　モータアプリケーション Java ソースファイルの一部

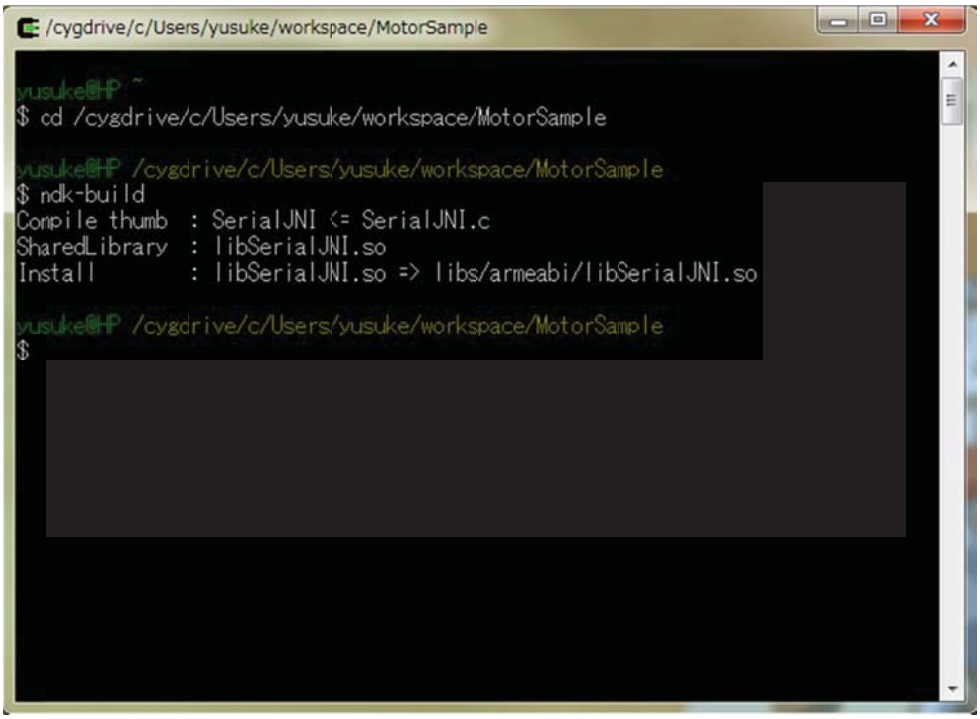


Figure III-x　ndk-build の様子

IV　台座の製作手順

ここでは, カメラの台座の作成法についてまとめる. また, 台座に使用した部品を Table IV-i にまとめた．以下製作手順である.

まず，サーボ ID を変更したモータの突起部分を切除した（Figure IV-i）．切除する際にモータのコードを傷つけないように配慮する．また，サーボ ID の変更法は第 5 章 5.3 駆動部におけるモータの詳細にて解説する．次に，突起部分を切除した 2 つのモータをモータケースに取り付ける.

Table IV-i　カメラ台座部品リスト

部品名	製品名
USB カメラ	Logicool Webcam C210
モータ	双葉電子工業製 RS304MD-FF （2 個）
サーボホーン	G-ROBOTS サーボホーン（5 個入）　（うち 2 個使用）
ワッシャー	POM ワッシャー 2x12x1mm（10 個入）　（うち 2 個使用）
	ワッシャー M2x6mm（10 個入）　（うち 2 個使用）
ラバーブッシュ	G-ROBOTS ラバーブッシュ（5 個入）（うち 1 個使用）
プラブッシュ	G-ROBOTS プラブッシュ（5 個入）（うち 1 個使用）
モータケース	G-ROBOTS GR-001 股関節サーボホルダー（L/R）セット （股関節サーボホルダー R，股関節サーボホルダー1 個のみ使用）
ロボットパーツ①	G-ROBOTS GR-001 肩（L/R）セット　（肩 L のみ使用）
ロボットパーツ②	G-ROBOTS GR-001 すね（L/R）セット　（すね L のみ使用）
アクリル接続部	アクリル角材 24x30x30mm
	アクリルパイプ φ8
	アクリルパイプ φ10
三脚	ケンコー ケータイ&デジカメポッド スーパーミニ[コーラル]
おもり	アルミ台形（3 個）

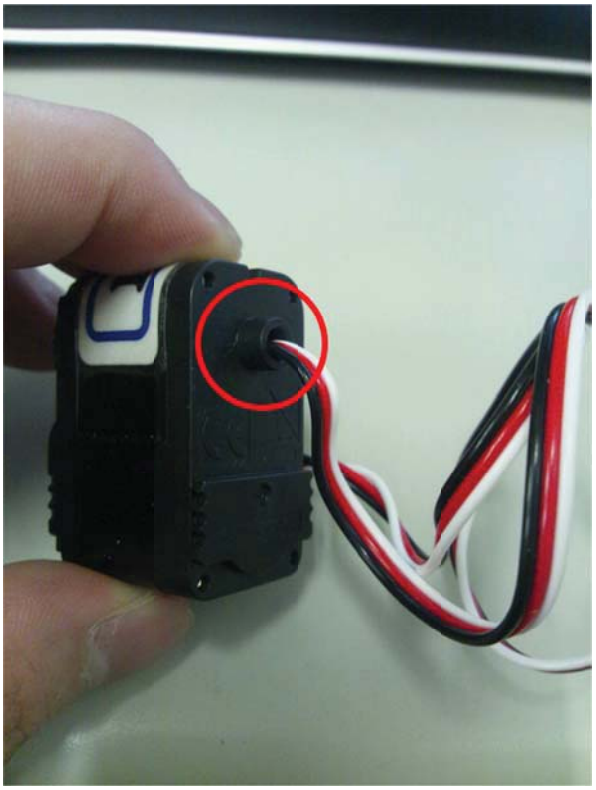


Figure IV-i　切除するモータの突起部

モータケースへの取り付けが完了したら，それぞれのモータにロボットパーツ①と②をとりつける（Figure IV-ii パーツ取り付け例を参考）．その際，モータのサーボホーンを Table IV-i に記したものに交換する．Figure IV-ii の赤で囲まれている部分が，それぞれモータのサーボホーンとロボットパーツ①②との接続部分であるが，空転防止のサーボホーンの突起とパーツのくぼみを合わせること，2 つのワッシャーを重ねて取り付けることに注意する．Figure IV-ii の青で囲まれている部分がサーボモータのコード側とロボットパーツ②の接続部であるが，まずはロボットパーツの穴にコードを通す．次にプラブッシュに切れ目をいれ，コードを通し，プラブッシュをパーツの穴にはめる．最後にラバーブッシュをコードに通し，Figure IV-ii のようにはめ込んで固定する．

次に USB カメラとロボットパーツ②の取り付けである．再利用できるように，カメラに直接穴をあけるなどということはせず，Figure IV ii の緑で囲まれている位置で接着力の強い両面テープで固定した．

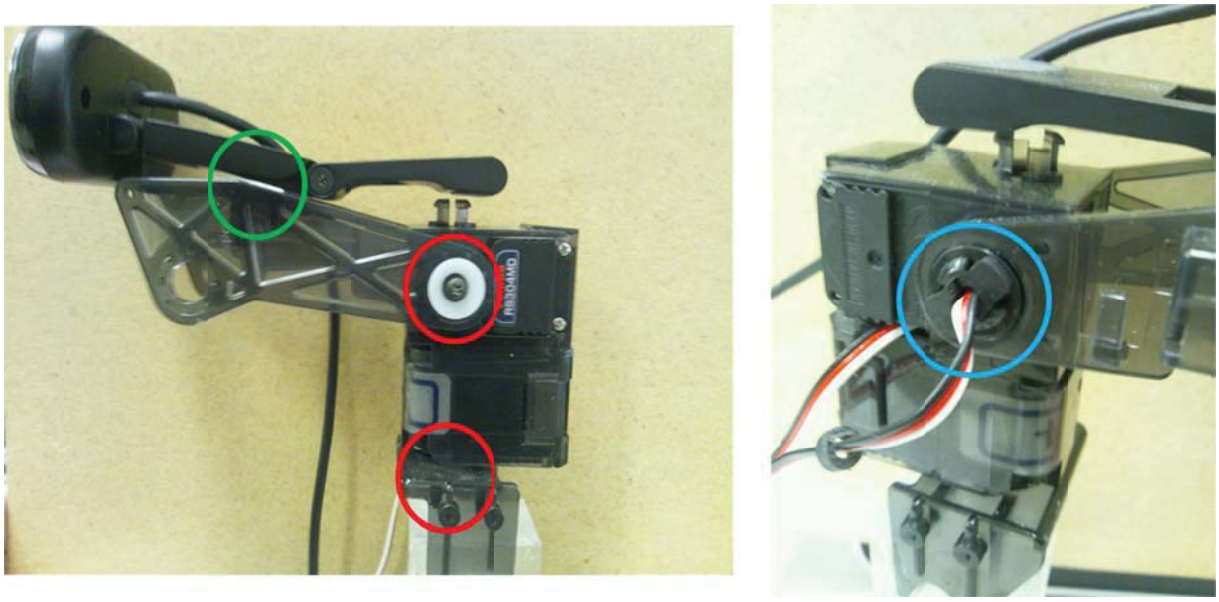


Figure IV-ii　パーツ取り付け例（左：左側面　右：右側面）

続いて，アクリルの接続部の加工である．まず，アクリル角材側面の上部から 10mm，左端から 15mm（中央）の位置にφ8，反対側にφ10 の穴をあける．また，アクリルパイプを適度な長さに切断する．

次に三脚のホルダー部分のネジを外して，ホルダー部分を三脚から取り外す（Figure IV-iii）．また，金属パテを用いて三脚の足の部分におもりを取り付ける（Figure IV-iv）．

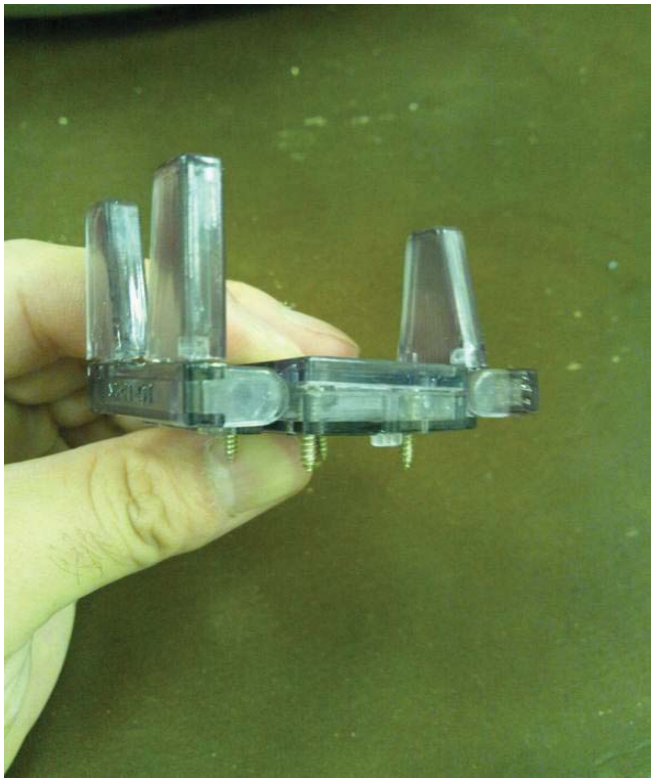


Figure IV-iii　取り外したホルダー部分

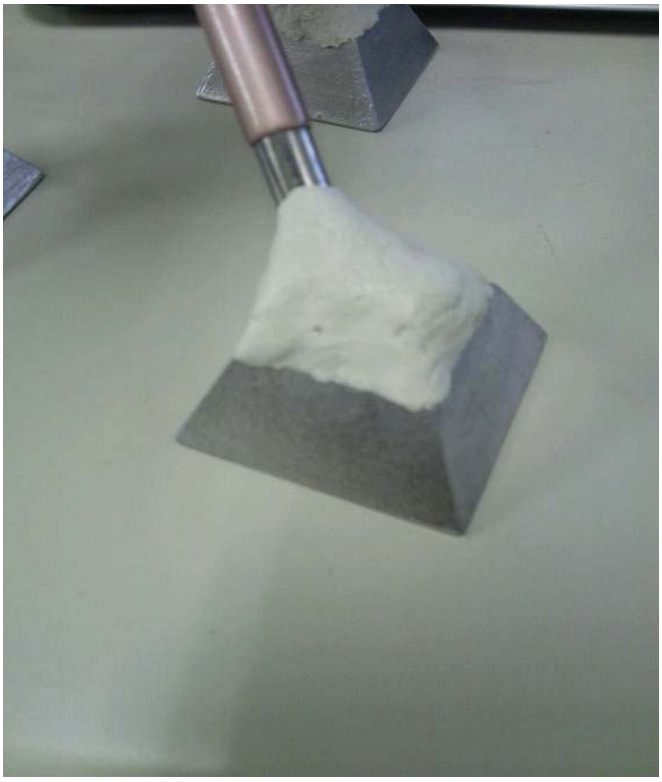


Figure IV-iv　おもりを取り付けた三脚の足

三脚のホルダーを取り外した部分に，アクリル角材を接着剤で固定する（Figure IV-v の緑で囲まれた部分）．また，接着剤が乾いて固定されたことが確認できたら，ロボットパーツ①とアクリル角材とアクリルパイプで接続する（Figure IV-v の赤で囲まれた部分）．

Figure IV-v　三脚とアクリル接続部

以上で，本研究で製作したカメラの台座の完成である．

Figure IV-vi　完成した台座

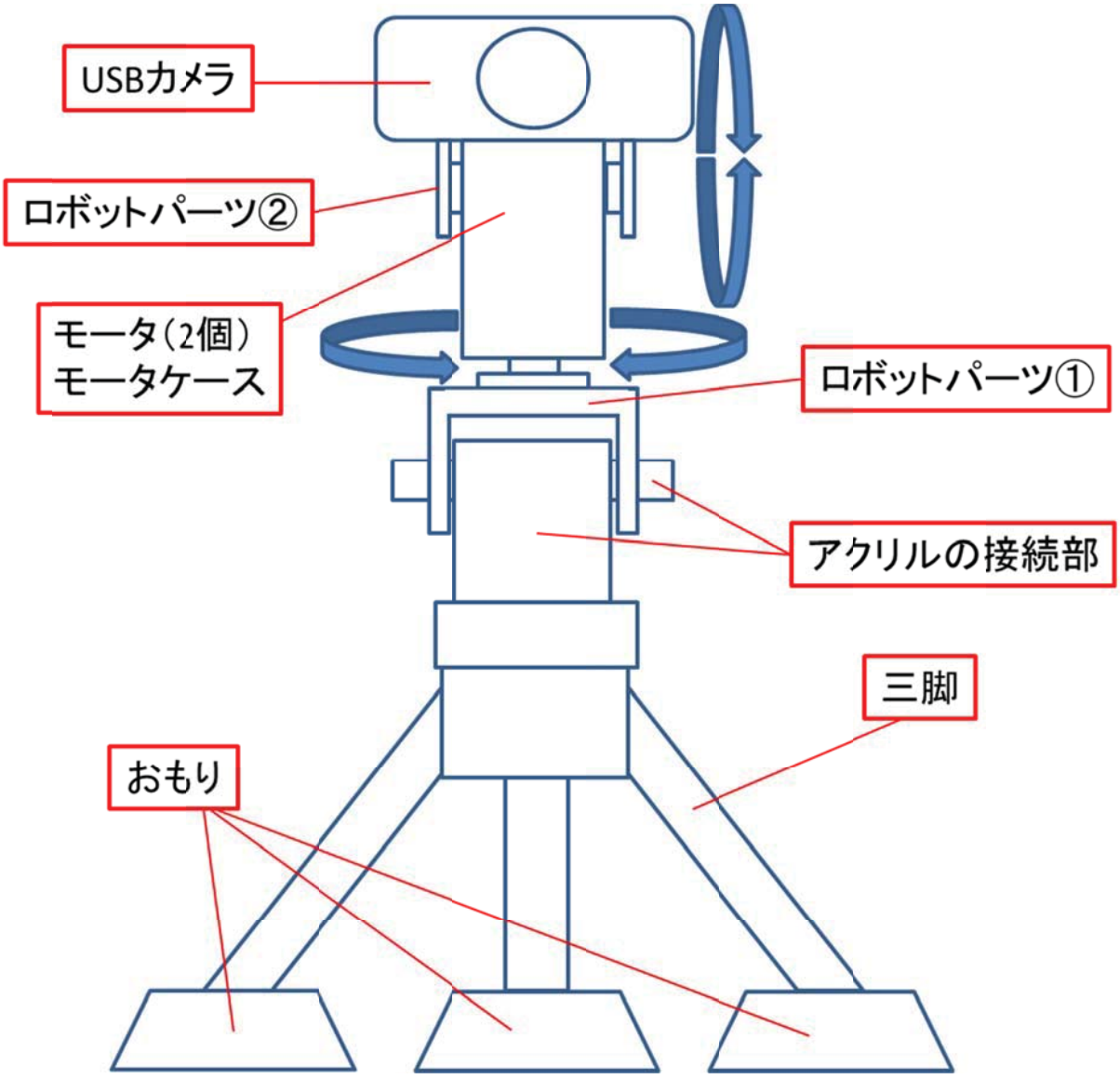


Figure IV-vii　台座の見取り図（正面）

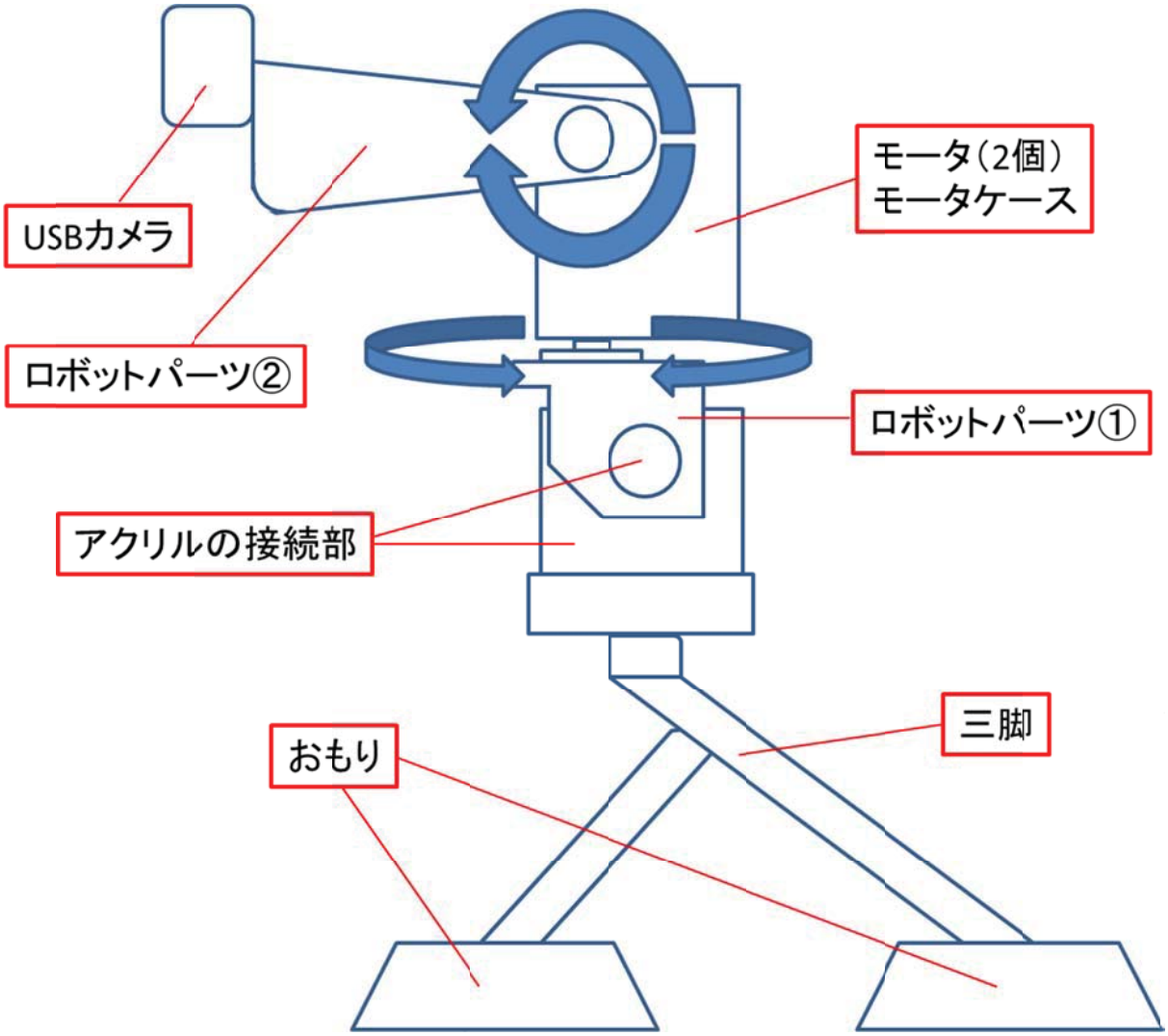


Figure IV-viii　台座の見取り図（側面）

V　RXTX ライブラリの利用

本章では Windows 上で動作する Java アプリケーションにて，シリアル通信を行う手段についてまとめる．まず，RXTX ライブラリの zip ファイルをダウンロードする．ダウンロード元は Using RXTX のウェブサイト[7] のダウンロードページより行う．Figure V-i のように画面左下の Downloads をクリックする．

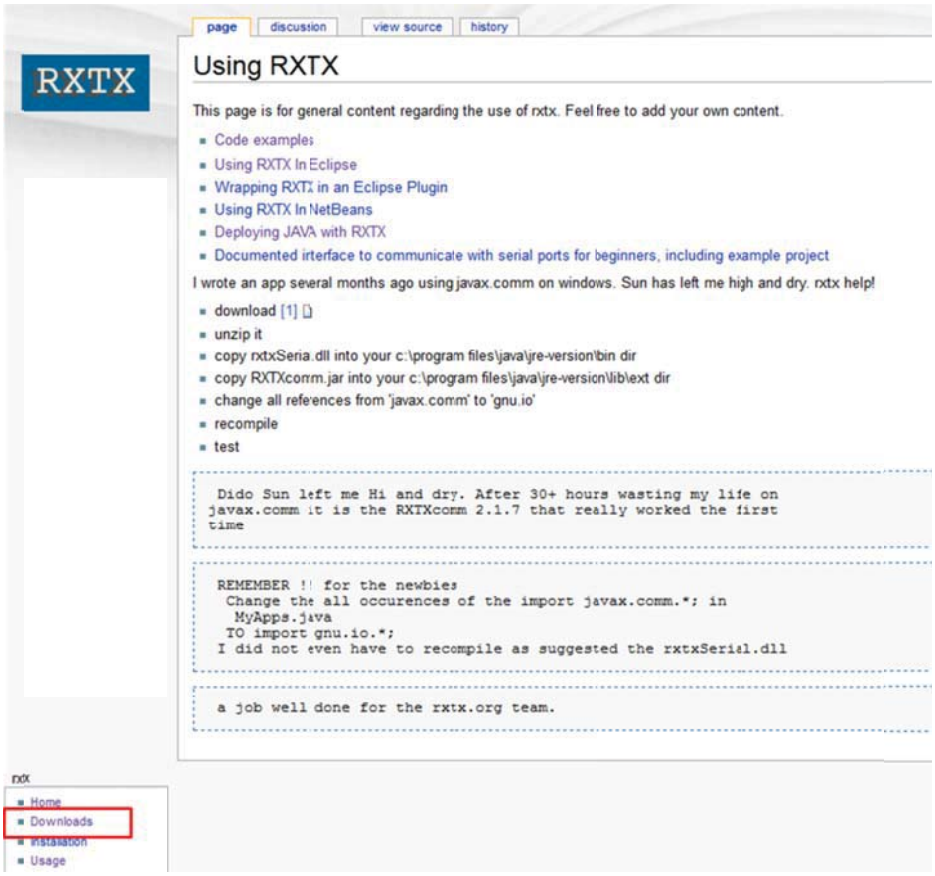


Figure V-i　Using RXTX トップページ

使用していた PC の OS が 64bit の Windows7 であったため，リンク先のダウンロードページの下部にある x64 Binaries の欄にあるリンクをクリックする（Figure V-ii）．

Figure V-ii　Downloads ページ下部，x64 Binaries リンク

リンク先のページの下部にある Downloads から Windows-x64 を選択（Figure V-iii）して ch-rxtx-2.2-20081207-win-x64.zip をダウンロードする．

Figure V-iii　zip ファイルダウンロードリンク

次に，ダウンロードした zip ファイルを解凍し，ライブラリをインストールする．解凍してできた ch-rxtx-2.2-20081207-win-x64 フォルダ内にテキストファイル以外に以下の 3 つのファイルがある．

rxtxSerial.dll
RXTXcomm.jar
RXTXcomm.jar

Install.txt を参考に, これらのファイルをそれぞれ Java のパスが通っているところにコピーする．まず，RXTXcomm.jar を C:¥Program Files¥Java¥jdk1.6.0_25¥jre¥lib¥ext にコピーする．次に，rxtxSerial.dll と RXTXcomm.jar を C:¥Program Files¥Java¥jdk1.6.0_25¥jre¥bin にコピーする．これで RXTX ライブラリのインストールが完了した．これにより，Java 言語でのシリアル通信が可能となった．

設定が完了した後，Eclipse 上で確認すると，JRE システム・ライブラリー内に先述の手順でインストールした jar ファイルが確認できる（Figure V-iv）．

Figure V-iv　RXTX 設定完了後の様子（Eclipse）

VI　Bluetooth コントローラ JS1H の利用法

本章では，本研究で利用した Bluetooth デバイスである Zeemote 製の JS1 H の利用法についてまとめる．まず Zeemote のウェブサイト[8]から Zeemote Android SDK をダウンロードする．必要事項を記入し，開発者登録を済ませれば無料でダウンロードできる．ダウンロードページで Zeemote SDK の使用許諾契約書に目を通し，ボックスにチェックを入れると，SDK のダウンロード欄が出てくるのでバージョンを選択してダウンロードする．本研究ではバージョン 1.6.0 をダウンロードしたが，今現在では最新版のバージョン 1.7.0 が登場している（Figure VI-i）．

Figure VI-i　Zeemote Android SDK のダウンロード

ダウンロードした zip ファイルを解凍すると，中に libs フォルダがある．libs フォルダの中の 4 つの jar ファイル(Figure VI-ii)を Android プロジェクトの libs フォルダ内にコピーする．

Figure VI-ii　libs ファイル内の jar ファイル

また，コピーした 4 つの jar ファイルを参照ライブラリに追加する．設定が完了すると，libs フォルダの中にライブラリが追加されているのが確認できる（Figure VI-iii）．

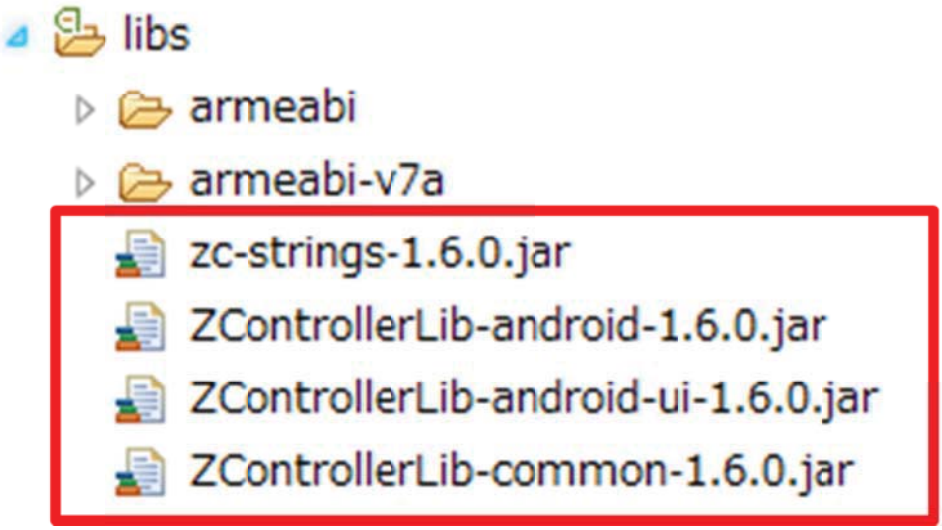


Figure VI-iii　参照ライブラリ設定完了後の様子（Eclipse）

続いて，AndroidManifest.xml ファイルの編集に入る．ファイルを開き，以下を追記する（Figure VI-iv）．

```
<uses-permission android:name="android.permission.BLUETOOTH">
</uses-permission>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN">
</uses-permission>
<activity android:name="com.zeemote.zc.ui.android.ControllerAndroidUi$Activity"/>
```

以上の変更が完了すれば，Zeemote Android SDK の設定は終了である．これにより，Android プロジェクトでライブラリを利用し，JS1 H を扱うことができるようになった．

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="jp.MotorCamera"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.BLUETOOTH"></uses-permission>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"></uses-permission>

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".test"
                  android:label="@string/app_name"
                  android:screenOrientation="landscape">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- Controller user interface. -->
        <activity android:name="com.zeemote.zc.ui.android.ControllerAndroidUi$Activity" />
    </application>
    <uses-permission android:name="android.permission.CAMERA" /><!-- check here -->
    <uses-feature android:name="android.hardware.camera" /><!-- check here -->
    <uses-feature android:name="android.hardware.camera.autofocus" /><!-- check here -->
    <uses-feature android:name="android.hardware.camera.flash" /><!-- check here -->
    <uses-sdk android:targetSdkVersion="8" android:minSdkVersion="8"></uses-sdk>
</manifest>
```

Figure VI-iv　追記した AndroidManifest.xml ファイル

参考文献

[1] android-development-environment（sola 氏によるページ）
http://code.google.com/p/android-development-environment/
[2] IT pro　第 1 回 Android の構造
http://itpro.nikkeibp.co.jp/article/COLUMN/20091127/341206/
[3] Happy my life 今さら JDK5 をインストールする方法
http://blog.cnu.jp/2011/05/31/install-jdk5/
[4] 工学院大学　電子素子・物理工学研究室（八王子）：金丸研
http://brain.cc.kogakuin.ac.jp/index-j.html
[5] 基礎から学ぶ組み込み Android 坂本俊之/出村成和/渡邊昌之 著　C&R 研究所（2011）
[6] Futaba 双葉電子工業株式会社
http://www.futaba.co.jp/product/index.html
[7] Using RXTX
http://rxtx.qbang.org/wiki/index.php/Using_RXTX
[8] Zeemote
http://www.aplix.co.jp/zeemote/jp/index.html
[9] Excel/Openoffice で学ぶフーリエ変換入門 金丸隆志 著　Softbank Creative（2011）
[10] h_kojima の日記　2011-03-29 Linux タッチパネル ドライバ（h_kojima 氏によるページ）
http://d.hatena.ne.jp/h_kojima/touch/20110329
[11]Cygwin
http://www.cygwin.com/
[12] android developers
http://developer .android. com/ sdk/ ndk/ index.html
[13] UsefullCode.net　”アンドロイド環境開発の構築（その 5）NDK のインストールと設定”
http://www.usefullcode.net/2010/12/android_sdk_inst05.html
[14] UsefullCode.net　”Android JNI プロジェクトをゼロから作る”
http://www.usefullcode.net/2010/12/android_jni.html
[15] UsefullCode.net　”Android NDK のサンプルプロジェクトをビルド/実行する”
http://www.usefullcode.net/2010/12/android_ndk_hello_jni.html

図表目次

付録図表目次

Android OS とカメラを用いた対象物追跡における処理の高速化

指導教員　金丸 隆志　准教授
AM-10053　高木 佑介

1. 緒言

Android は，スマートフォンやタブレット PC などの情報端末を主なターゲットとしたプラットフォームであり， 2007 年の発表以来注目を集めている．非常に汎用性が高く，様々なデバイスに移植可能であることと，誰でも無償で利用することができるオープンソースな OS であるということが魅力である．

また，近年 Android への注目が高まる中，それと同時に画像処理アプリケーションのニーズが高まってきている．セキュリティシステムや拡張現実への利用など，画像処理アプリケーションは現在も進歩を続けている．このことから，画像処理アプリケーションの有用性は高く，大きな可能性を秘めた分野であると言える．

本研究ではこれら 2 つを取り上げ，カメラを用いた対象物追跡を Android OS の利用法の 1 つとして提案する．対象物の追跡手段として，画像認識とモータ制御の技術を利用する．我々は Android OS を利用するにあたり，メインターゲットを組み込みデバイスとする．その理由として，Android のスマートフォン以外への利用法を提案することだけでなく，USB カメラやモータといったハードウェアは，スマートフォンでは利用するには手間がかかるということも挙げられる．このようにモータを動かし，ロボットを制御するような用途に Android の新たな利用の可能性があるのではないかと考えている．

2. 処理の高速化

処理を高速化するために，プログラムを書き換え，最も処理の速い方式を採用して対象物追跡のアプリケーションを作成する．JIT(Just-In-Time)コンパイラ ，JNI(Java Native Interface)，Neon 命令といった手法を用いて高速化を図る．

(a)JIT(Just-In-Time)コンパイラ

Java は JVM で動作させるため速度面で不利である(図 1)．そこで登場したのが JIT コンパイラである．これは，Java で用いられる実行時コンパイル機能の名称であり，中間コードから機械語への変換処理を実行直前にまとめて行う機能である．開発時にコンパイラで機械語に変換する場合とほとんど変わらない実行速度で実行できると言われている．Android ではバージョン 2.2 より搭載された．

(b) JNI(Java Native Interface)

JNI は，Java で記述されたプログラムと他の言語(C, C++)で書かれたコードを連携するためのインターフェースである．メリットとして既存のライブラリの活用ができることや，パフォーマンスの向上が挙げられる．JNI の利用法として，計算量の多いプログラムを部分的にネイティブコードに置き換えて高速化する場合の他に，ハードウェアアクセスする場合がある．特に，Java で作成したユーザーアプリケーションからでは直接ハードウェアにアクセスすることができないため，ハードウェアを利用するには JNI の利用は必須である(図 2)．

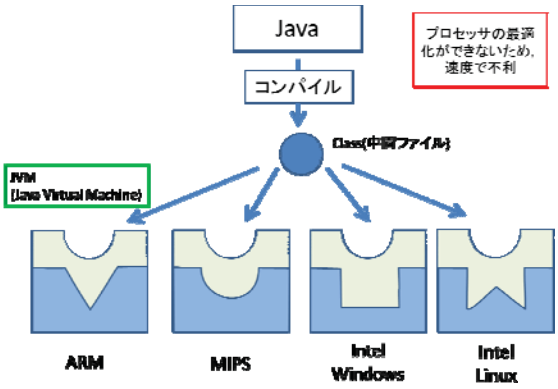


図 1　Java の仕組み

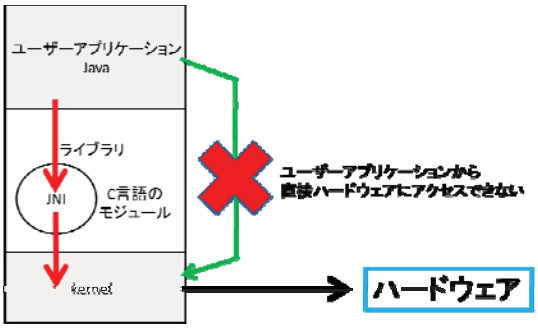


図 2　JNI によるハードウェアの利用

(c)Neon 命令

Neon 命令は ARM の SIMD（Single Instruction Multiple Data）命令セットで，1 つの命令でレジスターにセットされた複数のデータに対して一度に演算できる． ARMv6 で採用された SIMD 命令の拡張版で，新しい“ARMv7”アーキテクチャで使われる．本研究では図 3 のように 128bit レジスターを整数型 4 つ分として用いる．

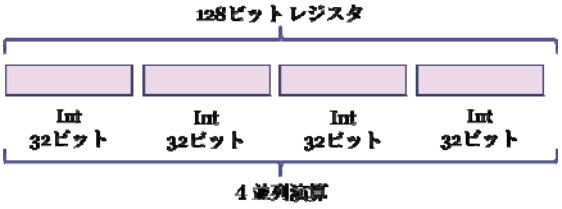


図 3　Neon 命令

3. 処理速度の検証

ここまで述べた技術を実際に画像処理に適用し，処理の高速化が図れるか検証する．本研究では次の 5 つの方法で簡単なカメラアプリケーション(図 4)を記述し，①から⑤になるにつれて速くなるという仮説のもと処理速度の比較を行った．その際に端末の性能比較も行った．対象は BeagleBoard，PandaBoard，Xperia である．

①Java のみ（JIT なし）
②Java のみ（JIT あり）
③JNI 利用：ユーザー処理部の一部を C 言語化
④JNI+Neon 利用：ユーザー処理部に ARM の Neon 命令を利用
⑤システム部でも Neon 利用

図 5 に測定結果の一部を示す．部分がシステムに依存度の高いシステム部，その他の部分をユーザー部とする．

図 4　カメラアプリケーション

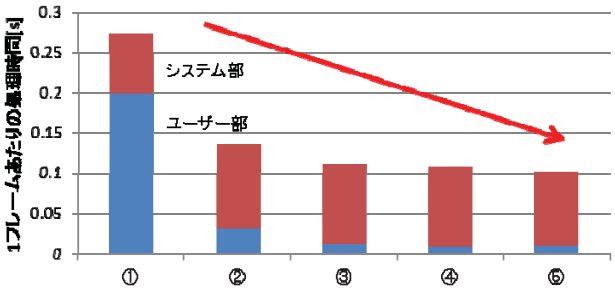


図 5　処理速度結果 I (BeagleBoard)

検証した結果，仮説の通り⑤「システム部でも Neon 利用したプログラム」が最も速いことが分かった．また，1 ピクセルあたりの計算能力を比較したところ，検証した 3 つのデバイスの中で最も計算能力が高いのは PandaBoard であることがわかった．

4. 対象物追跡

最も処理速度の速かったプログラムを利用して対象物追跡のアプリケーションを作成した．モータにより上下，左右に動く台座にカメラを取り付け，カメラで取得した画像から対象を認識し，その座標を取得する．取得した座標が画面の中央に移動するようにモータを動かし，対象を追跡する(図 6)．作成にあたり対象物を認識する手段として 2 つの方法をとった．Android に備わっているシステムを利用した顔認識を行う方法（FaceDetector）と，画像認識部を C 言語によるしたユーザー関数化する方法(テンプレートマッチングを利用)の 2 つである．また，Bluetooth デバイスにより遠隔操作も可能にした．これらのアプリケーションは BeagleBoard と PandaBoard のどちらでも動作するので，2 つの環境において処理速度の比較検証をする．図 7 に処理速度の測定結果の一部を示す．システム依存度の高いシステム部，対象物を認識する認識部，その他の部分をユーザー部とする．

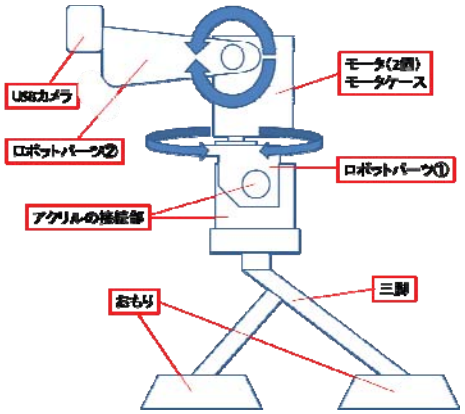


図 6　カメラの外観

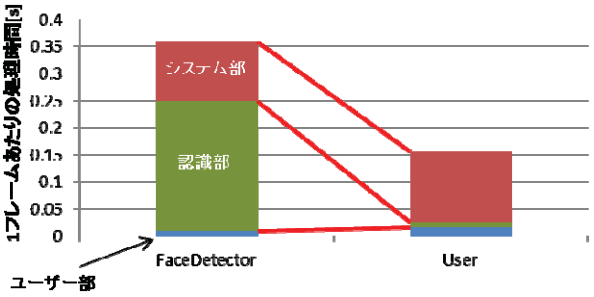


図 7　処理速度結果 II (BeagleBoard)

両デバイスにおいて，認識部をユーザー関数化することによる高速化に成功した．認識部の処理時間は BeagleBoard では 15 分の 1 に，PandaBoard では 23 分の 1 に短縮した．また，どちらのデバイスでもシステム全体にかかる時間は約半分になった．ユーザー部の処理時間が長くなっているのは，認識の際に一度小さくした画像をもとのサイズに戻して表示する工程のためだと考える．

5. 結言

カメラを利用した対象物追跡アプリケーションを作成した．また，その際に画像処理における処理の高速化に成功した．

画像処理の高速化手法は，「システム部でも Neon を利用したプログラム」が最も速いことがわかった．この結果に基づき，対象物追跡アプリケーションを作成した．それぞれ Android のシステムを利用した認識の手法と，テンプレートマッチングを利用した認識の手法を利用した 2 つのアプリケーションを作成した．システム関数を利用した認識処理を，認識部をユーザー関数化することで高速化に成功した．

最後に，処理速度向上のためのさらなるプログラムの改善を今後の課題とし，Android OS の利用の場を広げるためにもハードウェアの発展にも期待したいと考える．