修 士 論 文

手のジェスチャーを認識して駆動する外部装置の構築

Composition of Remote Control System Driven by Hand-Gesture

報 告 者

学籍番号:1205069

氏名:　　豊島　翔

指 導 教 員

綿森　道夫

平成 30 年 2 月 12 日

高知工科大学　大学院工学研究科電子・光システム工学コース

目次

第一章　　序論

1-1　研究目的

映画やアニメの中で、PC 等を手のジェスチャーだけで操作しているような場面を見たことが一度はあるのではないだろうか。最近では PC を手のジェスチャーだけで操作するというのは難しいことではなく、昨今ではデバイス（Leap Motion や Kinnect）さえ購入すれば誰もが体験できるようになっている。また、その直感的な操作方法は PC の操作だけでなく、アプリケーションでも多く見られるようになってきた。これは、無料のソフトで気軽にアプリケーションの開発ができ、尚且つ PC だけでなくスマートフォンの普及に伴ってアプリケーションを利用する人数が増加したからだと考えることができる。また、言葉の壁がある場合でも手の動き、体の動き（身振り手振り）は通じるといったことがあるように、動きでこちらの意図を理解させたい。例えば手話における手の動きを認識して翻訳したいということもあるだろう。翻訳機の様なモノまではいかなくても、定めた手の動きを認識して動作するプログラム、装置を開発することができれば、将来的に手話の正確な認識を行う足掛かりになるのではないか。

Leap Motion を用いたアプリケーションを自分で開発するとなると、開発環境の準備、外部装置の用意、装置との通信方法など乗り越えなければならない問題があることも、また事実である[1]。本研究においてはこれらを一通り勉強し、自分でアプリケーションの開発ができるようになったので、その過程を説明していきたいと思う。

また、外部装置として PIC16F688 と TWE-Lite を利用した装置を用いることにした[2]。PIC16F688 を選んだ理由としては、Leap Motion で得た情報をシリアル通信で送ったり、サーボモーターの制御、DC モーターの制御、LED の点灯、無線化などに必要な機能が備わっており、ピン数もそれほど多くなく設計しやすい、値段的にも高価なものではなく用意しやすいということから選択した。

PIC16F688 と Leap Motion のシリアル通信のやり取りは TWE-Lite を用いて行った[3]。これを選択した理由としては USB 接続ポートがあれば MoNoStick などを用いることによって容易に無線化することができるからである。

本研究では上で述べたような、手や指の骨格を検出できる Leap Motion を用いて指や手の動きをプログラムで判断し、いくつかの簡単な手話を検出して表示したり、離れた場所の端末にデータを送り、駆動するシステムの構築を目的とした。

1-2　特別研究を通して身についた事

本研究を通して以下の事が身についたと考える。

第一に Leap Motion を用いたプログラム開発へのより深い理解である。学士課程での研究では、Leap Motion のプログラム開発には cinder で用意されている Frame Work を利用していたが、本研究ではその FrameWork を用いずに基本からまたプログラム開発をやり直したため、Leap Motion で提供されているクラスの使用方法などより詳しく理解でき、より自由にプログラム開発ができるようになった。また Leap Motion から得た情報を描画する手段として OpenGL を選択したので OpenGL との連携方法も学んだ[4,5]。

第二に TWE-Lite を用いた無線通信の方法である。Arduino Uno を用いていた時は有線で接続をしてシリアル通信を行っていたが[6]、最終的に無線でシリアル通信をするという目的があった。Mono stick や TWE-Lite を使用して手軽に無線化できるということもあり、このことが無線化の方法に TWE-Lite を選んだ大きな理由である。TWE-Lite には様々なアプリが用意されており、その中のシリアル通信専用アプリの透過モードという生の情報を送るモードを使用してシリアル通信を行った[4]。詳しい説明は第三章で行う。

以上の二つが本研究の過程で身についた主な技術であり、その結果として本論文の完成に至った。

1-3　本研究の新規性について

Leap Motion を利用するためのプログラムについては、公表されているものが少なく、ほとんど 1～2 冊の書籍と、Leap Motion 社や Cinder 社がインターネット上で示しているサンプル例程度しか見つけられない[1]。また Leap Motion を用いて指文字の認識や手話の認識を試みているものは[7,8]のようにいくつか見つけることはできたが、実際の C 言語を用いてのプログラミング内容というのは見つけることができなかった。又、Arduino 用ライブラリーや Unity 用の Asset が存在するので、個人のプログラマはそれらのパッケージを利用してプログラムを開発することになる。しかしながらこれらのプログラムツールは、現代風のオブジェクト指向の概念で記述されており、それぞれの機能を各クラスの中にカプセル化して内包することによってクラスごとに分散して格納されている。その為、一種のテンプレートプログラムの様にいろいろな機能を付け加えるときには、必要なクラスの指定した場所を書き換えるという、まさにオブジェクト指向のプログラミング方法をとりざるをえない。確かに新たにプログラムを開発しようとするものは、継承を利用して開発すれば、先人のソースコードすら不要となり、秘匿性と開発効率の向上という両方が得られる。しかしながらこれではプログラム全体としては一体何をしているのかという見通しが全くたたなくなり、それこそ制御に関する大切な部分が隠ぺいされてしまう。これはオブジェクト指向言語の本質に関係する問題であり、知的所有権保護の観点からも大切なことである。しかしながら今回の様におよそ想定されていない拡張、すなわち Leap Motion の出力結果を別の外部装置に転送するなどといった場合にはどこを修正していいのか分からなくなる。そこで Leap Motion を利用したプログラムを従来の C 言語スタイルの構造化プログラミングスタイルで開発することにした。この方法によるプログラミングを他書やインターネットから見つけることができず、著者のオリジナルであり、1つ1つ動作を実際に確かめながらプロ

グラムを構築した。そしてこの作業をくみ上げていくことによって、Leap Motion の利用方法が明らかになっていったと考える。これが本研究における最大の新規性であろう。明確になった Leap Motion の動作プロセスを組み合わせて手話に相当する複数の手の動きを判断することが可能になった。Leap Motion は素晴らしいデバイスで、今回示した手話よりもより複雑な動作であっても比較的簡単に検出できるようになると思う。又、無線制御の外部デバイスの ON/OFF 等も今回の研究で示したので、手話によって自動でドアが開いたり、対応する音声アナウンスなどができるようになるのではないかと考える。

第二章　　LeapMotion の機構を Windows プログラムに組み込む

2-1　　LeapMotion とは

LeapMotion は 2012 年に LeapMotion 社から発売された、手のジェスチャーによる PC 操作を可能にする入力デバイスであり、マウスや、タッチパネルを使用せずに体感的、直感的に操作することができる。1/100mm 間隔で動作を認識することができる非常に高精度な三次元入力デバイスと言える[1]。図 2-1 に Leap Motion の外形を示す。

Leap Motion 自体は PC の手前に置く様に設計された小型の USB 周辺装置であり、二基の赤外線カメラと三つの赤外線照射 LED から構成されている。また、裏面にはグリップカバーが付いてあり、置いた時に滑らないようになっている。赤外線 LED に照らされた指、手を二基の赤外線カメラで撮影し、画像解析から指や手の三次元座標を割り出している。検知できる範囲としては半径 50cm ほど、中心角 110 度の空間で認識する。Leap Motion では両手とそれぞれに付随する指十本をそれぞれ別々に判別することが出来、指や手の動きを PC 上で再現したり、指やジェスチャーの軌跡を表示させたりすることができる。[1]また、Leap Motion の最新のバージョンでは初期のバージョンに比べて格段に指や手の追従性が向上しており、更に使いやすいデバイスとなった。本研究で使用した時点での最新のバージョンは ver.2.0 であった。Leap Motion の外観を図 2-1 に示す。Leap Motion の xyz 座標の正負は図 2-2 のようになっているので、3次元空間を考えるにはたえずこの座標を意識をしなければならない。

図 2-1 Leap Motion

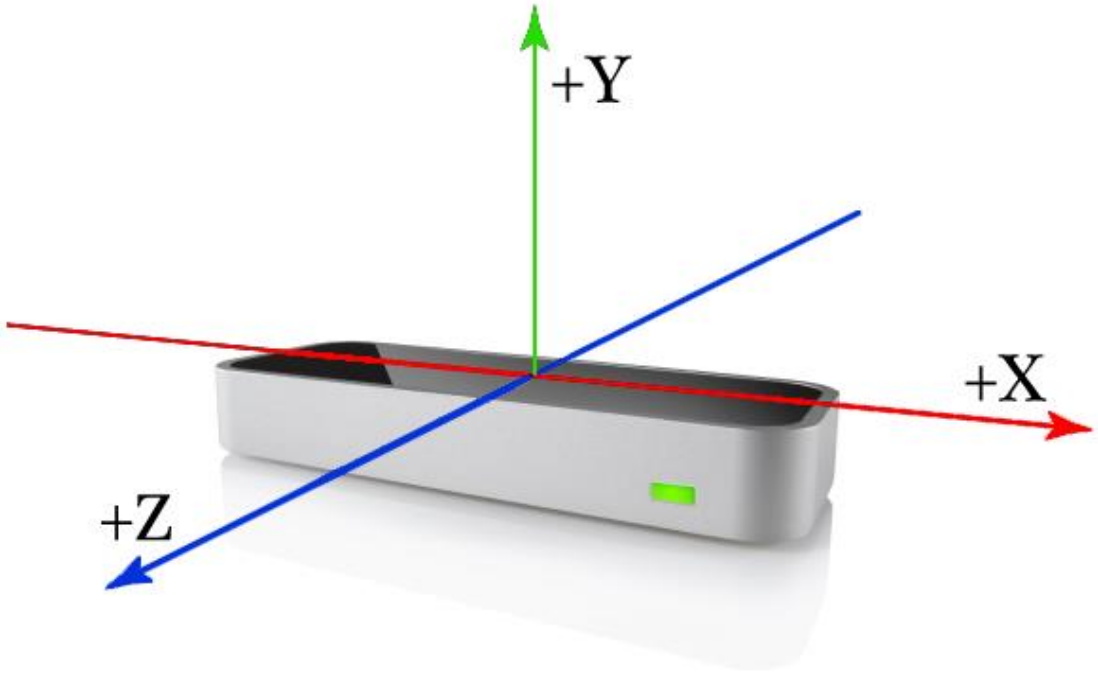


図 2-2　Leap Motion の XYZ 座標([9])

2-2　Leap Motion でアプリケーションを開発するための準備

最初に Leap Motion の公式サイト[10]からインストーラーをダウンロードしてくる。後は指示に従ってインストールを完了させる。次に、開発環境を整えるために Leap Motion の公式サイト[11]より無料で配布されている Leap Motion SDK をダウンロードしてくる。ここで注意しなければならないのはダウンロードは無償だが、ダウンロードするためにはこれもまた無償のサインアップをする必要がある。ダウンロードしてそのフォルダーを解凍すると Leap SDK というフォルダがあるので、それを適当なところに移す。筆者は C ドライブに移した。

次にこの章では Visual Studio 2013 を用いて開発していくので、Win32 アプリケーションで空のプロジェクトを作成する。Visual Studio 2013 において初期設定が必要である。

「すべての構成」の構成プロパティタブを開き、C/C++の追加のインクルードディレクトリで SDK の中の include フォルダを指定する。次にリンカーの追加のライブラリディレクトリで SDK の中の lib フォルダにある x86 フォルダを指定する。次に同じくリンカーの中の入力にある追加の依存ファイルで leap.lib を入力する。最後に Leap.dll へのパスを通すか、実行ファイルと同じ階層に Leap.dll を置く。これで環境が整った。

2-3　　LeapSDK について

ダウンロードした LeapSDK には LeapMotion の制御に必要となる関数がいくつかのクラスに分けて示されている。その内、特に重要であるのは Frame クラスで、このクラスの中に取得した手や指などの座標などの情報が入っている。ここでは今回のプログラム開発において比較的重要であったクラスについてのべる[1]。

2-3-1 Controller クラスで提供されている関数

Controlle クラスでは addListener()関数と removeListener()関数を利用して Listener クラスの変数を登録、削除すれば、イベント通知方式でプログラムを組むことができる。今回は定期的に TWE-Lite をアクセスする必要が生じたため、ポーリング方式でプログラムを組んだ。その為、これらの2つの関数は使っていない。ただし、Leap Motion を USB に接続したことを知るために isConnected()関数を利用した。また、ポリシー変更、ジェスチャー検出許可にも使用した。

・frame()・・・・フレームの取得をする関数。
・policyFlags()・・・動作ポリシーを取得する関数。
・setPolicyFlags()・・動作ポリシーを設定する関数。
・hasFocus()・・・・アプリケーションのフォーカス状態を取得する関数。
・addListener()・・・Leap::Listener を登録する関数。
・removeListener()・・Leap::Listener を削除する関数。
・config()・・・・設定に関する Leap::Controller クラスのインスタンスを取得する関数。
・enableGesture()・・・ジェスチャーの有効と無効の設定をする関数。
・isGestureEnabled()・・ジェスチャーの有効と無効の状態を取得する関数。
・isConnected()・・LeapMotion コントローラーの接続状態を取得する関数。
・isServiceConnected()・LeapMotion サービスの接続状態を取得する関数。
・devices()・・・・デバイスのリストを取得する関数。

2-3-2 Frame クラスで提供されている関数

Frame クラスで手や指、ジェスチャーなどを検出し、座標を OpenGL に渡す。本プログラムにおいて最も重要なクラスである。また、ポーリングによってプログラムを構築したので、前のデータを取得したフレームから更新されたかどうかを知る必要があり、そのためにこの id()関数を利用した。

・hand()・・・・指定した ID の手を取得する関数。
・hands()・・・このフレームで検出した手の一覧を取得する関数。
・pointable()・・指定した ID の指、ツールを取得する関数。
・pointables()・・このフレームで検出した指、ツールの一覧を取得する関数。
・finger()・・・・指定した ID の指を取得する関数。

・fingers()・・・このフレームで検出した指の一覧を取得する関数。
・tool()・・・・・指定した ID のツールを取得する関数。
・tools()・・・・・このフレームで検出したツールの一覧を取得する関数。
・gesture()・・・・指定したジェスチャーの ID を取得する関数。
・gestures()・・・指定したフレーム、もしくはこのフレームとの間で検出したジェスチャーの一覧を取得する関数。
・rotationAngle()・・指定したフレームとの間の回転角度を取得する関数。
・rotationAxis()・・指定したフレームとの間の回転軸を取得する関数。
・rotationMatrix()・・指定したフレームとの間の回転量を取得する関数。
・rotationProbability()・・回転が意図した動作かどうかを取得する関数。
・scaleFactor()・・・指定したっフレームとの間の拡大量を取得する関数。
・scaleProbability()・・拡大が意図した動作かどうかを取得する関数。
・translation()・・・指定したフレームとの間の移動量を取得する関数。
・translationProbability()・・移動が意図した動作かどうかを取得する関数。
・id()・・・・・・フレームの ID を取得する関数。
・currentFramesPerSecond()・・・現在のフレームレートを取得する関数。
・timestamp()・・・・LeapMotion が動作を開始してからの経過時間を取得する関数。
・isVaild()・・・・・フレームの有効、無効状態を取得する関数。
・interactionBox()・・InteractionBox クラスのインスタンスを取得する関数。

interactionBox()は Leap Motion の逆ピラミッドの座標系を直方体の座標系に変換するものであり、直方体の左下奥を(0,0,0)として 0 から 1 までの範囲で値を変化させるので、この値にウィンドウの幅や高さを掛けることによって指がウィンドウ上のどの位置にあるかを検出することができる。

2-3-3 Listener クラスで提供されている関数

Listener クラスは、Controller クラスと併用して、フレームが更新されたことを知るためのクラスである。初期のプログラムにおいてはこのクラスを用いてイベントリスナーに登録する方法を用いたが、TWE-Lite との併用において難が生じたので、結局利用することはなくなった。

・onInit()・・・・・リスナーの初期化処理を行う関数。
・onConnect()・・・Leap::Controlle クラスが Leap Motion と接続されたときに呼び出される関数。
・onDisconnect()・・Leap::Controlle クラスが切断されたときに呼び出される関数。
・onExit()・・・・・リスナーの終了処理を行う関数。
・onFrame()・・・・フレームのデータが更新されたときに呼び出される関数。
・onFocusGained() & onFocusLost()・・アプリケーションがアクティブ、非アクティブになったときに呼び出される関数。

・onDeviceChange()・・Leap Motion コントローラーの接続、切断が通知される関数。
・onServiceConnect() & onServiceDisconnect()・・Windows サービスとの接続、切断が通知される関数。

何らかの要因で Leap Motion サービスが停止してしまった時は、管理者権限でコマンドプロンプトを実行し、net start LeapService と打ち込み、手動でスタートさせる必要がある。Leap Motion の接続状態に関しては、Leap Motion コントローラーとアプリケーションが接続されていて、アプリケーションと Leap Motion サービスが接続されているという状態を意味している。onConnect や onServiceConnect は Leap::Controller クラスの状態と関連している。

第三章　　TWE-Lite とは

TWE-Lite というのは無線機能が備わったマイコンであり、TWE-Lite DIP と呼ばれるものと MoNoStick とよばれるものがある。TWE-Lite DIP はいわゆる一般的な IC の形にしたものであり、そのまま回路に組み込むことで簡単に無線化することができる。MoNoStick は USB メモリぐらいの大きさの USB 接続を可能にしたもので PC やスマートフォンなどで無線操作が可能になる。図 3-1 に TWE-Lite DIP の外観を、図 3-2 に MoNoStick の外観を示す。

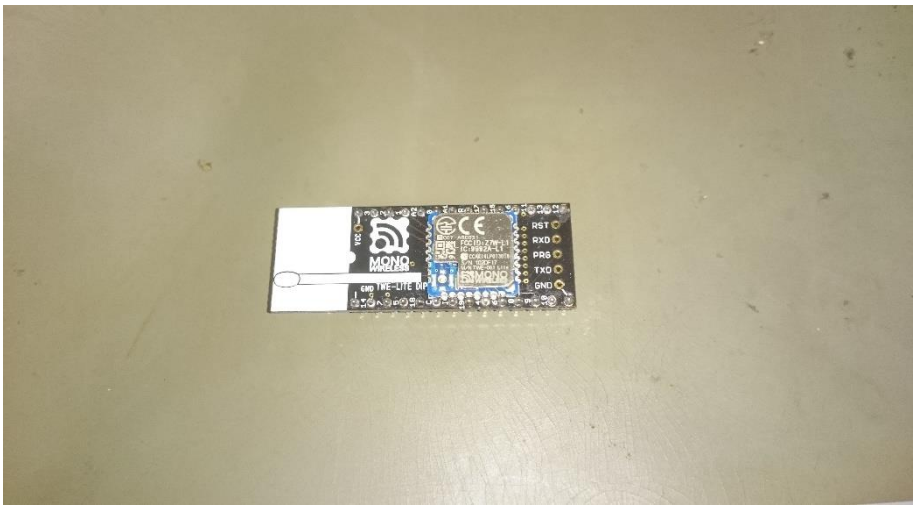


図 3-1 TWE-Lite DIP

図 3-2　MoNoStick

公式サイト [2]によるとTWE-Liteは、グローバル周波数で波長が短くアンテナを小型化可能な2.4GHz帯無線を採用しており、通信速度は250kbpsである。2.4GHz帯無線は日本国内に限らず世界中で使用可能となっている。また、省電力でもあり、2.3V～3.6Vの電圧で動作し、コイン電池で年単位の動作が可能となっている。TWE-Liteは32ビットのRISCマイコンを内蔵していて、UART、SPI、I2C、ADC、その他の豊富なインターフェースを使用可能であることで外部に制御用のマイコンが不要で、小型化や簡素化に繋がっている。日本国内の電波法認証（技適）も取得済みで免許無しで使用可能となっている。

また、TWE-Liteには「超簡単！TWE標準アプリ」というのがインストールされているのだが、他にもシリアル通信専用アプリやリモコン通信専用アプリなどが用意されていて、そちらにプログラムを書き換えることもできる。ただしプログラムを書き換えると各ピンの機能も変わるので注意する必要がある。電源の＋のピンと電源のーのピンはどのアプリでも共通である。以下に提供されているアプリをいくつか紹介する[3]。

Ⅰ.超簡単！TWE標準アプリ

工場出荷時に標準でインストールされている。

Ⅱ.シリアル通信専用アプリ

シリアル通信に長けたプログラム。Ⅰのプログラムではシリアルに I/O ポートの状態が刻々と出力されるようになっている為に自由な通信ができない。しかしこのプログラムでは、シリアルに入力されたデータを親子間でそのまま送受信できる。

Ⅲ.リモコン通信専用アプリ

アナログ入出力やI2C制御ピンの機能を無くして、これらのピン全てをデジタル入出力とすることで、12ピン分のON/OFF制御ができる。

Ⅳ.無線タグ

省電力の無線タグ用プログラム。各種センサーやスイッチの状態を、子機が低電力で送信して、親機側でデータ収集できる。

Ⅴ.オーディオ・信号通信専用アプリ

音声トランシーバーを実現するプログラム。最大8kHzのサンプリングレートでアナログ入力して、ADPCM圧縮して相手に届け、PWM出力する。音声専用CODEC ICを使用することなく、オペアンプなどの普通の部品で実現できる。

これらのプログラムをTWE-Lite DIPやMoNoStickに書き込む方法を説明する。

TWE-Lite DIPのプログラムを書き換えるにはTWE-Lite-R（トワイライター）という別の部品が必要になる。図3-3にTWE-Lite-Rの外観を示す。TWE-Lite-RにTWE-Lite DIPを取り付け、パソコンに接続するとCOMポートとして割り当てられるので、パソコン側に書き込むプログラムを公式サイトからダウンロードして用意し、専用の書き込みソフト「TWE-Liteプログラマ」を使用してプログラムを書き込むことができる。TWE-Lite DIPにはTWE-Lite-Rが必要であるが、DIPと同じ0.1インチ間隔でピンが外に出ているので、直接電子工作に組み込むことができる。一方で

MoNoStick は直接 USB に差し込んで内部プログラムを書き換えることができるが、信号線がピンの形で外部に出ておらず、電子工作には使いにくい。そこで本研究においては Leap Motion からのデータをパソコンで取得後に TWE-Lite に送るときには USB を経由して MoNoStick を用い、無線で伝送されたデータを受け取り、外部装置を動かす時には TWE-Lite DIP を電子工作に組み込んで利用することにした。

図 3-3 TWE-Lite-R

なお、MoNoStick にプログラムを書き込む際は、MoNoStick を直接パソコンに接続すれば COM ポートとして認識されるので、後は前述した TWE-Lite プログラマで MoNoStick が接続されている COM ポートを選択して書き込めばよい。

第四章　　Leap Motion を制御するプログラムについて

第二章では Leap Motion で用意されてある主なクラスについて紹介した。本章ではそれらを用いて実際に作成したプログラムを紹介していく。最初に初期変数宣言と Leap Motion との接続を確認する部分について図 4-1 に示す。

```
int64_t idnum_old = 0,idnum;
Leap::Controller controller;
Leap::Frame frame;
Leap::FingerList fingers;
Leap::Finger fing;
Leap::Hand hand;
Leap::HandList hands;
///
void main(int argc, char *argv[])
{
std::cout << "waiting for leapmotion setting " << controller.isConnected() << std::endl;
while (!controller.isConnected()){}
std::cout << "LeapMotion is ready to use!" << std::endl;
std::cout << "Press Enter to quit..." << std::endl;
std::cin.get();
}
```

図 4-1　共通部分

フレームが更新されたかどうかを知るための変数と手や指の検出で用いる変数をそれぞれのクラスの型で宣言している。また、main の共通部分としては、コントローラークラスから Leap Motion の接続状態を確認して、接続されていなければ接続されるまで待つという処理を行っている。すべてのプログラムはこの共通部分を使用している。

4-1　　指を検出

図 4-2 に握っていない状態、つまり開いている状態で手の指を数えてその本数を表示するプログラムの一部を示す。

```
Leap::Controller controller;
Leap::Frame frame;
Leap::FingerList fingers;
Leap::Finger fing;
                frame = controller.frame();
                idnum = frame.id();
                if (idnum > idnum_old){
                        idnum_old = idnum;
                        fingers = frame.fingers();
                        fingnum = fingers.count();
                        fc = 0;
                        for (i = 0; i < fingnum; i++){
                                fing = fingers[i];
                                if (fing.isExtended())fc++;}
                        std::cout << "Opened finger nums = " << fc << std::endl;
```

図 4-2 指を検出するプログラム

これから紹介していくプログラムと共通している点としては、Controller クラスからフレームを取得して、そのフレームの ID を用意した変数に入れておき、フレームの ID が前回のフレームより大きいとき、つまりフレームが更新されているならば、今のフレーム ID を idnum_old に格納しておき、様々な処理を行うようになっているところである。指の検出部分は、まず fingers にフレームから得られる指の一覧を与え、fingnum には検出した指の本数を与えている。次に検出した指の本数を Finger クラスの fing に与えてそれぞれの指が開いている状態であるならば fc の数を増やしている。認識した指の描画は OpenGL を用いて行っている[4][5]。特定の指を検出する方法は簡単で、上記のプログラムで言うと frame.fingers()の後に fingerType(Finger.FingerType.取得したい指)をつけるだけである。親指なら TYPE_THUMB、人差し指なら TYPE_INDEX、中指なら TYPE_MIDDLE、薬指なら TYPE_RING、小指なら TYPE_PINKY である。こうして指を認識している様子を図 4-3 に示す。

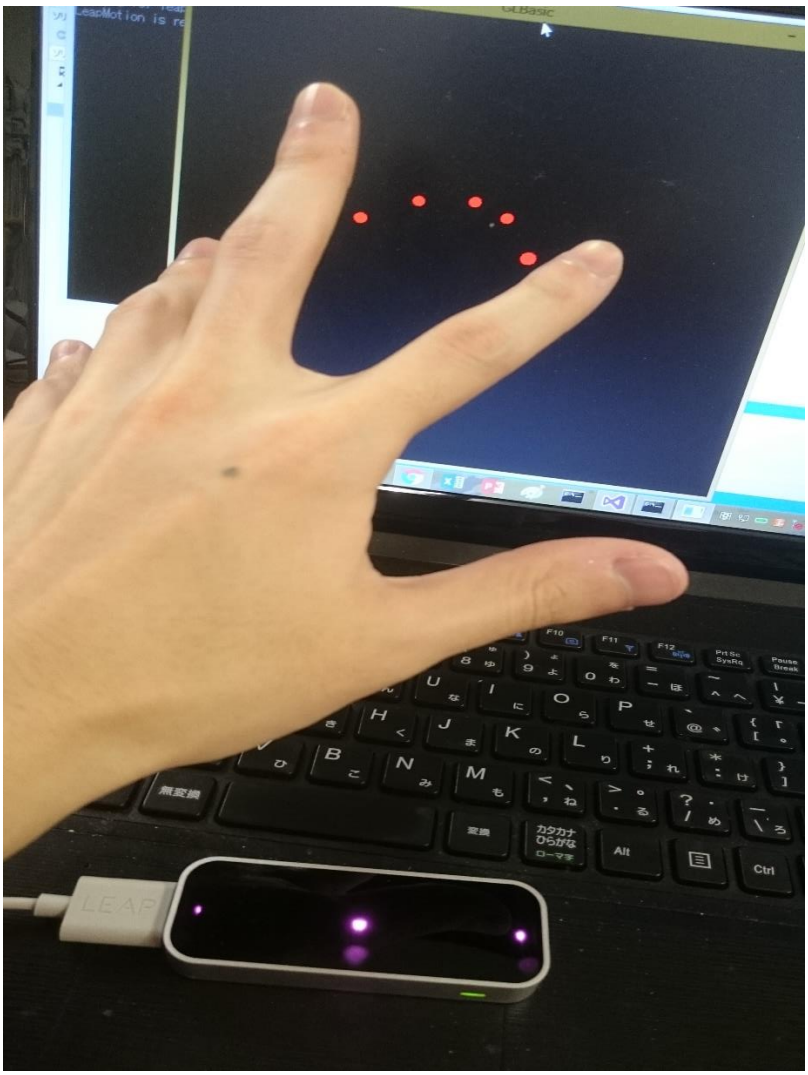


図 4-3　指を認識している様子

4-2　　手の検出

手を検出して、検出したその手が左手か右手かを判別するプログラムの一部を図 4-4に示す。手を検出するプログラムが指を検出するプログラムと共通している点は省いている。また、処理していることも指の検出とほとんど同じであり、まずフレームから手の一覧を取得し、その一覧より手の認識している数をカウントで取得し、取得した分の手の数だけの情報を取得する。そして認識した手が左手なら hand.isRight()が、右手なら hand.isLeft()が1になるので、それぞれで変数の値を変えて出力している。図 4-5 に指を検出している様子を示す。

```
Leap::Hand hand;
Leap::HandList hands;
                    fc = 0;
                    pc = 0;
                    hands = frame.hands();
                    hcount = hands.count();
                    for (i = 0; i < hcount; i++){
                            hand = hands[i];
                            if (hand.isRight())pc = 1;
                            if (hand.isLeft())pc = 0;
                    }
                    std::cout << "hand count = " << hcount << std::endl;
                    if (pc == 1)std::cout << "Right Hand " << std::endl;
                    if( pc == 0 ) std::cout << "Left Hand" << std::endl;
```

図 4-4　手の検出

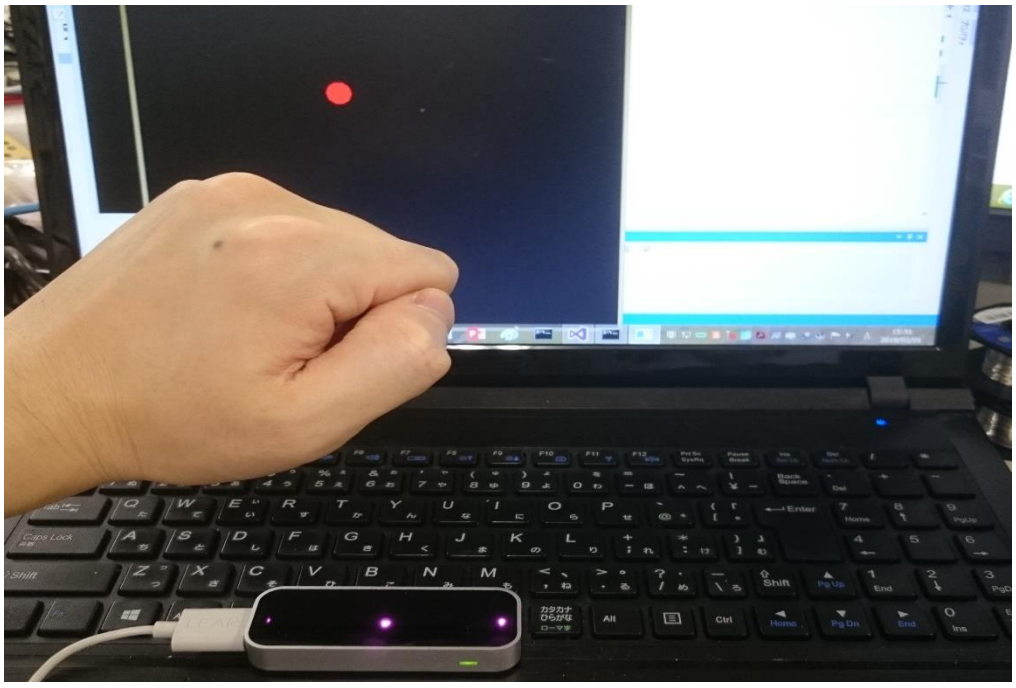


図 4-5　検出の様子

4-3　　ジェスチャーの検出

図 4-7にジェスチャーを検出するプログラムの一部示す。

まずジェスチャーを認識できるようにするには

controller.enableGesture(Gesture::Type::認識させたいジェスチャー);

という1行をプログラム中に記述しておく必要がある。スワイプジェスチャーを認識させる場合は TYPE_SWIPE、サークルジェスチャーを認識させたい場合は TYPE_CIRCLE を、スクリーンタップジェスチャーを認識させる場合は TYPE_SCREEN_TAP、キータップジェスチャーを認識させたい場合は TYPE_KEY_TAP と記述する。これにより記述したジェスチャーが有効になり、Leap Motion で認識することができるようになる。プログラム自体は手や指の検出のプログラムの応用で、まずフレームの更新ができればジェスチャーの一覧を取得することができる。次に認識したジェスチャーの数をカウントで取得して、検出した数だけジェスチャー型の gesture に入れる。そして認識したジェスチャーが本当にスワイプジェスチャーならば SwipeGesture 型の swipe に gesture を渡す。プログラムでは好みになるがすべての指のジェスチャーを認識すると繁雑になるので人差し指のスワイプだけ取得するようにしている。後半は指や手の検出と同様であり、ここでは左手のスワイプジェスチャーを認識したら出力するようにしている。ほかのジェスチャーを検出したい場合は前述した様に、各ジェスチャー型の変数に得られた gesture 変数の内容をコピーすることで動作するようになるので、これをプログラムに記述し、プログラム内の SwipeGetue の部分を対応するジェスチャーに変更することで認識するジェスチャーを変更可能である。図 4-6 に検出の様子を示す。

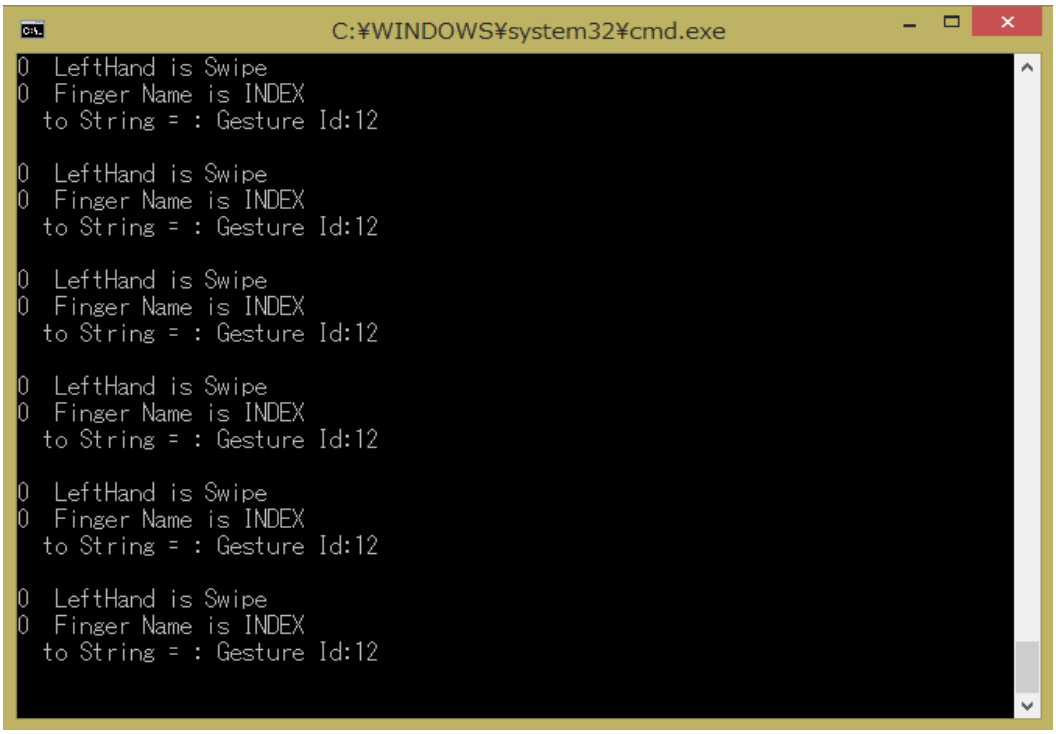


図 4-6　出力の様子

```
if (idnum > idnum_old){
                frame_old = controller.frame(idnum - idnum_old);
                idnum_old = idnum;
                if (frame_old.isValid()){
                        gestures = frame.gestures(frame_old);
                }
                else{
                        gestures = frame.gestures();
                }
                count = gestures.count();
                for (i = 0; i < count; i++){
                        gesture = gestures[i];
if(gesture.type()==Gesture::Type::TYPE_SWIPE){
SwipeGesture swipe(gesture);
fingers=frame.fingers().extended().fingerType(Leap::Finger::Type::TYPE_INDEX);
fcount = fingers.count();
for (n = 0; n < fcount; n++){
        finger = fingers[n];
        if (finger.hand().isLeft()){
                std::cout << n << "    LeftHand is Swipe " << std::endl;
        std::cout << n << "    Finger Name is INDEX " << std::endl;
    std::cout << "    to String = : " << swipe.toString() << "¥n" << std::endl;
                                                }
                                        }
```

図 4-7　ジェスチャーを認識するプログラムの一部

4-4　　モーションの組み合わせ

モーションの組み合わせは、先ほどのジェスチャープログラムの応用である。例えば左右のスワイプジェスチャーのうち、左から右にスワイプするものと、上下のスワイプジェスチャーのうち、下から上へスワイプするもの、以下右から左、上から下などのジェスチャーを順番に検知して組み合わせれば、1周するモーションになる。手話の解釈に関してはこのような考え方でプログラムを構築した。

```
for (n = 0; n < fcount2; n++){
finger2 = fingers2[n];
                    x = swipe.startPosition().x;
                    y = swipe.startPosition().y;
                    z = swipe.startPosition().z;
                    xp = swipe.position().x;
                    yp = swipe.position().y;
                    zp = swipe.position().z;
                    dx = swipe.direction().x;
                    dy = swipe.direction().y;
                    dz = swipe.direction().z;
                    dr = dx / sqrt(dx*dx + dy*dy + dz*dz);
                    dl = -dx / sqrt(dx*dx + dy*dy + dz*dz);
                    dup = dy / sqrt(dx*dx + dy*dy + dz*dz);

          dist = sqrt((xp - x)*(xp - x) + (yp - y)*(yp - y) + (zp - z)*(zp - z));
          ddown = -dy / sqrt(dx*dx + dy*dy + dz*dz);
glBegin(GL_LINES);
glVertex3f(x, y, z);
glVertex3f(xp, yp, zp);
glEnd();
glRasterPos3d(-100.0, -30.0, 0.0);

          if (dr > 0.96 && dist > 150 ){
                    state = 1;
          }if (ddown > 0.96){
                    kon = 1;
          }if (state == 1 && dl > 0.97 ){
                    dou = 1;
          }
```

図 4-8　モーション組み合わせプログラムの一部

図 4-8 にモーションを組み合わせたプログラムの一部を示す。ジェスチャープログラムの所で紹介したスワイプジェスチャーを応用すればある特定の動きを検出することも可能ではないかと思い、スワイプでも右への動き左への動き、上下の動きというのを考慮してみることにした。Swipe.position でスワイプを検出した後の座標を取得し、swipe.startPosition でスワイプジェスチャーの開始位置の座標を取得し、swaipe.direction でスワイプジェスチャーの方向ベクトルを取得する。それらの情報から左右への動き量、上下の動き量を算出することによってスワイプジェスチャーがどの方向にどれくらい動いたかというのを得ることができるようになった。そして「どういたしまして」の手話は目の前で手を左右に一往復移動させるというのものだったので、スワイプジェスチャーの右方向への動きを認識したら state を 1 として、state が 1 の状態で左の動きがきたら「どういたしまして」の手話だという認識にした。また「こんばんは」は手を上から下に顔の前あたりで下げる動きだったので、スワイプの下方向の動きを検出したら、「こんばんは」の手話という認識にした。

第五章　　TWE-Lite と PIC で端末を構成する

5-1　　PIC と TWE-Lite DIP 側の設定

最初、筆者は Arduino Uno と組み合わせて Leap Motion と連動する外部装置を作成しようと考えていたが、できるだけコンパクトに収めたいと考え、そこからシリアル通信やモーターの制御もできて大きさも手ごろで扱いやすい PIC16F688 を用いて TWE-Lite DIP と通信をすることにした。TWE-Lite DIP と MoNoStick の設定を PIC とシリアル通信するために変更した。変更内容と変更方法については後述する。図 5-1 に PIC16F688 の外観、図 5-2 にデータシートに記載されていた PIC16F688 のピン配置図、図 5-3 に MONO WIRELESS の公式サイトで提供されていたピン配置図を示す。

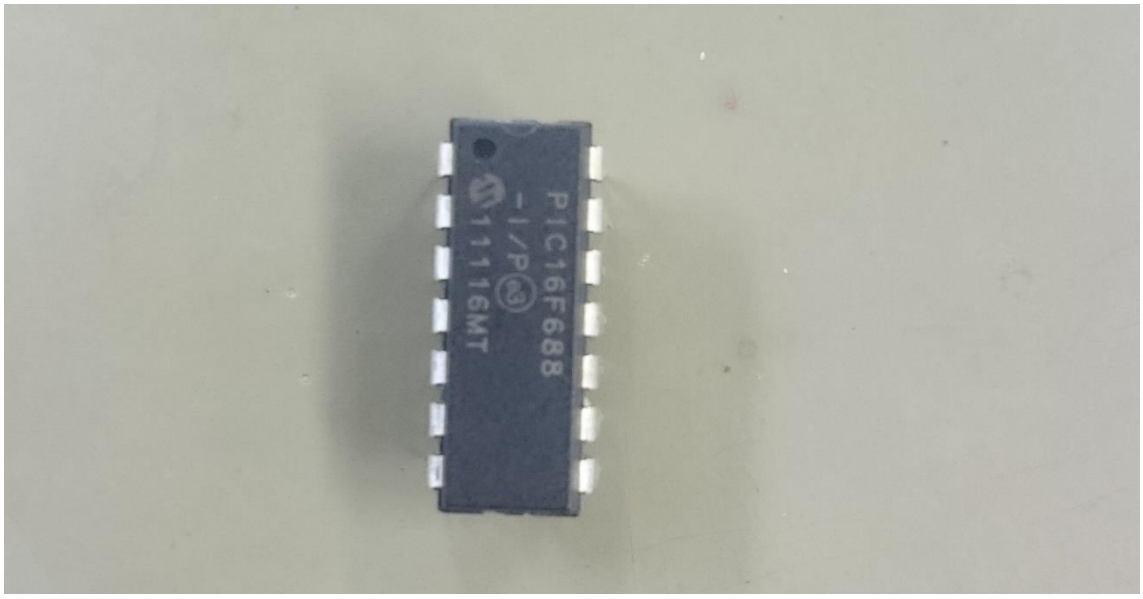


図 5-1　PIC16F688

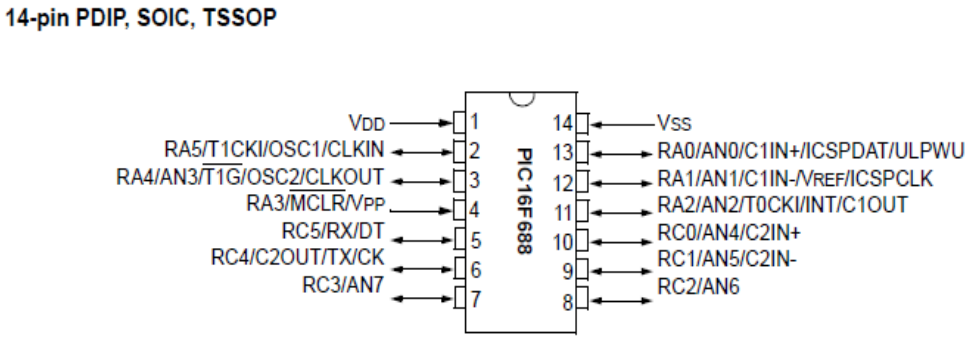
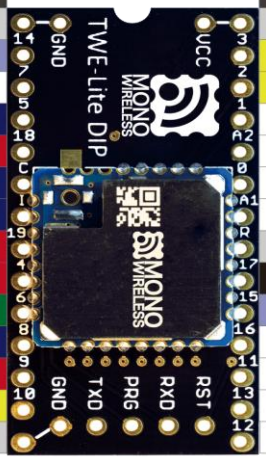


図 5-2　PIC16F688 のピン配置

ピン配置表

機能	信号名	シルク	ピン
電源グランド	GND	GND	1
I2C クロック	SCL	14	2
UART 受信	RX	7	3
PWM 出力 1	PWM1	5	4
デジタル出力 1	DO1	18	5
PWM 出力 2	PWM2	C	6
PWM 出力 3	PWM3	I	7
デジタル出力 2	DO2	19	8
デジタル出力 3	DO3	4	9
UART 送信	TX	6	10
PWM 出力 4	PWM4	8	11
デジタル出力 4	DO4	9	12
モード設定ビット1	M1	10	13
電源グランド	GND	GND	14



ピン	シルク	信号名	機能
28	VCC	VCC	電源 (2.3～3.6V)
27	3	M3	モード設定ビット3
26	2	M2	モード設定ビット2
25	1	AI4	アナログ入力 4
24	A2	AI3	アナログ入力 3
23	0	AI2	アナログ入力 2
22	A1	AI1	アナログ入力 1
21	R	RST	リセット入力
20	17	BPS	UART 速度設定
19	15	SDA	I2C データ
18	16	DI4	デジタル入力 4
17	11	DI3	デジタル入力 3
16	13	DI2	デジタル入力 2
15	12	DI1	デジタル入力 1

図 5-3　TWE-Lite DIP のピン配置図[12]

MoNoStick から送られてきた情報を TWE-Lite DIP から受け取るために UART 送受信のピンを使用する。PIC の TX ピンを TWE-Lite DIP の RX ピンに、TWE-Lite DIP の TX ピンを PIC の RX ピンに接続する。TWE-Lite DIP 側は後は電源の VCC と GND を接続して、20 番ピンをグランドに落とすだけである。20 番ピンをグランドに落とす理由に関しても後述する。PIC のほうに関しては RC0 と RC1 をモーターの制御に使用、12～14番ピンと4番ピンはプログラムの書き込みに使用、RC2 と RC3 は LED の点灯に割り振っている。

次に TWE-Lite DIP と MoNoStick 側の設定を変更していく。TWE-Lite DIP を TWE-Lite-R を用いて PC に接続すると前述通り COM ポートとして認識されるので、そこで TWE-Lite プログラマを用いてシリアル通信専用アプリを TWE-Lite DIP と MoNoStick に書き込む。その状態で TeraTerm などの COM ポートの状態を見られるものを立ち上げる。そこで+++と打ち込むと図 5-4 のような画面が出てくる。

```
--- CONFIG/TWE UART APP V1-02-15/SID=0x8102ddcf/LID=0x65 -E ---
 a: set Application ID (0x00010003)
 i: set Device ID (120=0x78)
 c: set Channels (20)
 x: set RF Conf (3)
 r: set Role (0x0)
 l: set Layer (0x1)
 b: set UART baud (9600)
 B: set UART option (8N1)
 m: set UART mode (T)
 k: set Tx Trigger (sep=0x0d, min_bytes=0 dly=100[ms])
 h: set handle name []
 C: set crypt mode (0)
 o: set option bits (0x00000000)
---
 S: save Configuration
 R: reset to Defaults
```

図 5-4　設定画面

この+++で入ったモードをインタラクティブモードという。このモードでは論理デバイス ID やモードの

変更、ボーレートの変更などの様々な設定ができる。ここでシリアル通信専用アプリに用意されてる4つのモードを紹介する。[3]

Ⅰ.チャット・モード

入力してから改行を打ち込むまでを、他の TWE-Lite DIP や MoNoStick に送信する。データが送信されるタイミングは改行を打った時である。チャット・モードにおいて親機や子機といった区別はなく、入力したものは全ての他の機器に送信される。なお、このチャット・モードではバイナリデータを送ることはできない。

Ⅱ.プロンプト無しチャット・モード

基本的な動作はチャット・モードと同様であるが、送信時にプロンプト文字が追記されず、入力された文字をそのまま送信する。送信のタイミングは改行入力時だけでなく、指定された文字が入力されたタイミングか、一定時間データが送信されなかった時かを設定することができる。なお、このモードではバイナリデータの送信もできる。

Ⅲ.透過モード

親機と子機とで1対1の通信を行う。入力されたデータは改行などを待つことなくすぐに送信される。送受信のデータは余分なデータは追加されることはなく、そのまま送られる。

Ⅳ.書式モード

透過モードでは、エラーチェックを行わないため、データが欠損する可能性がある。そこで、TWE-Lite DIP や MoNoStick でチェックサムを設けて、チェックサムが正しくなければエラーとするのが書式モードである。エラー時に再送する機能も備えられている。透過モードと違い、送信先を指定できるため、多対多の接続もできる。また、中継機を使用した距離の延長も可能である。

今回はこれらの中の透過モードを使用する。これは PIC 側で RS232C(UART)の内蔵ハードウェアを使用するのに適しているからである。設定方法に戻るが、まずインタラクティブモードに入って MoNoStick を親機に設定する。I キーを押して”121”と入力すれば親機に設定される。TWE-Lite DIP の方は 0 と入力し、子機に設定する。次に”m”キーを押すとモード選択ができるので、大文字の“T”を入力すると透過モードとなる。ちなみにこの時に“C”を入力すればチャット・モード、“D”を入力すればプロンプト無しチャット・モード、“A”キーを入力するとアスキー形式の書式モード、“B”を入力すればバイナリ形式の書式モードとなる。

5-2　　プログラム作成

次にシリアル通信をするためのプログラムを紹介する。

Web 上に Arduino Uno と Leap Motion でシリアル通信をする際に使用されたプログラムの例が紹介されていたので、これを元に作成した[6]。図 5-5 は実際の PC 側のプログラムの一部である。図 5-5、図 5-6 図 5-7 にプログラムを示す。

```
#define  TARGET_PORT  _T("¥¥¥¥.¥¥COM5")
#define  BUFF_SZ          (64)

char rbuf[BUFF_SZ];
char wbuf[BUFF_SZ];
HANDLE hCom;
int sendmessage(HANDLE hCom, char *wbuf, char *rbuf);
HANDLE make_handle();

void main(int argc, char *argv[])
{
        char moji[80];
        int n;

        hCom = make_handle();
        while (1){
                printf("送信するデータを入力してください。¥n");
                printf("改行だけ入力すると終了です。¥n");
                gets(moji);
                n = strlen(moji);
                printf("%s¥n", moji);
                printf("入力文字：%s  文字数：%d¥n", moji, n);
                if (n == 0)break;

                sendmessage(hCom, moji, rbuf);
        }
}
```

図 5-5　シリアル通信のプログラム①

```
HANDLE make_handle()
{
	HANDLE hCom = INVALID_HANDLE_VALUE;
	DCB	stDcb;
	COMMTIMEOUTS cto;
hCom=CreateFile(TARGET_PORT,GENERIC_READ|GENERIC_WRITE,0,NULL,
		OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL|
FILE_FLAG_WRITE_THROUGH, NULL);
	if (hCom == INVALID_HANDLE_VALUE){
		printf("Handle is not gotten error¥n");
		exit(1);
		//return INVALID_HANDLE_VALUE;}
	memset(&stDcb, 0, sizeof(DCB));
	stDcb.BaudRate = CBR_115200;
	stDcb.fParity = 1;
	stDcb.ByteSize = 8;
	stDcb.Parity = NOPARITY;
	stDcb.StopBits = ONESTOPBIT;
	stDcb.EofChar = 26;
	if (SetCommState(hCom, &stDcb) == FALSE){
		CloseHandle(hCom);
		printf("COM setting error ¥n");
		exit(1);
		//return INVALID_HANDLE_VALUE;}
	memset(&cto, 0, sizeof(COMMTIMEOUTS));
	cto.ReadTotalTimeoutConstant = 100;
	cto.WriteTotalTimeoutConstant = 200;
	if (SetCommTimeouts(hCom, &cto) == FALSE) {
		CloseHandle(hCom);
		printf("time out setting error ¥n");
		//return INVALID_HANDLE_VALUE;}
	return hCom;
}
```

図 5-6　シリアル通信のプログラム②

```
int sendmessage(HANDLE hCom, char *wbuf, char *rbuf)
{

	BOOL werr = FALSE, rerr = FALSE;
	DWORD wn = 0, rn = 1;
	int i = 0, cr_num = 0;
	if (!WriteFile(hCom, wbuf, (int)strlen(wbuf), &wn, NULL)) werr = TRUE;

	return (werr << 1) | rerr;

}
```

図 5-7　シリアル通信のプログラム③

MoNoStick を USB に挿入したときのターゲットポートは使用した PC によって変わるので、臨機応変に変更しなければならない。主要な動作は、文字の入力を待ち、“Enter”で改行入力されたら入力された文字と、文字数を表示して、入力された文字が“0”ならば終了。それ以外なら入力された文字をシリアル通信で送っていることである。

次に PIC 側のプログラムを紹介する。図 5-9 に PIC 側のプログラムの一部を示す。
PC 側からデータを受け取って、‘e’という文字列が来たら‘e’までの文字列を読み取り、その文字によってモーターの制御を行った。また 5-1 で述べたように PIC の RC0 と RC1 をモーターの制御に使用しているので、fugo が0かつ入力された数字が“0”の場合はモーターの停止と LED の消灯を行い、fugo が0かつ入力された数字が 2 の場合は緑色 LED の点灯、fugo が0かつ入力された数字が“3”の場合はモーターの回転、fugo が0かつ入力された数字が“4”の場合は赤色 LED の点灯、fugo が0かつ入力された数字が“5”の場合はモーターの回転(3 の時とは逆回転)と赤色 LED の消灯を行っている。図 5-8 に接続の様子を示す。

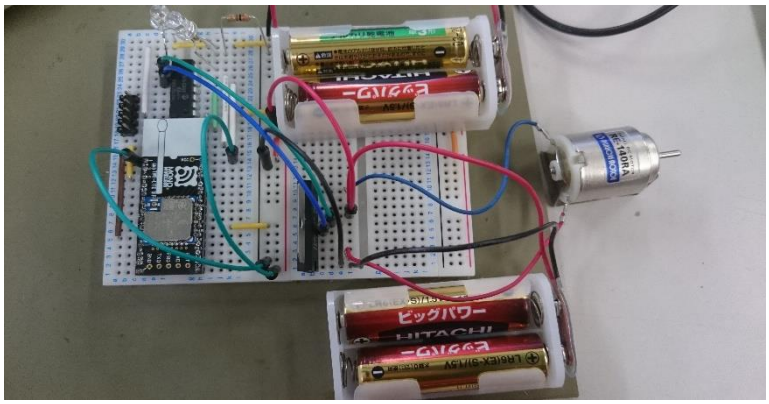


図 5-8　接続の様子

```
while(1)
        {if( dc > 0 ){data = getch();
                        if( data == 'e' ){if( num == 0 || fugo > 0 ){
                                        RC0 = 0;
                                        RC1 = 0;//motor stop
                                        RC3 = 0;
                                        RC2 = 0;}
                                if( num == 2 && fugo == 0 ){RC2 = 1;}
                                if( num ==3 && fugo == 0){
                                        RC0 = 1;
                                        RC1 = 0;}
                                if( num == 4 && fugo == 0 ){RC3 = 1;}
                                if( num   >= 5&& fugo == 0){
                                        RC0 = 0;
                                        RC1 = 1;
                                        RC2 = 0;
                                        RC3 = 1;}
                                num = 0;
                                fugo = 0;
                        }else if( data == '0' ){fugo = 1;num = 0;}
                        else if( data > '0' && data <= '9')num = data - '0';
        }
        }
```

図 5-9　PIC 側のプログラムの一部

これらにより TWE-Lite DIP と MoNoStick と PIC を組み合わせて、無線でシリアル通信を行うことができた。次章ではこれらを応用して外部装置制御と LeapMotion を組み合わせて動作をさせていくことについて説明する。

第六章　手、指の検出と最終作品

前章で TWE-Lite DIP と MoNoStick、PIC を用いたシリアル通信ができることが分かったので、最後に Leap Motion 側のプログラムを制作した。

6-1 手、指の描画

まず認識した手、指の骨格を描画したいので Leap の Bone クラスを使用した。Leap Motion で取得できる骨の一覧は図 6-1 のようになっている。

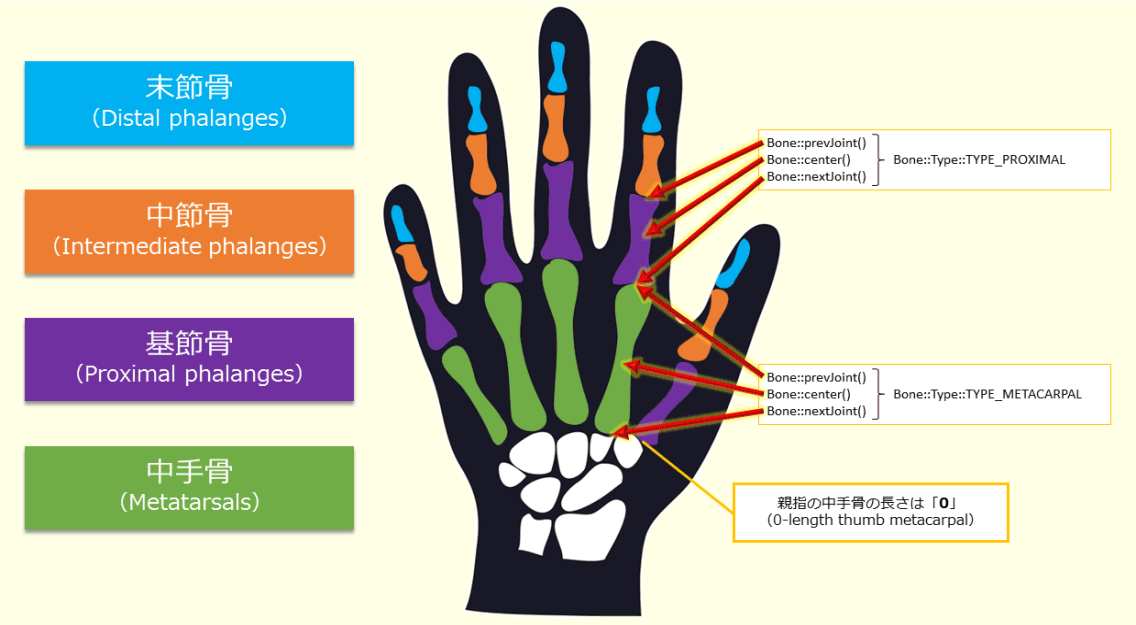


図 6-1　Leap Motion で認識する骨[13]

これらの指骨を描画するためにプログラム内で指骨の一覧を宣言しておく。図 6-2 に宣言部分を示す。

```
const Leap::Bone::Type boneType[] = {
        Leap::Bone::Type::TYPE_METACARPAL,
        Leap::Bone::Type::TYPE_PROXIMAL,
        Leap::Bone::Type::TYPE_INTERMEDIATE,
        Leap::Bone::Type::TYPE_DISTAL,
};
```

図 6-2　指骨の一覧

METACARPAL が中手骨で図 6-1 で言うと緑色の部分、PROXIMAL が基節骨で紫色の部分、INTERMEDIATE が中節骨でオレンジ色の部分、DISTAL が末節骨で水色の部分となっている。

次に指骨の描画部分のプログラムを図 6-3 に示す。

```
glColor3f(0.f, 1.f, 1.f);

glPushMatrix();
x = bone.center().x;
y = bone.center().y;
z = bone.center().z;
glTranslatef(x, y, z);
glutSolidSphere(4, 10, 10);
glPopMatrix();

glColor3f(0.f, 1.f, 0.f);

glPushMatrix();
x2 = bone.nextJoint().x;
y2 = bone.nextJoint().y;
z2 = bone.nextJoint().z;
glTranslatef(x2, y2, z2);
glutSolidCube(7);
glPopMatrix();

glColor3f(0.f, 0.f, 1.f);

glPushMatrix();
x3 = bone.prevJoint().x;
y3 = bone.prevJoint().y;
z3 = bone.prevJoint().z;
glTranslatef(x3, y3, z3);
glutSolidSphere(4, 10, 10);
glPopMatrix();

glColor3f(1.f, 0.f, 0.f);
```

図 6-3　指骨の描画部分

x～x3、y～y3、z～z3にそれぞれの骨の中心と両端の位置座標を与え、glTranslate()関数で描画場所を決めて、それぞれの図形を描画している。また、画面下部に認識している手の指の本数も表示している。実際の描画画面を図 6-4 に示す。

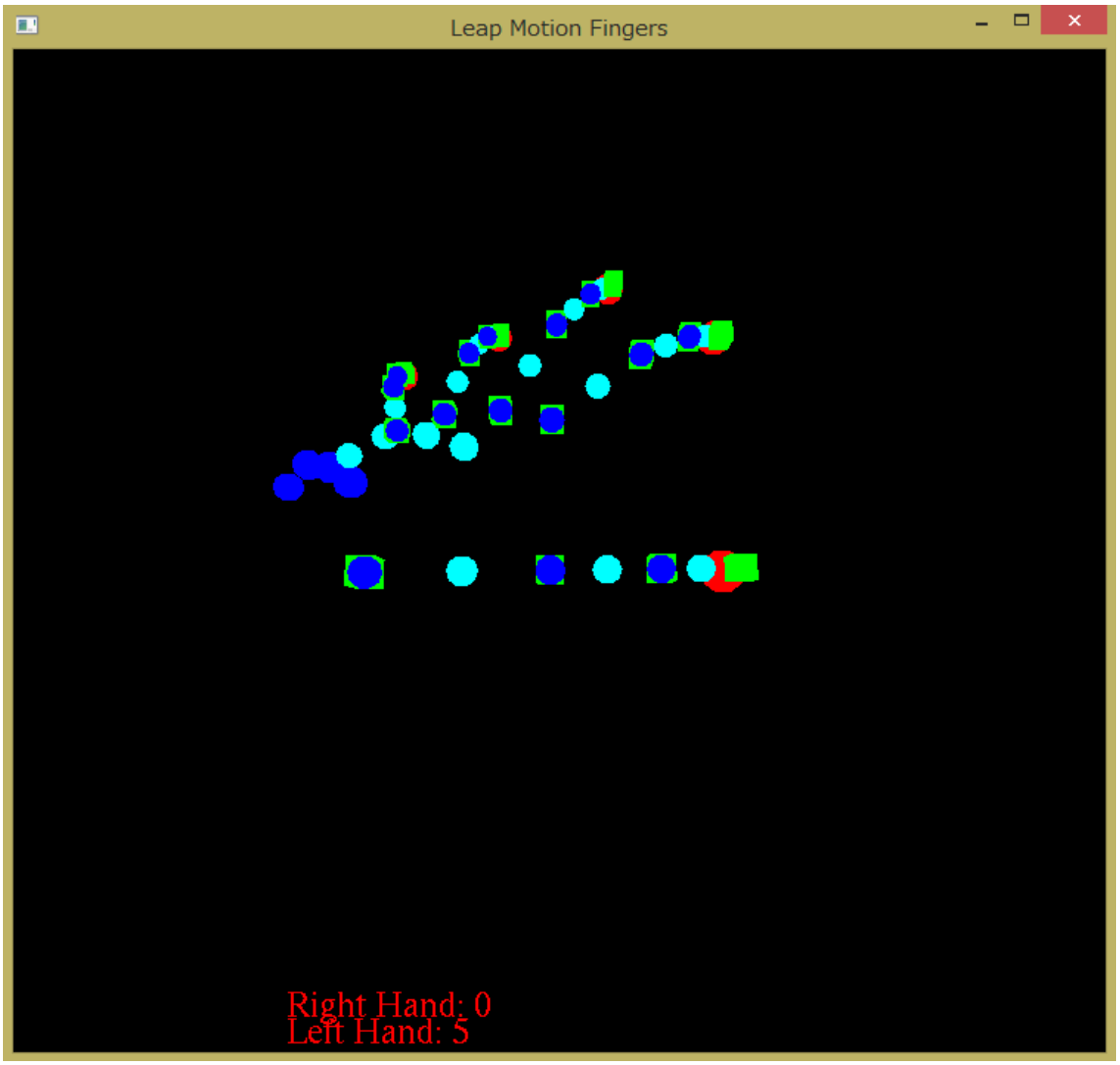


図 6-4　指骨の認識

これにより Leap Motion で現在認識している手の形、指の本数などが視覚的にすぐ分かるようになったと考える。

6-2 手の動き（ジェスチャー）の認識の応用

次に考えたのは Leap Motion でどんな動きを検出したときにシリアル通信でデータを送るかということである。手話の前進と、後進がそれぞれ「前へ手を突き出す」、と「後ろへ手を引く」だったのでそれをそのまま使用することにした。つまり手を前に出せばモーターが前へ回転し、後ろへ引けばモーターが逆回転する。そして手を上に振るとモーターが止まるようにした。その部分のプログラムを図 6-5 に示す。

```
df = dz / sqrt(dx*dx + dy*dy + dz*dz);
db = -dz / sqrt(dx*dx + dy*dy + dz*dz);

if (dr > 0.96 && dist > 150 ){
                                    state = 1;}
if (ddown > 0.96)
        kon = 1;
        sendmessage(hCom, "2e", rbuf);}
if (state == 1 && dup > 0.96){
        st = 1
        ba = 0;
        fr = 0;
        sendmessage(hCom, "0e", rbuf);}
if (state == 1 && dl > 0.97 ){
        dou = 1;
        sendmessage(hCom, "4e", rbuf);}
if (df > 0.96){
        fr = 0;
        ba = 1;
        st - 0;
        sendmessage(hCom, "5e", rbuf);}
if (db > 0.96){
        ba = 0;
        fr = 1;
        st = 0;
        sendmessage(hCom, "3e", rbuf);}
```

図 6-5 プログラムの一部

第四章の図 4-8 のプログラムに前方向への動きと後ろ方向への動きを検出するための変数を追加し、あとはそれぞれの動きの時々で sendmessage でデータを送っている。上述のプログラムでは 2e の時に「こんばんは」という手話を認識して LED の点灯を行っている。4e の時は「どういたしまして」の手話を認識して LED の点灯を行っている。また図 6-4 の時の描画を利用し、手が認識されていないときは画面上の文字を白文字で表し、手が認識されると文字に色が付くようにして、認識の有無を分かりやすくした。またどの手の動きも認識していないときは「No sign」と表示しているが、各手話や動きを認識したらそれに対応する部分が「Detect!」と表示されるようにした。認識されていないときの様子を図 6-6 に、認識されたときの様子を図 6-7 に示す。

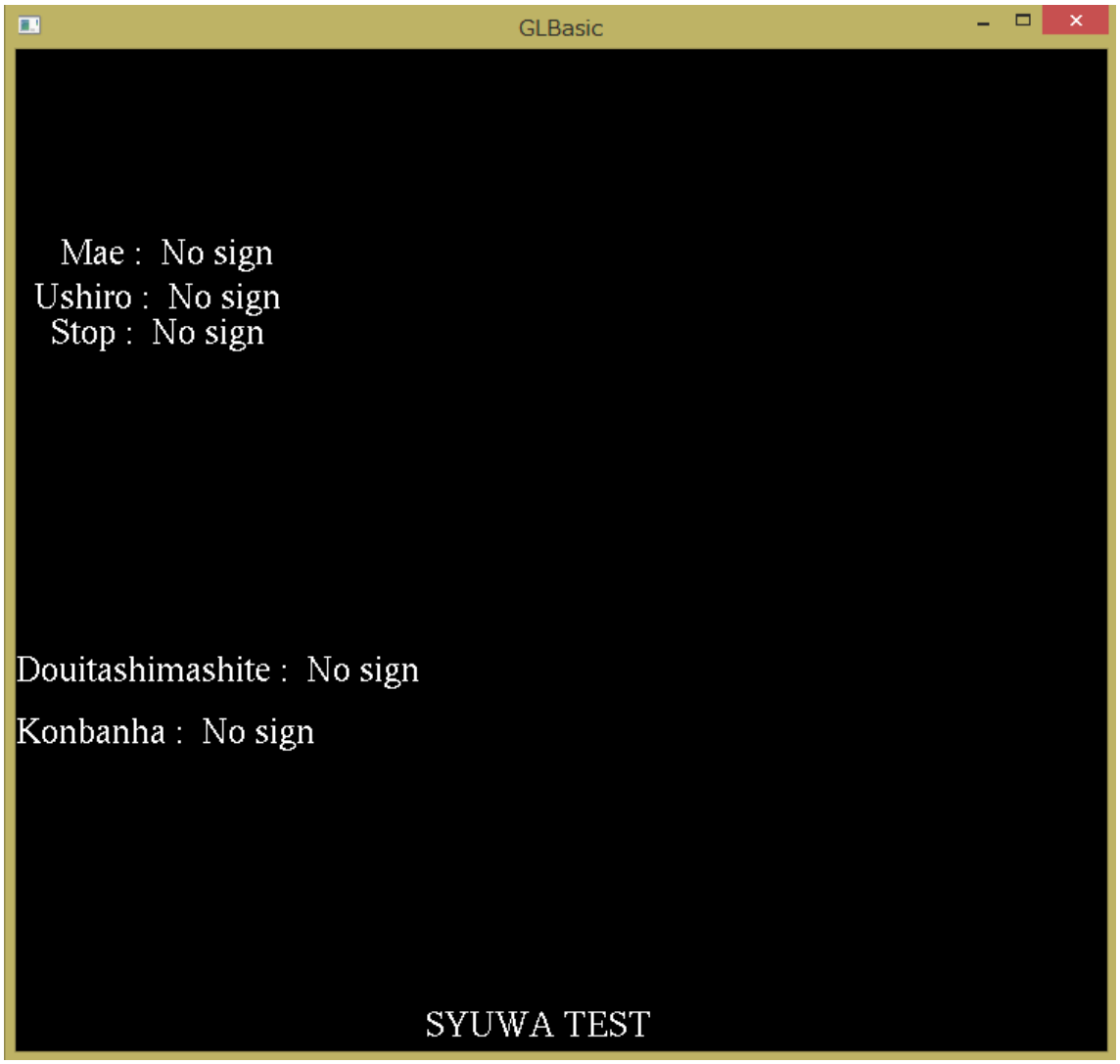


図 6-6　認識していないときの様子

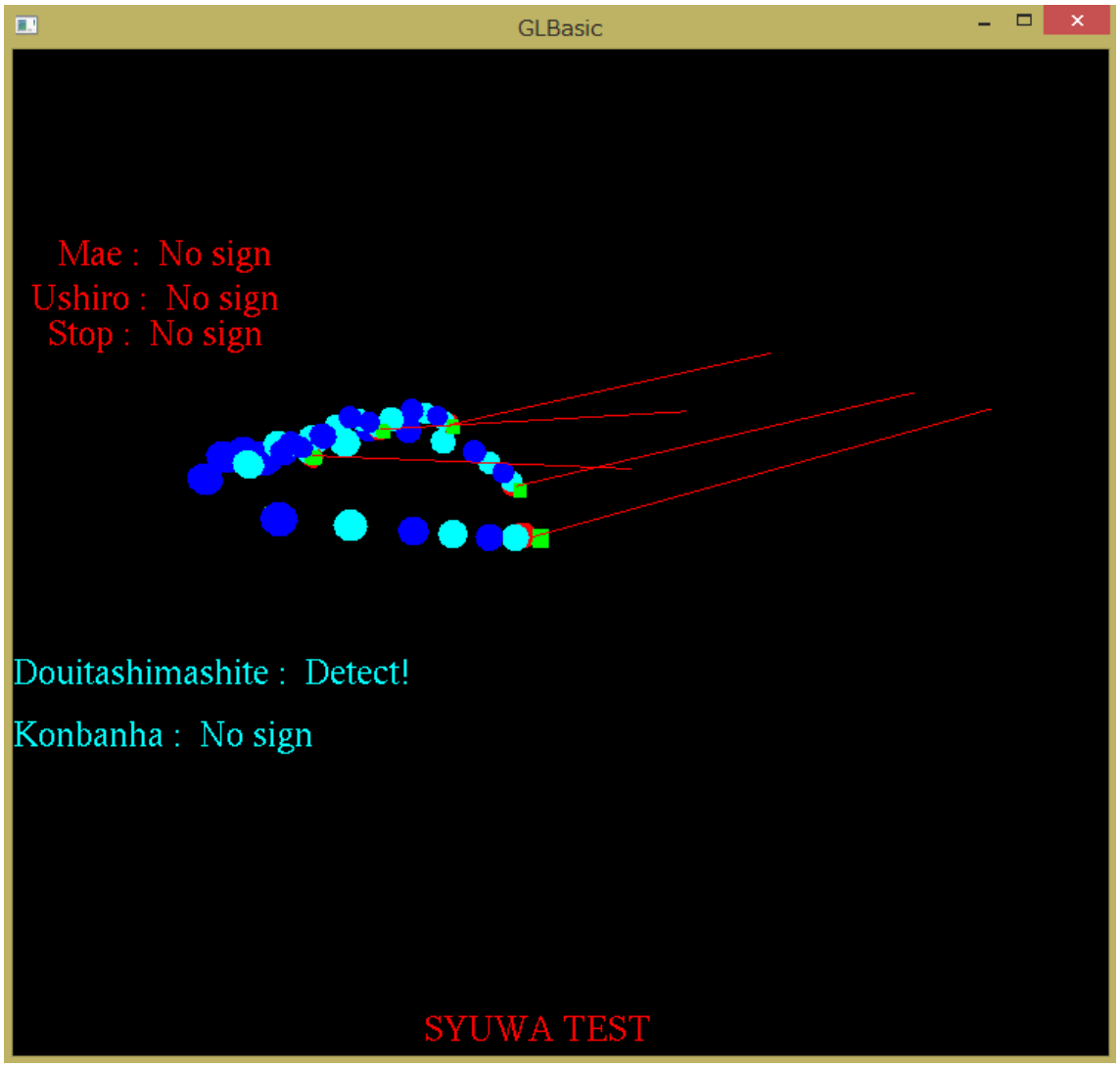


図 6-7　認識したときの様子

図 6-7 で表示されている赤い線はスワイプジェスチャーの軌跡であり、左右の動きを認識したので「どういたしまして」のところに「Detect!」と表示されている。またモーターは TAMIYA から販売されている「4WHEEL DRIVE CHASSIS KIT」に繋ぎ無線ラジコンカーとして動作するようにした。

第七章　考察

ここまでの長い過程を経て指や手の検出、描画を行い、Leap Motion を使用することによって直感的なシステムの操作ができるのではないかと考える。プログラムを構築していくことについては一定の知識や勉強が必要であるが、それでも Leap Motion のプログラムの開発は比較的容易な方だと筆者は思う。やり易さというのは重要で、それは多くの人が触れ易いということに繋がると思う。手や指といった三次元的なものの情報が扱えるデバイスがあってもその扱いが非常に難解なものであったらそのデバイスを用いたプログラムを開発できる人間は限られてしまう。。

また、出来上がったプログラム、ひいてはアプリの様なモノを使用する側となったとする。この時、使用する側としては分かりやすさが重要になってくると考える。今回制作したプログラムも含め、自らの体の一部で動作するというのは非常に分かりやすい手段であると考える。本研究では OpenGL とも連携させたので文字列の表示と合わせて、今現在自分のどの指や手が検出されているのか、どういう動きをしたのかという状況を視覚的に使用者に分かるようにする点に注力した。TWE-Lite と組み合わせることによって Leap Motion のある場所から手を動かすだけで今回はラジコンの前進、後進、ストップの操作が可能になる。今回は既存の模型の車で試したがサーボモーターなども組み合わせれば旋回など制御の幅も広がると感じた。

これらから Leap Motion だけではなく、様々なソフトであったり、自作の装置などと組み合わせて使用することにより、利便性や、できることがグンと広がり、さらに自由なプログラムや装置の構築ができるということが分かる。今回はスワイプジェスチャーを最終的に使用していったが、他にも認識可能なジェスチャーはあるのでそれらと組み合わせることによってさらに複雑な手の動きなども正確に検出できるのではないかと考える。

全体的な作品の開発を通して、左手、右手の検出及びどちらかの手のみを表示することができるようになった。指に関しても全ての指の検出と識別に成功した。指関節の検出と識別、いくつかのジェスチャーの検出と識別も可能になった。ジェスチャーに関しては、任意の指や手でどの位置からどの方向にどれだけ移動したかを検出できるようになったので、Leap Motion を用いた1通りの使い方に関してはプログラム可能になったと言える。批判的な部分としては、今回の手話の“どういたしまして”と“こんばんは”の検出が何度かに一度は失敗することが挙げられる。こちらの原因についてはよくわからず、Leap Motion 社提供の SDK の精度に依存しているのではないかと推察している。

最終作品のグラフィカルな点における出来栄えはお世辞にも良いとは言えず、こちらに関しては OpenGL に対する勉強不足及び作り込み不足である。しかしながらこの点に関しては本論の主題と異なる問題であるので、自ら設定した目標に到達したと考える。

第八章　結論

本研究の結論として、Leap Motion と PIC、TWE-Lite を用いることによって、無線でシリアル通信を行い、Leap Motion で認識した手の動きと連動させて動作させることに成功した。これにより本研究の目的とした Leap Motion を用いての手や指、骨格の検出や、いくつかの簡単な手話の認識、無線による離れた場所へのデータ転送からの外部装置の駆動というのは達成できたと言える。

謝辞

今回の特別研究に取り組むに当たり、懇切丁寧にご指導、ご鞭撻頂いた高知工科大学システム工学群の綿森道夫准教授に心よりの感謝を申し上げます。

参考文献

[1] 中村　薫、 Leap Motion プログラミングガイド[改訂版]、 I/O 編集部、 株式会社工学社、東京、 2015

[2]モノをつなぐ無線マイコンモジュール　TWELITE-トワイライト- MONO-WIRELESS.COM、MONOWIRELESS、 https://mono-wireless.com/jp/products/TWE-LITE/index.html 、1月　2018

[3] 大澤　文孝、 TWE-Lite ではじめるカンタン電子工作、 I/O 編集部、 株式会社工学社、東京、 2014

[4] MICC、 はじめての OpenGL[改訂版]、 I/O 編集部、 株式会社工学社、 東京、2012

[5] 橋本　洋志・小林　裕之、 図解　OpenGL　による3次元 CG アニメーション、株式会社オーム社、 東京、 2005

[6] とえら、 水飴信号　Starch Syrup Signal [Arduino と Leap Motion のシリアル接続]、とえら、http://sssignal.web.fc2.com/column/column17.html 、 6 月　2016

[7] 小林　実来・武田敦志、“Leap Motion Controller を用いた手話動作認識手法の実装と評価”、東北学院大学教養学部情報科学科

[8] 船坂真生子・石川由羽・髙田雅美・城和貴、“Leap Motion Controller　を用いた指文字認識”　、　奈良女子大学

[9] LEAP MOTION/ DEVELOPER, https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Overview.html

[10] LEAP MOTION 、https://www.leapmotion.com/setup

[11] LEAP MOTION/ DEVELOPER 、https://www.leapmotion.com/developers

[12] モノをつなぐ無線マイコンモジュール　TWELITE-トワイライト- MONO-WIRELESS.COM 、https://mono-wireless.com/jp/products/TWE-APPS/App_Twelite/index.html

[13] https://www.buildinsider.net/small/leapmotioncpp/002