令和５年度卒業研究報告書

身支度を楽しくするスマートミラーの作成

情報技術科　牛抱　伶奈

大久保　遥夏

小原　梨里花

指導教員　　飯坂　ちひろ

目次

第 1 章　はじめに 3

1.1 研究の背景 3

1.2 研究の目的 3

第 2 章　研究概要 4

2.1 システム構成 4

2.2 使用材料 6

2.3 開発環境 6

2.4 使用モジュール 7

2.4.1　Magic Mirror² 7

2.4.2　MagicMirror_scripts 7

2.4.3　MMM-Remote-Control 7

2.4.4　MediaPipe Hands 8

2.5 使用ライブラリ 9

2.5.1　ハンドジェスチャー 9

2.5.2　画像の切り抜き 10

2.5.3　画像の結合 11

第 3 章　環境構築 12

3.1 システムの環境構築 12

3.1.1　Visual Studio Code の導入 12

3.1.2　Raspberry Pi の OS 導入 19

3.1.3　Raspberry Pi での言語・地域設定 21

3.1.4　使用ライブラリの導入 22

3.2 Tera Term の環境設定.....23
3.2.1 Tera Term の導入.....23
3.2.2 ファイヤーウォール.....30
3.3 プロキシ設定.....31
第 4 章 システムの構築.....33
4.1 スマートミラーのセットアップ.....33
4.1.1 Magic Mirror2 の導入.....33
4.1.2 MagicMirror_scripts の導入.....33
4.1.3 MMM-Remote-Control の導入.....37
4.1.4 天気の設定.....39
4.2 ハンドジェスチャー.....42
4.2.1 ソースコードのダウンロード.....42
4.2.2 プログラムの説明.....45
4.3 服装提案システム.....59
4.3.1 背景の切り抜き.....59
4.3.2 画像の結合.....61
4.3.3 プログラムの説明.....63
第 5 章 おわりに.....69
5.1 考察.....69
5.2 成果と課題.....69
5.3 まとめ.....69
第 6 章 参考文献.....70

第１章　はじめに

1.1 研究の背景

現代社会では、IT 技術の進歩により私たちの日常生活はますます便利で快適になっている。その一方で、日々の忙しい生活の中で、自己管理や情報収集に使える時間を確保することは難しいと感じる人も少なからずいるはずだ。それぞれのちょっとした時間を有効活用し、IT 技術を使って楽しくかつ効率的に情報を得ることができれば、さらに便利になると考えた。

1.2 研究の目的

研究の背景から日々の生活に欠かせない「身支度」の時間に焦点を当てた。鏡で身支度をしながら時間や天候、気温などの情報を確認することができ、その日にぴったりな服装を提案してくれるスマートミラーを作成できないかと考えた。

また、私たちは基礎セミナーで Python を学んでおり、その知識を生かしたいと考えていた。そのため、主に Python を開発言語として使う Raspberry Pi を使用してスマートミラーを作成しようと考えた。

第 2 章　研究概要

2.1 システム構成

カメラでハンドサインを捉え、その映像をパソコンに送る。パソコンでは、MediaPipe Hands を用いてハンドサインを AI で認識し、その情報を Raspberry Pi に送る。Raspberry Pi では、ディスプレイの表示内容を出力しつつ、パソコンから指示されたスマートミラーの ON／OFF といった動作を行う。ディスプレイは、マジックミラーフィルムを貼り、使用する。鏡のように自分の姿が映り、日付や天気などが表示される。

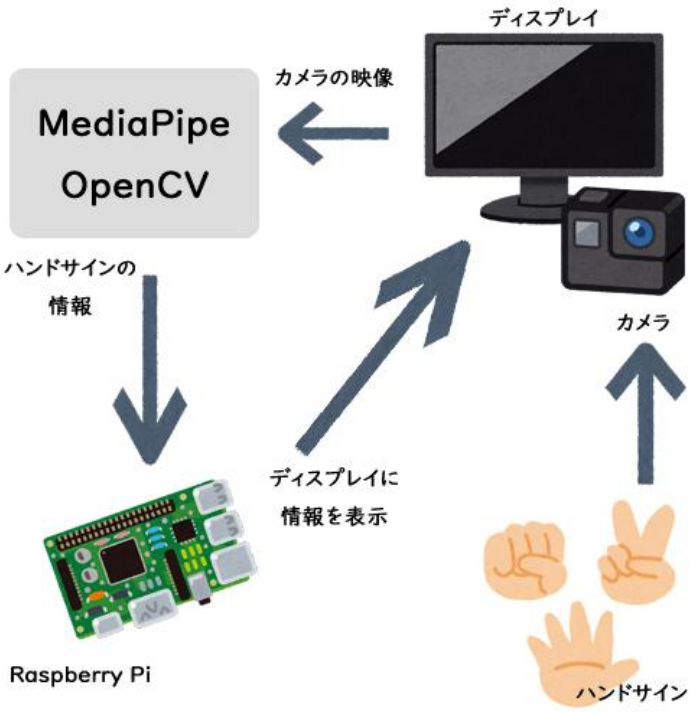

図 2.1　一連の流れ


Friday,December 22,2023
11:27
TASKS
用事 Tomorrow at 10:00
5℃
Sat 晴れ 1℃
Sun 晴れ 3℃
Mon くもり 4℃
Tue 晴れ 2℃
Wed くもり 2℃
Thu くもり/雪 1℃
今日のおすすめ服

図 2.2　完成イメージ

2.2 使用材料

材料は以下のとおりである。

- ディスプレイ：日付、天気などを表示。
- マジックミラーフィルム：ディスプレイに貼って使用。
- Raspberry Pi：日付、天気などを表示させるのため開発。
- Hand Gesture Recognition 用の PC：MediaPipe Hands を用いてハンドサインを AI で認識し、Raspberry Pi に送るために使用。

2.3 開発環境

また、開発環境は表 2.1 と表 2.2 のとおりである。

表 2.1　Raspberry Pi の開発環境

OS	Raspberry Pi OS Full（64-bit）
エディタ	Tera Term
モジュール	Magic Mirror² MagicMirror_scripts MMM-Remote-Control

表 2.2　Hand Gesture Recognition 用 PC の開発環境

OS	Windows10
エディタ	Visual Studio Code
開発言語	Python
モジュール	MediaPipe Hands

2.4 使用モジュール

2.4.1 Magic Mirror²

スマートミラーを作成するためのオープンソースのソフトウェアプロジェクト。ラズベリーパイ（Raspberry Pi）や他のシングルボードコンピューターを使用して、デジタルディスプレイをミラーとして使用し、様々な情報や機能を表示できるようにすることを目的としている。モジュールベースのアーキテクチャを採用しており、ユーザーが独自の機能を追加したり、デザインを変更したりするのが容易。

2.4.2 MagicMirror_scripts

Magic Mirror2 の簡易インストールスクリプト。モジュールのインストール、設定の変更、システムのアップデートなど、さまざまなタスクを効率的に実行するのに役立つ。

2.4.3 MMM-Remote-Control

Magic Mirror²の遠隔操作を可能にするための拡張機能。ウェブブラウザを介して Magic Mirror²のインターフェースにアクセスし、モジュールの非表示・表示、リロード、シャットダウン・再起動、設定の変更といった様々な操作を行うことができる。

2.4.4 MediaPipe Hands

オープンソースの機械学習ツールキットで、様々な視覚タスクに使用できる。Hands モデルは、カメラからの画像やビデオ入力を受け取り、その中の手の位置、姿勢、指の位置などをリアルタイムで検出する。手の追跡を行うための深層学習モデルを基にしており、手のポーズやジェスチャーの検出に利用される。簡単に利用できる API を提供し、開発者が手軽に手の追跡機能を導入できるようになっている。これにより、手の動きをリアルタイムで検知し、アプリケーションやデバイスに組み込むことが可能。

2.5 使用ライブラリ

2.5.1　ハンドジェスチャー

表 2.3　ハンドジェスチャーのライブラリ①

モジュール名	説明
cv2	OpenCV の Python バインディングで、画像や動画の処理ができる機能がまとめられたオープンソースライブラリ。
Numpy	Python で数値計算を効率的に行うためのライブラリ。主に多次元の配列や行列を操作するための機能を提供している。
csv	CSV 形式のファイルを扱うための標準ライブラリ。CSV 形式のデータの読み込みや書き込みを容易にするための機能を提供している。
TensorFlow	人工知能開発用の計算ライブラリ。数値計算やディープラーニングのための高度な数学演算を柔軟かつ効率的に行えるように設計されている。
warnings	プログラムの実行時に発生する警告メッセージを管理するための機能を提供する。
socket	ネットワーク通信を行うための基本的なソケット操作を提供する。
struct	バイナリデータと Python のデータ型との相互変換を行うための機能を提供する。
math	基本的な数学的な演算や関数を提供する。
Threading	スレッドベースの並行処理をサポートする。
os	ファイルやディレクトリの操作、プロセスの制御、環境変数の取得など、様々なオペレーティングシステム関連の機能を利用できる。
sys	Python のインタプリタや実行環境に関する情報を扱うためのライブラリ。

表 2.4　ハンドジェスチャーのライブラリ②

time	主に時間の計測、休止、フォーマット変換など、時間に関連する操作を行うために利用する。
requests	Python のサードパーティライブラリで、HTTP リクエストを簡単に送信し、HTTP レスポンスを処理するための機能を提供する。
vptree	データベースや機械学習のコンテキストで使われるデータ構造の一つ。

2.5.2　画像の切り抜き

表 2.5　背景切り抜きのライブラリ

モジュール名	説明
rembg	u2net と pymatting というツールを内部で使用し、背景からオブジェクトを切り抜くツール
cv2	2.5.1　ハンドジェスチャー　の通り

2.5.3　画像の結合

表 2.6　画像結合のライブラリ

モジュール名	説明
rembg	2.5.2　画像の切り抜き　の通り
Numpy	2.5.1　ハンドジェスチャー　の通り
cv2	2.5.1　ハンドジェスチャー　の通り
uuid	プログラムやデータベースなどで一意の識別子を作成するのに使う。
os	2.5.1　ハンドジェスチャー　の通り
random	様々なランダムな操作を提供する。

第 3 章　環境構築

3.1 システムの環境構築

3.1.1　Visual Studio Code の導入

Hand Gesture Recognition 用の WindowsPC に導入する。

図 3.1～図 3.6 に導入の手順を示す。

① Visual Studio Code のダウンロードページ（https://code.visualstudio.com/download）の Windows 用のリンクをクリックしてインストーラをダウンロードする。

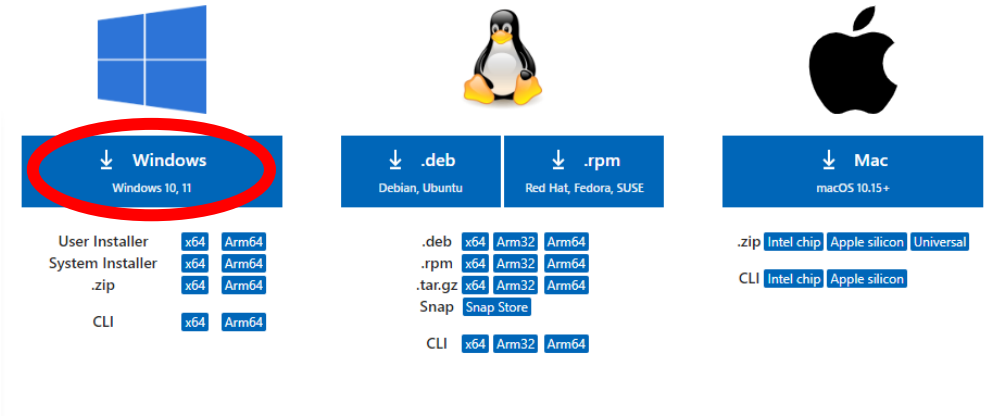

図 3.1　Visual Studio Code のダウンロードページ画面

② ダウンロードしたインストーラをダブルクリックで実行し、Visual Studi Code をインストールする。

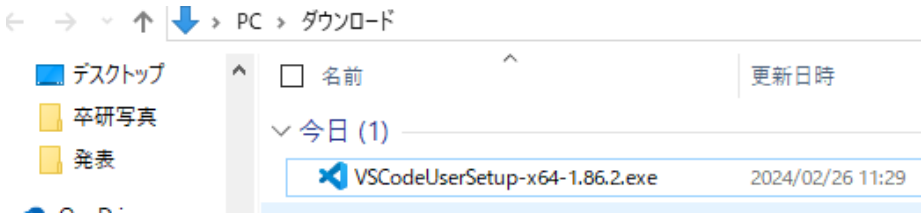

図 3.2　Visual Studio Code のダウンロードしたインストーラー

③　セキュリティ警告画面の「実行」をクリックする。

図 3.3　Visual Studio Code のセキュリティ警告画面

④　「同意する」を選択し「次へ」をクリックする。

図 3.4　Visual Studio Code の同意確認画面

⑤　フォルダーの指定は各自わかるところへ入れる。

必要であれば「デスクトップ上にアイコンを作成する」にチェックを入れ、「次へ」をクリックする。

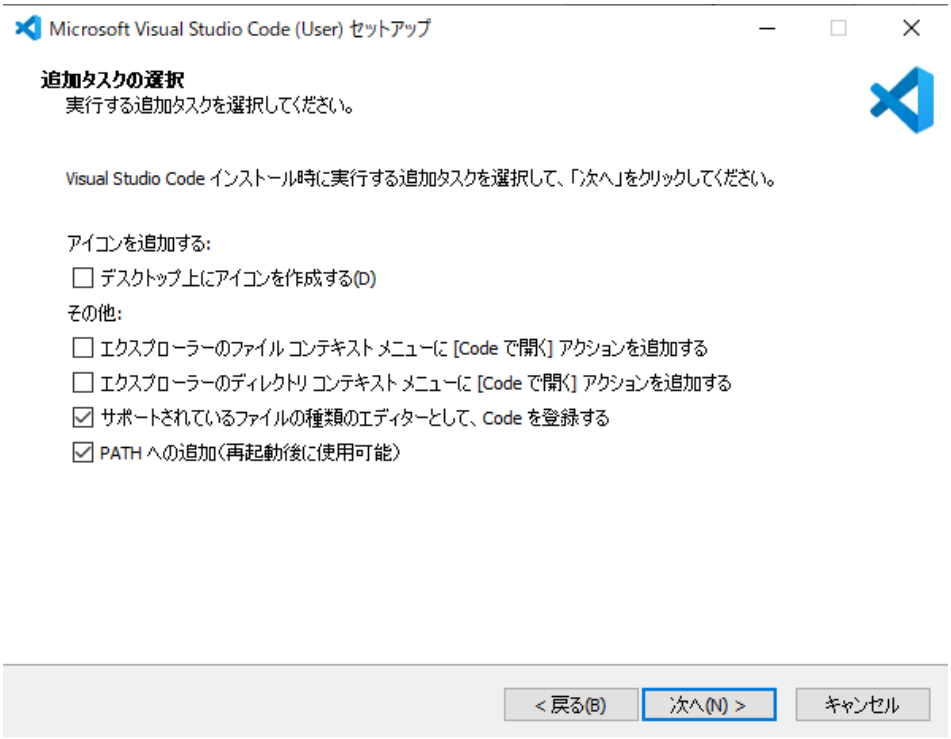

図 3.5　Visual Studio Code の追加タスク画面

⑥　「インストール」をクリックする。

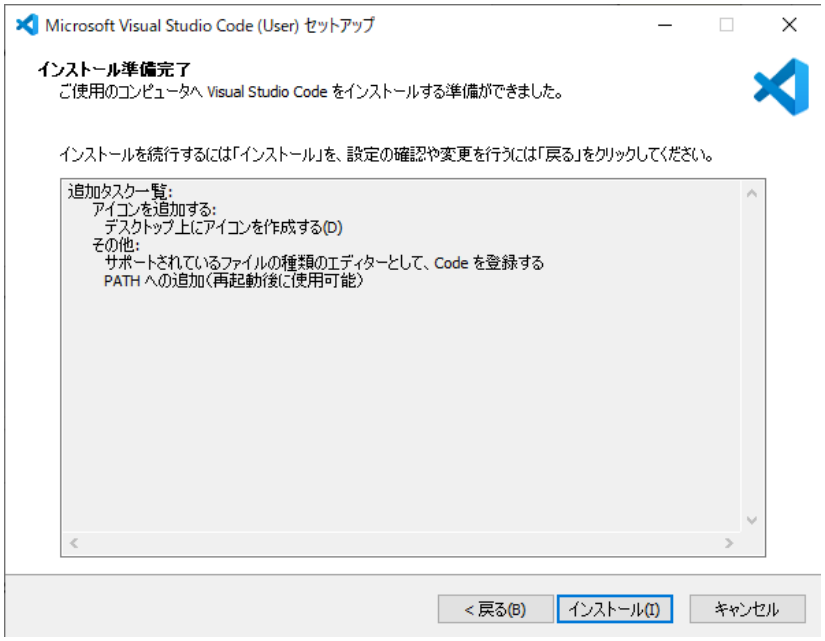

図 3.6　Visual Studio Code のインストール準備完了画面

Visual Studio Code の導入はおわり。

図 3.7～図 3.12 に日本語と Python の設定方法を示す。

① VSC を起動し、ウィンドウ左下の「Extensions」をクリックする。

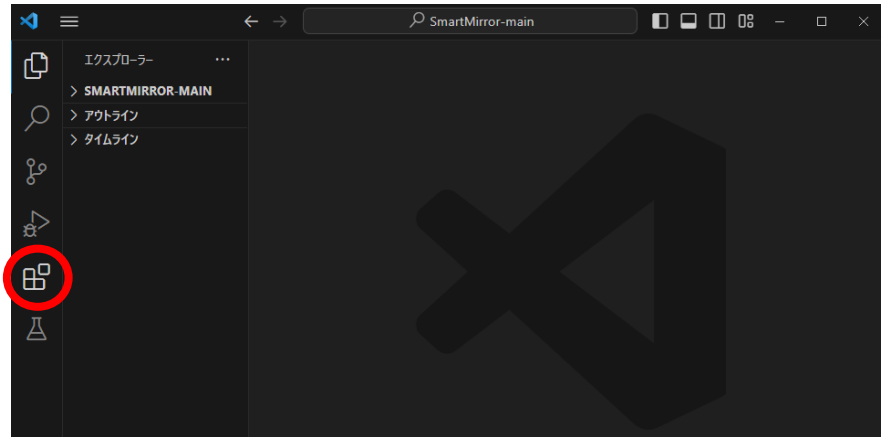

図 3.7　Visual Studio Code の起動画面

② 検索欄に「japanese」と入力し「Japanese Language Pac for VS Code」をインストールする。

図 3.8　Visual Studio Code の日本語導入画面

③ インストール後、右下に表示される「Change Language and Restart」ボタンをクリックして、Visual Studio Code を再起動する。

I. 日本語が反映されない場合は、左上の「View」から「Command Palette」をクリックする。

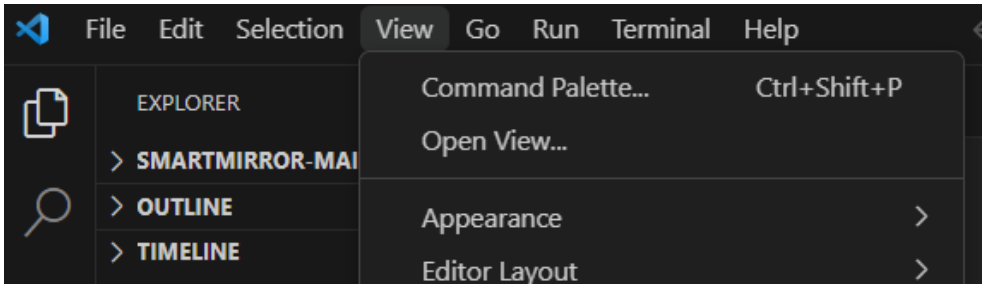

図 3.9　Visual Studio Code のコマンドパレット表示画面

II. コマンドパレットに 「display」と入力し、「Configure Display Language」をクリックする。

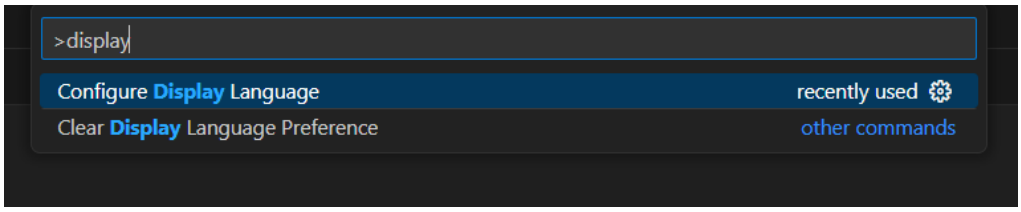

図 3.10　Visual Studio Code の「Configure Display Language」検索画面

III. 「日本語(ja)」を選択し、ダイアログに表示される「restart」をクリックする。

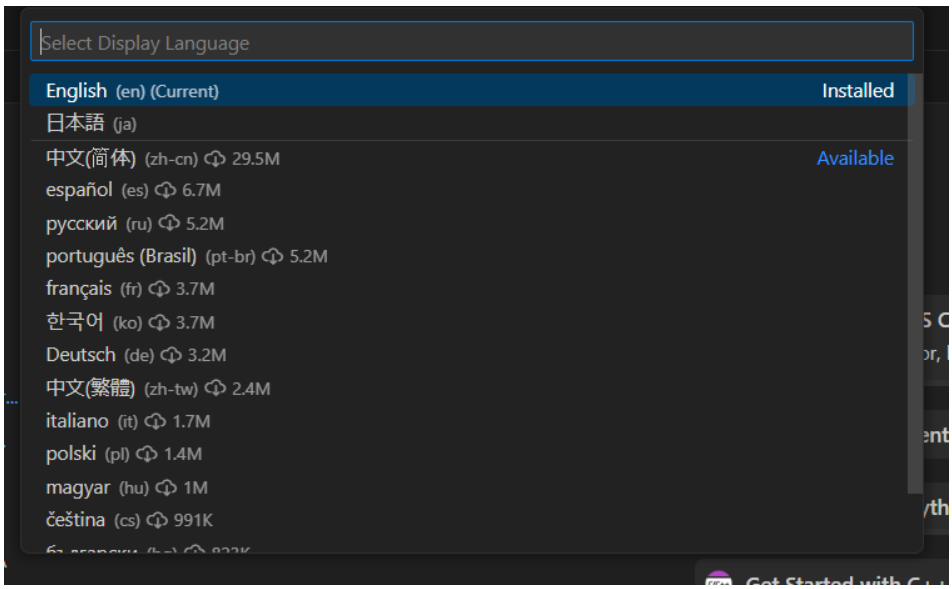

図 3.11　Visual Studio Code の日本語選択画面

④　ウィンドウ左下の「拡張機能」をクリックし、検索欄で「python」を入力する。

図 3.12　Visual Studio Code の Python 導入画面

「Python」をインストールし、日本語と Python の設定はおわり。

3.1.2　Raspberry Pi の OS 導入

OS は Raspberry Pi OS Full（64-bit）を使用する。

このOSを選択した理由は大きなメモリ空間にアクセスでき、一般的にパフォーマンスが向上する。また、大容量のメモリを効果的に利用できることで複雑なアプリケーションやデータ処理が必要な場合に特に有益である。

図 3.14～図 3.16 に OS 書き込みの手順を示す。

はじめに OS 導入用の SD カードを用意しておく。

① サイトから raspberry pi imager というアプリを WindowsPC にインストールする。

(https://www.raspberrypi.com/software/)

図 3.13　raspberry pi imager のアイコン

② インストールしたアプリを開き、管理者権限でインストールを実行する。

③ アプリが開けたら、OS を選ぶから、Raspberry Pi OS Full（64-bit）を選ぶ。

図 3.14　raspberry pi imager の起動画面

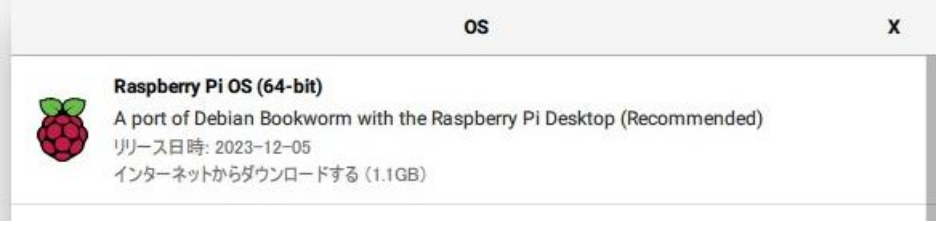

図 3.15 使用する raspberry pi OS

④　ストレージを選ぶから、使用する SD カードを選ぶ。

⑤　歯車マークの設定からユーザー名とパスワードの設定を行う。

図 3.16　ユーザー名とパスワードの設定画面

⑥　書き込むをクリックする。(書き込みには少し時間がかかる)

⑦　書き込んだ SD カードを Raspberry Pi に差し込み、起動する。

これで OS の導入はおわり。

3.1.3　Raspberry Pi での言語・地域設定

起動すると言語が英語になっているので日本語に変更する。地域も日本に設定する。

① 画面左上の Raspi マークをクリックして Settings→Raspberry Pi Configuration→ Localization を選ぶ。

② ロケールの設定から、「言語：ja(Japanese)」「国：JP(Japan)」「文字セット：UTF-8」に変更する。

③ タイムゾーンの設定から、「地域：Asia」「位置：Tokyo」に変更する。

④ キーボードの設定から、「モデル：Generic 105-key PC」「配列：Japanese」「種類：Japanese」に変更する。

⑤ 無線 LAN の国設定から「国：JP Japan」に変更する。

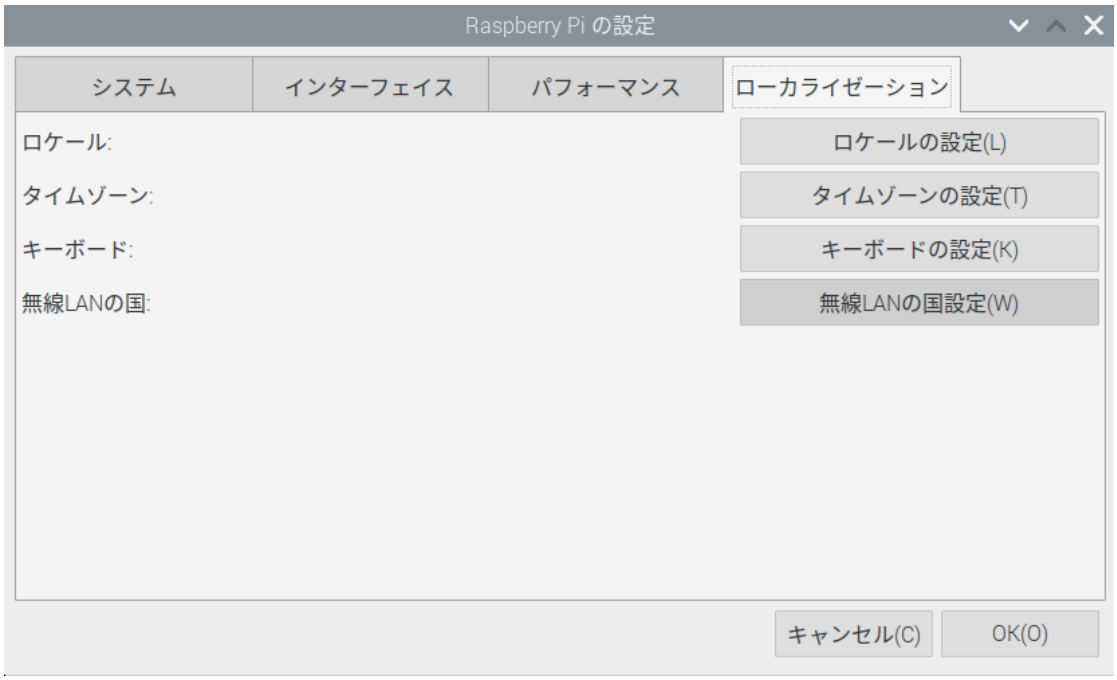

図 3.17　ローカライゼーションの設定画面

⑥ 全て変更し終わったら、再起動をする。

3.1.4　使用ライブラリの導入

使用ライブラリの導入方法を表 3.1 に示す。

Python インストール後、コマンドプロンプトまたは VSCode のターミナルで以下のコマンドを実行する。

表 3.1　ライブラリの導入コマンド一覧

コマンド	行う処理
set HTTP_PROXY=http://172.16.0.2:8080	校内プロキシの設定(HTTP)
set HTTPS_PROXY=http://172.16.0.2:8080	校内プロキシの設定(HTTPS)
pip install opencv-python	OpenCV のインストール
pip install numpy	NumPy のインストール
pip install tensorflow	TensorFlow のインストール
pip install scikit-learn	scikit-learn のインストール
pip install vptree	vptree のインストール
pip install requests	requests のインストール
pip install rembg	rembg のインストール

3.2 Tera Term の環境設定

3.2.1　Tera Term の導入

Tera Term とは、Windows 上で動作するターミナルエミュレータおよびシリアル通信ソフトウェアのこと。今回は Windows から Raspberry Pi にリモート接続するために使用する。

図 3.18～図 3.26 に導入の手順を示す。

Teraterm をダウンロード（https://ja.osdn.net/projects/ttssh2/releases/）し、デスクトップに保存する。(exe 形式のファイルをダウンロードするのを推進)

デスクトップに追加された図 3.18 のアイコンをダブルクリックする。

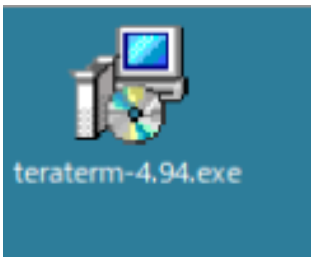

図 3.18　Tera Term のアイコン

選択言語を日本語にし、「OK」をクリックする。

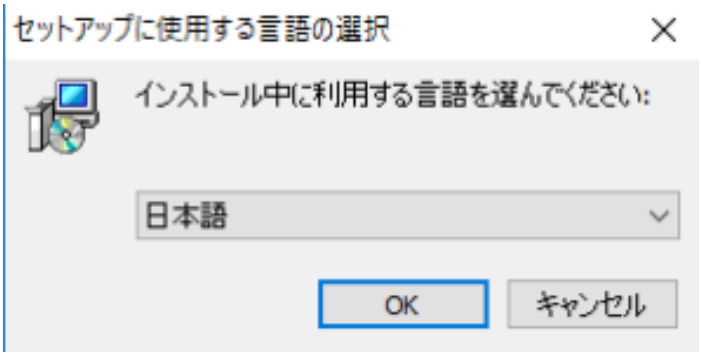

図 3.19　Tera Term の日本語選択画面

「次へ」をクリックする。

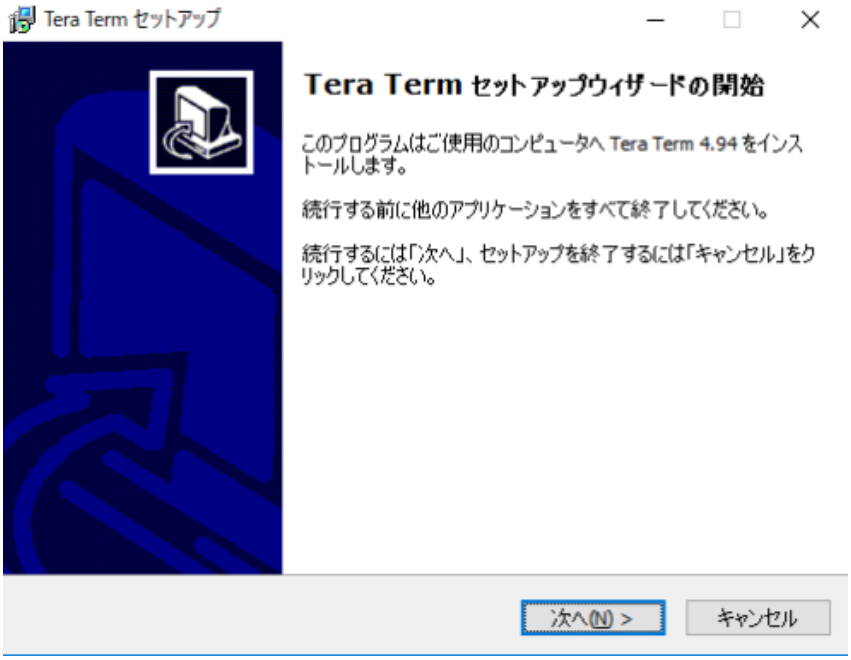

図 3.20　Tera Term のセットアップ画面

「同意する」を選択し「次へ」をクリックする。

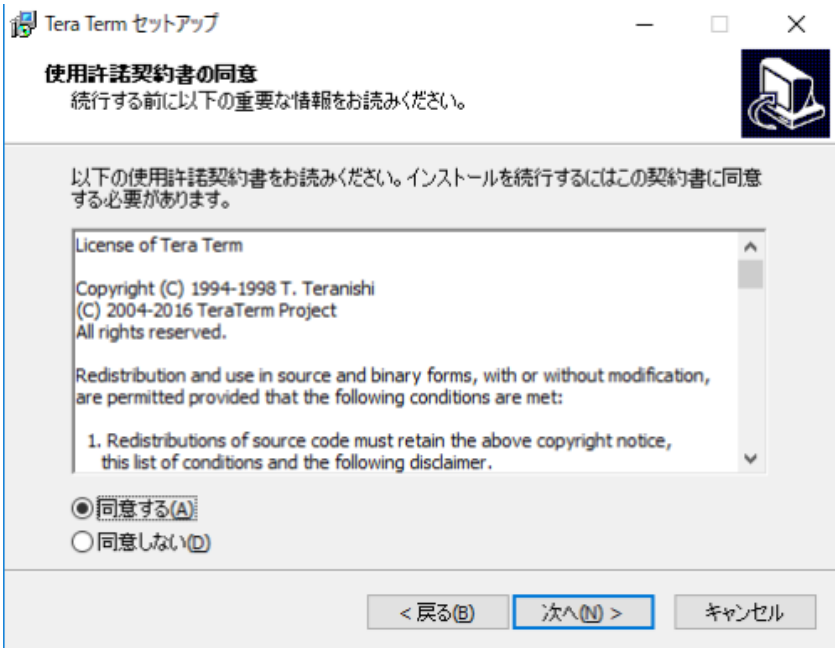

図 3.21　Tera Term の同意確認画面

フォルダーは各自わかるところに指定する。

「次へ」をクリックする。

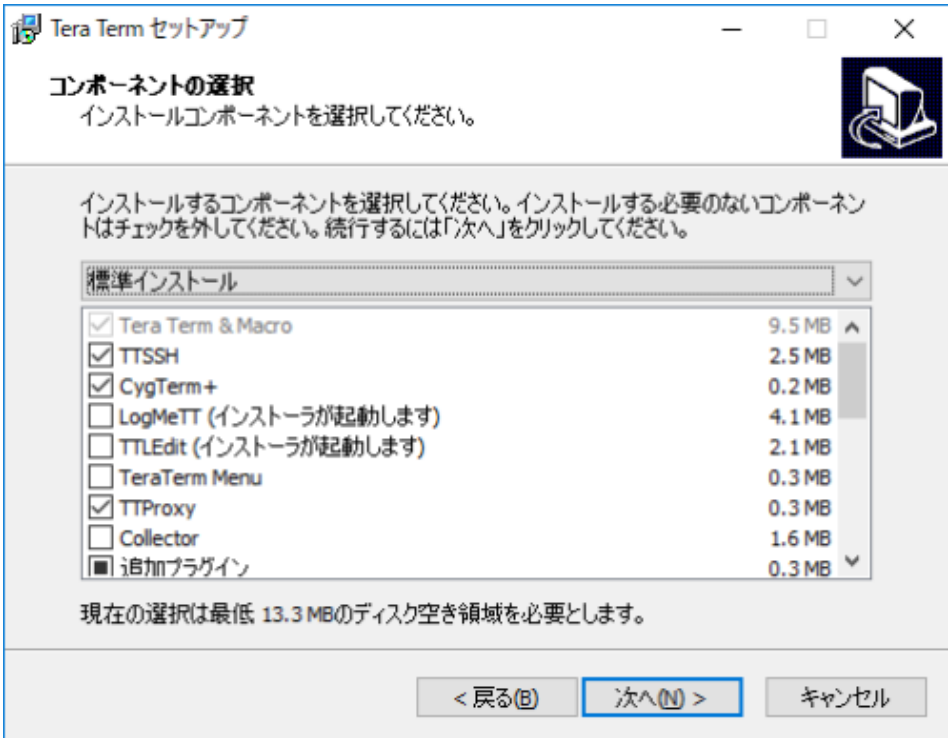

図 3.22　Tera Term のコンポーネント選択画面

言語の選択は「日本語」を選択し「次へ」をクリックする。

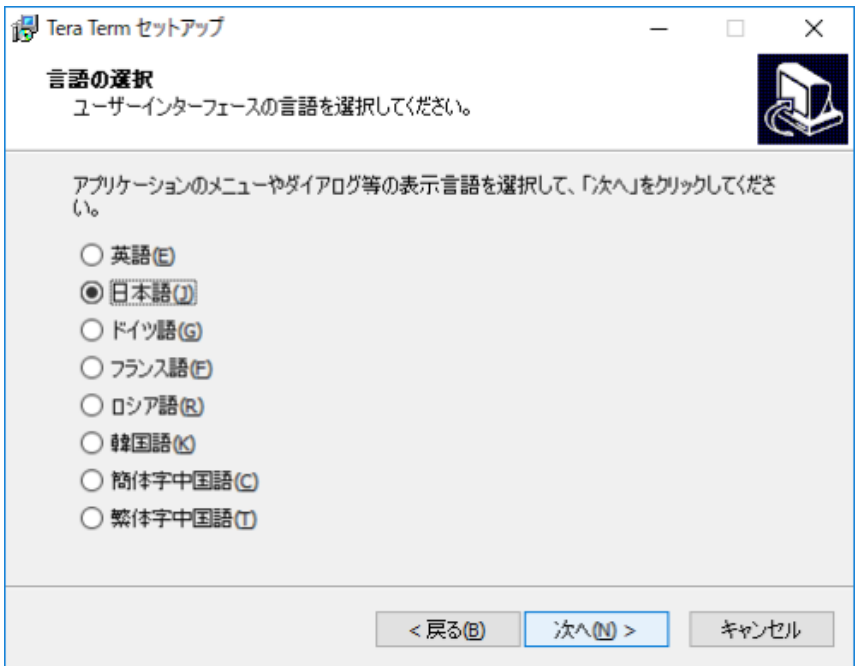

図 3.23　Tera Term の言語選択画面

「次へ」をクリックする。

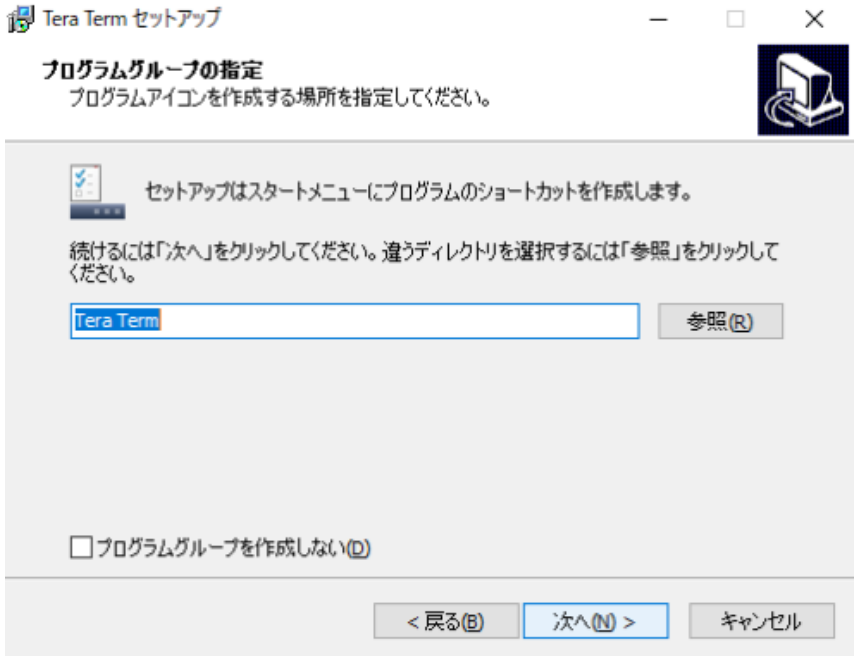

図 3.24　Tera Term のプログラムグループ指定画面

追加タスクの選択は必要に応じて選択し「次へ」をクリックする。

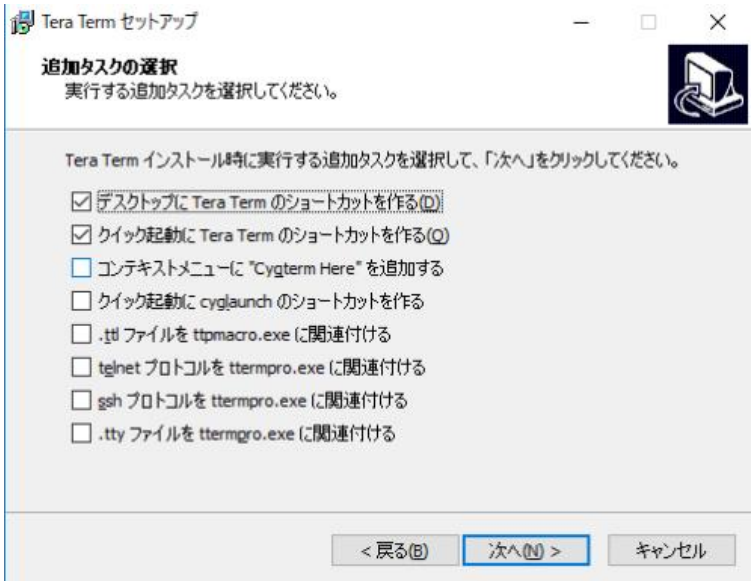

図 3.25　Tera Term の追加タスク選択画面

「インストール」をクリックする。

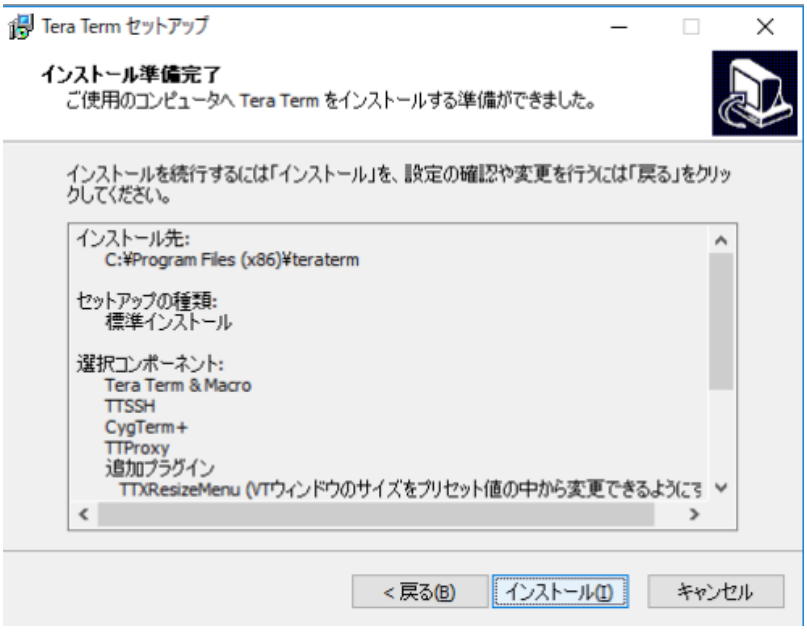

図 3.26　Tera Term のインストール準備完了画面

「完了」をクリックする。

図 3.27～図 3.29 に Raspberry Pi に接続する手順を示す。

Tera Term を起動後、ホストに設定した IP アドレスを入力し、「OK」をクリックする。

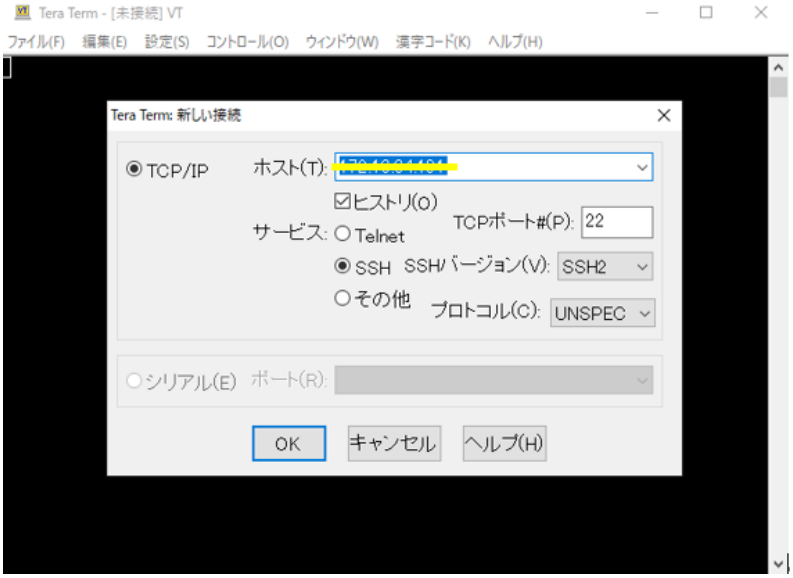

図 3.27　Tera Term の起動画面

「続行」をクリックする。

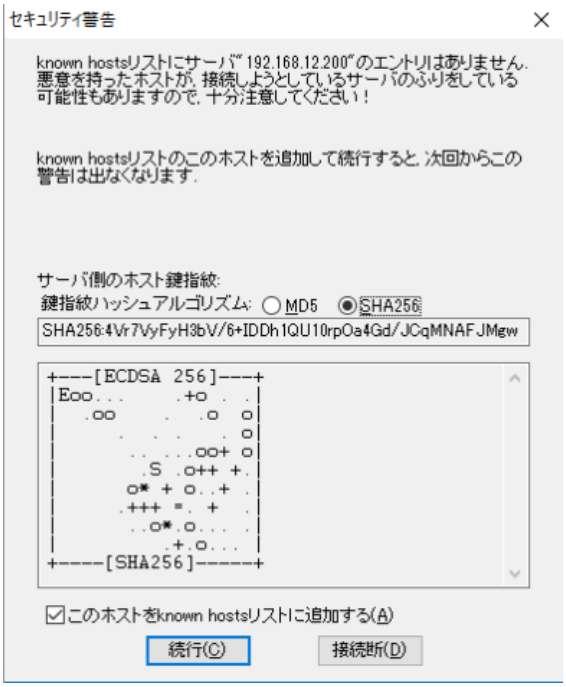

図 3.28　Tera Term のセキュリティ警告画面

設定したユーザ名とパスワードを入力し、「OK」をクリックする。

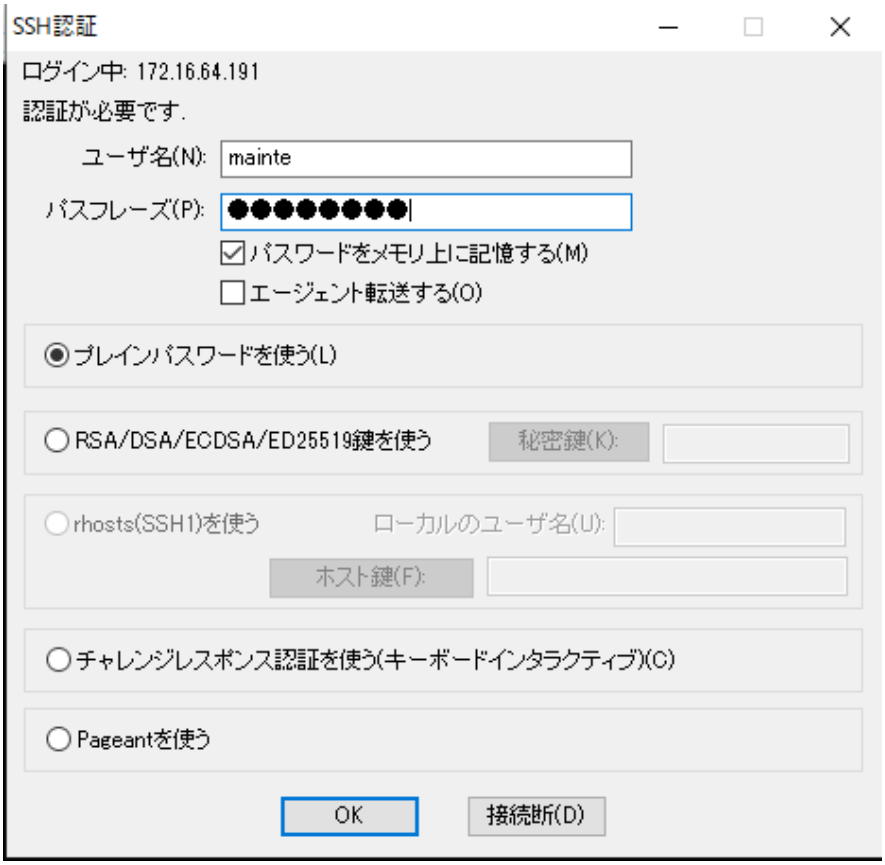

図 3.29　Tera Term の SSH 認証画面

3.2.2　ファイヤーウォール

別端末の Tera Term からミラー側にアクセスしようとするとホストに接続できないとエラー表示され、ファイヤーウォールに阻まれてしまう。

ファイヤーウォールとは、コンピュータやネットワークを守るガードのようなものである。

インターネットからのデータをチェックして、悪意のあるものは通さず、安全なものだけを通すことにより、コンピュータやネットワークが攻撃や不正なアクセスから守られる。

sudo ufw allow 22/tcp と sudo ufw allow 8080/tcp を実行するとステータスが図 3.30 のようになり、接続が可能になる。

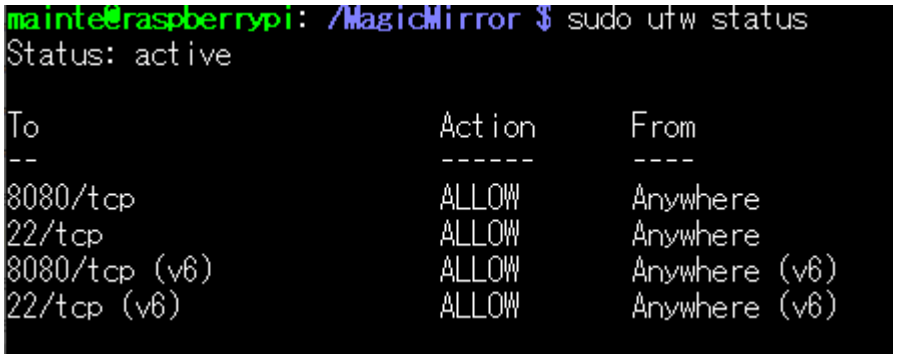

図 3.30　ファイヤーウォールのステータス

3.3 プロキシ設定

Raspberry Pi の方のコマンドプロンプロから下記のプロキシ設定を書き込む。

/etc/apt/apt.conf に下記コマンドを追加する。

Acquire::http::proxy"http://proxy.iwate-it.ac.jp:8080/";

Acquire::https::proxy"http://proxy.iwate-it.ac.jp:8080/";

Acquire::ftp::proxy"http://proxy.iwate-it.ac.jp:8080/";

Acquire::socks::proxy"http://proxy.iwate-it.ac.jp:8080/";

/etc/environment に下記コマンドを追加する。

http_proxy=http://proxy.iwate-it.ac.jp:8080/

https_proxy=http://proxy.iwate-it.ac.jp:8080/

ftp_proxy=http://proxy.iwate-it.ac.jp:8080/

.bashrc に下記コマンドを追加する。（mainte@ raspberrypi:~$にある）

export http_proxy=http://proxy.iwate-it.ac.jp:8080/

export https_proxy=http://proxy.iwate-it.ac.jp:8080/

export ftp_proxy=http://proxy.iwate-it.ac.jp:8080/

その後 NTP（Network Time Protocol）設定も行う。

これで時刻を現在のものにする。

/etc/systemd/timesyncd.conf に下記コマンドを追加する。

NTP=swsvbw01.iit.iwate-it.ac.jp

mainte@ raspberrypi:~$に戻り、下記コマンドを実行する。

sudo tymedatectl set-ntp true

sudo systemctl daemon-reload

sudo systemctl restart systemd-timesyncd.service

ソフトウェアアップデートも行う。

sudo apt update

sudo apt upgrade

第 4 章　システムの構築

4.1 スマートミラーのセットアップ

4.1.1　Magic Mirror2 の導入

スマートミラーのプラットフォームとなるソフトウェアの導入手順を以下に示す。

図 4.1 のコマンドで GitHub にある Magic Mirror2 のソースコードや情報をローカル環境に複製する。

```
# get sources
git clone https://github.com/MichMich/MagicMirror ↵
```

図 4.1　複数コマンド

4.1.2　MagicMirror_scripts の導入

MagicMirror_scripts を導入するための図 4.2 のコマンドを実行する。

```
# MagicMirrorのインストール
bash -c  "$(curl -sL
https://raw.githubusercontent.com/
sdetweil/MagicMirror_scripts/master/
raspberry.sh)"

# テスト
cd MagicMirror
cp config/config.js.sample config/config.js
npm start
```

図 4.2　導入と実行のコマンド

この時にプロキシのエラーが発生した。

そのため HTTP および HTTPS のプロキシ環境変数を設定し、MagicMirror_scripts を導入した。

```
export HTTP_PROXY=http://proxy.iwate-it.ac.jp:8080
export HTTPS_PROXY=http://proxy.iwate-it.ac.jp:8080
```

図 4.3　HTTP および HTTPS のプロキシ設定

MagicMirror_scripts 導入後、MagicMirror ディレクトリに移動する。

npm start のコマンドを実行し、MagicMirror を起動しようと試みるが electron が見つからないというエラーにより実行できない。

```
mainte@raspberrypi:~ $ cd MagicMirror
mainte@raspberrypi:~/MagicMirror $ npm start

> magicmirror@2.25.0 start
> DISPLAY="${DISPLAY:=:0}" ./node_modules/.bin/electron js/electron.js

sh: 1: ./node_modules/.bin/electron: not found
mainte@raspberrypi:~/MagicMirror $
```

図 4.4　npm start のエラー

Electron とは Electron とは Web 技術（HTML、CSS、JavaScript）を使用してデスクトップアプリケーションを開発するためのフレームワークである。

主な特徴として、クロスプラットフォーム（Windows、macOS、Linux）で動作するデスクトップアプリケーションを作成できる点が挙げられる。

クロスプラットフォームとは iOS、Android、Windows、macOS など異なる環境でも、同じ仕様のアプリケーションを動かすことが可能なフレームワークを指す。

指定フォルダ内に electron がないことを確認する。

図 4.5　ディレクトリ・ファイル一覧

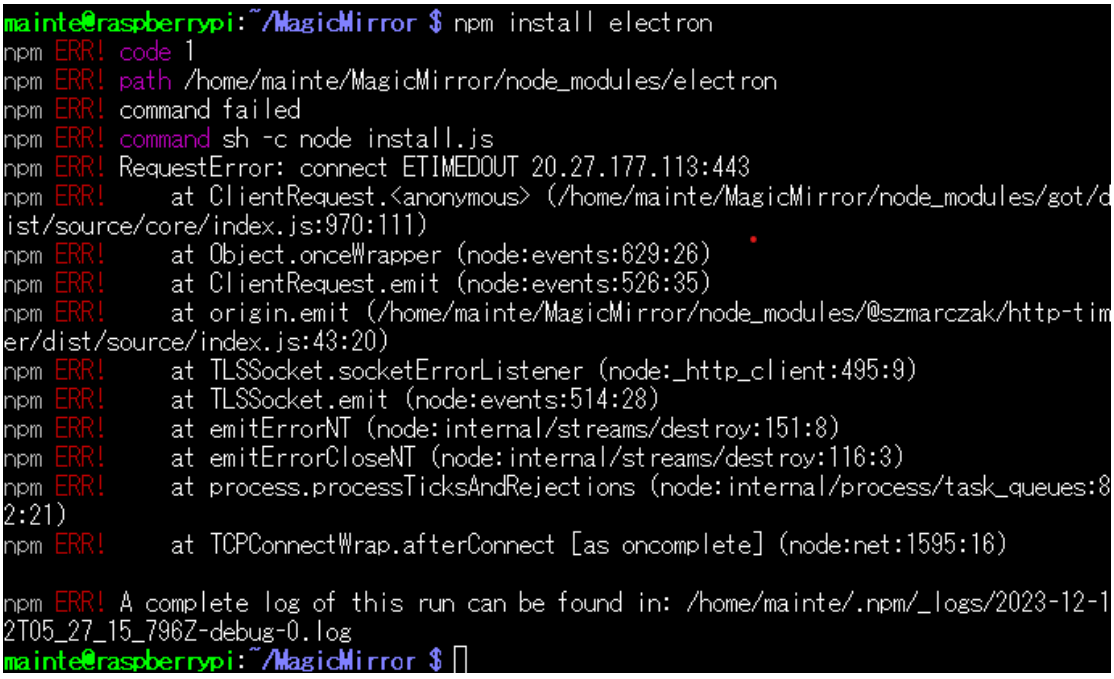

図 4.6　electron の導入エラー

npm install electoron コマンドで electron を導入しようと試みるがエラーにより実行できない。

調べたところ、プロキシの設定が原因でエラーが出ていることが分かった。

プロキシ設定のために下記のコマンドを実行する。

```
export ELECTRON_GET_USE_PROXY=true
export GLOBAL_AGENT_HTTPS_PROXY=http://proxy.iwate-it.ac.jp:8080
```

図 4.7　electron のプロキシ設定

このコマンドの 1 行目は Electron アプリケーションがプロキシを使用することの設定である。

2 行目はモジュールを使用して、アプリケーション全体で HTTPS 通信に対するプロキシを設定であり、これによりアプリケーション内のすべての HTTPS 通信が指定されたプロキシを経由して行われるという設定である。

その後 npm install electoron を実行すると electoron が追加できる。

4.1.3　MMM-Remote-Control の導入

Magic Mirror2 のモジュールをコマンドで制御するためのモジュールの導入を以下に示す。

①　　bash -c "$(curl -s https://raw.githubusercontent.com/Jopyth/MMM-Remote-Control/master/installer.sh)"のコマンドを実行する。

このコマンドは、指定された URL からスクリプトをダウンロードし、そのスクリプトが Bash に渡され、-c オプションによりそれがコマンドとして実行される。

②　　nano MagicMirror/config/config.js のコマンドを実行する。。

nano テキストエディタを使用して、MagicMirror プロジェクトの設定ファイルである config.js を変更するためのコマンド。

変更内容は図 4.8 の通り。

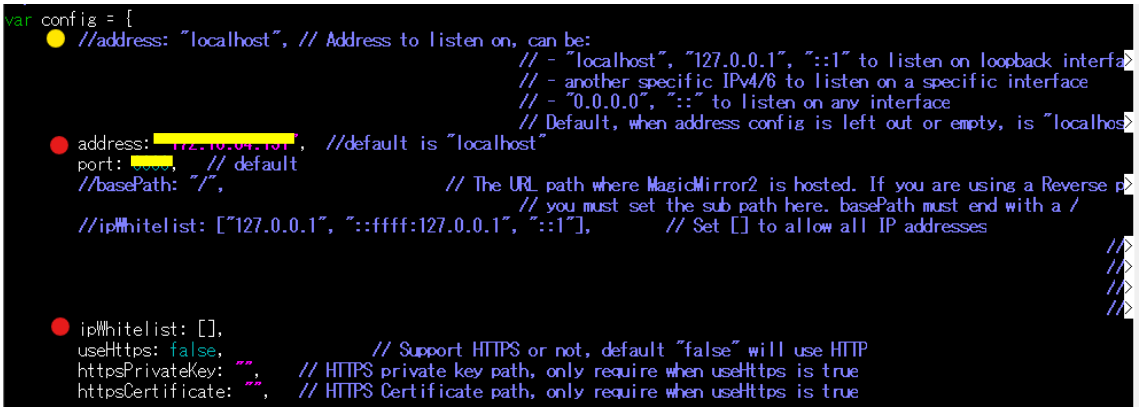

図 4.8　config.js の変更後

はじめに入力されていた address と basePath、ipWhitelist をコメントアウトし、使用している IP アドレス、ポート番号を入力。新しい ipWhitelist をつくる。

config の外、ファイルの最後に図 4.9 のものを書き込む。

```
{
        module: 'MMM-Remote-Control',
        // uncomment the following line to show the URL of the remote control on the mirror
        // position: 'bottom_left',
        // you can hide this module afterwards from the remote control itself
        config: {
                customCommand: {},  // Optional, See "Using Custom Commands" below
                showModuleApiMenu: true, // Optional, Enable the Module Controls menu
                // uncomment any of the lines below if you're gonna use it
                // customMenu: "custom_menu.json", // Optional, See "Custom Menu Items" below
                // apiKey: "",         // Optional, See API/README.md for details
        }
},
```

図 4.9　config.js に追加する設定

この部分は、MagicMirror² の config.js ファイルに追加される MMM-Remote-Control モジュールの設定を示している。

4.1.4　天気の設定

OpenWeatherMap とは、Web やモバイルアプリケーションの開発者に、現在の天候や予測履歴を含む各種気象データの無料 API を提供するオンラインサービスのことを指す。

このサービスを使用し、ディスプレイに天気の情報を表示させる。

図 4.10～図 4.12 にその手順を示す。

WindowsPC で OpenWeatherMap（https://openweathermap.org/）でアカウントを作成し、API キーを取得する。

図 4.10　OpenWeatherMap の API キー表示画面

OpenWeatherMap が提供している都市リストのファイルを、下記のコマンドで Raspberry Pi にダウンロードする。

wget https://bulk.openweathermap.org/sample/city.list.json.gz

下記のコマンドでファイルをダウンロードする。

gunzip city.list.json.gz

下記のコマンドで盛岡市の対応する ID を検索する。

grep "Morioka" -B1 -A2 city.list.json

図 4.11　盛岡市の ID 検索コマンドと検索結果

下記のコマンドで config.js ファイルのあるディレクトリに移動する。

cd /MagicMirror/config

下記のコマンドで config.js ファイルに API キーや ID を書き込む

nano config.js

図 4.12　天気表示の設定内容

変更内容は表 4.1 の通り

表 4.1　変更場所と変更内容

location	Morioka
locationID	盛岡市の対応する ID
apiKey	取得した API キー

4.2 ハンドジェスチャー

4.2.1　ソースコードのダウンロード

今回、ハンドサインを認識するプログラムは GitHub で公開されているものを WindowsPC にダウンロード使用した。(https://github.com/tech-life-hacking/SmartMirror)

GitHub とはソフトウェア開発プロジェクトやソースコードの共同作業を支援するためのウェブベースのプラットフォームのこと。

図 4.13~図 4.15 は GitHub からソースコードをダウンロードし、実際にプログラムを動かす手順を示す。

「Code」から「Dowload ZIP」を選択し zip ファイルをダウンロードする。

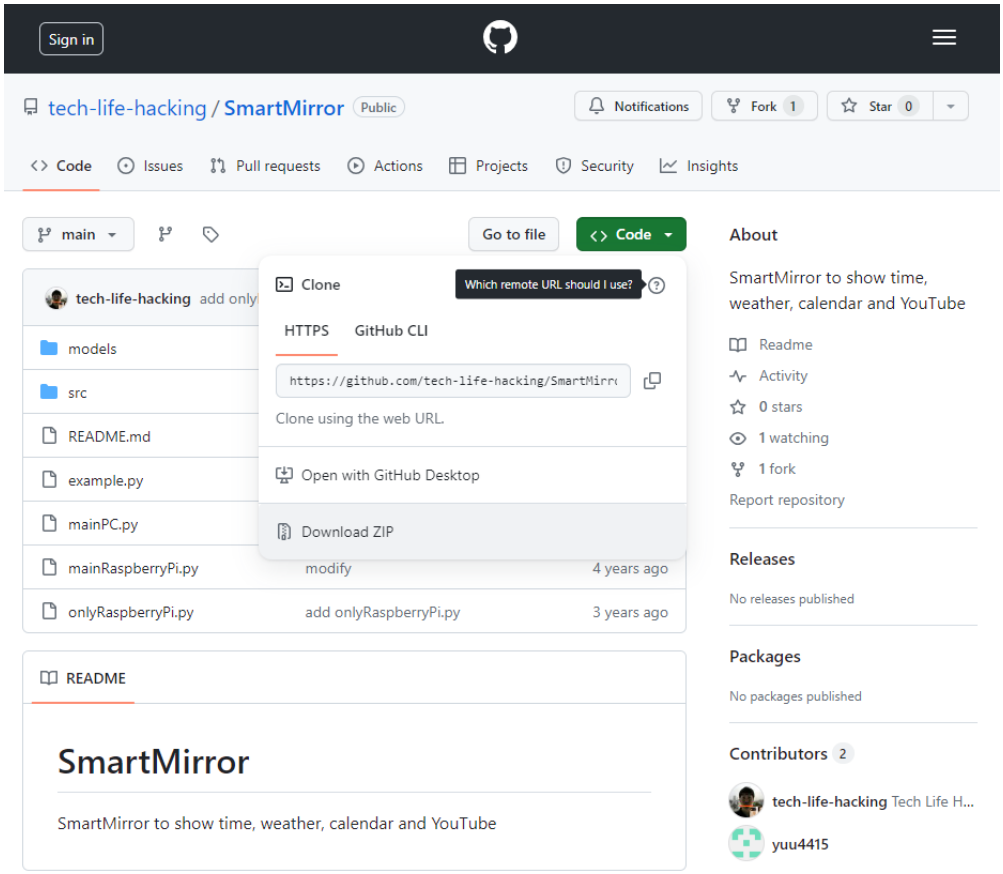

図 4.13　ハンドジェスチャーのソースコードのダウンロード画面

ファイルを解凍し、VSCode で解凍したファイルを開く。

動作を確認するため、example.py を実行する。(画像のようになれば正常に動作している)

図 4.14　example.py の実行結果

mainPC.py の中身を図 4.15 の通りに変更する。

```
147
148  def main():
149      cap = cv2.VideoCapture(0)
150      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
151          # s.connect((RaspberryPiIPAdress, 50007))
152          s.connect(('[redacted]', [redacted]))
```

図 4.15　mainPC.py のプログラム変更内容

RaspberryPiIPAdress：設定した IP アドレス

50007：設定したポート番号

mainPC.py を実行する。

4.2.2　プログラムの説明

図 4.16 は mainPC.py のディレクトリを示したものである。

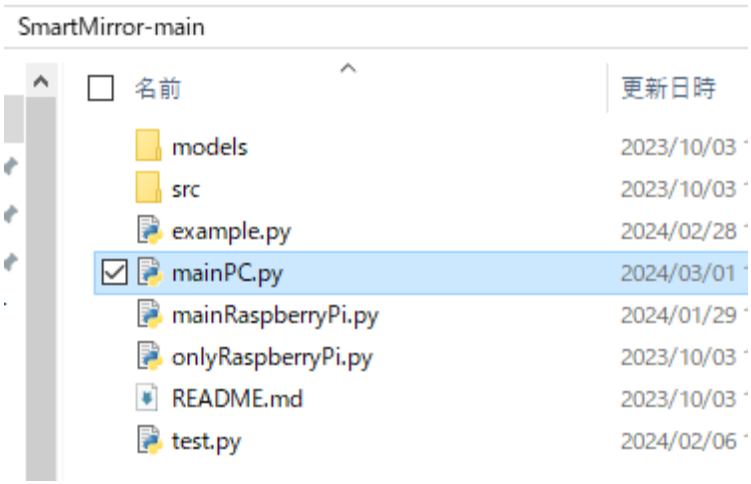

図 4.16　mainPC.py の保存場所

図 4.17～図 4.33 にハンドジェスチャー（mainPC.py）のプログラムを示す。

```
#!/usr/bin/env python

from __future__ import division

import cv2

import numpy as np

import socket

import struct

from src.hand_tracker import HandTracker

import vptree

from sklearn.preprocessing import normalize
```

図 4.17　mainPC.py の実行環境の設定部分

```
RaspberryPiIPAdress = 'YourRaspberryPiIPAdress'          #設定した IP アドレスを入力
WINDOW = "Hand Tracking"                                 #ウィンドウの名前を指定
PALM_MODEL_PATH = "models/palm_detection_without_custom_op.tflite"
                                                         #手のひら検出のモデルファイルパスを指定
LANDMARK_MODEL_PATH = "models/hand_landmark.tflite"
                                                         #手のランドマーク検出のモデルファイルパスを指定
ANCHORS_PATH = "models/anchors.csv"
                                                         #アンカーポイントのファイルパスを指定
MULTIHAND = True                                         #複数の手を検出
HULL = True                                              #凸包(※1)を表示
POINT_COLOR = (0, 255, 0)                                #ランドマークの色指定
CONNECTION_COLOR = (255, 0, 0)                           #線の色指定
HULL_COLOR = (0, 0, 255)                                 #凸包の色指定
THICKNESS = 2                                            #ランドマーク、線の太さ指定
HULL_THICKNESS = 2                                       #凸包(※1)の線の太さ指定
```

図 4.18　mainPC.py のプログラム設定やパラメータを定義する部分

※1 与えられた点をすべて包含する最小の凸多角形（凸多面体）のこと。

```
detector = HandTracker(                    #インスタンス作成
    PALM_MODEL_PATH,
    LANDMARK_MODEL_PATH,
    ANCHORS_PATH,
    box_shift=0.2,                         #手のひらの検出ボックスのシフト量指定
    box_enlarge=1.3                        #手のひらの検出ボックスの拡大率指定
)
max_size = 400                             #手のポーズデータを正規化（最大サイズ指定）
def get_pose(kp, box):
    x0, y0 = 0, 0
                #検出ボックスの原点の座標を指定（原点を画像の左上隅(0, 0)とする）
    box_width = np.linalg.norm(box[1] - box[0])
                #左上と右上の角のユークリッド距離(※2)を計算
    box_height = np.linalg.norm(box[3] - box[0])
                #左上と左下の角のユークリッド距離(※2)を計算
    x1 = ((kp[:, 0] - x0) * max_size) / box_width#スケーリングを行う(※3)
    #((ランドマークの x 座標 - 原点の y 座標) * 最大サイズ) / 検出ボックスの高さ
    y1 = ((kp[:, 1] - y0) * max_size) / box_height          #スケーリングを行う(※3)
    #((ランドマークの y 座標 - 原点の y 座標) * 最大サイズ) / 検出ボックスの高さ
    a = np.array([x1, y1])                 #x 座標と y 座標からなる配列 a を作成
    v = a.transpose().flatten()            #配列 a を転置し、1 次元配列に変換(※4)
    return normalize([v], norm='l2')[0]    #正規化してから返す
```

図 4.19　mainPC.py のインスタンスを作成とポーズデータの取得部分

※2 二点間の直線距離のこと。

※3 ランドマークのｘ座標、ｙ座標を、検出ボックスの高さを考慮してスケーリングする。

スケーリングとはシステムやサービスをより大規模に拡張すること。

※4 手のポーズデータを一次元のベクトルとして表現する。

```
indexes = ['swing', 'ok', 'index', 'three', 'thumbs',                #各ポーズの名前
            'palm_closed', 'palm_opened', 'peace', 'smile', 'fist']
poseData = np.array(          #(※5)
    [[0.18647491, 0.22245079, 0.1426551, 0.21015471, 0.10659113, 0.17722285, 0.07597828,
0.14441899, 0.04117138, 0.12755607, 0.13508488, 0.12111125, 0.12516799, 0.09906237,
0.13493404, 0.13594537, 0.14436085, 0.16026148, 0.16782215, 0.11746154, 0.15590905,
0.10302253, 0.1628355, 0.14403075, 0.16621838, 0.16670691, 0.1954005, 0.11966107, 0.18288804,
0.10100616, 0.18230077, 0.14286134, 0.18251303, 0.16901286, 0.22006123, 0.12585712,
0.21669258, 0.09424948, 0.21802386, 0.08854462, 0.22061957, 0.08287658],#swing
     [0.18434688, 0.23973041, 0.14691995, 0.22440134, 0.10763994, 0.19372458, 0.08462718,
0.15996101, 0.0919741, 0.13386023, 0.14644286, 0.13712727, 0.12869465, 0.10208318,
0.11398955, 0.10302045, 0.1063617, 0.11758749, 0.17654136, 0.12364753,
0.17334996,0.08817116, 0.16406605, 0.06903247, 0.15090216, 0.05939158, 0.2042279,
0.12392038, 0.20935623, 0.0892418, 0.20847576, 0.06383019, 0.20308884, 0.04911344,
0.22908612, 0.13730193, 0.23782973, 0.10968894, 0.24331368, 0.08895545, 0.24093,
0.07055891],#ok
     [0.20574305, 0.24668849, 0.17556497, 0.22605193, 0.16709151, 0.18426183,
0.1565399, 0.13807951, 0.14085615, 0.10864406, 0.13268977, 0.12998907, 0.10006527,
0.09176336, 0.07787868, 0.0682522, 0.06510728, 0.04197233, 0.16696938, 0.11354819, 0.1544783,
0.08169002, 0.15741036, 0.12510934, 0.16674335, 0.15652471, 0.19831311, 0.11199987,
0.18993277, 0.09362093, 0.18439568, 0.13545503, 0.18726359, 0.16281337, 0.22629688,
0.1143615, 0.21396741, 0.09988577, 0.20127536, 0.12347081, 0.19796878, 0.13589683],# index
```

図 4.20　mainPC.py の各ポーズデータを表す 2 次元の NumPy 配列部分①

[0.17428913, 0.22340391, 0.14914798, 0.20840863, 0.15048555, 0.18228617, 0.17473708, 0.1608064, 0.19158579, 0.14686246, 0.14201826, 0.13200931, 0.12749632, 0.10153376, 0.12069826, 0.07668609, 0.11883414, 0.05552163, 0.16612802, 0.1266895, 0.15665322, 0.08347182, 0.14908728, 0.0534117, 0.1459203, 0.03266382, 0.18750041, 0.13434874, 0.19842204, 0.1047664, 0.20909921, 0.07691436, 0.21895828, 0.05321266, 0.20351892, 0.15110564, 0.2007541, 0.13908943, 0.19469448, 0.15677244, 0.19387661, 0.17648656],# three

[0.18318445, 0.19968832, 0.14287076, 0.18600786, 0.11637019, 0.16152485, 0.08647456, 0.14779573, 0.05557882, 0.14432782, 0.14192424, 0.12375153, 0.13563914, 0.09971043, 0.14532968, 0.13380753, 0.15298971, 0.15120736, 0.1698368, 0.12254469, 0.16546489,0.09175989, 0.17154135, 0.12940182, 0.17324638, 0.14299075, 0.19456672, 0.12514232, 0.19334476, 0.09510231, 0.19214967, 0.13037139, 0.19076402, 0.1456383, 0.21474029, 0.13218189, 0.21210906, 0.10960651, 0.20567045, 0.13224842, 0.20259804, 0.14286903],# thumbs

[0.18403882, 0.24353707, 0.14810409, 0.22914967, 0.12417554, 0.18973013, 0.12259782, 0.14893835, 0.11102115, 0.11358896, 0.15214691, 0.1311901, 0.14718324, 0.08699801, 0.14754051, 0.06110825, 0.15094065, 0.04055923, 0.18231844, 0.12787095, 0.17720394,0.07999649, 0.17626061, 0.0497845, 0.17944943, 0.02243138, 0.2067777, 0.13276722, 0.20279329, 0.08606274, 0.20236166, 0.05536058, 0.20367333, 0.02794368, 0.23090097, 0.14432277, 0.22738936, 0.10891946, 0.22557151, 0.08665721, 0.22383213, 0.06285533],#palm_closed

図 4.21　mainPC.py の各ポーズデータを表す 2 次元の NumPy 配列部分②

```
            [0.15809512, 0.21386635, 0.13883535, 0.19705718, 0.13926474, 0.17525064,
0.16078032, 0.15910315, 0.18392389, 0.15248496, 0.13396172, 0.12849582, 0.12214344,
0.0915025, 0.11687374, 0.06437053, 0.11331773, 0.04076419, 0.15858937, 0.13062589,
0.17076254,0.08365408, 0.17719568, 0.0519051, 0.181212, 0.02521239, 0.17802356, 0.1422498,
0.19014316, 0.12382668, 0.18559735, 0.14721359, 0.18039312, 0.16166485, 0.19397253,
0.15961269, 0.20374444, 0.15435946, 0.1943716, 0.16737686, 0.18299277, 0.17574173],#peace
            [0.18620316, 0.20848493, 0.1534262, 0.21623465, 0.1260341, 0.19782164, 0.09503594,
0.19968312, 0.06283229, 0.20630055, 0.13811894, 0.12506329, 0.11950041, 0.08752148,
0.11655949, 0.06574876, 0.11584203, 0.04994573, 0.167047, 0.11787352, 0.17084832,0.07968003,
0.16905089, 0.10314258, 0.16184104, 0.12502938, 0.19173373, 0.12087494, 0.1971879,
0.08911833, 0.19370474, 0.116432, 0.185174, 0.13954724, 0.21385323, 0.12904313, 0.22037847,
0.1107641, 0.21645382, 0.12715677, 0.20674225, 0.14231086],# smile
            [0.16870468, 0.23211507, 0.12977799, 0.20814243, 0.09943, 0.16382496, 0.1123175,
0.12978834, 0.14672848, 0.13607521, 0.13753864, 0.13317615, 0.14097033, 0.09809432,
0.14065007, 0.12935257, 0.14333271, 0.14504507, 0.16329045, 0.12753517, 0.16547387,
0.08546145, 0.16095263, 0.12244097, 0.15828503, 0.13600505, 0.18647549, 0.13310245,
0.19047755, 0.09372779, 0.18434687, 0.12874961, 0.18357106, 0.14497022, 0.20711118,
0.14537352, 0.20915552, 0.11293839, 0.20329643, 0.13409531, 0.20128276, 0.14754838]])#fist
```

図 4.22　mainPC.py の各ポーズデータを表す 2 次元の NumPy 配列部分③

※5poseData は、各ポーズに対するポーズデータを表す 2 次元の NumPy 配列で、
各列はそのポーズの特徴量を示す。

```
def similarity(v1, v2):          #v1 と v2 の類似度を計算
    return np.dot(v1, v2) / (np.linalg.norm(v1) * np.linalg.norm(v2))
```

図 4.23　mainPC.py のポーズベクトル間の類似度を計算する部分

2 つのベクトルの内積をそのベクトルのノルムの積で割った値を返す。これにより、2 つのベクトルがどれだけ同じ方向を向いているかが測定される。類似度は -1 から 1 の範囲になるため、1 に近いほど類似しており、-1 に近いほど逆向きの関係にある。

```
def cosineDistanceMatching(poseVector1, poseVector2):
    cosineSimilarity = similarity(poseVector1, poseVector2)
    distance = 2 * (1 - cosineSimilarity)
    return np.sqrt(distance)
```

図 4.24　mainPC.py のポーズベクトル間の距離を計算する部分

2 つのポーズベクトル poseVector1 と poseVector2 の間のコサイン距離(※6)を計算する。具体的には、類似度を計算し、それを距離に変換する。距離は、類似度が 1 に近い場合は 0 に近く、類似度が -1 に近い場合は 2 に近くなる。そのため、距離が 0 に近いほどポーズベクトルは類似しており、2 に近いほど類似していない。

※6 ふたつのベクトル間の角度の違いを測定する方法である。
ふたつのベクトルの間の角度が小さいほど距離が小さくなる。距離が 0 に近い場合、
ベクトルは同じ方向を向き、距離が大きいほど、ベクトルは異なる方向を向く。

```
idx_m = np.mean(poseData, axis=1)  #手のポーズを表すベクトルの平均値を
                                                計算し、idx_m に格納
tree = vptree.VPTree(poseData, cosineDistanceMatching)
                                                # poseData を元に VP ツリーを構築
#手のポーズを表すベクトル間の接続を定義
#           8    12   16   20
#           |    |    |    |
#           7    11   15   19
#      4    |    |    |    |
#      |    6    10   14   18
#      3    |    |    |    |
#      |    5---9---13--17
#      2     ¥              /
#       ¥     ¥           /
#         1     ¥       /
#           ¥     ¥    /
#             ------0-
```

図 4.27　mainPC.py の Vantage-Point Tree を構築する部分

手のポーズデータを含む配列 poseData を使用して、Vantage-Point Tree を構築し、手のポーズを表すベクトル間の接続を定義する。

```
connections = [
    (5, 6), (6, 7), (7, 8),          #指の先端から手首にかけての関連付け
    (9, 10), (10, 11), (11, 12),     #人差し指の関連付け
    (13, 14), (14, 15), (15, 16),    #中指の関連付け
    (0, 5), (5, 9), (9, 13), (13, 17),  #手首から指の先端にかけての関連付け
    (0, 9), (0, 13)                  #手首から中指の先端にかけての関連付け
]
```

図 4.28　mainPC.py の手のジョイント間の接続を定義する部分

```
#手の外形を形成する手首から指の先端までの連結関係を定義
pseudo_hull_connections = [                                    #手の輪郭を定義
    (0, 17), (17, 18), (18, 19), (19, 20), (0, 1), (1, 2), (2, 3), (3, 4)]
#手首から凸包を構成する点までの接続関係を定義
hull_connections = [(4, 8), (8, 12), (12, 16), (16, 20)]       #凸包を定義


if HULL:
    #True：hull_connections に追加
    hull_connections += pseudo_hull_connections
else:
    #False：connections に追加
    connections += pseudo_hull_connections
```

図 4.29　mainPC.py の手の輪郭や凸包（convex hull）を定義するための接続情報を設定する部分

```
def handgesture(frame):
    image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)          # RGB 形式に変換
    hand = detector(image)                                  # RGB 形式の画像から手を検出
    if hand[0] is not None and len(hand) > 0:
        box = hand[2]                   #手の境界ボックスを取得
        bkp = hand[4]                   #手のランドマークを取得
        kp = get_pose(bkp, box)         #bkp、box を使用して手のポーズを推定
        res = tree.get_n_nearest_neighbors(kp, 1)[0]
                                        #推定されたポーズに最も近いポーズデータを検索・取得
        a = np.mean(res[1])             #最も近いポーズデータの距離の平均値を計算

        if res[0] < 0.2:        #取得したジェスチャーの姿勢ベクトルが 0.2 以下の場合
            idx = np.where(idx_m == a)[0][0]      #ジェスチャーのインデックス取得
            return idx          #検出されたジェスチャーのインデックスを返す

idxlist = []                    #idx を保持するための空のリスト
```

図 4.30　mainPC.py のフレームからジェスチャーを検出するための関数を定義する部分

```
def idxcounts(idx):
    idxlist.append(idx)       #idx を idxlist に追加

    if len(list(set(idxlist))) == 1 and len(idxlist) == 3:      #要素数が 3
        return idx                                              #idx を返す

    elif len(list(set(idxlist))) == 1 and len(idxlist) < 3:     #要素の種類が 1、かつ
                                                                #要素数が 3 未満
        return 0                                                #0 を返す

    elif len(idxlist) > 3:                                      #要素数がより大きい
        idxlist.clear()                                         # idxlist をクリア
        idxlist.append(idx)                                     #再び idx を追加
        return 0                                                # 0 を返す

    else:                                                       #要素数が 3 以下
        idxlist.clear()                                         # idxlist をクリア
        idxlist.append(idx)                                     #再び idx を追加
        return 0                                                # 0 を返す
```

図 4.31　mainPC.py の手のジェスチャーのインデックスをカウントおよび管理部分

```
def main():
    cap = cv2.VideoCapture(0)          #ビデオキャプチャデバイスを初期化
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                                # TCP/IP ソケット作成
        #設定した IP アドレスとポート番号を入力
        s.connect(('RaspberryPiIPAdress', ポート番号)) #(※7)
        while True:
            _, frame = cap.read()                    #カメラからフレームを読み込み
            try:
                idx = handgesture(frame)             #ジェスチャーを検出、idx に格納
                idx = idxcounts(idx)                 #検出ジェスチャーをカウント
                if indexes[idx] == 'fist':           #fist を検出した場合
                    s.sendall(b'TurnYouTube')
                        # YouTube を操作するコマンドを Raspberry Pi に送信(※8)
                elif indexes[idx] == 'palm_opened':  #palm_opened を検出した場合
                    s.sendall(b'TurnTV')
                        # TV を操作するコマンドを Raspberry Pi に送信
                else:                                #それ以外の場合
                    s.sendall(b'No Detected')
                        #メッセージを Raspberry Pi に送信
            except:          #例外が発生した場合は、処理を続行
                pass
```

図 4.32　mainPC.py のカメラ映像キャプチャ・ジェスチャーから対応アクションを実行する部分

※7 RaspberryPiIPAdress の部分にあらかじめ設定した IP アドレスを入力する。

IP アドレスの隣にポート番号も入力する。

※8 本研究では YouTube を操作するモジュールは導入していないため、動作しない。

```
            cv2.imshow('frame', frame)                    #画像をウィンドウに表示


            if cv2.waitKey(1) & 0xFF == ord('q'):         # q キーが押されたら
                break                                     #ループを抜けてプログラム終了


        cap.release()       #カメラキャプチャデバイスを解放して、プログラム終了


if __name__ == "__main__":          #スクリプトが直接実行された場合
    main()                          # main() 関数を呼び出す
```

図 4.33　mainPC.py のカメラからキャプチャしたフレームを表示する部分

4.3 服装提案システム

4.3.1　背景の切り抜き

ファイルパス：P:¥卒業研究¥Python_gazou¥test.py

OS：Windows

```
from rembg import remove
import cv2


def main():
    input_path = 'bobo.jpg'
    output_path = 'output_image.png'
    #  入力画像の読み込み
    input_image = cv2.imread(input_path)
    #  背景の除去
    output_image = remove(input_image)
    #  出力画像の保存
    cv2.imwrite(output_path, output_image)
#メイン関数の呼び出し
if __name__ == "__main__":
    main()
```

図 4.34　背景切り抜きのプログラム

rembg と OpenCV ライブラリを使用して画像の背景を除去するプログラムである。

rembg モジュールから remove 関数を、OpenCV モジュールから cv2 をインポートしている。

図 4.34 のプログラムは、実際には main.py 内に書き込み使用する。

図 4.35　背景の切り抜き

4.3.2　画像の結合

```
# 最大の高さを計算
max_height = max(input_image1.shape[0], input_image2.shape[0])


# 入力画像を最大の高さにリサイズ
input_image1_resized = cv2.resize(input_image1, (input_image1.shape[1], max_height))
input_image2_resized = cv2.resize(input_image2, (input_image2.shape[1], max_height))


# 背景を削除
output_image1 = remove(input_image1_resized)
output_image2 = remove(input_image2_resized)


# 画像 1 と画像 2 を横に結合
combined_image = np.concatenate([output_image1, output_image2], axis=1)


#結合された画像をターゲットの幅にリサイズ
target_width = 700
ratio = target_width / combined_image.shape[1]
combined_image_resized = cv2.resize(
    combined_image, (target_width, int(combined_image.shape[0] * ratio)))
```

図 4.37　画像の結合のプログラム（main.py 内から抜粋）

① 2 つの画像のうち、高さが最大のものを max_height に設定する。

② max_height に 2 つの画像のサイズを変更する。

③ remove 関数を使って、2 つの画像の背景を削除する。

④ concatenate(コンカットネイト)という関数を使って 2 つの画像を結合し、1 つの画像を生成する。

⑤ 新しく生成した画像の幅を指定し、その指定した幅をもとに画像の比率を計算する。

⑥ 計算した比率に画像のサイズを変更する。

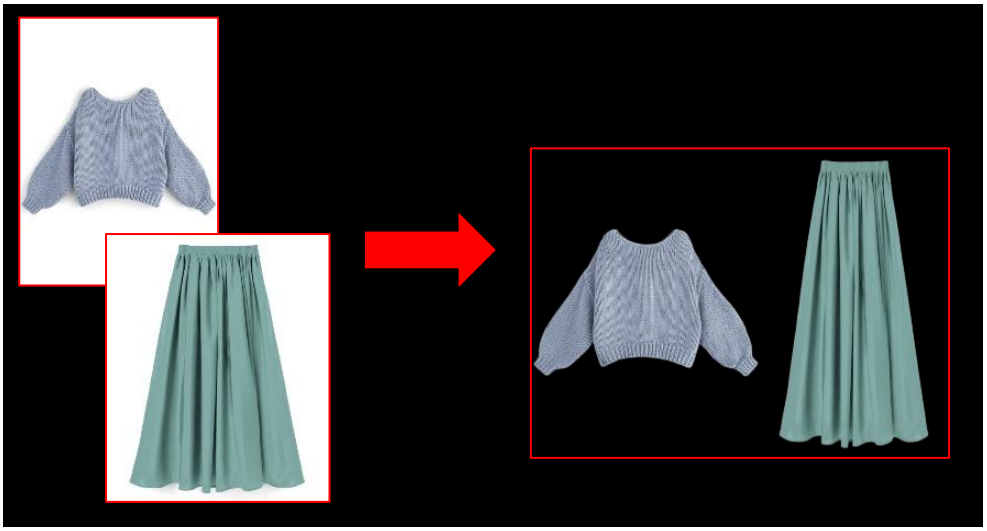

図 4.38　画像を結合

4.3.3　プログラムの説明

ファイルパス：P:¥graduation_research¥Python_gazou¥main.py

OS：Windows

ファイル構成：Python_gazou
- input_image1
- input_image2
- output_image
- main.py

```
from rembg import remove
import numpy as np
import cv2
import uuid
import os
import random


def remove_background(image):
    return remove(image)


def main():
    input_folder1 = 'P:¥¥graduation_research¥¥Python_upload¥¥input_image1'
    input_folder2 = 'P:¥¥graduation_research¥¥Python_upload¥¥input_image2'


    image_files1 = [f for f in os.listdir(input_folder1) if f.lower().endswith(('.png', '.jpg', '.jpeg'))]
    image_files2 = [f for f in os.listdir(input_folder2) if f.lower().endswith(('.png', '.jpg', '.jpeg'))]
    if not image_files1 or not image_files2:
        print("画像がありません。")
        return
```

図 4.39　main.py 内のファイルパス指定、リスト取得部分

input_folder1 と input_folder2 にそれぞれのファイルパスを指定する。(input_folder1 にはトップス、input_folder2 にはボトムスが保存されている。)

① os.listdir とリスト内包表記を使用して、各フォルダから対象の画像ファイル(.png、.jpg、.jpeg 拡張子を持つファイル）のリストを取得している。

② image_files1 または image_files2 が空の場合に、「画像がありません。」と表示してプログラムの実行を中断する処理を行っている。

```
# 画像フォルダからランダムに画像ファイル名を取得
    input_image_filename1 = random.choice(image_files1)
    input_image_filename2 = random.choice(image_files2)


    #ファイルパスの生成
    input_path1 = os.path.join(input_folder1, input_image_filename1)
    input_path2 = os.path.join(input_folder2, input_image_filename2)
```

図 4.40　main.py 内の画像読み込み

① random.choice を使用して、それぞれのフォルダ（input_folder1 および input_folder2)からランダムに画像ファイル名を選択する。

② os.path.join を使用して、各フォルダのパス（input_folder1 および input_folder2）と選択されたランダムな画像ファイル名を結合して、ファイルの絶対パスを生成する。

③ cv2.imread を使用して、生成されたファイルパスから画像を読み込む。

```
# 最大の高さを計算
max_height = max(input_image1.shape[0], input_image2.shape[0])


# 入力画像を最大の高さにリサイズ
input_image1_resized = cv2.resize(input_image1,
                                  (input_image1.shape[1], max_height))
input_image2_resized = cv2.resize(input_image2,
                                  (input_image2.shape[1], max_height))


# 背景を削除
output_image1 = remove(input_image1_resized)
output_image2 = remove(input_image2_resized)


# 画像 1 と画像 2 を横に結合
combined_image = np.concatenate([output_image1, output_image2], axis=1)


#結合された画像をターゲットの幅にリサイズ
target_width = 700
ratio = target_width / combined_image.shape[1]
combined_image_resized = cv2.resize(
    combined_image, (target_width, int(combined_image.shape[0] * ratio)))
```

図 4.41　main.py 内の画像の切り抜き、画像の結合

図 4.34　背景切り抜きのプログラムと図 4.37　画像の結合のプログラムの処理を行う。

```
# uuid を使用して一意の識別子を生成
    unique_id = str(uuid.uuid4())[:8]
    # 出力フォルダのパスを指定
    output_folder = 'P:¥¥graduation_research¥¥Python_upload¥¥output_image'
    # 出力ファイル名を作成
    output_path = os.path.join(output_folder, f'output_image_{unique_id}.png')
```

図 4.42　main.py 内の UUID 生成

① uuid.uuid4()を使用して一意の UUID（Universal Unique Identifier）を生成し、[:8]を使って最初の 8 文字を取り出す。

② 出力先のフォルダパス（output_folder）と、生成した一意の識別子を組み込んだ出力ファイルのパス（output_path）を指定している。

図

```
        # 透明な背景の出力画像を保存
        os.makedirs(output_folder, exist_ok=True)
        cv2.imwrite(output_path, combined_image_resized)

        #画像を表示
        saved_output_image = cv2.imread(output_path)
        cv2.imshow('Saved Output Image', saved_output_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

if __name__ == "__main__":
        main()
```

図 4.43　main.py 内の出力画像の保存、画像の出力

① s.makedirs を使用して、出力フォルダが存在しない場合に作成する。

② cv2.imwrite を使用して、背景が除去され、リサイズされた画像（combined_image_resiz)を指定されたファイルパスに保存する。

③ cv2.imread を使用して、保存された画像を読み込み、cv2.imshow で表示する。

④ cv2.waitKey(0)は、ユーザーが何かキーを押すまで待機する処理する。

⑤ cv2.destroyAllWindows()で、すべての OpenCV ウィンドウを閉じる。

第5章　おわりに

5.1 考察

今回の研究では、初めてRaspberry Piを使用したので、開発環境を設定するのに大変苦戦した。プロキシやネットワークに関するエラーが多く、修正に多くの時間を費やしてしまった。

5.2 成果と課題

そのため、ディスプレイには日付、時間、天気を表示できたが、予定やニュースを表示させることはできなかった。また、画像の背景切り抜きと、画像を組み合わせて表示させるプログラムは作成できた。しかし、それをRaspberry Piに組み込むことはできなかったため、服装を提案するシステムは導入できなかった。さらに、ハンドジェスチャーの認識は可能だが、Raspberry Pi への接続に問題があったため、ハンドサインによるディスプレイの画面操作はできなかった。

5.3 まとめ

研究を通して、プロキシやネットワーク環境の重要さを改めて理解した。また、背景切り抜き等のプログラム作成は、授業で学んだことを活かし、スムーズに進めることができたと感じている。

第 6 章　参考文献

AI で Hand Gesture 対応のスマートミラー！ – 機能とセットアップ –

https://www.techlife-hacking.com/?p=446

AI で Hand Gesture 対応のスマートミラー！ – 作成編 –

https://www.techlife-hacking.com/?p=479

Raspberry Pi の Magic Mirror: 完全なインストール ガイド

https://raspberrytips.com/magic-mirror-guide/

Proxy 環境で Electron インストールしようとしたら隠れた罠があったんです

https://qiita.com/LuckyRetriever/items/2f377b1ce34f7d12903c

【Python】画像の簡単切り抜き【Rembg】

https://plantprogramer.com/rembg/

Windows への Visual Studio Code のインストール方法

https://www602.math.ryukoku.ac.jp/Prog1/vscode-win.html

Raspberry Pi はじめての初期セットアップ(Debian 12：bookworm 版)

https://www.ingenious.jp/articles/howto/raspberry-pi-howto/raspberry-pi-basic-setup-bookworm/

【ゼロからわかる】Teraterm のインストールと使い方

https://eng-entrance.com/teraterm-install