**植物の自立化による自己育成システム**

**引き継ぎ書　ソフト編**

作成者：4BDA1208　田中達也

1. **目次**
2. 目次
3. ロボットのコンセプト
4. システムの仕様と必要なライブラリ
   1. Raspberry Pi のセットアップ
      1. SDカードのフォーマット
      2. SDカードにUbuntuMateをインストール
      3. Raspberry Pi 3でUbuntu Mateを起動
      4. パッケージ、ソフトの更新
      5. ライブラリの導入
      6. Raspberry pi のコンフィグ
      7. SSHで他PCからログインできるようにする
   2. Raspberry PiとArduinoを接続
5. 各プログラムの説明
   1. Arduinoのプログラム
   2. Raspberry Pi側のプログラム
      1. 初期設定
      2. スタート、ゴール地点設定
      3. Astar() Astarアルゴリズム
      4. 移動　main\_loop()
         1. read\_counter\_from\_arduino() エンコーダ読み取り
         2. calc\_odometry() オドメトリ計算
         3. move\_judgement() 移動判定
         4. calc\_rotation() 回転制御
         5. calc\_straight() 直進制御
         6. run\_pwm() PWM変換
      5. nextpointserch() 次の目的地を探す
         1. move()　環境情報を基に次の目的地を設定する
      6. Datap2p.py Paramiko関連の通信に関する関数
         1. ClientSetting() Paramikoのクライアントを作成、設定する
         2. SendData() ファイルを送信
         3. RecivedData() ファイルを受信
         4. PosRead() 相手の現在地取得
         5. Closing() 終了処理
      7. Nodesetting.py ノード管理に関する関数
         1. Node\_create() ノード作成
         2. Map\_view() マップの状態を表示
         3. Data\_write() データを書き込む
         4. Data\_read() データの読み込み
         5. Sensor\_read() センサの読み取り
         6. Data\_show() ノード情報を表示させる
         7. Dht11.py 温湿度センサの便利なモジュール
      8. \_\_init\_\_.py moduleディレクトリ内にある関数を使えるように
6. 現状確認している不具合
7. 今後の展望
8. **ロボットのコンセプト**

本研究のコンセプトは、複数の自律移動ロボットによって周辺環境を取得して、その環境情報を共有して植物の育成に適した位置へと能動的に移動するといったものです。前任者である黒木先輩が単体で周辺環境を取得し、移動するシステムを実現させていたので、これを複数で実現させようとしたものです。

　今回の研究では二台間での共有をさせますが、将来的には台数を増やすことも検討しています。

1. **システムの仕様と必要なライブラリ**

本システムは表1のような仕様となっており、システムの稼働に必要なライブラリは表2の通りです。

表 1 システムの仕様

|  |  |
| --- | --- |
| 使用するマイコン | Raspberry pi 3 B Model(主な制御)  Arduino Mega(エンコーダの読み取り) |
| OS | Ubuntu Mate 16.04.2 LTS  (Ubuntu Mate for the Raspberry Pi 3) |
| 開発言語 | Python 2.7 |
| 用いる通信方式 | SPI (A/Dコンバータ⇔Raspberry Pi)  Serial (Raspberry Pi ⇔ Arduino)  ssh (Raspberry Pi[A] ⇔ ルータ ⇔ Raspberry Pi[B]) |
| Raspberry Piの名前、IPアドレス、パスワード | User：pipipi  Password：oppai  IP：192.168.0.81 |
| User：pipipipi  Password：oppai  IP：192.168.0.90 |

表 2 稼働に必要なライブラリ

|  |  |
| --- | --- |
| **gpiozero** | A/Dコンバータの値を読み取る |
| **DHT11** | 温湿度センサの値を読み取る |
| **datetime** | 日付や時間の取得、表示形式の指定 |
| **Paramiko** | 二台間の通信に用いる |
| **ssh** | 他のクライアントに色々できる。 Paramikoで使う |
| **scp** | ファイルの受け渡しに用いる　デフォルトで入ってるかも？ |

* 1. **Raspberry Piのセットアップ**

Raspberry Pi 3 B ModelにUbuntu Mate を0から導入する際の手順を示します。次のサイト(<http://denshi.blog.jp/raspberry_pi3/ubuntu_mate>)を参考にして導入したので上手く行かないときはここを見ながら導入すると解決するかもしれません。

* + 1. **SDカードのフォーマット**

　SDFormatterというソフトを用いてUbuntu OSを入れるSDカードのフォーマットを行います。Raspberry PiにはmicroSDしか刺さりませんが、研究室のPCにはmicroSDが刺さる奴が居ないのでmicroSD→SDカードのコンバータを使ってください。Raspberry Piと書かれた箱の中にあると思います。

研究室にあるThinkpadのPCにならSDカードが刺さると思うのでそのPCを用いてセットアップを行うようにしてください。SDカードのフォーマットの手順は以下の通りです。

1. SDFormatterをインストール、起動(名前で検索すれば出てくる)
2. SDカードをPCに刺してSDカードのドライブ名を確認
3. 「Drive:」を②で確認したドライブ名にする
4. [オプション設定]→[消去設定]をイレースフォーマットに→[OK]
5. フォーマットを選択してOKボタン

　フォーマットする容量にも依存しますが一時間程度かかるので待ってる間に次章で用いる物をダウンロードしておくと良いと思います。

* + 1. **SDカードにUbuntu Mateをインストール**

　Ubuntu Mate for the Raspberry Pi 3(<https://ubuntu-mate.org/raspberry-pi/>)をダウンロードします。上記ページより

[Download Ubuntu Mate]→右下の[Raspberry Pi]→[16.04.2(Xenial)]→[ubuntu-mate-16.04.2-desktop-armhf-raspberry-pi.img.xz]の順にクリックするとダウンロードできると思います。若干ページのUIが変わってる可能性がありますが基本的にはこれで大丈夫です。拡張子が.torrentの物をダウンロードしないように注意してください。ダウンロードが完了したら解凍してimgファイルがあるか確認してください。

　次に、WIN32DiskImagerというソフトをインストールして起動します。検索すれば出ると思います。前章でフォーマットが終わってないなら終わるまで待ちます。

　起動したら以下の手順でSDカードにUbuntu Mateを書き込みます。

1. Image Fileに先ほど解凍したUbuntu Mateのimgファイルを選択
2. Deviceが書き込みたいSDカードになっているか確認
3. Writeを選択（警告文が出てきてもそのままOK）

　　　これでSDカードにUbuntu Mateが書き込めます。これも少々時間がかかります。

* + 1. **Raspberry Pi 3でUbuntu Mateを起動**

　ここからマウスとキーボードが必要になるので用意してUSBに挿しておいてください。その後は以下の手順通りです。

1. SDカードコンバータからmicroSDを抜き出し、Raspberry Piに挿す
2. HDMIケーブルをRaspberry Piとモニタに接続する
3. 電源ケーブルを接続する
4. 言語を日本語、タイムゾーンを日本、キーボードレイアウトを日本語に設定
5. 名前とパスワードは各自分かりやすい物にしてください。後で使います

以上の手順を行うとインストールが始まります。少しだけ時間がかかります。

* + 1. **パッケージ、ソフトの更新**

　Ubuntuのインストールが完了したら、研究室のWifiに接続してください。その後、ターミナルで以下のコマンドを順に実行してください。

|  |
| --- |
| sudo apt-get update  sudo apt-get upgrade |

これで各種ソフトが最新の物になります。ネットの速度にもよりますが軽く一時間はかかるので気長に待ちましょう。これを行うとFirefoxが開けなくなる相性問題（？）があるので注意してください。

* + 1. **ライブラリの導入**

　先ほど挙げた別途導入が必要なライブラリをインストールします。パッケージは既に最新の物になっているので大丈夫だと思いますが、もし上手く行かないのであれば3.1.4で使ったコマンドを打ってください。

**・pip**

これが無いとParamikoがインストールできません

|  |
| --- |
| sudo apt-get install python-pip |

**・git**

今回はDHT11をインストールするために使います。

|  |
| --- |
| sudo apt-get install git |

**・gpiozero**

A/Dコンバータから値を読み取る際に使います。

|  |
| --- |
| sudo apt-get install python-gpiozero |

**・ssh**

Paramikoの前提ライブラリの他、別のPCから作業をするときに使います。

|  |
| --- |
| sudo apt-get install openssh-server  sudo apt-get install ssh |

**・Paramiko**

前提モジュールとしてlibffi-devが必要になります。必ずインストールした後にParamikoをインストールしてください。結構ここでエラー吐かれやすいのでpipのアップデートやその他の必要なモジュール等があるかどうかをチェックしておくと良いかもしれません

|  |
| --- |
| sudo apt-get install libffi-dev  sudo pip install paramiko |

**・DHT11**

温湿度センサの読み取りをやる際に便利なライブラリです。

|  |
| --- |
| cd  git clone https://github.com/szazo/DHT11\_Python.git |

* + 1. **Raspberry pi のコンフィグ**

デフォルトのままではsshやSPIが使えないので使えるように設定します。以下のコマンドでコンフィグ画面を立ち上げます

|  |
| --- |
| sudo raspi-config |

　立ち上げたら、[Advanced Options]の項でSSH,Serial,SPIをenabledにします。

　次に、起動時に自動的にログインしてくれるようにしておきます。[システム]→[Administration]→[Users and Groups]→[Password]をNot asked on loginにします。その後、ターミナルに以下のコマンドを打ち込み、

|  |
| --- |
| sudo nano /usr/share/lightdm/lightdm.conf.d/60-lightdm-gtk-greeter.conf |

以下の文字を記載します。

|  |
| --- |
| autologin-user = (User名) |

これらを設定したら再起動して、自動的にデスクトップに行くことを確認してください。

　再起動後、IPアドレスを固定します。ターミナル上で設定する方法もあるらしいですが、Ubuntu付属のGUIを使って固定した方がやりやすいのでそちらで固定します。システムバーから[システム]→[設定]→[ネットワーク接続]へ進みます。

[無線]タブに先ほど接続した研究室のWifiがあると思いますのでそちらを選択し、右側にある[編集]をクリックしてIPv4を選択し、表3のように設定してください。

表 3 インターネット設定

|  |  |
| --- | --- |
| メソッド | 手動 |
| アドレス | 192.168.0.〇〇〇(ご自由に) |
| ネットマスク | 255.255255.0 |
| ゲートウェイ | 192.168.0.1 |
| DNSサーバ | 192.168.0.1 |

　既に作成した2基のアドレスは192.168.0.81と192.168.0.90を使用しているので、それ以外のIPアドレスにすることをおススメします。なるたけ後半のIPアドレスにしておくと被らずに済むと思います。

* + 1. **SSHで他PCからログインできるようにする**

　このままではいちいちHDMIケーブルやキーボード、マウスをモニタに接続しないと作業が出来ずにとても不便なので、研究室のPCからSSH接続して作業ができるようにします。Raspberry Piと同じネットワークでないと接続できないのでその点だけ注意してください。

**・Ubuntuから接続する場合（推奨）**

ターミナルから以下のコマンドを実行します。sshが入ってないぞ！と怒られたらsudo apt-get install sshを実行して入れてやってください。

|  |
| --- |
| ssh 接続先のUser名@IPアドレス  例) pipipi（IP:192.168.0.81）にssh接続したい時  ssh pipipi@192.168.0.81 |

成功したらPasswordを聞かれるので入力してやると、リモートPCでRaspberry Pi上のUbuntuに接続できます。ターミナル上でしか操作できないのでcdコマンドとか覚えておくと良いと思います。

**・Windowsから接続する場合**

WinOSから接続する場合はTeraTermから接続するのが一番楽かと思います。ダウンロードやインストールは端折るので各自導入してください。

　立ち上げると図1のような画面が出てくると思うので、[ホスト]の所に接続先のIPを入力してください。

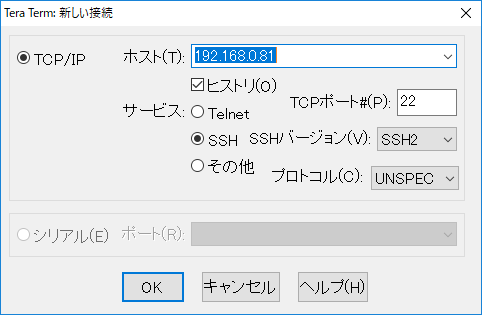


図 1　Tera Termの起動画面

[OK]を押すとUser、Passwordを要求されるので入力してください。初めて接続する場合はKeyが～と言われると思うので、[OK]を押すと接続が出来ると思います。

* 1. **Raspberry PiとArduinoを接続**

シリアルケーブルを用いてArduinoとRaspberry Piを接続します。接続した後、以下のコマンドを実行して認識できているかどうか確認してください。Raspberry Pi側からArduinoは「ttyACM0」という名前で認識されていると思います。

|  |
| --- |
| sudo ls /dev/ttyACM\* |

1. **各プログラムの説明**

それぞれのプログラムについて解説します。基本的にはコメントに書いてある通りなので、それぞれのプログラムが主に何をやっているのか、どのような流れで処理しているのかを概要的に解説します。

* 1. **Arduinoのプログラム**

Arduinoはエンコーダの読み取り専門のデバイスとして使っています。即応性が求められるので余計な機能は盛り込んでいません。

　エンコーダは200R/PでA相B相を出力するタイプで、4逓倍のプログラムを組んでいるのでエンコーダが1回転すると800カウントをする物になっています。Arduino Megaでは、値の変化があると自動的に読み取ってくれるpinが4つあるので、そこで測定をして変化時間当たり(dt=100ms)の回転数をRaspberry Pi側に送っています。

Serialの通信速度は速い方がいいので230400にしてあります。Raspberry Pi側もこの速さですんで違う値にしないでください。

　車輪の回転方向が逆な場合はプログラム中の56～76行目、時計/反時計の正負を逆にしてやると上手く行くと思います。時間当たりの回転数はRaspberry Pi側のオドメトリ計算に使います。

* 1. **Raspberry Pi側のプログラム**

Raspberry Pi側の大まかなプログラム処理の流れは図のようになります。ソースコードを開きつつ解説を読むようにしてください。

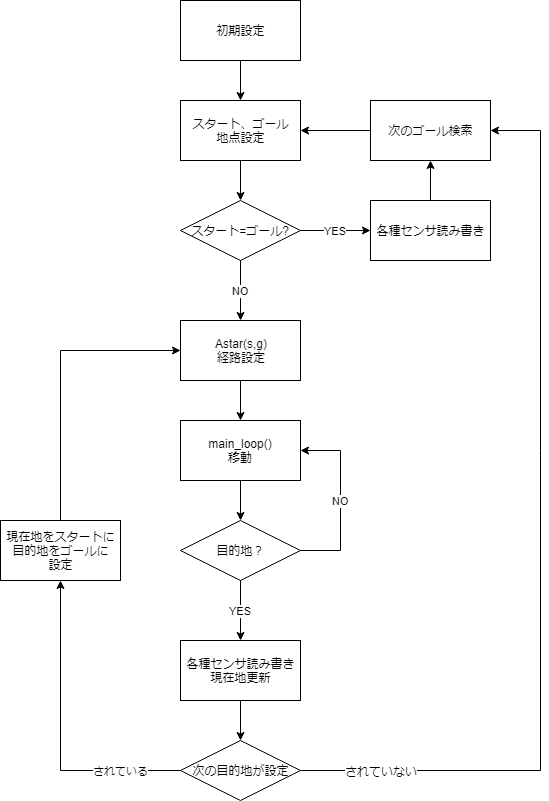


図 2　処理の流れ

* + 1. **初期設定**

　グローバル変数の宣言、シリアル通信の設定、オドメトリファイルの設定、PWM制御の設定、GPIOの宣言などを1~130行目までやっています。main()関数内では各変数の初期化と相方のロボットにssh接続を行っています。ClientSettingについては後程解説します。

基本的には各変数のコメント通りの意味になります。PWMの所にモータドライバにつなぐpin番号か書かれているので間違えないように接続してください。また、このプログラムはmap\_data.txtの情報を基にして行動します。先にmap\_data.txtを作成してから実行してください。0が通行不能、スペースが通行可能になります。

* + 1. **スタート、ゴール地点設定**

スタート地点とゴール地点を設定します。初回起動時には二台の現在地とゴール地点をそれぞれ入力してください。初回以外ではtonextpoint()関数で次の目的地(ゴール)を設定しています。

**・スタート＝ゴールの時**

各種センサを読み取り、現在地をposdata.txtに書き込んだ後に次の目的地を検索して三秒待機します。

**・スタート≠ゴールの時**

Astarアルゴリズムでゴールまでの経路を探索します。

* + 1. **Astar() Astarアルゴリズム**

　moduleフォルダ内のAstar.pyにあるAstar()関数を用いて現在地から目的地への経路探索を行います。斜め移動は許可しておらず、移動コストは室内の舗装された地面を移動することを考えているので全て1です。詳しい流れはプログラム中のコメントを見てください。

引数：スタートs[x,y],ゴールg[x,y]

戻り値：経路route

* + 1. **移動　main\_loop()**

　この関数では、移動に関する処理を行っています。処理の流れは図3の通りです。

引数：無し

戻り値：移動flag(1 or 0)

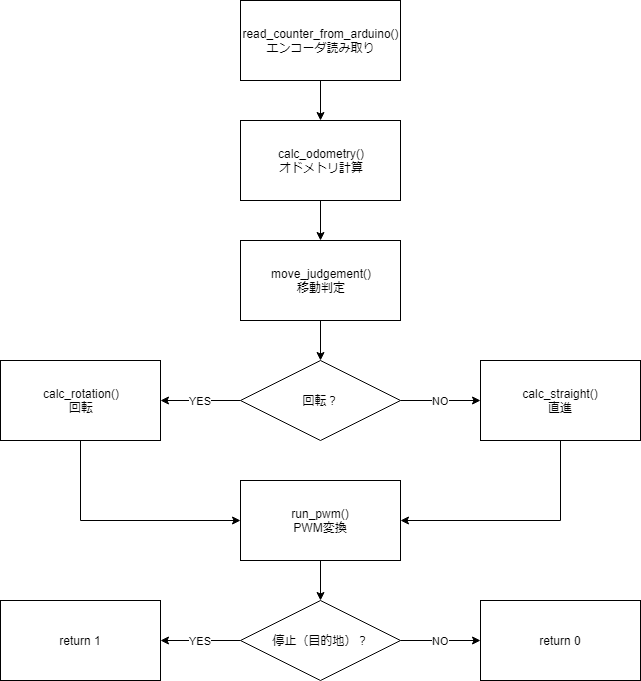


図 3　main\_loop()内のフローチャート

* + - 1. **read\_counter\_from\_arduino() エンコーダ読み取り**

Arduino Megaで読み取ったエンコーダの時間当たり回転数を読み込みます。Arduinoから送られてきた文字列はdt,左カウンタ,右カウンタの3つが半角スペース区切りで送られてくるので、これをstrip().split(“ “)で分割しています。

引数：無し

戻り値：サンプリング時間dt[ms],左車輪カウンタcounter\_l,右車輪カウンタcounter\_r

* + - 1. **calc\_odometry() オドメトリ計算**

　エンコーダを用いたオドメトリは以下の計算式で求められる。

1. 右車輪の角速度は,

で求めることができる。ここで、

Count(t):時刻tにおけるエンコーダの値[pulse]

Encoder:一回転当たりのエンコーダのパルス数[pulse]

Gear:ギア比（タイヤが一回転するときのエンコーダの回転数）

Δt:サンプリング周期[s]

である。左車輪も同様にして求められる。

1. ロボットの速度、角速度は次の計算式で求められる。

ここで、

v:並進速度[m/s]

ω:ロボットの回転角速度[rad/s]

:左車輪半径[m]

:右車輪半径[m]

:車軸間距離（トレッド）[m]

である。

1. サンプリング時間ごとに積分すると、誤差が生じて正確さに欠けるため、誤差を少なくするために台形積分を行う。現在の車体角度をθ、前回の車体角度を、前回の並進速度を、前回の車体角速度をとすると、現在の車体の位置は以下の式で求められる。

以上の式を落とし込んでやるとオドメトリが計算できます。多少誤差が乗るかもしれませんが調整してやってください。

引数：時間t[s] カウントcount[左カウント,右カウント]

戻り値：現在地ｘ[m],y[m],車体角度th[tad],並進速度v[m/h],車輪速度omega\_l,omega\_r

* + - 1. **move\_judgement() 移動判定**

直進、回転、後退、停止を判断する関数です。target\_distでゴールまでの距離を計算して、その距離がdiffdistで設定した値より小さければ停止させます。大きければtarget\_radでゴールまでの角度を計算して、diffradで設定した値より小さければ直進、大きければ回転させます。また、多少行き過ぎた場合は後退させるようにしてます。

　停止させる場合は、完全に停止させる「停止モード」と制動距離分を停止させる「制動距離モード」があります。後のPWM信号変換の項でも解説しますが、「停止モード」ではduty比を100に、「制動停止モード」では制動距離に応じて数値を変化させるようにしています。

　正直、ここは試行錯誤して作ったので改善の余地は大いにあります。是非改善してください。

引数：現在地np[x,y]　現在角度now\_rad[rad] 並進速度v[m/h]

戻り値：行動フラグ(1~6)　目標までの距離[m],　目標までの角度[rad]

表 4　行動ごとの戻り値

|  |  |
| --- | --- |
| 右旋回 | 1 |
| 左旋回 | 2 |
| 直進 | 3 |
| 後退 | 4 |
| 制動停止 | 5 |
| 停止 | 6 |

* + - 1. **calc\_rotation() 回転制御**

　PD制御を用いて回転する速度を制御します。移動判定で回転を指定されたらこの関数が呼び出されます。

　最終的な操作量の計算式は、

で求められます。これらの式をプログラムに落とし込むと289~305行のようになります。Kp,Kdは定数なのでこれを見つけるのが大変難しいです。頑張って探してください。

引数：行動フラグevent 目標までの角度target\_r[rad] 現在角度now\_rad[rad]

戻り値：制御量PD\_L,PD\_R

* + - 1. **calc\_straight() 直進制御**

直進する際に速度を制御する関数です。回転制御のPD制御に積分要素を足したPID制御で速度を制御しています。計算式は以下の通りになります。

こちらはKiが増える分もっと難しいです。頑張って良い組み合わせを探してください。

後退する際は計算した値に-1を乗じて負の数にしてます。

引数：行動フラグevent　目標までの距離target\_d[mm]

戻り値：制御量 PID\_L,PID\_R

* + - 1. **run\_pwm() PWM変換**

calc\_rotateまたはcalc\_straightで計算した制御量をPWMに変換してモータドライバが扱えるようにします。具体的には、duty比を調整してPWM出力を調整します。

　引数として受け取った制御量が100を超えている場合、duty比は100が上限なので99を上限としてリミット値的なものを適用します。下限値も然りで、あまりにも値が低すぎると車輪が回転してくれないことがあるので、適切な数値を下限値として設定してください。後は移動判定によって車輪を正転、逆転させています。

　例外として、制動停止時は制御量分のブレーキを効かしているのと、完全停止時はduty比を100として動かないようにしてあります。

引数：制御量 speed\_L,speed\_R　移動フラグevent 目標までの角度t\_rad, 車輪回転速度om\_r,om\_l

戻り値：無し

* + 1. **nextpointserch() 次の目的地を探す**

ロボットがゴール地点に到着した際、次の目的地を設定する関数です。module内のmove関数(pointmove.py内)を使用しています。ぶっちゃけmodule内に隔離する必要ないのでこのプログラムに統合してもいいと思います。

　move()関数で次の目的地を取得した後、相方のロボットに自分が取得した環境データnode\_data.txtをSendData()関数で送信し、PosRead()関数で相手の位置を取得します。

　その後、次の目的地に相手が居ないならそのまま戻り値に目的地を渡し、居るなら二番目に良い場所を目的地として戻り値に渡します。

引数：Paramikoクライアント client

戻り値：次の目的地 tugipoint[x,y]

* + - 1. **move()　環境情報を基に次の目的地を設定する**

　各ノードの情報を詰め込んだnode\_data.txtを参照して、植物が成長するに最も良いと思われる地点を探します。現状では明るさ、温湿度を参照して最も明るく、温かい地点に移動するようにしています。将来的には各植物の適した明るさ、温湿度のデータベースを作成してそれらを参照し、最も値が近い地点に移動するようにできるとより良いものになると思います。

　一番良い場所だけでは、既に相方が居ると衝突する可能性があるので、二番目に良い地点も検索するようにしています。台数を増やすならここを3,4番目と降順に良い場所リスト的なものを作成する必要があるかと思います。

引数：無し

戻り値：次の目的地 next\_position[x,y] 二番目に良い場所second\_position[x,y]

* + 1. **Datap2p.py Paramiko関連の通信に関する関数**

　moduleディレクトリ内にあるDatap2p.pyにある関数について解説します。この中にある関数は全てParamikoライブラリを用いたデータのやり取りに関するものです。Paramikoでもっと色々やりたい時はリファレンスを読むと良いと思います。

* + - 1. **ClientSetting() Paramikoのクライアントを作成、設定する**

Paramikoを使ってデータのやり取りをしよう！となると、SCPクライアントとsftpクライアント、scpクライアントを作っておく必要があります。この関数ではこのクライアントを作成すると共に、相方のロボットとssh接続していつでもデータを受け渡しできるようにするのが役目です。したがって、このシステムを稼働するときは相方のロボットも同ネットワーク内に居ないと稼働できないようになっています。改善できる余地はあるのでスタンドアロンで動かしたい場合は上手いことやってください。

　remortPathのpathは、cdのみのコマンドで移動するデフォルトpathにしておくと後々取り回しがしやすくなると思います。

引数：送り先 to

戻り値：sftpクライアントsftp, scpクライアントsc, sshクライアントssh

* + - 1. **SendData() ファイルを送信**

ファイルを相方に送ります。sftpクライアントを使うと送れます。Changednameで送った後の名前を変えることが出来ますが、path情報を付加すればデフォルト以外のディレクトリに送信することも可能です。

例）相方のデフォルトディレクトリ内にあるmoduleディレクトリに送信するとき

SendData(client, “file.py”, “module/sendedfile.py”)

送信した後のファイル名は同じ名前でもいいですが、引数は両方渡すようにしてください。

引数：クライアントclient 送りたいファイルsenddata 送信後の名前changedname

戻り値：無し

* + - 1. **RecivedData() ファイルを受信**

ファイルを相方から受け取ります。SCPクライアントを用いて受信します。

引数：クライアントclient, 受信ファイルremortpath, 受信後の名前recivename

戻り値：無し

* + - 1. **PosRead() 相手の現在地取得**

　相手の現在地を取得します。sshクライアントを使います。相方のposdata.txtを読み込んで、その値を戻します。

引数：クライアントclient

戻り値：相手の現在地 position[x,y,x,y]

* + - 1. **Closing() 終了処理**

Paramikoを用いた接続を終了する際に呼び出します。プログラムを終了させる時はコイツを置いとくようにしてください

引数：クライアント cli

戻り値：無し

* + 1. **Nodesetting.py ノード管理に関する関数**

各センサの値を収納するノードの作成やセンサの読み取り、書き込みを扱っている関数群です。ひとつのノードにはx,y,通行状態、照度、水分、温度、湿度、日付の情報が搭載されています。

* + - 1. **Node\_create() ノード作成**

Map\_data.txtを基にnodeを作成する関数です。Mapの縦横の長さに応じてnodeを作成して、それぞれの地点において通行可能ならTrue、ダメならFalseをstateの欄に入力し、作成日時はdatetimeを用いて入力します。このシステム内では初期設定の部分で使っています。

引数：無し

戻り値：map\_node

* + - 1. **Map\_view() マップの状態を表示**

Map\_data.txtの情報を基にして、現在のmapがどのような状態なのかを表示する関数です。

引数：無し

戻り値：無し

* + - 1. **Data\_write() データを書き込む**

Node\_data.txtに現在のnode\_data内の情報を書き込みます。データが入っていない（=None）場合は”None”の文字列を書き込むようにしてます。

引数：ノード情報map\_node

戻り値：無し

* + - 1. **Data\_read() データの読み込み**

相方から送られてきたノード情報を読み取り、自身のノード情報と照合して日時が新しい情報は更新する関数です。ただ、情報が新しいといってもデータが無い場合はスルーするようにしてあります。

引数：自分のノード情報map\_data

戻り値：無し

* + - 1. **Sensor\_read() センサの読み取り**

各センサの読み取りを行い、ノードに情報をぶちこみます。Gpiozeroを使うと通常A/Dコンバータの値を読み取るにはbitをずらす等の処理をしなければならないのが、instance.read()を用いるとすれぞれのchannelの電圧？を返してくれるようになります。使用する前にきちんと値が取得できているかテストしてから使うようにしてください。全てが0.0のままなのは接触不良か回路の設計ミスの可能性があります。

引数：ノード情報map\_node 現在地mx,my

戻り値：無し

* + - 1. **Data\_show() ノード情報を表示させる**

現在のノードに入ってる情報を一通り表示させる関数です。適当にUI作ったのであまりにも桁が細かいデータがあるとレイアウトが崩れます。

引数：ノード情報map\_node

戻り値：無し

* + - 1. **Dht11.py 温湿度センサの便利なモジュール**

Gitで取得したdht11.pyですが、moduleディレクトリに入れてください。nodeSetting.py内で温湿度の値を読み取るのに使ってます。

* + 1. **\_\_init\_\_.py moduleディレクトリ内にある関数を使えるように**

ここにmoduleディレクトリ内にある関数を書き込んでおくと、importする際に呼び出せるようになります。もし既存の関数以外の物をこのmoduleディレクトリ内で作成して、プログラムで使用したいならば、ここに関数とファイル名を他の書式に倣って書き込んでください。

1. **現状確認している不具合**
2. **お互いの現在地が把握できていない**

Posdata.txtで現在地とこれから行く予定の場所の２つを共有しているはずなのに、何回か衝突する。同期するタイミングがあまり良くないのかもしれない。

1. **車輪が回転しない**

あまりにも機体が重すぎると、PWMのDuty比を100にしてもうんともスンとも言わずに車輪が回転しなくなるので、適切なモータと重量にしてあげよう。それ以外にも配線ミスによるものもあるので配線は確実に行うように。

1. **オドメトリが多少ずれる**

回転する際に、オドメトリ計算では180°回転していることになっているが実際に実測すると5°程回転が足りなかったりすることがよくある。少々の移動ならあまり問題は無いのだが、完全に自律移動で長時間動作するロボットを目指しているので誤差は結構侮れない大きさになる。原因は不明。

1. **一定時間稼働すると大きく一回転する**

Arduino側で送信しているカウンタがオーバーフローしている可能性がある。対策できるならしておいた方がいい。

1. **行動範囲がwifiの届く距離に限られる**

双方の通信が研究室wifiを介している関係上、行動範囲がwifiの届く範囲の限られてしまいます。片方のRaspberry Piを親機化させて稼働させる手もありかもしれません。

1. **あらかじめmapを設定しておかないと動けない**

逐一周辺状況を読み込んでmapを生成しながら移動できたらより自律移動の完成度が高まるかと思います。となるとROS使うことになると思います。正直移動やオドメトリに関してはROS使った方が個人的にはいいと思います。

1. **今後の展望**

　このシステムは室内で植物が自律的に行動して成長することを目的としています。今回の研究ではこのシステムを複数化させて情報を共有し、より効率的に育成させようといったものです。今後は台数をもっと増やして、本当に効率化できたのかを検証することでより説得力が増すのではないかと考えています。

　また、現段階では周辺情報の取得ができていないので、レーザーか何かを搭載して壁を検知し、自動的にマップを作成できるとより自律化が進んで人間の手間が省けるのではないかと考えています。是非ともこの二点を改善してほしいです。