Raspberry Pi Pico を使った郵便通知システムの製作

髙原 佑介　　野口 照太
森川 皓正

１．研究概要

私たちはセンサと Raspberry Pi を用いて日常で使うことができるシステムを作りたいと考えた。

まずは研究の基盤として、今や日常生活に不可欠となっている連絡手段である LINE を活用しようと思い、日常でより便利にしたいと思うものの意見を出し合った。

話し合いを重ねた結果、通信販売などが普及している現代において自分が家にいなくても郵便物が届いたことがわかるシステムがあると便利だと考え、郵便通知システムについて研究し製作することにした。

郵便通知システムを作成するにあり、年間計画（表１）を立てて、活動した。

表１　年間計画

４月：研究内容の決定　年間計画作成
５月：センサを用いた基板作成
６月：センサが感知するための プログラム作成　企画発表
７月：LINE 通知するためのプログラム作成
８月：基板に合わせたネットワークの設定 及びネットワークを使用するための プログラム作成
９月：プログラムの完成　基板との接続
１０月：発表用の箱製作　基板の取り付け
１１月：文化祭　報告書作成
１２月：報告書完成、提出
１月：課題研究発表会

２．研究の具体的内容

（１）開発環境(Thonny)
　　言語(pyhton)

（２）設計

今回のシステムは図１の流れで動作せるようにした。

超音波センサにより距離を測定する
↓
手紙が投函されると、検出した距離が、設定した数値を下回る。
↓
LINENotify のサービスで設定したトークルームに通知が送信される。

図１　動作の流れ

（３）使用したセンサ

写真１　HC-SR04

・センサの仕組み

今回のシステムには超音波距離センサである HC-SR04（写真１）を使用した。
HC-SR04 は超音波の反射時間を利用して非接触で距離を測るモジュール。
外部からトリガパルスを入力すると超音波パルス(８波)が送信され、出力された反射時間を[音速×出力して戻ってくる時間÷2]の公式にあてはめ、マイコン(Arduino など)で計算することによって距離を測ることができる（図２）。

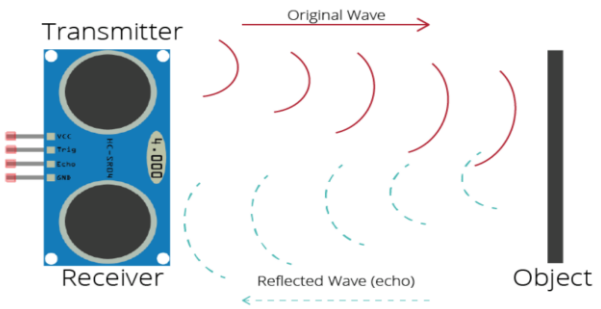

図２　センサの仕組み

・センサの接続

ブレットボード上に Raspberry Pi Pico を接続し HC-SR04 の仕様に合わせて基板に配線、郵便物が入る箱につけやすいように基板から配線を伸ばした。

・センサが距離を測定するためのコード

図３の 24 行目～29 行目で超音波センサの Trig ピンを制御して、センサに超音波を起動させている。41 行目～42 行目で距離の計算をしている。44 行目から 46 行目で測定値が有効な場合の処理をしている（図３）。

distance() 関数はセンサなどを使って距離を測定し、その値を返す。例えば、distance() 関数が距離を測定できなかった場合やエラーが発生した場合に None が返されることがある。

```
19 # 距離のしきい値(単位:センチメートル)
20 THRESHOLD_DISTANCE = 20  # 距離が20cm未満になったら通知
21 MAX_VALID_DISTANCE = 500  # 500cm以上の測定は無視する
22
23 # 距離を測定する関数
24 def distance():
25     TRIG.low()
26     time.sleep_us(2)
27     TRIG.high()
28     time.sleep_us(10)
29     TRIG.low()
30
31     # ECHOピンがHighになるのを待つ
32     while not ECHO.value():
33         pass
34     time1 = time.ticks_us()
35
36     # ECHOピンがLowになるのを待つ
37     while ECHO.value():
38         pass
39     time2 = time.ticks_us()
40
41     during = time.ticks_diff(time2, time1)
42     dist = during * 340 / 2 / 10000
43
44     if dist > MAX_VALID_DISTANCE:
45         print("無効な距離の読み取り: %.2f cm" % dist)
46         return None
47     return dist
```

図３　（プログラム１）

（４）Raspberry Pi Pico W

LINE に通知するために Wi-Fi に接続する必要があったため、無線機能を搭載している Raspberry Pi Pico W を使用した（図４）。

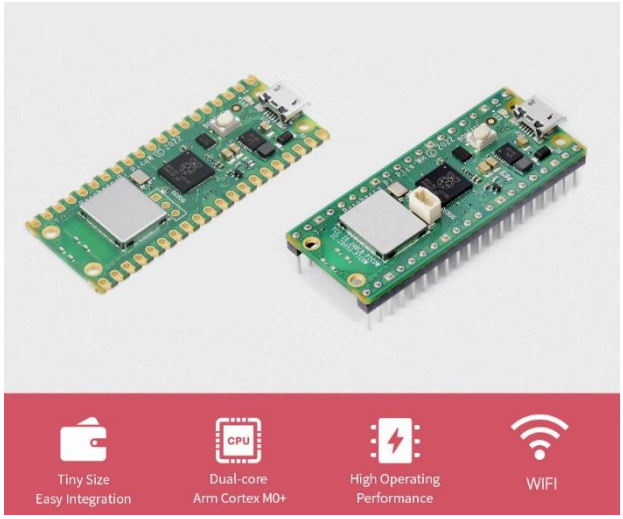

図４　Raspberry Pi Pico W

（５）LINE との通信

センサが取得した情報を LINE に送信するために、参考にしたサイトの方式に合わせて LINE が提供している LINENotify というサービスを利用した。LINENotify の web サイトからトークンを発行し、RaspberryPiPico に書き込んで連携をした。

・LINE 通知を送信するためのプログラム

図５のコードでは、

・51 行目

この行は、send_line_notify という名前の関数を定義している。この関数は、message という引数を受け取る。message は LINE Notify に送信したいメッセージの内容である。

・52 行目

try ブロックは、例外（エラー）が発生する可能性のあるコードを囲むためのものである。もしエラーが発生した場合、except ブロックが実行される。ここでは、LINE Notify への送信処理でエラーが起きる可能性を考慮して囲んでいる。

・53～56 行目

この行から始まるコードは、LINE Notify に送信するリクエストのヘッダを設定している。

Content-Type ヘッダは､送信するデータの形式を示す。この場合、application/x-www-form-urlencoded は URL エンコードされたフォームデータとしてリクエストを送信することを意味する。

Authorization ヘッダは、認証情報を指定するために使用される。Bearer + TOKEN で、LINE Notify のアクセストークン（TOKEN は事前に LINE のコンソールで生成された文字列）を指定している。このアクセストークンを使用して、LINE Notify API に認証されたリクエストを送信できる。

・57、58 行目

この行は、送信するデータを辞書形式で作成している。message キーに通知したい内容（引数 message）を設定する。式の contents を URL エンコードされた文字列に変換する｡LINE Notify の API はこの形式でデータを受け取る。

エンコード処理は MicroPython のリポジトリのソースコードから移植した。

・59～61 行目

ここで実際に POST リクエストを送信している。

urequests.post は、指定された URL（https://notify-api.line.me/api/notify）に POST リクエストを送る。

headers=line_header で先ほど作成したヘッダーを指定している。

data=data で送信するメッセージ（data）を指定している。

このリクエストは LINE Notify API にメッセージを送信するためのものである。

req.close() は､urequests.post で開かれた接続を閉じるためのコード。リクエストを送信した後、接続を適切に閉じることでリソースを解放する。

・62、63 行目

except ブロックは、try ブロック内でエラーが発生した場合に実行される。Exception はすべての一般的な例外を捕まえることができ、e はエラー内容を格納します。もし何らかのエラーが発生した場合、この行でエラーメッセージを表示する。エラーの詳細は e に格納されているため、それを出力することでデバッグが可能である。

```
50 # LINE通知を送信する関数
51 def send_line_notify(message):
52     try:
53         line_header = {
54             'Content-Type': 'application/x-www-form-urlencoded',
55             'Authorization': 'Bearer ' + TOKEN
56         }
57         contents = { 'message' : message}
58         data = urlencode(contents)
59         req = urequests.post('https://notify-api.line.me
60         /api/notify', headers=line_header, data=data)
61         req.close()
62     except Exception as e:
63         print("通知エラー:", e)
```

図５　（プログラム２）

・メインループ

・107～114 行目（図６）

その後、dis が None であるかどうかを確認している。None は、何も返されていないことを意味する。もし dis が None の場合、プログラムが再試行する前に短い時間待機するため試行錯誤して動作に問題がなく待機時間を設けられる time.sleep(0.5) を使用して 0.5 秒待機している。continue は、現在のループの残りの部分をスキップし、次のイテレーションに進む命令である。もし dis が None の場合、距離の測定ができなかったため、次のループに進むように指示している。距離が正常に測定できた場合、dis の値をフォーマットして表示している。%.2f は小数点以下２桁で表示するという意味である。例えば、距離が 50.1 cm の場合、「距離: 50.10 cm」と表示される。

・116～119 行目（図７）

dis が図６のプログラムで定義したあるしきい値 THRESHOLD_DISTANSE よりも小さいかどうかを判定している。この条件が True であれば、特定のアクションを実行する。例えば、距離がしきい値を下回った場合、何かが近づいてきたことを示す場合などである。もし距離がしきい値 THRESHOLD_DISTANCE を下回った場合、send_line_notify 関数を呼び出して LINE 通知を送る。通知のメッセージは「メールが届きました!!」である。LINE 通知を送った後、通知が頻繁に送信されるのを防ぎ、次のアクションに移る前に少し待機するため、１秒間の待機時間では一回の動作で複数の通知が来ることがあったため、５秒間プログラムを停止させる。

・122～124 行目（図８）

この行では、現在の時間（time.time()）と last_weather_notify の差が weather_interval より大きいかどうかをチェックしている。time.time() はエポックからの経過時間（秒）を返し、last_weather_notify は前回天気情報を通知した時刻。weather_interval は通知を送る間隔である。

もし、最後に天気通知を送った時間から指定された間隔（weather_interval）が経過していれば、天気通知を送信する。天気通知を送信する関数 send_weather_notify() を呼び出している。この関数は、天気に関する情報を取得し、ユーザに通知を送る処理を行う。天気通知を送信した後、last_weather_notify に現在の時刻を代入している。これにより、次回天気通知を送信するための基準時刻が更新される。

```
107             # 距離の測定
108             dis = distance()
109
110             if dis is None:
111                 time.sleep(0.5)
112                 continue
113
114             print("距離: %.2f cm" % dis)
```

図６　（プログラム３）

```
116         # 距離がしきい値を下回った場合に通知
117         if dis < THRESHOLD_DISTANCE:
118             send_line_notify("メールが届きました！！")
119             time.sleep(5)
```

図７　（プログラム４）

```
121         # 1分ごとの天気通知
122         if time.time() - last_weather_notify > weather_interval:
123             send_weather_notify()
124             last_weather_notify = time.time()
```

図８　（プログラム５）

（６）天気予報の送信

定期的に天気予報の通知が来る機能があれば便利だと思い追加で作成した。天気予報の送信には、OPEN WEATHER を利用した。OPEN WEATHER の公式サイトでトークンを発行して、Raspberry Pi Pico に書き込んだ。
これにより、１時間に１回、岡山県の天気や気温が送信される。

（７）部品

・Raspberry Pi Pico
・HC-SR04（超音波距離センサ）
・ジャンパー線
・USB アダプター
・A4 トレイ

（８）配線

回路図（図９）を参考にしながら、Raspberry Pi Pico や HC-SR04 の配線を行った（写真２）。

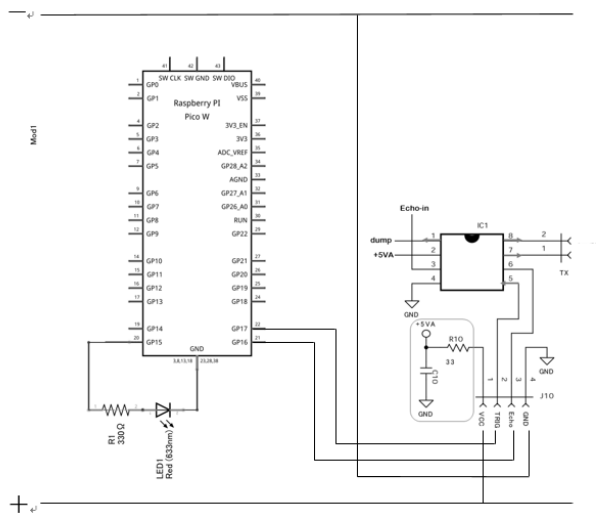

図９　回路図

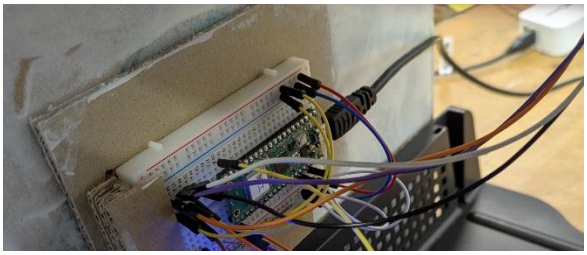

写真２　作成した回路

（９）郵便ポストの作成

段ボールを利用し、より距離センサで正確に距離が測定できるように工夫した。具体的には、距離センサを入れる専用の穴を作り、基板を取り付けるくぼみも作った。デザインも本物の郵便ポストをイメージし、カラースプレーで色を付けた（写真３）。

写真３　完成した郵便ポスト

（10）LINE 通知の受信

郵便ポストに紙が投函される（測定距離が一定距離を下回る）と、ほとんど時差がなく、設定した LINE のトークルームにメッセージが表示される（写真４）。

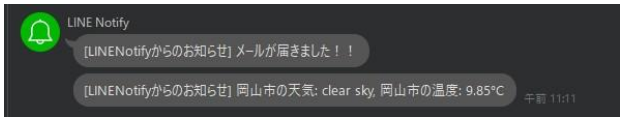

写真４　実際の動作

（11）今後の展望

今は一時的にブレッドボード上で動作させているので、ユニバーサル基板に回路を移して、どこでも使えるようにしたい。電源についてもコンセントがなくても使えるようにするためにバッテリをつなげて動作させるようにしたい。

今回は参考にしたサイトに合わせて LINE Notify を使用したがサービスが終了することが決定しているため、同じく LINE が提供している Message API のアカウントを作成して移行できるようにしたい。

４．研究のまとめ

今回の課題研究を通して、Raspberry Pi の通信の仕組みやセンサの使い方について学ぶことができた。センサを使っていく中で、もっといろんなものに活用できると思った。

Python の知識をつけることができたので、とても有意義な課題研究になった。また、授業で学んだブレッドボードの配線やはんだ付けなどの知識が役に立ってよかった。実際に作業を進めていく中で、様々な課題があり、計画通りいかないことも多かった。途中で機能を追加したり、削除したりしながら進めていくこと、年間を通して活動することの難しさも実感した。しかし、班員と協力して、たくさん意見を交換しながら難しい時期も乗り越えることができた。卒業後も、周りの人とのコミュニケーションを大事にしながらいろんな困難を乗り越えていきたいと思う。

参考文献

Raspberry Pi Pico のピンの電気的な仕様
https://zenn.dev/ythk/articles/ythk-raspico-pins

レッスン 23: 超音波センサーモジュール (HC-SR04)
https://docs.sunfounder.com/projects/umsk/ja/latest/04_pi_pico/pico_lesson23_ultrasonic.html

【ラズパイピコ】Raspberry Pi Pico W を使用してスマホに LINE 通知する方法（LINE Notify）
https://matsunosuke4989.com/raspberry-pi-pico-w-line-notify/

LINE で OpenWeatherMap の天気情報を通知する仕組みの解説
https://kaizen-factory.jp/line_notify_openwheathermap/

LINE Notify
https://notify-bot.line.me/ja/

Thonny　Python IDE for beginner
https://thonny.org/

文字化けの対策
https://github.com/micropython/micropython-lib/blob/master/unix-ffi/urllib.parse/urllib/parse.py

https://github.com/micropython/micropython-lib/blob/master/python-stdlib/collections-defaultdict/collections/defaultdict.py

最新 Pico W 対応！ラズパイ Pico 完全ガイド（ラズパイマガジン）

ラズベリーパイ Pico/Pico W 攻略本（ボード・コンピュータ・シリーズ）
（Interface 編集部）