

```
1 # 1. 初期設定パート *****
2
3 # モジュール読み込み *****
4 import time # 時間に関するPython標準モジュールです。
5 import RPi.GPIO as GPIO # Raspberry PiのGPIO（汎用入出力）を制御するためのPythonモジュールです。
6 import os # os機能に関するPython標準モジュールです。
7 import subprocess # プロセス管理に関するPython標準モジュールです。
8 import cv2 # OpenCVのPythonモジュールです。
9 import socket # socket通信に関するPython標準モジュールです。
10 import threading # マルチスレッド処理に関するPython標準モジュールです。
11 import pandas # データの処理及び分析を用意にするPythonモジュールです。
12 import csv # CSVデータ読み書きに使用するPython標準モジュールです。
13 from Adafruit_BME280 import * # センサメーカー(Adafruit)がGitHub (https://github.com/adafruit/Adafruit_Python_BME280) に公開しているPythonモジュールです。
14 from grove_gesture_sensor import * # GitHub (https://github.com/DexterInd/GrovePi/tree/master/Software/Python/grove_gesture_sensor) に公開されているPythonモジュールです。
15 from upspackv2 import * # GitHub (https://github.com/rcdrones/UPSPACK_V3/) に公開されているPythonモジュールです。
16 import datetime # 日付と時刻に関するPython標準モジュールです。
17 import requests # HTTPリクエストに関するPythonモジュールです。
18 import wikipedia # ウィキペディア検索用Pythonモジュールです。
19 from flask import (Flask, render_template, request, redirect, session, send_from_directory, Response) # FlaskのPythonモジュールです。
20 from gtts import gTTS # "google text to speech"のPythonモジュールです。
21 import random # 乱数生成に使用します。
22
23 # SIROの行動制御 *****
24
25 mode = 1
26 mode_string = ["テスト", "のんびり", "聞き耳", "ラジオ", "ラジコン", "ライブカメラ", "***", "***", "***", "***"]
27 # SIROの行動をモードで制御します。
28 # モードはまだ5つしかありませんが今後追加していく予定です。
29
30 g_parameter = [0, 0, 0, 0, 0]
31 # SIROの行動制御に汎用的に使用するパラメータです。
32
33 siro_silent = 1 # おしゃべり禁止
34 siro_dontmove = 1 # お散歩禁止
35 # テスト時などSIROの行動を制限したい場合に使用するパラメータです。
36 # 2つしかないですが今後追加していく予定です。
37
38 communication = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
39 # 家族とのコミュニケーションの進み具合を記憶するためのパラメータです。
40 # IDNo.ごとに記憶します。
41
42 thread_loop = 1 # サブスレッド終了制御フラグ
43 # サブスレッドの終了を制御するためのフラグです。
44
45 list_week = ['月曜日', '火曜日', '水曜日', '木曜日', '金曜日', '土曜日', '日曜日']
46 # 曜日の認識に使用します。
47
48 # GPIOの設定 *****
49
50 GPIO.setmode(GPIO.BCM)
51 # GPIOを使用する際にポート番号で指定するかピン番号で指定するかを設定しています。
52 # 例えばGPIO2はポート番号で指定すると"2", ピン番号で指定すると"3"になります。
53 # ここではポート番号で指定するモードを選択しています。
54
55 # 基本機能1 移動する DCモータ制御 モータドライバ DRV8835 *****
56
57 MOTOR_B_IN1 = 12 # モータ左 ENABLE
58 MOTOR_B_IN2 = 16 # モータ左 PHASE
59 MOTOR_A_IN1 = 20 # モータ右 ENABLE
60 MOTOR_A_IN2 = 21 # モータ右 PHASE
61 # モータドライバDRV8835の接続先とGPIOのポート番号を紐づけています。
62
63 GPIO.setup(MOTOR_A_IN1, GPIO.OUT)
64 GPIO.setup(MOTOR_A_IN2, GPIO.OUT)
65 GPIO.setup(MOTOR_B_IN1, GPIO.OUT)
66 GPIO.setup(MOTOR_B_IN2, GPIO.OUT)
67 # GPIOを入力、出力どちらで使用するかを設定しています。ここではすべて出力に設定しています。
68
69 MOTOR_A_PWM = GPIO.PWM(MOTOR_A_IN1, 50)
70 MOTOR_B_PWM = GPIO.PWM(MOTOR_B_IN1, 50)
71 # GPIOの12と20をPWM出力に設定しています。
72 # パルス周期はとりあえず50Hzに設定しています。
```

```
73
74 siro_motor_speed = 50 # Duty比
75 # PWMのDuty比(0~100)でモーター印加電圧を変化させ回転速度を制御します。
76 # とりあえず50に設定しています。
77
78
79 def siro_motor(command):
80     # SIROを走行させるための関数です。
81     # 引数commandに従い左右のモータの回転速度、回転方向を制御します。
82
83     if command == "go": # まえ
84         if not uss_alert: # 障害物アラートフラグが立っている場合は前進を禁止します。
85             MOTOR_A_PWM.start(siro_motor_speed)
86             MOTOR_B_PWM.start(siro_motor_speed)
87             GPIO.output(MOTOR_A_IN2, GPIO.LOW)
88             GPIO.output(MOTOR_B_IN2, GPIO.LOW)
89     elif command == "back": # うしろ
90         MOTOR_A_PWM.start(siro_motor_speed)
91         MOTOR_B_PWM.start(siro_motor_speed)
92         GPIO.output(MOTOR_A_IN2, GPIO.HIGH)
93         GPIO.output(MOTOR_B_IN2, GPIO.HIGH)
94     elif command == "right": # 右回転
95         MOTOR_A_PWM.start(siro_motor_speed)
96         MOTOR_B_PWM.start(siro_motor_speed)
97         GPIO.output(MOTOR_A_IN2, GPIO.HIGH)
98         GPIO.output(MOTOR_B_IN2, GPIO.LOW)
99     elif command == "left": # 左回転
100         MOTOR_A_PWM.start(siro_motor_speed)
101         MOTOR_B_PWM.start(siro_motor_speed)
102         GPIO.output(MOTOR_A_IN2, GPIO.LOW)
103         GPIO.output(MOTOR_B_IN2, GPIO.HIGH)
104     elif command == "stop": # 停止
105         MOTOR_A_PWM.stop()
106         MOTOR_B_PWM.stop()
107         GPIO.output(MOTOR_A_IN2, GPIO.LOW)
108         GPIO.output(MOTOR_B_IN2, GPIO.LOW)
109
110
111 # 基本機能2 発話する OpenJtalk *****
112
113 sp_vol = 50
114 # スピーカー音量を調整するためのパラメータです。50~100で調整します。
115
116 speaking = 0 # 発話中フラグ
117 # 発話中、サウンド再生中はフラグを立てます。発話中フラグが立っている間は音声認識結果をキャンセルします。
118
119
120 def siro_speak(serif):
121     # 発話用の関数です。引数serif（文字列）を音声合成して発話します。
122
123     global sp_vol, speaking
124     # グローバル宣言しています。
125     # 参照だけなら本当はいらないのですが筆者はとりあえず宣言しておくことにしています。
126
127     cmd = "amixer cset numid=3 "+str(sp_vol)+"%"
128     subprocess.call(cmd.split())
129     # amixer（標準コマンド）を使ってスピーカー音量を設定しています
130     # subprocess.callでコマンドを実行しています。
131
132     # voice = " -m /usr/share/hts-voice/nitech-jp-atr503-m001/nitech_jp_atr503_m001.htsvoice" # 音声ファイル 男性の声
133     voice = " -m /usr/share/hts-voice/mei/mei_normal.htsvoice" # 音声ファイル 女性の声
134     dic = " -x /var/lib/mecab/dic/open-jtalk/naist-jdic" # 辞書ファイル
135     # 使用する音声ファイルと辞書ファイルを指定しています。
136
137     option = " -r 0.8 -fm 5"
138     # OpenJtalkの音声パラメータを設定しています。指定しない場合はデフォルト値になるようです。
139     # -a オールパス値/声の質が変わります。
140     # 小さいとキテレツ大百科のコロスケみたいな声になります。大きいとおっさんみたいな声になります。1.0の前後で調整します。
141     # -b ポストフィルター係数/ 使い方がよくわかりませんでした。
142     # -r スピーチ速度係数/ 話すスピードが変わります。1.0がデフォルトなのでその前後で調整します。
143     # -fm 追加ハーフトーン/ 声の高さが変わります。0.0 がデフォルトです。-15~15で調整しました。
144     # -u 有声/無声境界値/ 使い方がよくわかりませんでした。
145     # -jm スペクトラム系列内変動の重み/ 抑揚が変わります。
146     # 小さくするとロボットみたいな感じに、大きくするとミニオンみたいな声になります。1.0がデフォルトで0.5~5.0で調整します。
147     # -jf F0系列内変動の重み/ 音量が変わります。0.5~2.0で調整します。
148
```

```
149 cmd = "echo " + serif + " | open_jtalk" + dic + voice + option + " -ow /tmp/siro_openjtalk.wav"
150 os.system(cmd)
151 # 文字列serifから音声ファイルsiro_OpenJtalk.wavを生成しています。
152 # 音声ファイルは/tmp に保存し発話の都度ファイルを更新します。
153 # 文字列serifを標準入力でOpen_Jtalkに入力するのにechoコマンドとパイプ（|）を使用しています。
154 # os.systemを使ってコマンドを実行しています。
155
156 speaking = 1
157 # 発話中は発話中フラグを立てます。この間は音声認識結果をキャンセルします。
158
159 cmd = "aplay /tmp/siro_openjtalk.wav"
160 subprocess.call(cmd.split())
161 # Open_Jtalkで生成した音声ファイルsiro_OpenJtalk.wavをaplay(標準コマンド)で再生しています。
162
163 speaking = 0
164 # 発話が終了したら発話中フラグを下げます。
165
166
167 # 基本機能3 サウンド再生 *****
168
169 def siro_sound(sound):
170     # サウンド再生用の関数です。
171     # 引数soundで音源ファイルを選択して再生します。
172     # 音源ファイルはmp3で、mpg321というソフトを使用しています。
173
174     global sp_vol, speaking
175
176     speaking = 1
177     # サウンド再生中は発話中フラグを立てます。この間は音声認識結果をキャンセルします。
178
179     if sound == "ごきげん":
180         # 引数soundで再生する音源を指定します。
181
182         cmd = "amixer cset numid=3 "+str(sp_vol*0.8)+"%"
183         subprocess.call(cmd.split())
184         # amixer (標準コマンド) を使ってスピーカー音量を設定しています。
185         # パラメータsp_volに係数をかけて音源毎に音量を微調整しています。ここでは0.6掛けしています。
186         # subprocess.callでコマンドを実行しています。
187
188         cmd = "mpg321 /home/kuro/kurosSIRO/data/sound/R2D2_1.mp3"
189         subprocess.call(cmd.split())
190         # 音声ファイルを指定してmpg321で再生しています。
191         # subprocess.callでコマンドを実行しています。
192
193     # 以下、同様に音源を追加します。
194     elif sound == "タイマー":
195         cmd = "amixer cset numid=3 "+str(sp_vol*1.0)+"%"
196         subprocess.call(cmd.split())
197         cmd = "mpg321 /home/kuro/kurosSIRO/data/sound/alertm.mp3"
198         subprocess.call(cmd.split())
199     elif sound == "ピー":
200         cmd = "amixer cset numid=3 "+str(sp_vol*0.6)+"%"
201         subprocess.call(cmd.split())
202         cmd = "mpg321 /home/kuro/kurosSIRO/data/sound/pi.mp3"
203         subprocess.call(cmd.split())
204     elif sound == "はと":
205         cmd = "amixer cset numid=3 "+str(sp_vol*0.9)+"%"
206         subprocess.call(cmd.split())
207         cmd = "mpg321 /home/kuro/kurosSIRO/data/sound/hatodokei.mp3"
208         subprocess.call(cmd.split())
209     elif sound == "メッセージ消去":
210         cmd = "amixer cset numid=3 "+str(sp_vol*0.6)+"%"
211         subprocess.call(cmd.split())
212         cmd = "mpg321 /home/kuro/kurosSIRO/data/sound/vanish.mp3"
213         subprocess.call(cmd.split())
214     elif sound == "カメラ":
215         cmd = "amixer cset numid=3 "+str(sp_vol*0.9)+"%"
216         subprocess.call(cmd.split())
217         cmd = "mpg321 /home/kuro/kurosSIRO/data/sound/camera.mp3"
218         subprocess.call(cmd.split())
219
220     speaking = 0
221     # サウンド再生が終了したら発話中フラグを下げます。
222
223     # 以下は未使用です。
224     # cmd = "mpg321 /home/kuro/kurosSIRO/data/sound/R2D2_1.mp3"    # ごきげんにおどけた感じ
```

```
225     # cmd = "mpg321 /home/kuro/kurosSIR0/data/sound/R2D2_2.mp3" # ごきげんに何か訴える感じ
226     # cmd = "mpg321 /home/kuro/kurosSIR0/data/sound/R2D2_3.mp3" # なにかを完了させた感じ
227     # cmd = "mpg321 /home/kuro/kurosSIR0/data/sound/R2D2_4.mp3" # 困った感じ
228     # cmd = "mpg321 /home/kuro/kurosSIR0/data/sound/R2D2_5.mp3" # ちょっと興奮した感じ
229     # cmd = "mpg321 /home/kuro/kurosSIR0/data/sound/R2D2_6.mp3" # 怒って訴える感じ
230     # cmd = "mpg321 /home/kuro/kurosSIR0/data/sound/R2D2_7.mp3" # # ガンパっている感じ
231
232
233 # 基本機能4 顔認識 *****
234
235 face_cascade = \
236     cv2.CascadeClassifier('/home/kuro/kurosSIR0/haarcascades/haarcascade_frontalface_alt.xml')
237 # 顔認識学習モデル（カスケード分類器）をファイルを指定して読み込んでいます。
238
239 cap = cv2.VideoCapture(0)
240 # クラスのインスタンス生成
241
242 SCREEN_WIDTH = 640 # 画像サイズ WIDTH
243 SCREEN_HEIGHT = 480 # 画像サイズ HEIGHT
244 cap.set(3, SCREEN_WIDTH)
245 cap.set(4, SCREEN_HEIGHT)
246 # カメラの解像度を設定しています。
247 # 解像度を大きくすると画像が鮮明になりますが処理速度が遅くなります。
248
249 cap.set(cv2.CAP_PROP_FPS, 30)
250 # FPSを設定しています。
251 # 使用したUSBカメラはFPS30で固定なので設定してもあまり意味がないのですが一応明示しています。
252
253 cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)
254 # バッファサイズを最小の1に設定しています。
255 # こうすることでバッファ分のフレームの遅延がなくなります。
256
257 minW = 50
258 minH = 50
259 # 認識可能な顔のサイズを設定しています。
260 # 大きくすると遠くの顔が認識できなくなりますが誤認識が少なくなります。
261 # 逆に小さくすると遠くの顔も認識できるようになりますが壁の模様を顔と間違える等の誤認識が増えます。
262
263 recognizer = cv2.face.LBPHFaceRecognizer_create()
264 # 顔判別クラスのインスタンス生成
265
266 recognizer.read('/home/kuro/kurosSIR0/trainer.yml')
267 # 作成した顔判別機械学習モデルを読み込んでいます。
268
269 font = cv2.FONT_HERSHEY_SIMPLEX
270 # 画像に文字を記載する際のフォントを指定しています。
271
272 face_counter = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
273 # 顔判別カウンターです。
274 # 判別した顔のIDNo.をカウントアップしていきます。
275 # ID番号は 0: 自分 1: 妻 2: 長男 3: 次男 9: 不審者 としています。 IDNo.4~8は未割り当てです。
276
277 face_position = [[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]
278 # 顔を検出したら顔の位置とサイズを求め、位置関係（左右前後）の把握に使用します。
279
280 # ret, frame = cap.read()
281
282
283 def siro_facerecog():
284     # 顔認識処理を行う関数です。
285
286     global thread_loop, face_counter, face_position
287
288     while thread_loop:
289         # メイン処理終了時にthread_loopを0にしてループから脱出させてサブスレッドを終了させます。
290
291         ret, frame = cap.read()
292         # カメラ画像を取得しています。
293
294         grayimg = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
295         # 顔認識を行う前に画像をモノクロ処理して情報量を減らします。
296
297         facerect = face_cascade.detectMultiScale(grayimg, minSize=(minW, minH))
298         # 顔認識を行っています。
299
300         if len(facerect) > 0:
```

```

301         # 顔を見つけたら
302
303         for (x, y, w, h) in facerect:
304             # 顔を一つ一つ確認します。
305
306             cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
307             # 顔を検出した箇所を四角で囲んでいます。
308
309             id_count, confidence = recognizer.predict(grayimg[y:y+h, x:x+w])
310             # 顔判別を行います。
311             # id_countは判別した顔のIDNo.です。confidenceは顔判別の信頼度です。
312
313             cv2.putText(frame, str(id_count), (x+5, y-5), font, 1, (255, 255, 255), 2)
314             cv2.putText(frame, str(round(confidence, 1)), (x+5, y+h-5), font, 1, (255, 255, 0), 1)
315             # 判別した顔のIDNo.と顔判別の信頼度を画面に表示します。
316
317             if confidence >= 100:
318                 # 顔判別の信頼度が低い場合は不審者と判断します。
319                 id_count = 9
320
321             face_counter[id_count] = face_counter[id_count]+2
322             # 顔判別カウンターをカウントアップします。
323
324             face_position[id_count][0] = x+w/2-SCREEN_WIDTH/2
325             face_position[id_count][1] = w
326             # 顔の位置（中心が0で右が+左が-）と顔のサイズ（すなわち近さ）を把握します。
327
328         for i in range(10):
329             face_counter[i] = face_counter[i]-1
330             # 顔がなくなるとカウントを1ずつ減らしていきます。
331
332             if face_counter[i] > 10:
333                 face_counter[i] = 10 # 上限10
334             if face_counter[i] < 0:
335                 face_counter[i] = 0 # 下限0
336             # カウントは0～10としています。
337
338         cv2.imshow('camera capture', frame)
339         # カメラ画像をウインドウに表示させています。
340
341         cv2.waitKey(100)
342         # 処理周期を100msecに設定しています。
343
344     cap.release()
345     cv2.destroyAllWindows()
346     # サブスレッド終了時、カメラをリリースしウインドウを閉じます。
347
348
349 th_facerecog = threading.Thread(target=siro_facerecog, daemon=True)
350 th_facerecog.start()
351 print("face recognition start.....",)
352 # 定義した関数をサブスレッドで起動します。
353 # 顔認識処理は常時処理で且つ処理が重いのでメイン処理に影響を与えない様にサブスレッドで処理します。
354
355
356 # 基本機能5 音声認識 Julius *****
357
358 cmd = "bash /home/kuro/kurosSIRO/julius/julius_module.sh"
359 subprocess.Popen(cmd.split())
360 print("Julius server start.....")
361 # Juliusをモジュールモードで起動します。
362 # subprocess.Popenを使って並列処理の子プロセスとして実行しています。
363
364
365 time.sleep(5)
366 # Juliusの起動を待ちます。5秒は適当です。
367
368 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
369 HOST = "localhost" # JuliusサーバーのIPアドレス,自分自身なのでlocalhost
370 PORT_julius = 10500 # Juliusサーバーの接続ポート番号
371 client.connect((HOST, PORT_julius))
372 print("Conecting Julius server.....",)
373 # IPアドレスとポート番号を指定してクライアントとしてJuliusサーバーに接続します。
374
375 words = [['', ''], ['', ''], ['', ''], ['', ''], ['', ''], ['', ''], ['', ''], ['', ''], ['', ''], ['', '']]
376 # 音声認識結果を格納するためのリストです。

```

```
377
378 bufsize = 1024
379 # Juliusサーバーからデータを読み込む際の1回あたりのデータサイズ[byte]です。
380
381 voice_recog = 0
382 # 音声認識結果読み込み完了フラグです。
383 # フラグが立ったらメインスレッドで音声認識結果にリアクションします。
384
385
386 def julius_crient():
387     # Juliusサーバーから音声認識結果を読み込む関数です。
388
389     global thread_loop, words, voice_recog, speaking
390
391     while thread_loop:
392         # メイン処理終了時にthread_loopを0にしてループから脱出させてサブスレッドを終了させます。
393
394         if voice_recog == 0:
395             # 音声認識読み込み完了フラグが下がっていたらJuliusからデータの読み込みを開始します。
396
397             julius_data = ""
398             NG = 0
399             # まずは初期化します。
400
401             while julius_data.find("</RECOGOUT>") == -1 and julius_data.find("<RECOGFAIL/>") == -1:
402                 julius_data = julius_data + client.recv(1024).decode('utf-8')
403                 # Juliusからデータを読み込みます。データの中に音声認識完了を示すキーワード</RECOGOUT>
404                 # 若しくは音声認識失敗を示すキーワード<RECOGFAIL/>のどちらかが現れたら読み込みを終了します。
405
406                 if speaking == 1: # データ読み込み中に発話中フラグが立ったら音声認識結果をキャンセルします。
407                     NG = 1
408
409                 if thread_loop == 0: # thread_loopが0になった場合は、読み込みを中止し、whileループから脱出します。
410                     break
411
412             # ここからJuliusサーバーから取得したデータ（XML形式）の中から認識した単語"WORD"とその信頼度"CM"の値を抽出します。
413             words = [['', ''], ['', ''], ['', ''], ['', ''], ['', ''], ['', ''], ['', ''], ['', ''], ['', ''], ['', '']]
414             counter = 0
415             # まずは初期化します。
416             for line in julius_data.split('\n'): # 1行ごとに探す
417                 index = line.find('WORD="')
418                 if index != -1: # "WORD="が見つかったら以下を実施
419                     words[counter][0] = line[index + 6:line.find('"' , index+6)] # "WORD"の値を格納
420                     index = line.find('CM="')
421                     words[counter][1] = line[index + 4:line.find('"' , index+4)] # "CM"の値を格納
422                     counter = counter + 1 # カウントアップ
423
424             if NG == 1:
425                 print("発話中のため音声認識結果をスキップ")
426                 # 発話中だった場合は音声認識結果をスキップします。
427             else:
428                 voice_recog = 1
429                 # 音声認識結果読み込みフラグを立ててメインスレッドでの処理を待ちます。
430
431                 print(julius_data)
432                 # Juliusサーバーから取得した生データの中身を確認します。
433
434                 time.sleep(0.1)
435                 # 処理周期を調整しています。
436
437             client.close()
438             # サブスレッド終了時、Juliusサーバとの通信を終了します。
439
440
441 th_voicerecog = threading.Thread(target=julius_crient, daemon=True)
442 th_voicerecog.start()
443 print("Voice recognition start....",)
444 # 定義した関数をサブスレッドで起動します。
445 # 音声認識処理は音声入力待ちが発生するのでメイン処理に影響を与えない様にサブスレッドで処理します。
446
447
448 # 基本機能6 ロボットの記憶 *****
449
450 # パーソナルデータ *****
451
452 personal_data = pandas.read_csv('/home/kuro/kurosSIRO/data/personal.csv', header=0).values.tolist()
```

```
453 # pandasを使ってCSVデータを配列で読み込みます。
454
455 name = [""]*len(personal_data) # 名前
456 yobi = [""]*len(personal_data) # よびかた
457 suki = [0]*len(personal_data) # 好感度
458 for i in range(len(personal_data)):
459     name[i] = personal_data[i][1]
460     yobi[i] = personal_data[i][2]
461     suki[i] = int(personal_data[i][3])
462 # プログラム中で使用しやすい様に別の変数名に変換します。
463 # suki(好感度)は数値パラメータなので数値として扱える様、整数型に変換しています。
464
465 # 予定データ（週ごと） *****
466
467 schedule1_data = pandas.read_csv('/home/kuro/kurosSIRO/data/schedule1.csv', header=0).values.tolist()
468 # 同様にpandasを使ってCSVデータを配列で読み込みます。
469
470 # 予定データ（日ごと） *****
471
472 schedule2_data = pandas.read_csv('/home/kuro/kurosSIRO/data/schedule2.csv', header=0).values.tolist()
473 # 同様にpandasを使ってCSVデータを配列で読み込みます。
474
475 schedule2_data_all = ""
476 for i in range(len(schedule2_data)):
477     schedule2_data_all = schedule2_data_all + str(schedule2_data[i][0]) + "/" + str(schedule2_data[i][1]) + "/" +
478     str(schedule2_data[i][2]) + "\t" + str(schedule2_data[i][3]) + "\n"
479
480 # こちらは予定データを一つの文字列にまとめたものです。SIROのWebアプリで使します。
481
482
483 # 基本機能7 障害物検知 *****
484
485 USS_TRIG = 24 # 超音波発生トリガー
486 USS_ECHO = 25 # エコー受信
487 # 超音波センサの接続先とGPIOのポート番号を紐づけています。
488
489 GPIO.setup(USS_TRIG, GPIO.OUT)
490 GPIO.setup(USS_ECHO, GPIO.IN)
491 # トリガー送信を出力、エコー受信を入力に設定しています。
492
493
494 uss_distance = 0 # 障害物までの距離 cm
495
496
497 def siro_uss():
498     # 障害物との距離を計測する関数です。
499
500     global uss_distance
501
502     GPIO.output(USS_TRIG, GPIO.HIGH)
503     time.sleep(0.00001)
504     GPIO.output(USS_TRIG, GPIO.LOW)
505     # トリガー信号（10μ秒）を打って超音波を送信します。
506
507
508     while GPIO.input(USS_ECHO) == GPIO.LOW:
509         uss_start = time.time()
510         # 照射エコーを検知しなくなるまで待機します。
511
512
513     while GPIO.input(USS_ECHO) == GPIO.HIGH:
514         uss_end = time.time()
515         # 反射エコーを検知するまで待機しています。
516
517
518     uss_distance = (uss_end - uss_start)*17015
519     # 超音波を送信してから障害物に当たって跳ね返った反射波を受信するまでの時間から障害物までの距離を算出しています。
520     # 17015は34029 cm/s（音速）÷2です。往復距離なので最後に2で割っています。
521
522
523 # 基本機能8 気温・湿度検知 *****
524
525 BME280_sensor = BME280(t_mode=BME280_OSAMPLE_8, p_mode=BME280_OSAMPLE_8,
526                        h_mode=BME280_OSAMPLE_8)
527 # BME280クラスのインスタンス生成、初期化を行っています。
528
529
530 temp = 0.0 # 温度
531 humid = 0.0 # 湿度
532 temp_offset = -2.1
533 humid_offset = 2.0
534 WBGT = 0.0 # 暑さ指数
```

```
528 # 家の温湿度計と差があるのでオフセットをつけて調整しています。
529
530
531 def siro_ondo():
532     # 気温、湿度を検知、WBGTを算出する関数です。
533
534     global temp, humid, WBGT, temp_offset, humid_offset
535
536     temp = BME280_sensor.read_temperature() + temp_offset
537     humid = BME280_sensor.read_humidity() + humid_offset
538     # クラスメソッドを使ってセンサから気温と湿度を読み込んでいます。
539
540     WBGT = 0.725*temp + 0.0368*humid + 0.00364*temp*humid - 3.246
541     # WBGT（暑さ指数）は熱中症予防を目的とした湿度、日射・輻射など周辺の熱環境、気温の3つを取り入れた指標です。
542     # 本来のWBGT計算式は複雑なのですが、ここでは屋内前提の近似式を使っています。
543
544
545 # 基本機能9 ジェスチャー認識 *****
546
547 gesture_result = 0
548 # ジェスチャー認識結果を保存する変数です。
549 # ジェスチャーを検知すると1～9の数字が返ってきます。ジェスチャー未検知の場合は0が返ってきます。
550 # 0:nothing
551 # 1:Forward
552 # 2:Backward
553 # 3:Right
554 # 4:Left
555 # 5:Up
556 # 6:Down
557 # 7:Clockwise
558 # 8:anti-clockwise
559 # 9:wave
560
561 list_gesture = ['なし', '前', '後', '右', '左', '上', '下', '時計回り', '半時計回り', 'バイバイ']
562
563 g = gesture()
564 g.init()
565 # gestureクラスのインスタンス生成、初期化を行っています。
566
567 # 基本機能10 バッテリー残量検知 *****
568
569 version = ""
570 vin = "" # UPSへの入力：UPSが充電中であれば“GOOD”, 充電していなければ“NG”となります。
571 vout = 0 # UPS出力電圧[mV]
572 batcap = 0 # バッテリー残量 0～100[%]
573
574 UPS = UPS2("/dev/ttyAMA0")
575 # UPS2クラスのインスタンス生成、初期化処理を行っています。
576
577
578 def siro_battery():
579     # バッテリー残量を検知する関数です。
580
581     global version, vin, vout, batcap
582
583     version, vin, batcapstr, voutstr = UPS.decode_uart()
584
585     vout = int(voutstr)
586     batcap = int(batcapstr)
587     # 数値に変換します。
588
589
590 # 基本機能11 CPU異常検知 *****
591
592 # 温度監視 *****
593
594 cpu_temp = 0 # CPU温度[℃]
595
596
597 def siro_cpu_temp():
598     # CPU温度[℃]を取得する関数です。
599
600     global cpu_temp
601
602     cmd = "vcgencmd measure_temp"
603     # Raspberry PiのCPU温度を調べるコマンドです。
```

```
604
605     result = subprocess.check_output(cmd.split())
606     # コマンドを実行してその結果を取得しています。
607
608     result = result.decode("utf-8")
609     # コマンド実行結果をデコードして文字列に変換しています。
610
611     index = result.find('temp=')
612     index2 = result.find('')
613     cpu_temp = float(result[index+5:index2])
614     # 文字列 temp=xx.x℃ から数値の部分xx.xを抽出して最後にfloatで文字列を数値に変換しています。
615
616
617 # クロックダウン監視 *****
618
619 cpu_lowvolt = 0
620 cpu_clockdown = 0
621 # CPU電圧低下、クロックダウン検出フラグです。
622 # 異常を検出したらフラグを立てます。
623
624
625 def siro_cpu_clockdown():
626     # CPU電圧低下・クロックダウンを検知する関数です。
627
628     global cpu_clockdown, cpu_lowvolt
629
630     cmd = "vcgencmd get_throttled"
631     # Raspberry PiのCPU電圧低下・クロックダウンの有無を調べるコマンドです。
632
633     result = subprocess.check_output(cmd.split())
634     # コマンドを実行してその結果を取得しています。
635
636     result = result.decode("utf-8")
637     # コマンドの結果をデコードして文字列に変換しています。
638
639     # コマンド実行結果は以下のようになります。
640     # "throttled=0x■■■■■■■■"
641     # 0x0      正常
642     # 0x50000   過去に低電圧状態になった。現在は正常
643     # 0x50005   現在、低電圧状態
644     # 0x80000   過去に熱によりクロックダウンした。現在は正常
645     # 0x80008   現在、熱によりクロックダウン中
646     # 0xD000D   現在、低電圧状態かつ熱によりクロックダウン中
647
648     if ("0x50005" in result): # 現在、低電圧状態にある。
649         cpu_lowvolt = 1
650         cpu_clockdown = 0
651     elif ("0x80008" in result): # 現在、熱によりクロックダウンしている。
652         cpu_lowvolt = 0
653         cpu_clockdown = 1
654     elif ("0xD000D" in result): # 現在、低電圧状態かつ熱によりクロックダウンしている。
655         cpu_lowvolt = 1
656         cpu_clockdown = 1
657     elif (result == "throttled=0x0\n"): # 異常なし
658         cpu_clockdown = 0
659         cpu_lowvolt = 0
660     # 電圧低下、クロックダウンを検知したら検出フラグを立てます。
661
662
663 # 基本機能12 IPアドレス確認 *****
664
665 localipaddress = ""
666 globalipaddress = ""
667 # ローカルIPアドレスとグローバルIPアドレス
668
669
670 def siro_localipaddress():
671     # ローカルIPアドレスを取得する関数です。
672
673     global localipaddress
674
675     cmd = "hostname -I"
676     # ローカルIPアドレスを取得するコマンドです。
677
678     result = subprocess.check_output(cmd.split())
679     # コマンドを実行してその結果を取得しています。
```

```

680
681     result = result.decode("utf-8")
682     # コマンド実行結果をデコードして文字列に変換しています。
683
684     index = result.find('\n')
685     localipaddress = result[0:index-1]
686     # 文字列の最後の改行とスペースを除いています。
687
688
689 def siro_globalipaddress():
690     # グローバルIPアドレスを取得する関数です。
691
692     global globalipaddress
693
694     cmd = "curl ifconfig.me"
695     # グローバルIPアドレスを取得するコマンドです。
696
697     result = subprocess.check_output(cmd.split())
698     # コマンドを実行してその結果を取得しています。
699
700     result = result.decode("utf-8")
701     # コマンド実行結果をデコードして文字列に変換しています。
702
703     globalipaddress = result
704
705
706 # 基本機能13 写真撮影 *****
707
708 def siro_camera():
709     # カメラで写真を撮影する関数です。
710
711     now = datetime.datetime.now()
712     # 日付と時刻を取得しています。
713
714     filename = "/home/kuro/kurosSIRO/data/picture/" + now.strftime("%Y-%m-%d_%H-%M-%S") + ".jpg"
715     # 日付と時刻からファイル名を作成しています。
716
717     ret, frame = cap.read() # 画像取得
718     cv2.imwrite(filename, frame) # 画像保存
719     # カメラ画像を取得しファイル名を指定して保存しています。
720
721
722 # 基本機能14 音声録音 *****
723
724 def siro_voice_record(sec, filename):
725     # 音声録音を行う関数です。
726     # 引数secで録音時間(秒)で指定し、引数filenameでファイル名を指定します。
727
728     cmd = "arecord -f cd -d " + str(sec) + " /home/kuro/kurosSIRO/data/message/" + filename + ".wav"
729     # arecord という標準コマンドを使って録音を行います。
730     # arecorodのオプション: -f cd :CDと同じ音質 -d 録音時間指定
731     # 事前に音声ファイルを保存するフォルダ(筆者の場合は/home/kuro/kurosSIRO/data/message)を準備しておきます。
732
733     subprocess.call(cmd.split())
734     # コマンドを実行し録音を開始します。
735
736
737 # 基本機能15 radiko視聴 *****
738
739 radiko_ch = 0
740 radiko_ch_list = ["KISSFMKOB", "802", "CCL"]
741 # チャンネル選択用の変数とチャンネルリストです。
742 # radikoで無料視聴できるのはネット接続したエリア内の放送局です。筆者の場合は兵庫県の3局になります。
743
744 radio_vol = 70
745 # ラジオの音量 50-100 で調整
746
747 radio_on = 0
748 # ラジオのON/OFFを示すフラグです。
749
750
751 def siro_radio(command):
752     # radikoの再生、停止を行う関数です。引数commandで操作内容を指定します。
753
754     global radiko_ch, radiko_ch_list, radio_on, radio_proc
755

```

```

756 if (command == "start"):
757     # 引数commandが"start"なら再生します。
758
759     cmd = "amixer cset numid=3 " + str(radio_vol) + "%"
760     subprocess.call(cmd.split())
761     # ラジオの音量をセットします。
762
763     cmd = "bash /home/kuro/kurosSIRO/play_radiko.sh -t radiko -s " + radiko_ch_list[radiko_ch]
764     # radiko再生用シェルスクリプトを起動するコマンドです。チャンネルを指定してradikoを再生します。
765
766     radio_proc = subprocess.Popen(cmd.split())
767     # subprocess.Popenを使って並列処理の子プロセスとしてシェルスクリプトを実行しています。
768     # ラジオを停止（子プロセスをKILL）する際にプロセスIDが必要となるので
769     # 戻り値のオブジェクトをグローバル変数radio_procに保存しておきます。
770
771     radio_on = 1
772     # ラジオONフラグを立てます。
773     print("ラジオON", radiko_ch_list[radiko_ch], radiko_ch)
774
775 elif (command == "stop"):
776     # 引数commandが"stop"なら停止します。
777
778     cmd = "ps ho pid --ppid=" + str(radio_proc.pid)
779     # 親プロセスPID（プロセス番号）を指定してその子プロセスのPIDを表示させるコマンドです。
780
781     proc2 = subprocess.Popen(cmd.split(), stdout=subprocess.PIPE)
782     proc2.wait()
783     # subprocess.Popenを使ってコマンドを実行しています。
784
785     for line in proc2.stdout:
786         cmd = 'kill -s SIGINT ' + line.decode()
787         subprocess.call(cmd.split())
788         # 子プロセスを1つずつKILLします。
789
790     cmd = 'kill -s SIGKILL ' + str(radio_proc.pid)
791     subprocess.call(cmd.split())
792     # 子プロセスのKILLが終わったら最後に親プロセス（シェルスクリプト）をKILLします。
793
794     radio_on = 0
795     # ラジオONフラグを下げます。
796     print("ラジオOFF")
797
798
799 # 基本機能16 LINE発信 LINENotify *****
800
801 LINETOKEN = "[]"
802 # LINE通知用トークン ※自分のLINEのマイページから取得します。
803
804 url_LINENotifyAPI = "https://notify-api.line.me/api/notify"
805 # LINENotifyのAPIのURLです。
806
807
808 def siro_Line(LINE_message):
809     # LINENotifyを使って自分のLINEにメッセージを通知する関数です。
810     # 引数LINE_messageで通知するメッセージ（文字列）を指定します。
811
812     headers = {'Authorization': 'Bearer ' + LINETOKEN}
813     payload = {'message': LINE_message}
814     requests.post(url_LINENotifyAPI, headers=headers, data=payload)
815     # LINENotifyAPIにPOSTリクエストを送っています。
816
817
818 def siro_Line_file(LINE_message, LINE_file):
819     # LINENotifyを使って自分のLINEにメッセージ通知+ファイル送信を行う関数です。
820     # 引数LINE_messageで通知する文字列を指定し、引数LINE_fileで送信するファイルを指定します。
821
822     headers = {'Authorization': 'Bearer ' + LINETOKEN}
823     payload = {'message': LINE_message}
824     files = {"imageFile": open(LINE_file, "rb")}
825     requests.post(url_LINENotifyAPI, headers=headers, params=payload, files=files)
826     # LINENotifyAPIにPOSTリクエストを送っています。
827
828
829 # 基本機能18 ウィキペディア検索 *****
830
831 wiki_result = ""

```

```
832 # ウィキペディア検索結果を保存する変数です。
833
834
835 def wikipedia_search(word):
836     # ウィキペディア検索を行う関数です。引数word（文字列）で検索ワードを指定します。
837
838     global wiki_result
839
840     wikipedia.set_lang('ja')
841     # 言語に日本語を設定しています。
842
843     candidate_list = wikipedia.search(word)
844     # 引数wordをウィキペディア検索しています。
845
846     if not candidate_list:
847
848         wiki_result = "該当する単語が存在しませんでした。"
849         # 該当する単語がなければ該当なしと回答します。
850
851     else:
852         try:
853             wiki_result = wikipedia.summary(candidate_list[0])
854             # 要約部分を抽出しています。
855
856         except Exception as e:
857             # エラーの場合もプログラムが停止しないよう例外処理を行います。
858             wiki_result = "検索エラーです。混乱してます。"
859
860
861 # 基本機能19 Webアプリ flask *****
862
863 id_pwd = {"****": "*****"}
864 # WebアプリのログインIDとパスワードを設定しています。（****は任意に設定可能です。）
865
866 app = Flask(__name__, static_folder='/home/kuro/kurosSIRO/data', template_folder='/home/kuro/kurosSIRO/templates')
867 # Flaskクラスのインスタンスを生成しています。
868 # 同時にHTMLファイル他コンテンツファイルの保存場所を指定しています。
869 # テンプレートファイルの保存場所はデフォルトですが、一応、明記しています。
870
871 app.config['TEMPLATES_AUTO_RELOAD'] = True
872 app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0
873 # HTMLファイル、画像ファイルなどコンテンツファイルの読み込み時にキャッシュを使用しない様に設定しています。
874
875 key = os.urandom(21)
876 app.secret_key = key
877 # ログイン情報のセッション管理に使用する秘密鍵を生成しています。
878
879
880 # 以降はルーティング設定です。Webブラウザからのリクエストに対してWebサーバーがどのような処理を行うかを設定します。
881
882 # ルート *****
883 @app.route('/')
884 # Webブラウザがルート（/）をリクエストした場合の処理です。
885 def login_check():
886
887     if not session.get('login'):
888         return redirect('/login')
889         # loginセッションがFalseすなわちログイン前であればログイン画面（/login）に移行します。
890     else:
891         return redirect('/menu')
892         # loginセッションがTrueすなわちログイン済であればメニュー画面（/menu）に移行します。
893
894
895 # ログイン画面 *****
896 @app.route('/login')
897 # Webブラウザがログイン画面（/login）をリクエストした場合の処理です。
898 def login():
899
900     return render_template("login.html")
901     # Flaskのレンダリングテンプレートという機能を使ってログイン画面を表示します。
902     # HTMLファイルは事前に所定のフォルダ（筆者の場合は/home/kuro/kurosSIRO/templates）に保存しておきます。
903
904
905 # ログインチェック *****
906 @app.route('/logincheck', methods=['POST'])
907 # WebブラウザがPOSTメソッドで/logincheckをリクエストした場合の処理です。
```

```
908 # 上記リクエストはログイン画面でログインボタンを押した場合に発生します。
909 # その動作はHTMLファイル（login.html）側での設定になります。
910 def logincheck():
911
912     user_id = request.form['user_id']
913     password = request.form['password']
914     # POSTされたフォーム入力データ（ユーザーIDとパスワード）を取得しています。
915
916     if user_id in id_pwd:
917         if password == id_pwd[user_id]:
918             session['login'] = True
919         else:
920             session['login'] = False
921     else:
922         session['login'] = False
923     # ユーザーID、パスワードともに一致すればloginセッションをTrueにします。
924
925     if not session.get('login'):
926         return render_template("login.html", warning="ユーザーID,パスワードが一致しません。")
927         # loginセッションがFalseなら"ユーザーID,パスワードが一致しません。"と警告します。
928         # Flaskのレンダリングテンプレート機能でlogin.htmlの中の { {warning} } という変数に警告文を反映させます。
929     else:
930         return redirect('/menu')
931         # loginセッションがTrueならメニュー画面(/menu)に移行します。
932
933
934 # メニュー画面 *****
935 @app.route('/menu')
936 # Webブラウザがメニュー画面（/menu）をリクエストした場合の処理です。
937 def menu():
938
939     if not session.get('login'):
940         return redirect('/login')
941         # loginセッションがFalseすなわちログイン前ならログイン画面に移行します。
942     else:
943         return render_template("menu.html")
944         # loginセッションがTrueすなわちログイン済であれば
945         # Flaskのレンダリングテンプレート機能を使ってメニュー画面を表示します。
946
947
948 # ログアウト処理 *****
949 @app.route('/logout')
950 # Webブラウザが/logoutをリクエストした場合の処理です。
951 # 上記リクエストはメニュー画面でlogoutボタンを押したとき場合に発生します。
952 # その動作はHTMLファイル（menu.html）側での設定になります。
953 def logout():
954
955     session.pop('login', None)
956     # loginセッションを削除します。
957
958     return redirect('/login')
959     # ログイン画面に移行します。
960
961
962 # ルーティング設定が終わったらWebサーバーを起動します。
963 def httpserver():
964     # Webサーバーの起動を行う関数です。
965
966     app.run(host='0.0.0.0', port=5000)
967     # Webサーバーを起動しています。
968     # host='0,0,0,0'と指定することでローカルネットワークの外からもWebサーバーにアクセスすることが可能になります。
969     # 0.0.0.0はすべてのホストからのアクセスを受け付けることを意味します。
970     # デフォルト設定では自分自身からしかアクセスできないようになっています。
971     # ポート番号を5000に指定しています。デフォルトで5000なのですが、一応、明示しています。
972
973
974 th_flask = threading.Thread(target=httpserver, daemon=True)
975 th_flask.start()
976 print("SIRO Web application by Flask start...")
977 # 定義した関数をサブスレッドで起動します。
978 # Webサーバーの起動はメイン処理と並列処理となるようサブスレッドで処理します。
979
980
981 # 応用機能1,2 障害物監視、タッチセンサ *****
982
983 uss_alert = 0
```

```
984 # 障害物アラートフラグです。障害物との距離が10cmを下回るとフラグを立てます。
985
986 uss_touch = 0
987 # タッチ検出フラグです。障害物との距離が5cm以下になるとタッチと判断しフラグを立てます。
988
989
990 def siro_usskanshi():
991     # 障害物監視、タッチ検出を行う関数です。
992     # 障害物との距離が10cmを下回った場合、障害物アラートフラグを立てます。
993     # SIROが移動中の場合は一旦モータを停止させます。
994     # また障害物との距離が5cm以下の場合、タッチ検出フラグを立てます。
995
996     global uss_distance, uss_alert, uss_touch, thread_loop
997
998     while thread_loop:
999         # メイン処理終了時にthread_loopを0にしてループから脱出させてサブスレッドを終了させます。
1000
1001         siro_uss()
1002         # まずは障害物までの距離を計測します。
1003
1004         if uss_distance < 10:
1005             # 10cmを下回ると障害物アラートフラグを立てます。
1006             if uss_alert == 0:
1007                 siro_motor("stop")
1008                 # 障害物アラート検出直後はモータを一度停止させます。
1009                 uss_alert = 1
1010             else:
1011                 uss_alert = 0
1012
1013         if uss_distance <= 5:
1014             # 5cm以下でタッチ検出フラグを立てます。
1015             uss_touch = 1
1016         else:
1017             uss_touch = 0
1018
1019         time.sleep(0.5)
1020         # 0.5秒ごとに監視を行います。
1021
1022
1023 th_uss = threading.Thread(target=siro_usskanshi, daemon=True)
1024 th_uss.start()
1025 print("Obstacle detection start.....")
1026 # 定義した関数をサブスレッドで起動します。
1027 # 障害物監視はリアルタイム性が求められる為、処理の遅れが発生しない様、サブスレッドで処理します。
1028
1029
1030 # 応用機能3 気温・湿度監視 *****
1031
1032 temp_kanshi_flag = 0
1033 # 一定時間（10分）ごとに気温・湿度の監視を行うための実行フラグです。
1034 # 監視を行ったらフラグを立て一定時間(10分)待ちます。
1035
1036 tempkanshitime = 0
1037 # 監視を行った時刻を記録するための変数です。
1038
1039
1040 def siro_ondokanshi():
1041     # 気温・湿度・WBGTの監視を行う関数です。
1042     # 気温・湿度を検知し気温またはWBGTが高いと”暑くないですか？”などと声掛けをします。
1043
1044     global temp_kanshi_flag, tempkanshitime
1045
1046     now_UNIX = time.time()
1047     # 現在の時刻（UNIX秒）を取得します。
1048
1049     if temp_kanshi_flag == 0:
1050         # 実行フラグが下がっていたら監視を行います。
1051
1052         siro_ondo()
1053         # 気温と湿度を取得します。
1054
1055         if WBGT < 30 and WBGT > 25:
1056             serif = "現在、気温{0:0.1f}度、湿度{1:0.1f}パーセント、WBGTがすこし高いです。エアコン調整しませんか？".format(temp, humid)
1057             siro_speak(serif)
1058             # 気温とWBGTが高いと声掛けをします。
1059         elif WBGT >= 30:
```

```
1060         serif = "現在、気温{0:0.1f}度、湿度は{1:0.1f}パーセント、WBGTが高いです。エアコンきいてますか?".format(temp, humid)
1061         siro_speak(serif)
1062     elif temp < 30 and temp > 28:
1063         serif = "現在、気温{0:0.1f}度、湿度は{1:0.1f}パーセント、暑くないですか?".format(temp, humid)
1064         siro_speak(serif)
1065     elif temp >= 30:
1066         serif = "現在、気温{0:0.1f}度、湿度は{1:0.1f}パーセント、暑いです.".format(temp, humid)
1067         siro_speak(serif)
1068     # もっと増やしていく予定ですが今のところはここまでです。
1069
1070     temp_kanshi_flag = 1
1071     tempkanshitime = now_UNIX
1072     # 実行フラグを立てて監視時刻を記録します。
1073
1074     if now_UNIX - tempkanshitime > 600:
1075         # 前回の監視から10分経過したら実行フラグを下げます。
1076         temp_kanshi_flag = 0
1077
1078
1079 # 応用機能4 バッテリー残量監視 *****
1080
1081 battery_kanshi_flag = 0
1082 # 一定時間（5分）ごとにバッテリーの監視を行うための実行フラグです。
1083 # 監視を行ったらフラグを立て一定時間(5分)待ちます。
1084
1085 batterykanshitime = 0
1086 # 監視を行った時刻を記録するための変数です。
1087
1088
1089 def siro_batterykanshi():
1090     # バッテリー残量の監視を行う関数です。
1091     # バッテリー残量を検知し30%を下回るとバッテリー低下を知らせます。
1092
1093     global batcap, battery_kanshi_flag, batterykanshitime
1094
1095     now_UNIX = time.time()
1096     # 現在の時刻（UNIX秒）を取得します。
1097
1098     if battery_kanshi_flag == 0:
1099         # 実行フラグが下がっていたら監視を行います。
1100
1101         siro_battery()
1102         # バッテリー残量を取得します。
1103
1104         if batcap < 30:
1105             # バッテリー残量が30%を下回ると警告します。
1106             serif = "バッテリーが{0:0.1f}パーセントです。 배터리が低下しています.".format(batcap)
1107             siro_speak(serif)
1108
1109             battery_kanshi_flag = 1
1110             batterykanshitime = now_UNIX
1111             # 実行フラグを立てて監視時刻を記録します。
1112
1113             if now_UNIX - batterykanshitime > 300:
1114                 # 前回の監視から5分経過したら実行フラグを下げます。
1115                 battery_kanshi_flag = 0
1116
1117
1118 # 応用機能5 CPU異常監視 *****
1119
1120 cpu_temp_kanshi_flag = 0
1121 cputempkanshitime = 0
1122 # 一定時間（5分）ごとにCPU温度監視を行うための実行フラグです。
1123 # 監視を行ったらフラグを立て一定時間(5分)待ちます。
1124
1125 clockdown_kanshi_flag = 0
1126 clockdownkanshitime = 0
1127 # 一定時間（1分）ごとに電圧低下・クロックダウン監視を行うための実行フラグです。
1128 # 監視を行ったらフラグを立て一定時間(1分)待ちます。
1129
1130
1131 def siro_cpu_temp_kanshi():
1132     # CPUの異常監視を行う関数です。
1133
1134     global cpu_temp, cpu_temp_kanshi_flag, cputempkanshitime
```

```
1136 now_UNIX = time.time()
1137 # 現在の時刻（UNIX秒）を取得します。
1138
1139 if cpu_temp_kanshi_flag == 0:
1140     # 実行フラグが下がっていたら監視を行います。
1141
1142     siro_cpu_temp()
1143     # CPU温度を取得します。
1144
1145     if cpu_temp > 50:
1146         # CPU温度が50℃を上回ると警告します。
1147
1148         serif = "CPU温度、{0:0.1f}度です。CPU温度が上昇しています。".format(cpu_temp)
1149         siro_speak(serif)
1150
1151         cpu_temp_kanshi_flag = 1
1152         cputempkanshitime = now_UNIX
1153         # 実行フラグを立てて監視時刻を記録します。
1154
1155 if now_UNIX - cputempkanshitime > 300:
1156     # 前回の監視から5分経過したら実行フラグを下げます。
1157     cpu_temp_kanshi_flag = 0
1158
1159
1160 def siro_cpu_clockdown_kanshi():
1161
1162     global cpu_lowvolt, cpu_clockdown, clockdown_kanshi_flag, clockdownkanshitime
1163     # 電圧低下・クロックダウンの監視を行う関数です。
1164
1165     siro_cpu_clockdown()
1166     # 電圧低下・クロックダウン発生有無を確認します。
1167
1168     now_UNIX = time.time()
1169     # 現在の時刻（UNIX秒）を取得します。
1170
1171     if clockdown_kanshi_flag == 0:
1172         # 実行フラグが下がっていたら監視を行います。
1173
1174         if cpu_lowvolt == 1 and cpu_clockdown == 0:
1175             serif = "低電圧状態です。"
1176             siro_speak(serif)
1177
1178             clockdown_kanshi_flag = 1
1179             clockdownkanshitime = now_UNIX
1180             # 実行フラグを立てて監視時刻を記録します。
1181
1182         elif cpu_clockdown == 0 and cpu_clockdown == 1:
1183             serif = "クロックダウン中です。"
1184             siro_speak(serif)
1185
1186             clockdown_kanshi_flag = 1
1187             clockdownkanshitime = now_UNIX
1188             # 実行フラグを立てて監視時刻を記録します。
1189
1190         elif cpu_clockdown == 1 and cpu_clockdown == 1:
1191             serif = "低電圧状態です。さらにクロックダウン中です。"
1192             siro_speak(serif)
1193
1194             clockdown_kanshi_flag = 1
1195             clockdownkanshitime = now_UNIX
1196             # 実行フラグを立てて監視時刻を記録します。
1197
1198     if now_UNIX - clockdownkanshitime > 60:
1199         # 前回の監視から1分経過したら実行フラグを下げます。
1200         clockdown_kanshi_flag = 0
1201
1202
1203 # 応用機能6 IPアドレスLINE通知 *****
1204 def siro_ipaddressLINE():
1205     # Webアプリで使用するIPアドレスをLINE通知する関数です。
1206
1207     global localipaddress, globalipaddress
1208
1209     siro_localipaddress()
1210     # ローカルIPアドレスを取得します。
1211
```

```
1212 LINE_message = "http://" + localipaddress+":5000"
1213 siro_Line(LINE_message)
1214 # LINEで通知します。
1215
1216 siro_globalipaddress()
1217 # グローバルIPアドレスを取得します。
1218
1219 LINE_message = "http://" + globalipaddress+":5000"
1220 siro_Line(LINE_message)
1221 # LINEで通知します。
1222
1223
1224 # 応用機能7 写真撮影LINE通知 *****
1225
1226 def siro_camera_Line():
1227     # カメラで写真を撮ってLINE通知する関数です。
1228
1229     now = datetime.datetime.now()
1230     # 日付と時刻を取得します。
1231
1232     filename = "/home/kuro/kurosSIRO/data/picture/" + now.strftime("%Y-%m-%d_%H-%M-%S")+".jpg"
1233     # 日付と時刻からファイル名を作成します。
1234
1235     ret, frame = cap.read() # カメラ画像取得
1236     cv2.imwrite(filename, frame) # 画像保存
1237     # カメラ画像を取得しファイル名を指定して保存しています。
1238
1239     siro_Line_file(filename, filename)
1240     # 画像をLINE通知します。
1241
1242
1243 # 応用機能8 あいさつ *****
1244
1245 def siro_greeting(who):
1246     # SIROにあいさつをさせる関数です。
1247     # 引数whoであいさつする人をIDNo. で指定します。指定しない場合は100を入力します。
1248
1249     global yobi
1250
1251     now = datetime.datetime.now()
1252     # 時刻を取得します。
1253
1254     if who == 100:
1255         # あいさつをする人を指定しない場合は100
1256
1257         if now.hour >= 0 and now.hour < 9:
1258             siro_speak("おはようございます。")
1259         elif now.hour >= 9 and now.hour < 18:
1260             siro_speak("こんにちわです。")
1261         elif now.hour >= 18 and now.hour < 21:
1262             siro_speak("こんばんわです。")
1263         elif now.hour >= 21 and now.hour < 24:
1264             siro_speak("ねむたいです。")
1265         # 時刻によって適切なあいさつをします。
1266
1267     else:
1268         # あいさつをする人をIDNo. で指定します。
1269         if now.hour >= 0 and now.hour < 9:
1270             siro_speak(yobi[who] + "、おはようございます。")
1271         elif now.hour >= 9 and now.hour < 18:
1272             siro_speak(yobi[who] + "、こんにちわです。")
1273         elif now.hour >= 18 and now.hour < 21:
1274             siro_speak(yobi[who] + "、こんばんわです。")
1275         elif now.hour >= 21 and now.hour < 24:
1276             siro_speak("ねむたいです。")
1277
1278
1279 # 応用機能9 予定を伝える(曜日ごと) *****
1280
1281 def siro_schedule1(who, when):
1282     # 予定データ(曜日)から予定を抽出する関数です。
1283     # 引数whoで誰の予定かを選択し、引数whenでいつの予定を確認するかを選択します。
1284
1285     global schedule1_data
1286
1287     now = datetime.datetime.now()
```

```

1288 # 日付と時刻を取得します。
1289
1290 yotei = ""
1291
1292 if who == 100:
1293     # 人を選択しない場合は100を入力します。
1294
1295     if when == "今日":
1296         for i in range(len(schedule1_data)):
1297             if schedule1_data[i][now.weekday()+3] != "x":
1298                 yotei = yotei + schedule1_data[i][2] + 'の' + schedule1_data[i][now.weekday() + 3] + "、"
1299             if yotei == "":
1300                 yotei2 = "今日は予定はありません。"
1301             else:
1302                 yotei2 = "今日の予定は" + yotei + "以上です。"
1303     if when == "明日":
1304         for i in range(len(schedule1_data)):
1305             if now.weekday() == 6:
1306                 if schedule1_data[i][3] != "x":
1307                     yotei = yotei + schedule1_data[i][2] + "の" + schedule1_data[i][3] + "、"
1308                 else:
1309                     if schedule1_data[i][now.weekday() + 4] != "x":
1310                         yotei = yotei + schedule1_data[i][2] + "の" + schedule1_data[i][now.weekday() + 4] + "、"
1311             if yotei == "":
1312                 yotei2 = "明日は予定はありません。"
1313             else:
1314                 yotei2 = "明日の予定は" + yotei + "以上です。"
1315     if when == "明後日":
1316         for i in range(len(schedule1_data)):
1317             if now.weekday() == 5:
1318                 if schedule1_data[i][3] != "x":
1319                     yotei = yotei + schedule1_data[i][2] + "の" + schedule1_data[i][3] + "、"
1320             if now.weekday() == 6:
1321                 if schedule1_data[i][4] != "x":
1322                     yotei = yotei + schedule1_data[i][2] + "の" + schedule1_data[i][4] + "、"
1323                 else:
1324                     if schedule1_data[i][now.weekday()+5] != "x":
1325                         yotei = yotei + schedule1_data[i][2] + "の" + schedule1_data[i][now.weekday() + 5] + "、"
1326             if yotei == "":
1327                 yotei2 = "明後日は予定はありません。"
1328             else:
1329                 yotei2 = "明後日の予定は" + yotei + "以上です。"
1330
1331 else:
1332     # 人を選択する場合
1333     if when == "今日":
1334         if schedule1_data[who][now.weekday()+3] != "x":
1335             yotei = yotei + schedule1_data[who][2] + 'の' + schedule1_data[who][now.weekday() + 3] + "、"
1336         if yotei == "":
1337             yotei2 = "今日は予定はありません。"
1338         else:
1339             yotei2 = "今日の予定は" + yotei + "です。"
1340     if when == "明日":
1341         if now.weekday() == 6:
1342             if schedule1_data[who][3] != "x":
1343                 yotei = yotei + schedule1_data[who][2] + "の" + schedule1_data[who][3] + "、"
1344             else:
1345                 if schedule1_data[who][now.weekday() + 4] != "x":
1346                     yotei = yotei + schedule1_data[who][2] + "の" + schedule1_data[who][now.weekday() + 4] + "、"
1347             if yotei == "":
1348                 yotei2 = "明日は予定はありません。"
1349             else:
1350                 yotei2 = "明日の予定は" + yotei + "です。"
1351     if when == "明後日":
1352         if now.weekday() == 5:
1353             if schedule1_data[who][3] != "x":
1354                 yotei = yotei + schedule1_data[who][2] + "の" + schedule1_data[who][3] + "、"
1355             if now.weekday() == 6:
1356                 if schedule1_data[who][4] != "x":
1357                     yotei = yotei + schedule1_data[who][2] + "の" + schedule1_data[who][4] + "、"
1358                 else:
1359                     if schedule1_data[who][now.weekday()+5] != "x":
1360                         yotei = yotei + schedule1_data[who][2] + "の" + schedule1_data[who][now.weekday() + 5] + "、"
1361             if yotei == "":
1362                 yotei2 = "明後日は予定はありません。"
1363             else:

```

```
1364         yotei2 = "明後日の予定は" + yotei + "です。"
1365
1366     return yotei2
1367
1368
1369 # 応用機能10 予定を伝える（日ごと）*****
1370
1371 def siro_schedule2(when):
1372     # 予定データ(日)から予定を抽出する関数です。
1373     # 引数whenでいつの予定を確認するかを選択します。
1374
1375     global schedule2_data
1376
1377     now = datetime.datetime.now()
1378     # 日付と時刻を取得します。
1379
1380     yotei = ""
1381
1382     if when == "今日":
1383         for i in range(len(schedule2_data)):
1384             if schedule2_data[i][0] == now.year:
1385                 if schedule2_data[i][2] == now.day:
1386                     yotei = yotei + schedule2_data[i][3] + "、"
1387
1388             if yotei == "":
1389                 yotei2 = "今日は予定はありません。"
1390             else:
1391                 yotei2 = "今日は" + yotei + "です。"
1392
1393     elif when == "今月":
1394         for i in range(len(schedule2_data)):
1395             if schedule2_data[i][0] == now.year:
1396                 if schedule2_data[i][1] == now.month:
1397                     if schedule2_data[i][2] >= now.day:
1398                         yotei = yotei + str(schedule2_data[i][1]) + "月" + str(schedule2_data[i][2]) + "日、" +
1399                             str(schedule2_data[i][3]) + "、"
1400
1401             if (yotei == ""):
1402                 yotei2 = "今月の予定はありません。"
1403             else:
1404                 yotei2 = "今月の予定は" + yotei + "です。"
1405
1406     elif when == "今年":
1407         for i in range(len(schedule2_data)):
1408             if schedule2_data[i][0] >= now.year:
1409                 if schedule2_data[i][1] >= now.month:
1410                     if schedule2_data[i][2] >= now.day:
1411                         yotei = yotei + str(schedule2_data[i][1]) + "月" + str(schedule2_data[i][2]) + "日、" +
1412                             str(schedule2_data[i][3]) + "、"
1413
1414             if yotei == "":
1415                 yotei2 = "今年の予定はありません。"
1416             else:
1417                 yotei2 = "今年の予定は" + yotei + "です。"
1418
1419     elif when == "来月":
1420         if now.month == 12:
1421             month = 1
1422             year = now.year + 1
1423         else:
1424             month = now.month + 1
1425             year = now.year
1426         for i in range(len(schedule2_data)):
1427             if schedule2_data[i][0] == year:
1428                 if schedule2_data[i][1] == month:
1429                     yotei = yotei + str(schedule2_data[i][1]) + "月" + str(schedule2_data[i][2]) + "日" +
1430                         str(schedule2_data[i][3]) + "、"
1431
1432             if yotei == "":
1433                 yotei2 = "来月の予定はありません。"
1434             else:
1435                 yotei2 = "来月の予定は" + yotei + "です。"
1436
1437     elif when == "明日":
1438         now = now + datetime.timedelta(days=1)
1439         for i in range(len(schedule2_data)):
1440             if schedule2_data[i][0] == now.year:
1441                 if schedule2_data[i][1] == now.month:
1442                     if schedule2_data[i][2] == now.day:
```

```
1437         yotei = yotei + schedule2_data[i][3] + "、"
1438
1439     if yotei == "":
1440         yotei2 = "明日の予定はありません。"
1441     else:
1442         yotei2 = "明日は" + yotei + "です。"
1443
1444     elif when == "明後日":
1445         now = now + datetime.timedelta(days=2)
1446         for i in range(len(schedule2_data)):
1447             if schedule2_data[i][0] == now.year:
1448                 if schedule2_data[i][1] == now.month:
1449                     if schedule2_data[i][2] == now.day:
1450                         yotei = yotei + schedule2_data[i][3] + "、"
1451
1452         if yotei == "":
1453             yotei2 = "明後日の予定はありません。"
1454         else:
1455             yotei2 = "明後日は" + yotei + "です。"
1456
1457     return yotei2
1458
1459
1460 # 応用機能11 鳩時計 *****
1461
1462 hatoclock_flag1 = 0
1463 hatoclock_flag2 = 0
1464 # 鳩時計行動フラグです。
1465 # 0分になったら鳩時計（0分）を鳴らしてflag1を立てます。0分以外になったらフラグをおろして次に備えます。
1466 # 30分になったら鳩時計（30分）を鳴らしてflag2を立てます。30分以外になったらフラグをおろして次に備えます。
1467
1468 hatoclocktime = 0
1469 # 鳩時計を鳴らした時刻を記録するための変数です。
1470
1471
1472 def siro_hatoclock():
1473     # 鳩時計として時刻を知らせる関数です。
1474
1475     global hatoclock_flag1, hatoclock_flag2, hatoclocktime
1476
1477     now = datetime.datetime.now() # datetimeオブジェクト
1478     now_UNIX = time.time() # UNIX秒
1479     # 時刻を取得します。
1480
1481     if now.minute == 0 and hatoclock_flag1 == 0:
1482         # 0分になったら時刻の回数ポッポッポッと鳴きます。
1483
1484         if now.hour > 12:
1485             for i in range(now.hour - 12):
1486                 siro_sound("はと")
1487             serif = str(now.hour - 12) + "時です。"
1488             siro_speak(serif)
1489         else:
1490             for i in range(now.hour):
1491                 siro_sound("はと")
1492             serif = str(now.hour) + "時です。"
1493             siro_speak(serif)
1494
1495         hatoclock_flag1 = 1
1496         hatoclocktime = now_UNIX
1497         # 何度も鳴かないようにフラグを立てて時刻(UNIX秒)を記録します。
1498
1499     if now.minute == 30 and hatoclock_flag2 == 0:
1500         # 30分になったら1回だけ鳴きます。
1501
1502         siro_sound("はと")
1503         if now.hour > 12:
1504             serif = str(now.hour - 12) + "時半です。"
1505         else:
1506             serif = str(now.hour) + "時半です。"
1507         siro_speak(serif)
1508
1509         hatoclock_flag2 = 1
1510         hatoclocktime = now_UNIX
1511         # 何度も鳴かないようにフラグを立てて時刻(UNIX秒)を記録します。
1512
```

```
1513     # 1分経過したらフラグを下げます。
1514     if hatoclock_flag1 == 1 and now_UNIX - hatoclocktime > 60:
1515         hatoclock_flag1 = 0
1516     if hatoclock_flag2 == 1 and now_UNIX - hatoclocktime > 60:
1517         hatoclock_flag2 = 0
1518
1519
1520 # 応用機能12 メッセージ録音 *****
1521
1522 messages = [[0, 0, datetime.datetime.now()], [0, 0, datetime.datetime.now()], [0, 0, datetime.datetime.now()]]
1523 # メッセージの有無、誰のメッセージか？、録音した日時を保存する変数です。
1524 # メッセージは3件まで録音可能とします。
1525 # まだ100%完成していませんが、とりあえずメッセージの録音、再生、消去はできるようになりました。
1526
1527
1528 def siro_message(command, kenme):
1529     # メッセージの録音、再生、削除を行う関数です。
1530     # 引数commandで録音か再生または削除かを選択します。
1531     # 削除する場合は引数kenmeで何件目のメッセージを削除するかを指定します。
1532
1533     global messages
1534
1535     if command == "録音":
1536
1537         for i in range(3):
1538             if messages[i][0] == 0:
1539                 # messageの空を確認します。
1540
1541                 siro_speak("15秒でメッセージをどうぞ。")
1542                 siro_sound("ピー")
1543                 cmd = "arecord -f cd -d 15 /home/kuro/kurosSIRO/data/message/{0:d}.wav".format(i)
1544                 subprocess.call(cmd.split())
1545                 siro_speak("録音を終了します。")
1546                 # arecordコマンドで録音します。
1547                 # arecorodのオプション： -f cd :CDと同じ音質 -d 録音時間
1548
1549                 messages[i][0] = 1
1550                 # 録音済フラグを立てます。
1551
1552                 break
1553                 # メッセージを録音したらforループから脱出します。
1554
1555             if i == 2:
1556                 # messageに空きがない場合は諦めます。
1557                 siro_speak("既に3件のメッセージが録音されています。")
1558
1559     elif command == "再生":
1560
1561         aaa = 0
1562         for i in range(3):
1563             aaa = aaa + messages[i][0]
1564             # メッセージの有無を確認します。
1565
1566         if aaa == 0:
1567             # メッセージが無い場合
1568             siro_speak("メッセージはありません。")
1569
1570         else:
1571             # メッセージがある場合
1572             serif = str(aaa) + "件のメッセージがあります。"
1573             siro_speak(serif)
1574
1575             for i in range(3):
1576                 if messages[i][0] == 1:
1577                     serif = str(i+1) + "件目のメッセージを再生します。"
1578                     siro_speak(serif)
1579                     cmd = "aplay /home/kuro/kurosSIRO/data/message/{0:d}.wav".format(i)
1580                     subprocess.call(cmd.split())
1581                     # 順番にメッセージを再生します。
1582
1583             serif = "メッセージは以上です。"
1584             siro_speak(serif)
1585
1586     elif command == "消去":
1587         # 引数kenmeで選択したメッセージを消去します。
1588         # kenmeが10の場合はすべてのメッセージを消去します。
```

1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664

```

    if kenme == 1:
        siro_speak("一件目のメッセージを消去します。")
        messages[0][0] = 0
        # 録音済フラグを下げます。
        # 音声ファイルは削除せずにそのままです。新たにメッセージを録音する場合はファイルを上書きします。
        siro_sound("メッセージ消去")

    elif kenme == 2:
        siro_speak("二件目のメッセージを消去します。")
        messages[1][0] = 0
        siro_sound("メッセージ消去")
    elif kenme == 3:
        siro_speak("三件目のメッセージを消去します。")
        messages[2][0] = 0
        siro_sound("メッセージ消去")
    elif kenme == 10:
        siro_speak("すべてのメッセージを消去します。")
        for i in range(3):
            messages[i][0] = 0
        siro_sound("メッセージ消去")

# 応用機能13 タイマー *****

timers = [[0, "", time.time()], [0, "", time.time()], [0, "", time.time()]]
# タイマーセットの有無、セット時間、タイマーセット時刻を保存する変数です。
# タイマーは3件までセット可能とします。

def siro_timer():
    # タイマー処理を行う関数です。

    global thread_loop, timers

    while thread_loop:
        # メイン処理終了時にthread_loopを0にしてループから脱出させてサブスレッドを終了させます。

        now_UNIX = time.time()
        # 時刻 (UNIX秒) を確認します。

        for i in range(3):

            if timers[i][0] == 1 and now_UNIX-timers[i][2] > 0:
                # タイマーセットの有無を確認し、タイマーがセットされていればタイマーセット時刻を過ぎていないかを確認します。

                siro_sound("タイマー")
                serif = "{0:d}件目、{1:s}のタイマーを終了します.".format(i+1, timers[i][1])
                siro_speak(serif)
                # タイマーを鳴らします。
                # 本当は"タイマーストップ"というまでタイマーを鳴らし続けたいのですが、SIROはサウンドを鳴らしながらの音声認識ができません。。。
                # Alexa (スマートスピーカー) は本当に優秀です。

                timers[i][0] = 0
                # タイマーをリセットします。

            time.sleep(1)

th_timer = threading.Thread(target=siro_timer, daemon=True)
th_timer.start()
print("Timer start.....")
# 定義した関数をサブスレッドで起動します。
# タイマー処理はリアルタイム性が求められる為、処理の遅れが発生しない様、サブスレッドで処理します。

def siro_timer_set(command, minutes, kenme):
    # タイマーのセット、リセットを行う関数です。
    # 引数commandでセットかリセットを選択し、引数minutesでタイマー時間を指定します。
    # タイマーをリセットする場合は引数kenmeで何件目のタイマーをリセットするかを指定します。

    global timers

    now_UNIX = time.time()
    # 時刻 (UNIX秒) を確認します。
```

```

1665     if command == "セット":
1666
1667         for i in range(3):
1668             if timers[i][0] == 0:
1669                 # timerの空さを確認します。
1670
1671                 serif = minutes + "のタイマーをセットします。"
1672                 siro_speak(serif)
1673
1674                 timers[i][0] = 1
1675                 # タイマーフラグを立てます。
1676
1677                 timers[i][1] = minutes
1678                 # セット時間（分）をタイマーに入力します。
1679
1680                 index = minutes.find('分')
1681                 minutes2 = int(minutes[0:index])
1682                 # "〇分"から数値部分〇を抽出します。
1683
1684                 timers[i][2] = now_UNIX + minutes2*60
1685                 # セット時刻を入力します。
1686
1687                 break
1688                 # タイマーをセットしたらforループから抜けます。
1689
1690             if i == 2:
1691                 siro_speak("既に3件のタイマーがセットされています。")
1692                 # タイマーがすでに3件セット済であればその旨を伝えます。
1693
1694 elif command == "確認":
1695     # タイマーの有無を確認します。
1696
1697     aaa = 0
1698     for i in range(3):
1699         aaa = aaa + timers[i][0]
1700
1701     if aaa == 0:
1702         siro_speak("タイマーはセットされていません。")
1703         # タイマーがセットされていなければその旨を伝えます。
1704
1705     else:
1706         serif = str(aaa) + "件のタイマーがあります。"
1707         siro_speak(serif)
1708
1709         for i in range(3):
1710             if timers[i][0] == 1: # タイマーの空さを確認
1711                 serif = str(i+1) + "件目"+timers[i][1] + "のタイマーがあります。のこり" + str(int(timers[i][2] - now_UNIX)) + "秒です。"
1712                 siro_speak(serif)
1713                 # タイマーがセットされていれば、何分のタイマーがのこり何秒かを伝えます。
1714
1715 elif command == "キャンセル":
1716
1717     if kenme == 1:
1718         # 引数kenmeで何件目のタイマーをキャンセルするか選択肢します。
1719         siro_speak("一件目のタイマーをキャンセルします。")
1720         siro_sound("メッセージ消去")
1721         timers[0][0] = 0
1722         # タイマーフラグを下げます。
1723
1724     elif kenme == 2:
1725         siro_speak("二件目のタイマーをキャンセルします。")
1726         siro_sound("メッセージ消去")
1727         messages[1][0] = 0
1728
1729     elif kenme == 3:
1730         siro_speak("三件目のタイマーをキャンセルします。")
1731         siro_sound("メッセージ消去")
1732         messages[2][0] = 0
1733
1734     elif kenme == 10:
1735         # 全件キャンセルする場合はkenme=10とします。
1736         siro_speak("すべてのタイマーをキャンセルします。")
1737         siro_sound("メッセージ消去")
1738         for i in range(3):
1739             timers[i][0] = 0

```

```
1740
1741
1742 # 応用機能14 Webアプリ Auto mode *****
1743
1744 @app.route('/automode', methods=['GET'])
1745 # Automode画面です。
1746 # SIROの制御プログラムの動作チェックに使用しています。
1747 # 顔認識処理の状態、音声認識結果、各種センサの値,CPU温度他各種パラメータなどを表示します。
1748 def automode():
1749
1750     return render_template("automode.html", mode=mode_string[mode], siro_dontmove=siro_dontmove, siro_silent=siro_silent,
1751                             communication=communication, speaking=speaking, face_counter=face_counter,
1752                             face_position=face_position, words=words, uss_distance=int(uss_distance), uss_alert=uss_alert,
1753                             uss_touch=uss_touch, temp=int(temp), humid=int(humid), WBGT=int(WBGT), batcap=batcap,
1754                             gesture_result=list_gesture[gesture_result],
1755                             cpu_temp=cpu_temp, cpu_lowvolt=cpu_lowvolt, cpu_clockdown=cpu_clockdown,
1756                             globalipaddress=globalipaddress, localipaddress=localipaddress,
1757                             messages=messages, timers=timers)
1758
1759     # Flaskのレンダリングテンプレートという機能を使ってAutomode画面を表示します。
1760
1761
1762 # 応用機能14 Webアプリ RadioControl *****
1763
1764 @app.route('/radiocontrol', methods=['GET'])
1765 # RadioControl画面です。
1766 # SIROをラジコンで操作できます。
1767 # カメラ画像とコントローラーを表示します。
1768 def radiocontrol():
1769
1770     global mode, g_parameter
1771
1772     mode = 4 # ラジコンモード
1773     g_parameter = [0, 0, 0, 0, 0] # 汎用パラメータ初期化
1774     # ラジコンモードに移行します。
1775
1776     return render_template("radiocontrol.html")
1777
1778     # Flaskのレンダリングテンプレートという機能を使ってRadioControl画面を表示します。
1779
1780
1781
1782 @app.route('/motor_go', methods=['GET'])
1783 # コントローラーの前進ボタンがクリックされたときの処理です。
1784 def motor_go():
1785
1786     siro_motor("go")
1787     # 前進します。
1788
1789     return ('OK', 200)
1790
1791     # HTTPリスポンスを返します。
1792     # 画面更新は行いません。
1793
1794
1795 @app.route('/motor_left', methods=['GET'])
1796 # コントローラーの左回転ボタンがクリックされたときの処理です。
1797 def motor_left():
1798
1799     siro_motor("left")
1800     # 左回転します。
1801
1802     return ('OK', 200)
1803
1804     # HTTPリスポンスを返します。
1805     # 画面更新は行いません。
1806
1807
1808 @app.route('/motor_right', methods=['GET'])
1809 # コントローラーの右回転ボタンがクリックされたときの処理です。
1810 def motor_right():
1811
1812     siro_motor("right")
1813     # 右回転します。
1814
1815     return ('OK', 200)
1816
1817     # HTTPリスポンスを返します。
1818     # 画面更新は行いません。
1819
1820
1821 @app.route('/motor_back', methods=['GET'])
```

```
1813 # コントローラーの後進ボタンがクリックされたときの処理です。
1814 def motor_back():
1815
1816     siro_motor("back")
1817     # 後進します。
1818
1819     return ('OK', 200)
1820     # HTTPリスポンスを返します。
1821     # 画面更新は行いません。
1822
1823
1824 @app.route('/motor_stop', methods=['GET'])
1825 # コントローラーの停止ボタンがクリックされたときの処理です。
1826 def motor_stop():
1827
1828     siro_motor("stop")
1829     # 停止します。
1830
1831     return ('OK', 200)
1832     # HTTPリスポンスを返します。
1833     # 画面更新は行いません。
1834
1835
1836 @app.route('/texttospeak', methods=['POST'])
1837 # コントローラーの発話ボタンがクリックされたときの処理です。
1838 def texttospeak():
1839
1840     text = request.form['text']
1841     # 発話する文字列を取得します。
1842
1843     siro_speak(text)
1844     # 発話します。
1845
1846     return ('OK', 200)
1847     # HTTPリスポンスを返します。
1848     # 画面更新は行いません。
1849
1850
1851 # カメラ画像表示 *****
1852 # MotionJPEGという仕組みを使用しています。
1853 # サーバーからJPEG画像を連続して送ることでWebブラウザ上で動画になります。
1854 # Pythonではジェネレータ (yield文) を使って実現します。
1855 # HTMLファイルに  100:
1905         radio_vol = 100
1906     # 音量パラメータradio_volを上げます。最大100です。
1907
1908     cmd = "amixer cset numid=3 " + str(radio_vol) + "%"
1909     subprocess.call(cmd.split())
1910     # 音量セットします。
1911
1912     return ('OK', 200)
1913     # HTTPレスポンスを返します。
1914     # 画面更新は行いません。
1915
1916
1917 @app.route('/radio_voldown', methods=['GET'])
1918 # コントローラーの▼ボタンがクリックされたときの処理です。
1919 def radio_voldown():
1920
1921     global radio_vol
1922
1923     radio_vol = radio_vol - 10
1924     if radio_vol < 40:
1925         radio_vol = 40
1926     # 音量パラメータradio_volを下げます。最小40です。
1927
1928     cmd = "amixer cset numid=3 " + str(radio_vol) + "%"
1929     subprocess.call(cmd.split())
1930     # 音量セットします。
1931
1932     return ('OK', 200)
1933     # HTTPレスポンスを返します。
1934     # 画面更新は行いません。
1935
1936
1937 @app.route('/radio_ch1', methods=['GET'])
1938 # コントローラーの▶▶ボタンがクリックされたときの処理です。
1939 def radio_ch1():
1940
1941     global radiko_ch
1942
1943     radiko_ch = radiko_ch + 1
1944     if radiko_ch == 3:
1945         radiko_ch = 0
1946     # 選局パラメータradio_chを+1
1947
1948     siro_radio("stop")
1949     siro_radio("start")
1950     # ラジオを停止、再生することでチャンネル変更しています。
1951
1952     return render_template("radiko.html", radiko_ch=radiko_ch_list[radiko_ch])
1953     # 放送局のロゴを変更するため、画面更新します。
1954
1955
1956 @app.route('/radio_ch2', methods=['GET'])
1957 # コントローラーの◀◀ボタンがクリックされたときの処理です。
1958 def radio_ch2():
1959
1960     global radiko_ch
1961
1962     radiko_ch = radiko_ch - 1
1963     if radiko_ch == -1:
1964         radiko_ch = 2
```

```
1965     # 選局パラメータradio_chを-1
1966
1967     siro_radio("stop")
1968     siro_radio("start")
1969     # ラジオを停止、再生することでチャンネル変更しています。
1970
1971     return render_template("radiko.html", radiko_ch=radiko_ch_list[radiko_ch])
1972     # 放送局のロゴを変更するため、画面更新します。
1973
1974
1975 @app.route('/radio_startstop', methods=['GET'])
1976 # コントローラーの▶/■ボタンがクリックされたときの処理です。
1977 def radio_startstop():
1978
1979     global radio_on
1980
1981     if radio_on == 1:
1982         siro_radio("stop")
1983     else:
1984         siro_radio("start")
1985     # ラジオ再生フラグradio_onを確認し再生中なら停止し停止中なら再生します。
1986
1987     return ('OK', 200)
1988     # HTTPリスポンスを返します。
1989     # 画面更新は行いません。
1990
1991
1992 # 応用機能14 Webアプリ Schedule1 *****
1993
1994 ALLOWED_EXTENSIONS = {'txt', 'csv'}
1995 # アップロードを許可するファイル形式を指定しています。
1996
1997
1998 def allwed_file(filename):
1999     # 許可されたファイルかどうかを確認する関数です。
2000     # 引数filenameでファイル名を指定します。
2001     # OKなら 1、だめなら0を返します。
2002
2003     return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
2004
2005
2006
2007
2008 @app.route('/schedule1', methods=['GET'])
2009 # Schedule1画面です。
2010 # 予定データ（週ごと）の確認、編集ができます。
2011 # 予定データ（週ごと）を表示します。また、編集の為のダウンロードボタンとアップロードボタンがあります。
2012 def schedule1():
2013
2014     return render_template("schedule1.html", schedule1_data=schedule1_data)
2015     # Flaskのレンダリングテンプレートという機能を使ってSchedule1画面を表示します。
2016
2017
2018
2019
2020 @app.route('/schedule1_download', methods=['GET'])
2021 # ダウンロードボタンがクリックされたときの処理です。
2022 def schedule1_download():
2023
2024     downloadDir = '/home/kuro/kurosSIRO/data'
2025     downloadFile = 'schedule1.csv'
2026     downloadFileName = 'schedule1.csv'
2027
2028     return send_from_directory(downloadDir, downloadFile, as_attachment=True,
2029                               attachment_filename=downloadFileName, cache_timeout=0, mimetype="text/csv")
2030     # 予定データ（週ごと）ファイルを送ります。
2031     # cache_timeout=0 でキャッシュを無効にしています。(キャッシュを無効にしないと最新ファイルがダウンロードされません。)
2032
2033
2034
2035
2036 @app.route('/schedule1_upload', methods=['POST'])
2037 # アップロードボタンがクリックされたときの処理です。
2038 def schedule1_upload():
2039
2040     global schedule1_data
2041
2042     uploadDir = '/home/kuro/kurosSIRO/data'
2043     uploadFile = 'schedule1.csv'
2044
2045     if 'file' not in request.files:
2046         return render_template("schedule1.html", schedule1_data=schedule1_data)
```

```
2041         # ファイルが選択されていない場合は何もせずに画面を再読み込みします。
2042     file = request.files['file']
2043
2044     if allwed_file(file.filename) == 0:
2045         # 許可されたファイルでない場合は何もせずに画面を再読み込みします。
2046         return render_template("schedule1.html", schedule1_data=schedule1_data)
2047     else:
2048         file.save(os.path.join(uploadDir, uploadFile))
2049         # 許可されたファイルであればファイルを保存します。
2050
2051         schedule1_data = pandas.read_csv('/home/kuro/kurosSIRO/data/schedule1.csv', header=0).values.tolist()
2052         # # pandasを使ってCSVデータを配列で読み込みます。
2053
2054         return render_template("schedule1.html", schedule1_data=schedule1_data)
2055         # 画面を更新します。
2056
2057
2058 # 応用機能14 Webアプリ Schdule2 *****
2059
2060 @app.route('/schedule2', methods=['GET'])
2061 # Schedule2画面です。
2062 # 予定データ（日ごと）の確認、編集ができます。
2063 # 予定データ（日ごと）を表示します。また、編集の為のダウンロードボタンとアップロードボタンがあります。
2064 def schedule2():
2065     return render_template("schedule2.html", schedule2_data_all=schedule2_data_all)
2066     # Flaskのレンダリングテンプレートという機能を使ってSchedule2画面を表示します。
2067
2068
2069 @app.route('/schedule2_download', methods=['GET'])
2070 # ダウンロードボタンがクリックされたときの処理です。
2071 def schedule2_download():
2072     downloadDir = '/home/kuro/kurosSIRO/data'
2073     downloadFile = 'schedule2.csv'
2074     downloadFileName = 'schedule2.csv'
2075
2076     return send_from_directory(downloadDir, downloadFile, as_attachment=True,
2077                               attachment_filename=downloadFileName, cache_timeout=0, mimetype="text/csv")
2078     # 予定データ（日ごと）ファイルを送ります。
2079     # cache_timeout=0 でキャッシュを無効にしています。(キャッシュを無効にしないと最新ファイルがダウンロードされません。)
2080
2081
2082 @app.route('/schedule2_upload', methods=['POST'])
2083 # アップロードボタンがクリックされたときの処理です。
2084 def schedule2_upload():
2085
2086     global schedule2_data, schedule2_data_all
2087     uploadDir = '/home/kuro/kurosSIRO/data'
2088     uploadFile = 'schedule2.csv'
2089
2090     if 'file' not in request.files:
2091         return render_template("schedule2.html", schedule2_data_all=schedule2_data_all)
2092         # ファイルが選択されていない場合は何もせずに画面を再読み込みします。
2093     file = request.files['file']
2094
2095     if allwed_file(file.filename) == 0:
2096         # 許可されたファイルでない場合は何もせずに画面を再読み込みします。
2097         return render_template("schedule2.html", schedule2_data_all=schedule2_data_all)
2098     else:
2099         file.save(os.path.join(uploadDir, uploadFile))
2100         # 許可されたファイルであればファイルを保存します。
2101
2102         schedule2_data = pandas.read_csv('/home/kuro/kurosSIRO/data/schedule2.csv', header=0).values.tolist()
2103         # pandasを使ってCSVデータを配列で読み込みます。
2104
2105         schedule2_data_all = ""
2106         for i in range(len(schedule2_data)):
2107             schedule2_data_all = schedule2_data_all + str(schedule2_data[i][0]) + "年" + str(schedule2_data[i][1])\
2108                 + "月" + str(schedule2_data[i][2]) + "日: " + str(schedule2_data[i][3]) + "\n"
2109
2110         return render_template("schedule2.html", schedule2_data_all=schedule2_data_all)
2111         # 画面を更新します。
2112
2113
2114 # 応用機能14 Webアプリ ウィキペディア検索 *****
2115 # メニュー画面で検索ワードを入力して"ウィキペディア検索"ボタンを押したときの処理です。
2116 @app.route('/wikipedia', methods=['POST'])
```

```
2117 def wiki():
2118
2119     word = request.form['word']
2120     # 検索ワードを取得します。
2121
2122     wikipedia_search(word)
2123     # ウィキペディア検索します。
2124
2125     siro_speak2(wiki_result)
2126     # 検索結果をSIROに読み上げさせます。
2127
2128     return render_template("wikipedia.html", word=word, result=wiki_result)
2129     # 検索結果を画面表示します。
2130
2131
2132 def siro_speak2(serif):
2133     # SIROに発話させる関数パート2です。
2134     # ウィキペディア検索結果読み上げ用に作成しました。
2135     # googletexttospeechというPythonモジュールを使用しています。
2136     # こちらは長文のセリフでも使用できます。(逆にOpenJtalkは長文だとエラーになります。)
2137     # 発話の開始と同時に検索結果画面が表示されるようsubprocess.Popenを使って並列処理で音声再生します。
2138
2139     global sp_vol
2140
2141     tts = gTTS(serif, lang="ja")
2142     tts.save("/tmp/out.mp3")
2143     # googletexttospeechを使って音声ファイルを作成します。
2144
2145     cmd = "amixer cset numid=3 "+str(sp_vol)+"%"
2146     subprocess.call(cmd.split())
2147     # 音量を設定します。
2148
2149     cmd = "mplayer -speed 1.2 -af scaletempo /tmp/out.mp3"
2150     subprocess.Popen(cmd.split())
2151     # 作成した音声ファイルを再生します。
2152     # subprocess.Popenを使って並列処理で再生します。
2153
2154
2155 # 2. メイン処理パート *****
2156 # ここからようやくメイン処理に入ります。
2157
2158 print("SIRO Start.....",)
2159 siro_speak("しろ起動します。")
2160
2161 siro_ipaddressLINE()
2162 # 応用機能6 IPアドレスLINE通知
2163
2164 try: # try,except 構文, ctrl +C (キーボード割り込み)で終了処理に移ります。
2165     while True: # メイン処理ループです。
2166
2167         # 共通処理 *****
2168
2169         siro_batterykanshi()
2170         # 応用機能4 バッテリー残量監視
2171
2172         siro_cpu_temp_kanshi()
2173         # 応用機能5 CPU温度監視
2174
2175         siro_cpu_clockdown_kanshi()
2176         # 応用機能5 CPU異常監視
2177
2178         # 1. のんびりモード *****
2179         if mode == 1:
2180
2181             print("のんびりモード。")
2182
2183             # 初期処理 *****
2184             if g_parameter[0] == 0:
2185
2186                 siro_motor_speed = 70 # モータ速度を設定します。
2187                 g_parameter[0] = 1
2188
2189             # 応用機能3 気温・湿度監視 *****
2190             siro_ondokanshi()
2191
2192             # 応用機能11 鳩時計 *****
```

```
2193 siro_hatoclock()
2194
2195 # 応用機能15 コミュニケーション *****
2196 # 家族の顔を見つけるとコミュニケーションします。
2197 # コミュニケーションの進度を表すパラメータ (communication) で行動を制御します。
2198
2199 for i in range(4):
2200     if face_counter[i] >= 8:
2201         if communication[i] == 0:
2202             siro_greeting(i) # 応用機能8 あいさつ
2203             communication[i] += 1 # コミュニケーション進度+ 1
2204         elif communication[i] == 1:
2205             serif = siro_schedule1(i, "今日") # 応用機能9 予定を伝える
2206             siro_speak(serif)
2207             communication[i] += 1
2208             # とりあえず骨子だけ完成。。つづきは今後作成予定です。。
2209
2210 # 応用機能16 巡回 *****
2211 # 家の中を巡回します。とりあえずランダムウォークです。
2212 # 本当は自己位置推定, マッピングなどを行いたいのですが、そのためにはまずセンサの増強が必要です。。
2213 # とりあえず骨子だけ完成させてつづきは今後作成予定です。。
2214
2215 if siro_dontmove != 1:
2216     ransuint = random.randint(1, 2000) # 乱数生成
2217     if ransuint >= 1 and ransuint < 10:
2218         siro_motor("go") # 前進
2219         time.sleep(4)
2220         siro_motor("stop")
2221     elif ransuint >= 10 & ransuint < 20:
2222         siro_motor("right") # 右旋回
2223         time.sleep(2)
2224         siro_motor("stop")
2225     elif ransuint >= 20 and ransuint < 30:
2226         siro_motor("left") # 左旋回
2227         time.sleep(2)
2228         siro_motor("stop")
2229     elif ransuint >= 30 and ransuint < 40:
2230         siro_motor("right") # 右小旋回
2231         time.sleep(1)
2232         siro_motor("stop")
2233     elif ransuint >= 40 and ransuint < 50:
2234         siro_motor("left") # 左小旋回
2235         time.sleep(1)
2236         siro_motor("stop")
2237
2238 # 応用機能17 顔サーチ *****
2239 # 家族の顔を見つけると振り向きます。
2240
2241 for i in range(4):
2242     if face_counter[i] >= 8 and face_position[i][0] != 0:
2243
2244         if face_position[i][0] < - face_position[i][1]/2:
2245             siro_motor("left")
2246             time.sleep(0.4)
2247             siro_motor("stop")
2248         elif face_position[i][0] > face_position[i][1]/2:
2249             siro_motor("right")
2250             time.sleep(0.4)
2251             siro_motor("stop")
2252
2253 # 応用機能18 ひとりごと *****
2254 # ランダムに独り言を言います。
2255 if siro_silent != 1:
2256     ransuint = random.randint(1, 1000) # 乱数生成
2257     if ransuint == 1:
2258         siro_speak("しろちゃんです。") # 自己紹介
2259     elif ransuint == 2:
2260         siro_ondo()
2261         serif = "げんざい、きおん、{0:0.1f}ど。しつど、{1:0.1f}パーセント".format(temp, humid) # 気温と湿度を伝える。
2262         siro_speak(serif)
2263     elif ransuint == 3:
2264         serif = siro_schedule2("今月") # 今月の予定を伝える。
2265         siro_speak(serif)
2266         # とりあえず骨子だけ完成。。つづきは今後作成予定です。。
2267
2268 # 応用機能19 音声認識へのリアクション *****
```

```

2269 if voice_recog == 1:
2270     # 音声認識完了フラグが立っていたら、音声認識結果wordsを確認してリアクションします。
2271
2272     for i in range(10):
2273         print('WORD: {0:s}{1:s}\n'.format(words[i][0], words[i][1]))
2274         # 音声認識結果を表示して確認します。
2275
2276     # 名前を呼ばれたときのリアクション
2277     if 'しろ' in words[1][0] and '['/s]' == words[2][0] and float(words[1][1]) > 0.8:
2278         siro_speak("なんですか？")
2279
2280     # ワードへのリアクション
2281     for i in range(10):
2282
2283         if 'ラジオ' in words[i][0] and float(words[i][1]) > 0.9:
2284
2285             siro_ipaddressLINE()
2286             # 応用機能6 IPアドレスLINE通知
2287             # ラジオ操作にWebアプリを使用するのでIPアドレスをLINE通知します。
2288
2289             mode = 3 # ラジオモード
2290             g_parameter = [0, 0, 0, 0, 0] # 汎用パラメータ初期化
2291             # ラジオモードに移行します。
2292             break
2293
2294         elif 'IPアドレス' in words[i][0] and float(words[i][1]) > 0.8:
2295
2296             siro_ipaddressLINE()
2297             # 応用機能6 IPアドレスLINE通知
2298             break
2299
2300         elif 'こんにちは' in words[i][0] and float(words[i][1]) > 0.9:
2301
2302             siro_greeting(100)
2303             # 応用機能8 あいさつ
2304             break
2305
2306         elif 'こんばんは' in words[i][0] and float(words[i][1]) > 0.9:
2307
2308             siro_greeting(100)
2309             # 応用機能8 あいさつ
2310             break
2311
2312         elif '予定' in words[i][0] and float(words[i][1]) > 0.9:
2313             # 応用機能9,10 予定を伝える
2314
2315             for j in range(len(words)):
2316                 if '今日' in words[j][0] and float(words[j][1]) > 0.8:
2317
2318                     serif = siro_schedule1(100, '今日')
2319                     siro_speak(serif)
2320                     serif = siro_schedule2('今日')
2321                     siro_speak(serif)
2322                     break
2323
2324                 elif '明日' in words[j][0] and float(words[j][1]) > 0.8:
2325
2326                     serif = siro_schedule1(100, '明日')
2327                     siro_speak(serif)
2328                     serif = siro_schedule2('明日')
2329                     siro_speak(serif)
2330                     break
2331
2332                 if '明後日' in words[j][0] and float(words[j][1]) > 0.8:
2333
2334                     serif = siro_schedule1(100, '明後日')
2335                     siro_speak(serif)
2336                     serif = siro_schedule2('明後日')
2337                     siro_speak(serif)
2338                     break
2339
2340                 if '今月' in words[j][0] and float(words[j][1]) > 0.8:
2341
2342                     serif = siro_schedule2("今月")
2343                     siro_speak(serif)
2344                     break

```

```

2345
2346         if '来月' in words[j][0] and float(words[j][1]) > 0.8:
2347
2348             serif = siro_schedule2("来月")
2349             siro_speak(serif)
2350             break
2351
2352         if '今年' in words[j][0] and float(words[j][1]) > 0.8:
2353
2354             serif = siro_schedule2("今年")
2355             siro_speak(serif)
2356             break
2357
2358     elif 'しゃしん' in words[i][0] and float(words[i][1]) > 0.9:
2359
2360         siro_speak("写真を撮ります。")
2361         siro_camera_Line()
2362         # 応用機能7 写真撮影LINE発信
2363         siro_sound("カメラ") # シャッター音
2364         break
2365
2366     elif 'メッセージ' in words[i][0] and float(words[i][1]) > 0.9:
2367         # 応用機能12 メッセージ
2368
2369         for j in range(len(words)):
2370             if '録音' in words[j][0] and float(words[j][1]) > 0.8:
2371                 siro_message("録音", 0)
2372                 break
2373             elif '再生' in words[j][0] and float(words[j][1]) > 0.8:
2374                 siro_message("再生", 0)
2375                 break
2376             elif '消去' in words[j][0] and float(words[j][1]) > 0.8:
2377                 siro_speak("全部ですか？それとも何件目ですか？")
2378                 mode = 2 # 聞き耳モードに以降
2379                 g_parameter = [0, 1, 0, 0, 0] # 聞き耳を立てる対象を汎用パラメータで指定します。
2380                 break
2381
2382     elif 'タイマー' in words[i][0] and float(words[i][1]) > 0.9:
2383         # 応用機能13 タイマー
2384
2385         for j in range(len(words)):
2386             if 'かけて' in words[j][0] and float(words[j][1]) > 0.6:
2387                 for k in range(len(words)):
2388                     if '分' in words[k][0] and float(words[k][1]) > 0.4:
2389                         siro_timer_set("セット", words[k][0], 0)
2390                         break
2391             elif ('おしえて' in words[j][0] and float(words[j][1]) > 0.8) or ('あと何分' in words[j][0] and
2392 float(words[j][1]) > 0.8):
2393                 siro_timer_set("確認", "", 0)
2394                 break
2395             elif 'キャンセル' in words[j][0] and float(words[j][1]) > 0.8:
2396                 siro_speak("全部ですか？それとも何件目ですか？")
2397                 mode = 2 # 聞き耳モードに以降
2398                 g_parameter = [0, 2, 0, 0, 0] # 聞き耳を立てる対象を汎用パラメータで指定します。
2399                 break
2400
2401     elif ('名前' in words[i][0] and float(words[i][1]) > 0.6) or ('誰' in words[i][0] and float(words[i][1])
2402 > 0.8):
2403
2404         siro_speak("しろちゃんです。")
2405         break
2406
2407     elif 'ただいま' in words[i][0] and float(words[i][1]) > 0.8:
2408
2409         siro_speak("おかえりなさい。")
2410         break
2411
2412     elif 'ってきます' in words[i][0] and float(words[i][1]) > 0.8:
2413
2414         siro_speak("いってらっしゃい")
2415         break
2416
2417     elif 'おかえり' in words[i][0] and float(words[i][1]) > 0.9:
2418
2419         siro_speak("おかえりなさい")
2420         break

```

```
2419
2420 elif 'おはよう' in words[i][0] and float(words[i][1]) > 0.9:
2421
2422     siro_speak("おはようございます。")
2423     break
2424
2425 elif 'おっす' in words[i][0] and float(words[i][1]) > 0.9:
2426
2427     siro_speak("おっす。おらしろちゃん。")
2428     break
2429
2430 elif 'しずかにして' in words[i][0] and float(words[i][1]) > 0.9:
2431
2432     siro_speak("しずかにします。")
2433     siro_silent = 1
2434     break
2435
2436 elif '温度' in words[i][0] and float(words[i][1]) > 0.9:
2437
2438     serif = "{0:0.1f}どです。しつどは{1:0.1f}%です。".format(temp, humid)
2439     siro_speak(serif)
2440     break
2441
2442 elif '湿度' in words[i][0] and float(words[i][1]) > 0.9:
2443
2444     serif = "現在のしつどは、{0:0.1f}%です。".format(humid)
2445     siro_speak(serif)
2446     break
2447
2448 elif ('今何時' in words[i][0] and float(words[i][1]) > 0.9) or ('今の時刻' in words[i][0] and
float(words[i][1]) > 0.9):
2449
2450     now = datetime.datetime.now() # 現在の日付時刻を取得
2451     if now.hour <= 12:
2452         serif = str(now.hour) + "時" + str(now.minute) + "分です。"
2453     else:
2454         serif = str(now.hour-12) + "時" + str(now.minute) + "分です。"
2455     siro_speak(serif)
2456     break
2457
2458 elif '何曜' in words[i][0] and float(words[i][1]) > 0.7:
2459     now = datetime.datetime.now() # 現在の日付時刻を取得
2460     for j in range(len(words)):
2461         if '今日' in words[j][0] and float(words[j][1]) > 0.8:
2462             serif = "今日は、" + list_week[now.weekday()] + "です。"
2463             siro_speak(serif)
2464             break
2465         elif '昨日' in words[j][0] and float(words[j][1]) > 0.8:
2466             if now.weekday() == 0:
2467                 serif = "昨日は、" + list_week[6] + "です。"
2468             else:
2469                 serif = "昨日は、" + list_week[now.weekday()-1] + "です。"
2470             siro_speak(serif)
2471             break
2472         elif '明日' in words[j][0] and float(words[j][1]) > 0.8:
2473             if now.weekday() == 6:
2474                 serif = "明日は、" + list_week[0] + "です。"
2475             else:
2476                 serif = "明日は、" + list_week[now.weekday()+1] + "です。"
2477             siro_speak(serif)
2478             break
2479         elif '明後日' in words[j][0] and float(words[j][1]) > 0.8:
2480             if now.weekday() == 5:
2481                 serif = "あさっては、" + list_week[0] + "です。"
2482             elif now.weekday() == 6:
2483                 serif = "あさっては、" + list_week[1] + "です。"
2484             else:
2485                 serif = "あさっては、" + list_week[now.weekday()+2] + "です。"
2486             siro_speak(serif)
2487             break
2488 elif ('何月' in words[i][0] and float(words[i][1]) > 0.9) or ('何日' in words[i][0] and
float(words[i][1]) > 0.9):
2489     now = datetime.datetime.now() # 現在の日付時刻を取得
2490     for j in range(len(words)):
2491         if '今日' in words[j][0] and float(words[j][1]) > 0.8:
2492             serif = "今日は、" + str(now.month) + "月" + str(now.day) + "日、" + list_week[now.weekday()]
```

```

2493         + "です。"
2494         siro_speak(serif)
2495         break
2496     elif '昨日' in words[j][0] and float(words[j][1]) > 0.8:
2497         now = now - datetime.timedelta(days=1)
2498         serif = "昨日は、" + str(now.month) + "月" + str(now.day) + "日、" + list_week[now.weekday()]
2499         + "です。"
2500         siro_speak(serif)
2501         break
2502     elif '明日' in words[j][0] and float(words[j][1]) > 0.8:
2503         now = now + datetime.timedelta(days=1)
2504         serif = "明日は、" + str(now.month) + "月" + str(now.day) + "日、" + list_week[now.weekday()]
2505         + "です。"
2506         siro_speak(serif)
2507         break
2508     elif '明後日' in words[j][0] and float(words[j][1]) > 0.8:
2509         now = now + datetime.timedelta(days=2)
2510         serif = "あさっては、" + str(now.month) + "月" + str(now.day) + "日、" +
2511             list_week[now.weekday()] + "です。"
2512         siro_speak(serif)
2513         break
2514
2515     elif 'ありがと' in words[i][0] and float(words[i][1]) > 0.6:
2516
2517         siro_speak("どういたしまして。")
2518         break
2519
2520     elif 'どうも' in words[i][0] and float(words[i][1]) > 0.9:
2521
2522         siro_sound("ごきげん")
2523         break
2524
2525     elif 'たすかります' in words[i][0] and float(words[i][1]) > 0.9:
2526
2527         siro_speak("とんでもございません。")
2528         break
2529
2530     elif 'いいこ' in words[i][0] and float(words[i][1]) > 0.8:
2531
2532         siro_sound("ごきげん")
2533         break
2534
2535     elif 'かしこい' in words[i][0] and float(words[i][1]) > 0.7:
2536
2537         siro_sound("ごきげん")
2538         break
2539
2540     elif 'かわいい' in words[i][0] and float(words[i][1]) > 0.7:
2541
2542         siro_sound("ごきげん")
2543         break
2544
2545     elif '天気' in words[i][0] and float(words[i][1]) > 0.9:
2546         # わからないことはとりあえずAlexa（スマートスピーカー）に聞きます。
2547         # Alexaは優秀なのでSIROの声にも反応してくれます。
2548
2549         for j in range(len(words)):
2550             if '今日' in words[j][0] and float(words[j][1]) > 0.8:
2551                 siro_speak("あれくさ。今日の天気はどうですか？")
2552                 break
2553             elif '明日' in words[j][0] and float(words[j][1]) > 0.8:
2554                 siro_speak("あれくさ。明日の天気はどうですか？")
2555                 break
2556             elif '明後日' in words[j][0] and float(words[j][1]) > 0.8:
2557                 siro_speak("あれくさ。あさっての天気はどうですか？")
2558                 break
2559
2560     elif '何の日' in words[i][0] and float(words[i][1]) > 0.9:
2561         # わからないことはとりあえずAlexa（スマートスピーカー）に聞きます。
2562         # Alexaは優秀なのでSIROの声にも反応してくれます。
2563
2564         for j in range(len(words)):
2565             if '今日' in words[j][0] and float(words[j][1]) > 0.8:
2566                 siro_speak("あれくさ。今日はなんの日ですか？")
2567                 break
```

```
2565
2566         elif '昨日' in words[j][0] and float(words[j][1]) > 0.8:
2567             siro_speak("あれくさ。昨日はなんの日ですか？")
2568             break
2569
2570         elif '明日' in words[j][0] and float(words[j][1]) > 0.8:
2571             siro_speak("あれくさ。明日はなんの日ですか？")
2572             break
2573
2574         elif '明後日' in words[j][0] and float(words[j][1]) > 0.8:
2575             siro_speak("あれくさ。あさってはなんの日ですか？")
2576             break
2577
2578     voice_recog = 0
2579     # 音声認識結果にリアクションしたら音声認識結果読み込み完了フラグを下げます。
2580     # そうすることでJuliusサーバーからのデータ読み込みが再開します。
2581
2582     # 応用機能20 ジェスチャー認識へのリアクション *****
2583
2584     gesture_result = g.return_gesture()
2585     # ジェスチャー入力結果を読み込みます。
2586
2587     # if gesture_result == 1:
2588     #     どんなリアクションさせるかは現在作成中です。とりあえず骨子部分だけ完成させました。
2589
2590     time.sleep(0.5)
2591     # 処理周期を調整します。
2592
2593     elif mode == 2: # 聞き耳モード *****
2594         # 状況別に特定のワードを聞き分けさせたい場合に使用するモードです。
2595         # 汎用パラメータg_parameter[1]で状況を判別します。
2596         # g_parameter[1]: 1 : 消去するメッセージの回答待ち
2597         #                     2 : キャンセルするタイマーの回答待ち
2598
2599         print("聞き耳モード。")
2600
2601         # 応用機能19 音声認識へのリアクション *****
2602         if voice_recog == 1:
2603             # 音声認識完了フラグが立っていたら、音声認識結果wordsを確認してリアクションします。
2604
2605             for i in range(10):
2606                 print('WORD: {0:s}{1:s}\n'.format(words[i][0], words[i][1]))
2607                 # 音声認識結果を表示して確認します。
2608
2609                 if g_parameter[1] == 1:
2610                     # 応用機能12 メッセージ /消去するメッセージの回答待ち
2611
2612                     if '全部' in words[i][0] and float(words[i][1]) > 0.9:
2613
2614                         siro_message("消去", 10) # 全メッセージを消去
2615                         mode = 1 # のんびりモードに移行
2616                         g_parameter = [0, 0, 0, 0, 0]
2617                         break
2618
2619                     if '一件目' in words[i][0] and float(words[i][1]) > 0.9:
2620                         siro_message("消去", 1) # 1件目のメッセージを消去
2621                         mode = 1 # のんびりモードに移行
2622                         g_parameter = [0, 0, 0, 0, 0]
2623                         break
2624
2625                     elif '二件目' in words[i][0] and float(words[i][1]) > 0.9:
2626                         siro_message("消去", 2) # 2件目のメッセージを消去
2627                         mode = 1 # のんびりモードに移行
2628                         g_parameter = [0, 0, 0, 0, 0]
2629                         break
2630
2631                     elif '三件目' in words[i][0] and float(words[i][1]) > 0.9:
2632                         siro_message("消去", 3) # 3件目のメッセージを消去
2633                         mode = 1 # のんびりモードに移行
2634                         g_parameter = [0, 0, 0, 0, 0]
2635                         break
2636
2637                 if g_parameter[1] == 2:
2638                     # 応用機能12 タイマー /キャンセルするタイマーの回答待ち
2639
2640                     if '全部' in words[i][0] and float(words[i][1]) > 0.9:
```

```
2641
2642         siro_timer_set("キャンセル", "", 10) # 全タイマーをキャンセル
2643         mode = 1 # のんびりモードに移行
2644         g_parameter = [0, 0, 0, 0, 0]
2645         break
2646
2647     if '一件目' in words[i][0] and float(words[i][1]) > 0.9:
2648         siro_timer_set("キャンセル", "", 1) # 1件目のタイマーをキャンセル
2649         mode = 1 # のんびりモードに移行
2650         g_parameter = [0, 0, 0, 0, 0]
2651         break
2652
2653     elif '二件目' in words[i][0] and float(words[i][1]) > 0.9:
2654         siro_timer_set("キャンセル", "", 2) # 2件目のタイマーをキャンセル
2655         mode = 1 # のんびりモードに移行
2656         g_parameter = [0, 0, 0, 0, 0]
2657         break
2658
2659     elif '三件目' in words[i][0] and float(words[i][1]) > 0.9:
2660         siro_timer_set("キャンセル", "", 3) # 3件目のタイマーをキャンセル
2661         mode = 1 # のんびりモードに移行
2662         g_parameter = [0, 0, 0, 0, 0]
2663         break
2664
2665     voice_recog = 0
2666     # 音声認識結果にリアクションしたら音声認識結果読み込み完了フラグを下げます。
2667
2668     # 一定時間経過したらのんびりモードに戻ります。
2669     g_parameter[2] = g_parameter[2] + 1 # カウントアップ
2670     if g_parameter[2] > 30: # とりあえず約15秒
2671         mode = 1 # のんびりモードに移行します。
2672         g_parameter = [0, 0, 0, 0, 0] # 汎用パラメータ初期化
2673
2674     time.sleep(0.5)
2675     # 処理周期を調整します。
2676
2677 elif mode == 3: # ラジオモード *****
2678
2679     print("ラジオモード。")
2680
2681     # 初期処理 *****
2682     if g_parameter[0] == 0:
2683
2684         siro_speak("ラジオモード。")
2685         siro_radio("start")
2686         g_parameter[0] = 1
2687         # ラジオをつけて初期処理完了フラグを上げます。
2688
2689     # 応用機能20 ジェスチャー認識へのリアクション *****
2690
2691     gesture_result = g.return_gesture()
2692     # ジェスチャー入力結果を読み込みます。
2693
2694     if gesture_result == 5: # 上:音量を上げます。
2695
2696         radio_vol = radio_vol + 10
2697         if radio_vol > 100:
2698             radio_vol = 100
2699
2700         cmd = "amixer cset numid=3 "+str(radio_vol)+"%"
2701         subprocess.call(cmd.split())
2702         # 音量を設定してます。
2703
2704     elif gesture_result == 6: # 下:音量を下げます。
2705
2706         radio_vol = radio_vol - 10
2707         if radio_vol < 40:
2708             radio_vol = 40
2709         # 40%を最小に設定しています。
2710
2711         cmd = "amixer cset numid=3 " + str(radio_vol) + "%"
2712         subprocess.call(cmd.split())
2713         # 音量を設定してます。
2714
2715     elif gesture_result == 3: # 右:局変更します。
2716
```

```

2717         radiko_ch = radiko_ch + 1
2718     if radiko_ch == 3:
2719         radiko_ch = 0
2720
2721     siro_radio("stop")
2722     siro_radio("start")
2723     # 一旦ラジオを停止し、再び再生することでチャンネルを変更しています。
2724     # スマートではないですがこうすることでradiko再生シェルスクリプトをそのまま流用できます。
2725
2726 elif gesture_result == 4:    # 左：局変更します。
2727
2728     radiko_ch = radiko_ch - 1
2729     if radiko_ch == -1:
2730         radiko_ch = 3
2731
2732     siro_radio("stop")
2733     siro_radio("start")
2734
2735 elif gesture_result == 7:    # 右回り:音量を上げます。
2736
2737     radio_vol = radio_vol + 10
2738
2739     if radio_vol > 100:
2740         radio_vol = 100
2741
2742     cmd = "amixer cset numid=3 " + str(radio_vol) + "%"
2743     subprocess.call(cmd.split())
2744     # 音量を設定してます。
2745
2746 elif gesture_result == 8:    # 左回り:音量を下げます。
2747     radio_vol = radio_vol - 10
2748
2749     if radio_vol < 40:
2750         radio_vol = 40
2751         # 40%を最小に設定しています。
2752
2753     cmd = "amixer cset numid=3 " + str(radio_vol) + "%"
2754     subprocess.call(cmd.split())
2755     # 音量を設定してます。
2756
2757 elif gesture_result == 1:    # 手を近づける：停止 or 再生
2758
2759     if radio_on == 1:
2760         siro_radio("stop")    # ラジオ停止
2761     else:
2762         siro_radio("start")    # ラジオ再生
2763
2764 elif gesture_result == 9:    # 手をふる。:ラジオモード終了
2765
2766     if radio_on == 1:
2767         siro_radio("stop")    # ラジオ停止
2768     siro_speak("ラジオモードを終了します。")
2769     mode = 1    # のんびりモードに移行
2770     g_parameter = [0, 0, 0, 0, 0]    # 汎用パラメータ初期化
2771
2772 # 応用機能21 タッチ認識へのリアクション *****
2773 if uss_touch == 1:
2774
2775     # 超音波センサでタッチを検出したらラジオモードを終了してのんびりモードに戻ります。
2776     if radio_on == 1:
2777         siro_radio("stop")    # ラジオ停止
2778     siro_speak("ラジオモードを終了します。")
2779     mode = 1    # のんびりモードに移行
2780     g_parameter = [0, 0, 0, 0, 0]    # 汎用パラメータ初期化
2781
2782     time.sleep(0.5)
2783     # 処理周期を調整します。
2784
2785 elif mode == 4:    # ラジコンモード *****
2786
2787     print("ラジコンモード。")
2788
2789     # 初期処理 *****
2790     if g_parameter[0] == 0:
2791
2792         siro_speak("ラジコンモード。")

```

