平成 30 年度　特別研究報告書

冷蔵庫内の氷量を自動管理するシステムの検討

龍谷大学　理工学部　情報メディア学科

T150536　松永　昂生

指導教員　三好　力　教授

内容梗概

現在、冷蔵庫はどの世帯にも 1 台はあり、多くの人は何か飲み物を飲もうとした時に氷を使おうとしたら氷ができておらず氷がない状態に出くわし、困ったことがあるだろう。その原因として既存の冷蔵庫では、氷の量が少ない時にそれを伝えるユーザーに伝える術を持っていないことだと考えた。そのため、氷がないという状態にならないようにさせる自動管理システムを検討した。本研究では、超音波センサーを用いて氷量を測定し、取得した情報を自宅にある Wifi などを用いてインターネット上にアップロードし、氷などの量が少ない場合にはインターネットを通じて自動的に利用者へ通知を送る手法を提案した。そこで、Raspberry Pi 3、Google メール,スプレッドシートを利用し、自動的に測定、記録、アップロード、メール送信を行うプログラムを作成し、超音波センサーで取得したデータを Google スプレッドシートにアップロードし、氷の量が少ない場合には Gmail を用いてメールを送信させるという提案したシステムの実現に必要な機能が動作するか実験した。

目次

第 1 章　はじめに

近年、IoT と呼ばれる技術が進歩し様々なモノがインターネットに繋がっている。IoT とは、Internet of Things の略称で、モノのインターネットという意味であり、今までインターネットに繋がっていなかったモノがコミュニケーションを行うための情報伝達路を作ることである。しかし、身近にある家電などで IoT が使われているモノはテレビやエアコン、照明などもともとリモートコントローラーがあるモノがスマートフォンなどと繋がることができるモノが多い。それに比べ、冷蔵庫や洗濯機などリモートコントローラーがない家電には IoT が使われていないモノがまだ多くあり、これらに IoT を導入する事でさらに生活を豊かにすることができると考えられる。そこで本研究では冷蔵庫に着目した。

現在、冷蔵庫はどの世帯にも 1 台はある。冷蔵庫の用途は食材を冷蔵保存して食材の鮮度を保ったり、冷凍保存して長期保存したり、水分を冷やしたりと生活の中で様々なことによく使われる生活必需品である。そのような中、冷蔵庫で氷を作ることはほとんどの人がしているだろう。氷は清涼飲料水やお酒を飲むとき、料理や患部を冷やすなど主にものを冷やすこと使われ、夏場のかき氷などがでてくるような暑い時期になるとかき氷に氷を使うためだけでなく、ものを冷やすためにも氷の使用頻度が増える傾向あり、必然的に冷蔵庫で氷を作る機会が増え、一人暮らしではなく、世帯のところは人が多い分氷の消費量は増えるであろう。氷が必要な時に氷を作るために使われるのが家庭用の大型冷蔵庫についてある自動製氷機能である。自動製氷機能は給水タンクの中に水を入れておくだけで、給水タンク内に水がある限り、自動で氷を作り、貯氷ケースに貯めることができる。しかし給水タンク内の水が無くなっていると氷を作れないという欠点がある。その欠点により多くの人は何か飲み物を飲もうとした時に氷を使おうとしたら給水タンクに水がなく、氷を作ることができていなくて、困ったことがあるだろう。このように氷が不足している事態を避けるためには、給水タンク内に水がどのくらい入っていて、貯氷ケースにはどの程度氷が貯まっているかを把握する必要がある。そのためには氷を使う際に毎回冷蔵庫内の給水タンクの水量と冷凍庫の貯氷ケースに溜まっている氷の量を確認していないとできない。冷蔵庫に対しては中の食品の消費期限の管理、在庫の食品による料理のレシピの提案などの研究が行われている。しかし製氷機能についての研究がほとんどない。冷蔵庫に IoT を導入し、スマートフォンから貯氷ケース内の氷の有無や給水タンクの水量の情報をインターネットにアップロードすることで家の中や外などどこにいてもスマートフォンがあれば家の冷蔵庫内の氷の量を確認ができ、氷が少なくなると自動的にユーザーに通知を送信して知らせることができるシステムを提案する。このシステムを使うことで、いつでも現在家にある氷の量を把握することができ、また通知により貯氷ケースに氷がなく、給水タンクにも水が入ってなくて氷が作れてないという状況を把握できれば、帰り道に氷を買って帰るなり、帰ってすぐに給水タンクに水を補充したりすることができる。スマートフォンと冷蔵庫を IoT

技術を使い、繋げることにより氷がない状況でも柔軟に対応できることにより氷不足という事態が起こらないこと期待できる。

さらにインターネット上の天気予報からその日の天気や気温を自動で取得し、その情報から体感温度などを算出することでその日に使われる氷の使用量を推定することもできる。そうすることで利用者が氷の残量を自発的に現在の氷の量を確認できるだけではなく、暑い日には氷を多く使うことを推測して、利用者に向けて早めに通知を出すなどできることが期待できる。

第2章　既存技術

2.1 自動製氷機

現在の冷蔵庫の自動製氷機の氷を作る基本的なシステムは給水タンクにある水が冷凍室にある製氷皿に注がれ、数時間後にはブロック状の氷となり貯氷ケースに貯められるようになっている。(図 1)また、自動製氷機には氷の量を調べるセンサーがついており、給水タンクに十分な水があり、製氷皿に氷ができている際に、センサーが貯氷ケース内の氷の量を検知し、貯氷ケースにも十分な氷があるときは氷を作らないようになっている。この貯氷ケースに十分な氷があるか検知するセンサーは製氷皿の下に検知レバーというものがついている。(図 2)氷を作る段階で、製氷皿から氷を貯氷ケースに氷を落とすときに、この検知レバーが貯氷ケースのところに降りて、検知レバーに氷が触れると氷があると認識して製氷皿から貯氷ケースに氷を移さず、検知レバーになにも触れなければ、氷がないと認識して、製氷皿から貯氷ケースに氷を移す仕組みになっている。

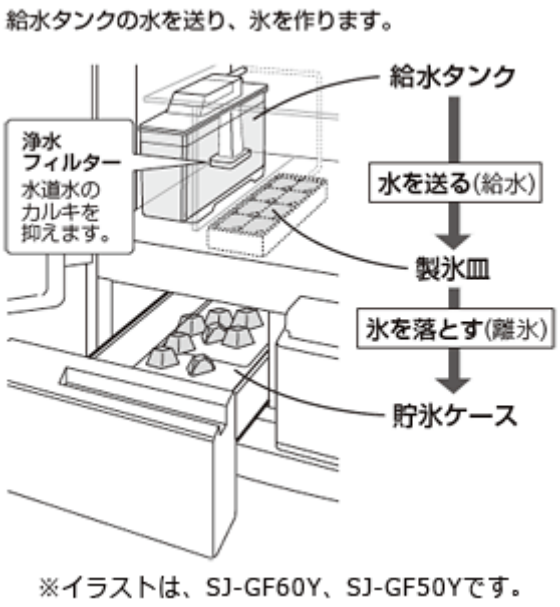

図 2.1： 自動製氷機の製氷システム(*1)

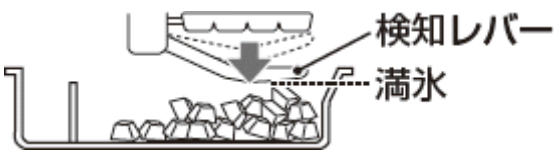

図 2.2： 検知レバーの仕組み(*1)

2.2 既存の IoT 冷蔵庫

既存技術として他に、2017 年 3 月 14 日にシャープが発表した人工知能を搭載した IoT 冷蔵庫「SJ-TF49C」などがある。(*2)

この IoT 冷蔵庫は人口知能により、使い続けることでクラウドがユーザーの生活リズムを学習して、必要な情報を音声によって対話することや無線 LAN を通じて情報を外に伝えることできる。

またスマートフォン用アプリの「COCORO KICHEN」を利用すれば、購入したものをアプリに入力すれば、その購入登録から冷蔵庫内にあると推測される食品を把握する。また食材の買い物の傾向などを学習し、定期購入食材の購入予測日から在庫が切れそうなことを伝え、食材が残っている場合にはその食材や冷蔵庫内の食品を使った献立をユーザーに伝える。

2.3 冷蔵庫に関する研究

冷蔵庫に関する研究が複数ある。まず1つ目は龍谷大学の研究で冷蔵庫内の食品の自動認識及び管理に関する研究があった。(*6)この研究では冷蔵庫内に web カメラを 2 台以上設置し、冷蔵庫が開いたときにつく庫内灯センサーを利用し、冷蔵庫内が明るい時に録画を行い、撮影した動画を解析し1つの食品が出し入れされる前後のフレームを抽出する。抽出した静止画像を背景差分により物体の検出を行い、予めテンプレートに登録してある静止画像とマッチングさせ物体が何であるかを認識させるという研究である。

次に京都大学の研究で食材管理のための荷重特徴を用いた食材固定という研究がある。(*7)この研究は食材をカメラで撮影された画像により認識し、荷重センサボードを開発し、それで重さを測ることでそれがどこに、どのくらいあるかを認識し、携帯端末を通して、ユーザーにその情報を提示する。

2.4 既存技術の問題点

既存技術にはいくつかの問題点がいくつか考えられる。

まず、2.1 の現存する冷蔵庫の自動製氷システムについて、氷を作る工程で余分に氷を作りすぎないようにするための検知レバーがあるが検知レバーは、レバーが下がる途中で氷に触れることで満氷であると認識する仕様になっているが、これでは満氷であるかどうかしかわからず、現在の氷の量を正確に測ることができない。また既存の給水タンクには現在の水量を測ることはできず、給水タンク内の水が十分な量がない時に利用者に知らせる術がない。そのため利用者が冷蔵庫内にある給水タンク内の水の量を直接確認していないと気付かないうちに氷がなくなっているという状況になる。また氷の量が少ないが氷の分布が検知レバー側に偏っていた場合、検知レバーは氷が十分にあると認識するが実際には氷の量自体は少ないということが起こり得る。

また 2.2 で記述したシャープの IoT 冷蔵庫はほかの冷蔵庫に比べ値段が高いためなかなか入手することが難しく、冷蔵庫外からアプリでデータを登録することにより冷蔵庫内の情報を冷蔵庫が認識し管理できるがこの冷蔵庫の製氷機に関する部分は製氷における氷の種類や大きさの調節、清掃などはできるが、氷の量を測り、氷の量を外部に伝えることができない。食材管理に関する研究はカメラを用いているが製氷機内は常に暗く、開閉時に点灯することがないのでカメラによる撮影は厳しい。

第 3 章　提案手法

第 2 章で述べた問題点を解決する策として、製氷機の製氷システムの改良法を検討した。

まず、貯氷ケース内の氷の量が検知レバーでしか測定できず、正確な氷の量が測れないという問題は超音波センサー使用し、超音波センサーから氷までの距離を測り、この超音波センサーを複数設置し、測定することで立体的に氷の量を測ることで正確な氷の量を測ることができ、正確な氷の量がわからない問題を解決できると考えた。

また、給水タンク内の水量も同様に超音波センサーで測定することで水の水位を正確に測ることができると考えた。

次に現在の氷や水の量をの情報を外部へ伝えることができない問題は、超音波センサーで取得した情報を自宅にある Wifi などを用いてインターネット上にアップロードすることで外部からでも、取得情報をスマートフォンなどを通じて確認することができる。また氷や水の量が少ない場合にはインターネットを通じて自動的に利用者に通知を送る機能を追加すれば良いと考えた。

さらに氷の量の変化を記録し、気象データから氷の使用量とその日の気温または天気などの関係性からどのような時に氷を使うのか学習させ、氷をよく使う時期や時間帯を把握いき、よく使われる時期には氷の量を早めに通知をすることで常に適切な氷の必要量を保ち、氷がきれるという状態の無くすことができると考えた。

3.1 超音波センサーによる氷の量及び水量の測定

氷の量を測定するために超音波センサーを貯氷ケースに設置し、氷を作るための水が給水タンク内にどのくらいあるか調べるために給水タンクにも超音波センサーを設置し、それぞれの数値を測定する装置を検討した。

センサーは上部から下部に向けて超音波が飛ぶように設置する。センサーは貯氷ケースには等間隔に 3 台以上設置することが好ましい。複数台超音波センサーを設置することで、貯氷ケース内の氷の分布を具体的に調べることで氷の偏りによって起こっていた間違った満氷認識を防ぐことができる。貯氷ケース内のセンサーはすべて同時に測定を行い、それぞれの距離を測定する。10 分毎に測定を行うように設定をする。そして、水が入っている給水タンク内にも超音波センサーを設置し水量を測れるようにする。給水タンク内の超音波センサーは貯氷ケース内の氷が一定の基準値よりも少ない時に氷を作ることができる状態かを調べるために水位を測定する。

3.2 測定データをインターネット上にアップロード

3.1 で測定した氷や水の量は Wifi などを通じて測定した日時とともにインターネット上に記録し、共有することで氷や水の量をどこからでも把握できるようにする。各データを前回のデータと比較することで氷の量の変化をわかりやすくする。

3.3 氷の使用量と気象データの関連から推測

インターネット経由でその日の気温や湿度、天候などの気象データを取得し認識させる。気象データから体感温度を算出し、その結果から閾値を設定する。体感温度が高ければ普段より氷を多めに使うと推測し、閾値を基本より高く設定する。

また氷の量が変化したタイミングの気象データや時間帯及びから氷が減った時の時期や過去の取得データと氷の使用量からその日に使用されそうな氷の量を推測することでデータがたまるにつれてより正確によく使う時間帯や時期を推測していくことが可能である。

3.4 氷の量が適切であるか判断

予め貯氷ケースと給水タンクそれぞれに基本の閾値を設定する。貯氷ケース内のすべての超音波センサーで測定した結果、1 つでも閾値を下回った際には現在貯氷ケース内に氷は少ないと判断させる。そして給水タンク内の超音波センサーを実行させる。また気象データから推測した結果によって閾値を変化させる。

給水タンクでの測定結果が設定していた閾値を下回った際には、3.2 の手法でユーザーに伝える。

3.5 3.1 から 3.4 における工程のフローチャート

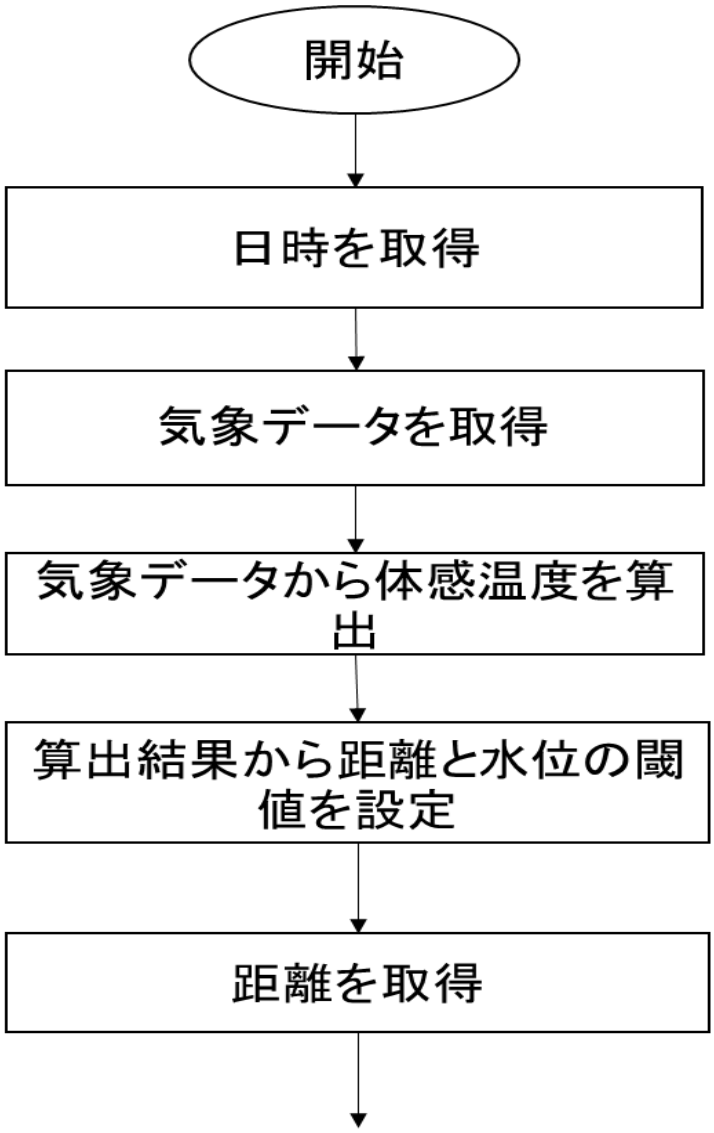

図 4.1： 製氷機自動管理化システムのフローチャート①

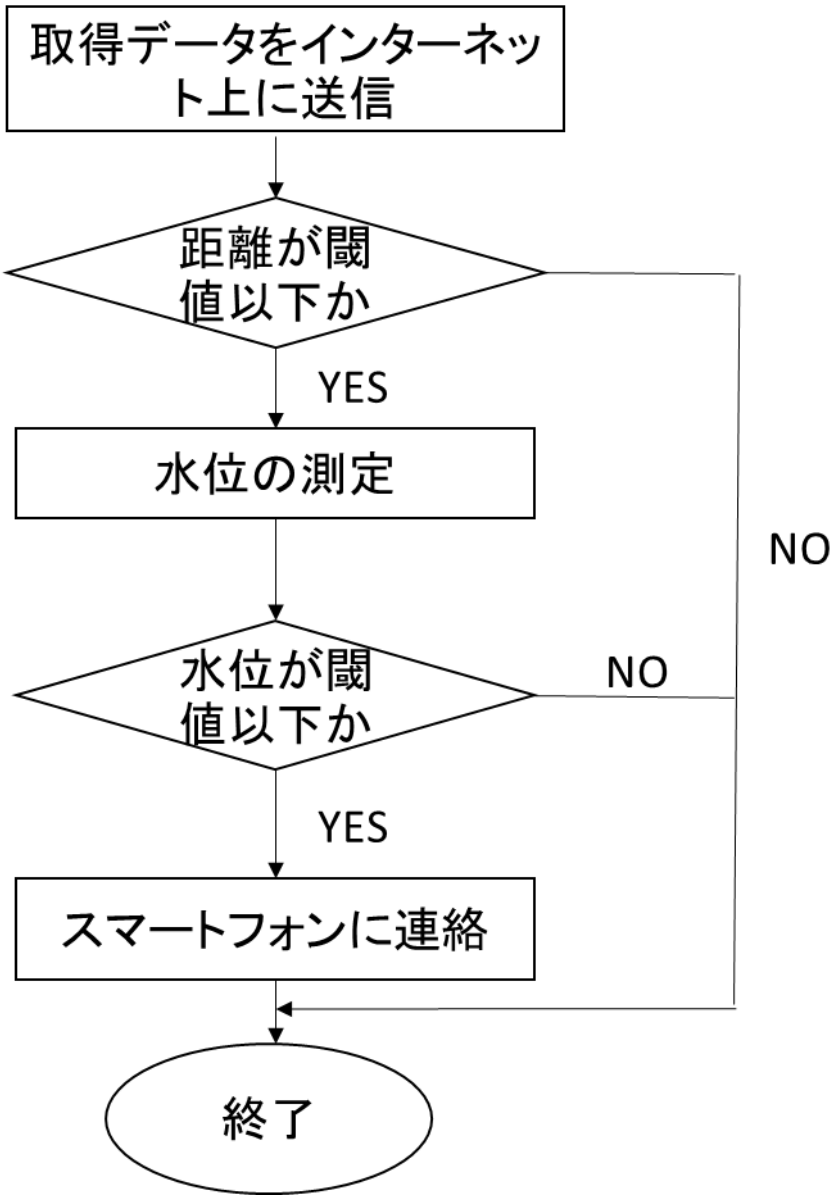

図 4.2： 製氷機自動管理化システムのフローチャート②

第 4 章　実験

4.1　実験概要

本実験は、第 3 章で述べたアルゴリズムから超音波センサーで氷の量を測定し、超音波センサーで測定した氷の量のデータをインターネット上に記録し、氷の量が閾値を超えた際にスマートフォンに連絡をするということが行うことができるのかを確かめるため第 3 章で述べたアルゴリズムを参考に作成したプログラムを動かし、以上のことができるのか調べるため次の実験を行った。

また、実験環境は Raspberry Pi3,Raspbian 9.3,Python 2,7,13 を使用する。超音波センサーは HC-SR04 を使用し、メールはGmailを用いる。測定データをインターネットにアップロードするために Google スプレッドシートを用いる。Google スプレッドシートとは、Google が提供するオンラインストレージサービスの一つで、複数の端末から Google ドライブ 上にあるスプレッドシートのファイルを共有するという事できる。ファイルを共有することができるのでオンライン環境であればどこにいてもファイルを操作することができるものである。

図 4.1　実験装置

4.2　実験 1

本実験で扱う超音波センサーHC-SR04 の精度を調べるため、超音波センサーから約 5cm 離れたところに壁を置き、壁との距離を 100 回測定し、HC-SR04 の推定精度を調べる。

4.3　実験 2

超音波センサーで測定した測定データを Google スプレッドシートにアップロードできるのか調べるために計 6 種類の測定を行う。

高さ 7.2cm，縦 21.8cm ,横 10.5cm 透明な容器を用意する。今回、氷では時間がたつにつれて溶けてしまい正確に測り続けることができないため、氷の代わりに小石を用意する。

次に実験方法として小石を入れた容器の両端から 3.5cmの場所と左端から 11cmの場所を中央地点とし、左からそれぞれ地点 A，B，C とし一つの超音波センサーをすべて高さ 10cmのところから容器に対して水平になるように地点Aから順次各地点1つずつ超音波センサーを移動させ測定する。それぞれの測定結果を容器の中が何も入っていない状態の時の測定結果から差を算出し、小石と超音波センサーまでとの距離を求め、その差を貯氷ケースの底から氷のある高さの数値として測定したデータを予め共有してある Google スプレッドシートに測定したデータに記入させる。記入させる場所は地点 ABC に合わせて Google スプレッドシートに記入する場所を変更させるため随時プログラムの修正を行い、それぞれ適切な場所に記入を行う。

様々な状況で測定したいため、随時小石の量を変化させながら、故意に小石の分布を偏らせたり、量を少なくさせ、パターン 1 からパターン 6 を対象に小石を配置し、測定を行う。

図 4.2　パターン 1 の小石の状態（満タン状態）

図 4.3　パターン 2 の小石の状態（全体的に少ない状態）

図 4.4　パターン 3 の小石の状態（左がやや少ない状態）

図 4.5　パターン 4 の小石の状態（満タンではないが量はある状態）

図 4.6　パターン 5 の小石の状態（左側が少ない状態）

図 4.7　パターン 6 の小石の状態（中央が少ない状態）

4.4　実験 3

グラフで実際の量を判断できるか調べるため、測定したデータを棒グラフで表し、各状況の実際の量の写真とグラフを比較し、小石の量や分布が判断できるのか確かめる。

測定結果をもとに手動で Google スプレッドシート上のデータをグラフ化し、現在の氷の分布を 3 つの地点から測定した結果をもとにグラフを用いて小石量を表示する。

4.5　実験 4

氷量が少ないと判断した際に Gmail にて自動送信ができるのか調べるため、各地点で閾値を3.0cmと設定し、3.0cm以下を測定するとGmailを用いて自動でメールを行わせる。

第 5 章　実験結果と考察

5.1　実験 1 の結果

4.2 の工程で 100 回測定した結果を以下に示す。

表 5.1 小石が一定の時の測定結果

回数(回)	距離(cm)						
1	4.97	26	4.96	51	4.98	76	5.01
2	4.97	27	4.97	52	4.98	77	4.98
3	5.01	28	4.98	53	4.98	78	4.98
4	5.02	29	5.01	54	4.99	79	4.99
5	4.97	30	4.98	55	4.99	80	4.97
6	4.96	31	4.97	56	4.99	81	5.01
7	4.99	32	4.99	57	4.99	82	4.97
8	4.96	33	4.99	58	4.99	83	4.96
9	4.94	34	4.99	59	4.96	84	4.98
10	4.97	35	4.97	60	4.99	85	5.01
11	4.97	36	4.99	61	4.97	86	4.99
12	4.96	37	4.96	62	4.99	87	4.98
13	4.99	38	4.98	63	5.01	88	4.98
14	4.99	39	4.96	64	4.99	89	4.97
15	4.96	40	4.98	65	4.98	90	4.96
16	4.96	41	4.98	66	4.98	91	4.99
17	4.99	42	4.99	67	4.99	92	4.98
18	4.98	43	4.99	68	5.01	93	4.98
19	4.99	44	4.97	69	4.97	94	4.99
20	4.97	45	5.01	70	4.99	95	4.96
21	4.96	46	4.99	71	4.97	96	4.99
22	4.97	47	5.01	72	4.99	97	4.97
23	4.98	48	4.99	73	4.99	98	4.99
24	4.99	49	4.98	74	4.99	99	4.99
25	4.99	50	4.96	75	4.97	100	4.97

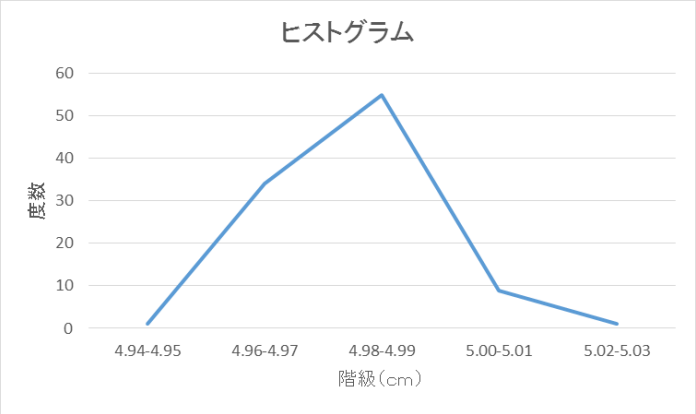

図 5.1　表 5.1 のヒストグラム

次に表 5.1 で測定した結果を信頼度 95%で求めた統計を示す。

表 5.2　表 5.1 の統計結果

平均(cm)	4.981515
標準偏差(cm)	0.014976
標準誤差(cm)	0.001498
t値	1.96
信頼区間(cm)	±0.002935
上限(cm)	4.98445
下限(cm)	4.97858

上記の結果により、超音波センサーHC-SR04 の標準誤差は 0.001498cmであり、精度がかなり高いことが分かった。

5.2 実験 2 の結果

小石の量が様々な状態での測定結果を以下に示す。

表 5.3　小石の量が様々な状態での測定結果

小石分布パターン	A(cm)	B(cm)	C(cm)
1	6.78	6.34	6.59
2	3.33	4.27	2.71
3	3.41	6.33	5.98
4	6.71	4.77	5.33
5	2.89	4.62	5.33
6	6.61	2.72	4.29

超音波センサーで測定したデータを Google スプレッドシートにアップロードすることが可能であった。また端末を経由して、外部から端末を通じてデータを確認することできた。

5.3 実験 3 の結果

表 5.2 で得たデータをグラフ化したものとそれぞれの測定時刻での実際の小石の量を次に示す。

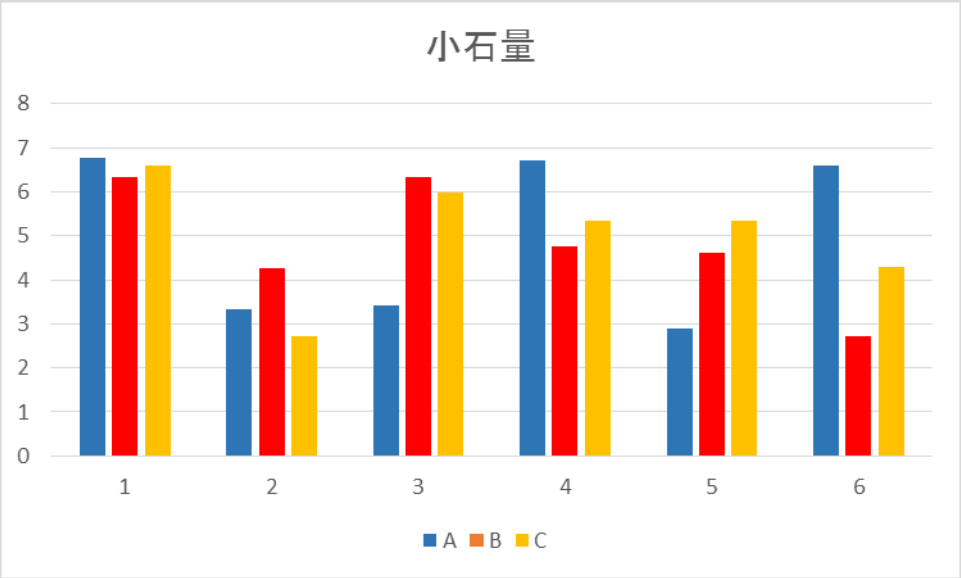

図 5.1　表 5.3 で測定した結果を視覚化したグラフ

グラフの縦棒の左から青が地点A、赤が地点B、黄色が地点Cでの測定結果である。

図 5.1のグラフと図 4.2 から図 4.7 までの実際の写真と比べるとグラフの形と実際の小石の状態がパターン 1,5,6 の小石の時とグラフと写真を比べると抽象的であるがグラフで見える状態と写真ものと一致する。しかし、パターン 2,3,4 の時は実際の量とは少し違う状態になった。

5.4 実験 4 の結果

次に測定結果が閾値を下回った際に受信したメールを以下に示す。

図 5.8　量が少ないと判断した際に送信されたメール

閾値3.0cmを下回った際に図 5.8 のように自動でメールを送信することができた。

5.5 考察

実験 1 の結果から超音波センサーHC-SR04 は精度が高いことが分かった。実験 2 の結果から超音波センサーにて測定結果を Google スプレッドシートにデータを送信できた。実験 4 では閾値を下回った際にメールを自動送信することができた。しかし実験 3 の結果である測定したデータをグラフ化したものと実験 2 のパターンの実際の量を比較した際に一部少し違うような形になった。このようなことになる原因として考えられるのは小石の形や表面が歪であるため、測定した地点がところどころ開いている空間や小石の斜面などを測定したためこのようなことが起こったと考えられる。また測定する地点が 3 点だけでは少なかったため正確な情報を測定できなかったことも考えられる。また超音波センサーの測定地点が石と石の隙間を測定したりすると実際よりを少ないことになったことなどが考えられる。超音波センサーで測定する地点を増やすことで小石の量をグラフで可視化した際に、3 点の時よりも具体的に小石の量を測定することができるだろう。

第 6 章　おわりに

本研究では、冷蔵構内の氷の量を自動で管理させるために、超音波センサーを用いて数分おきに、氷の量を測定し、そのデータを Google スプレッドシートに記入し、グラフを用いて可視化させ、設定していた閾値を下まわった際にメールを送信するというアルゴリズムの提案をした。そのため、超音波センサーで測定したデータを自動で Google スプレッドシートに記入し、閾値を下回った際に自動でメールする、また 3 つの地点から測定してデータをグラフ化して可視化できるのかを実験した。実験結果から自動で Google スプレッドシートに記入させることができ、メールも送信させることができたことにより第3章で提案したアルゴリズムが可能であることが期待できる。

本実験では超音波センサーは1つだけしか使っておらず、超音波センサーでの測定や測定データからグラフの作成、Google スプレッドシートへのデータ入力の場所の変更を手動で行っているため、今後は、複数のセンサーの同時に使用、測定データを自動でグラフ化させる機能、これらの工程を 10 分おきに行うようにループ作業化させることにより、完全に自動管理させることが、今後の課題である。

謝辞

今回の研究、論文作成を行うにあたり、ご指導頂きました三好力教授に深謝いたします。

また多忙の中、日頃の議論にご協力くださった同三好研究室の皆様や、学友の皆様に心より感謝いたします。

参考文献

(*1) SHARP　自動製氷機のいろいろな疑問
http://www.sharp.co.jp/support/refrigerator/doc/point_seihyo.html

(*2) SHARP　SJ-TF49C の特長
http://www.sharp.co.jp/reizo/feature/sjtf49c

(*3) Google スプレッドシート
https://gsuite.Google.com/intl/ja/products/sheets/?amp;&gclid=CLjZzIW1nt4CFcXPvAod3KQEhQ&gclsrc=ds

(*4)水耕栽培用に Raspberry Pi で水温、室温、湿度、水位を計測して Google スプレッドシートにアップしてみた
https://qiita.com/tama_qiita/items/d0eab2f2802517cc7279

(*5) Gmail で簡単にメール送信
http://make.bcde.jp/python/gmail%E3%81%A7%E7%B0%A1%E5%8D%98%E3%81%AB%E3%83%A1%E3%83%BC%E3%83%AB%E9%80%81%E4%BF%A1/

(*6) 平成 27 年度　特別研究報告書 「冷蔵庫内食品の自動認識及び管理の考察」
龍谷大学　理工学部　情報メディア学科　T120430　川村　紘菜

(*7)社団法人　電子情報通信学会　信学技報 「食材管理のための荷重特徴を用いた食材固定」 加茂田玲奈　上田真由美　舩冨卓哉　飯山将晃　美濃導彦

付録

実行プログラム

```
# -*- coding: utf-8 -*-
import sys
import datetime
import time
import RPi.GPIO as GPIO
import sqlite3
sys.path.append('/home/pi/python_sens/')
from spredsheet import SpreadSheet
from hcsr04 import hcsr04
from sqlite_connect import SqliteConnect
from gmail import gmail
def hcsr04_result(pPinTrig,pPinEcho,pHight,pTemp):
    instance = hcsr04(pPinTrig,pPinEcho)
    d = instance.result(pTemp)
    l = pHight - d
    return l
def sqlite_insert(t0,l):
    instance = SqliteConnect();
    instance.insert(t0,l)
def test(t0,l):
    print("日付："+str(t0))
    print("高さ：{:.4}".format( l)+"cm")
def spreadSheet_insert(t0,l):
    KEY_FILENAME ='/home/pi/python_sens/gs.json'
    SHEET_ID ='記入する Google スプレッドシートのシート
ID'
    APPEND_RANGE = 'Sheet1!A1:B1'
    APPEND_LENGTH=5
Sheet=SpreadSheet(KEY_FILENAME,SHEET_ID,APPEND_
RANGE,APPEND_LENGTH)
    sheet.append(["{0:%Y-%m-%d %H:%M:%S}".format(t0),l])
def main():
    Hight =#容器の高さ(cm)
    GPIO_TRIG = 17 #HC-SR04
    GPIO_ECHO = 27 #HC-SR04
    #t0:日時,l:距離
    t0 = datetime.datetime.now()
    l=0.0
    #距離計測
    l = hcsr04_result(GPIO_TRIG,GPIO_ECHO,Hight,t2)
    test(t0,l)
    #ローカル DB へ書き込み
    sqlite_insert(t0,l)
    #GoogleSpredSheet へ書き込み
    #spreadSheet_insert(t0,l)
    spreadSheet_insert(t0,l)
  if l<3.0
  gamil()
    #処理終了
    sys.exit()
if __name__ == '__main__':
   main()
```

超音波センサーのプログラム

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time
import math
class HCSR04(object):
  def __init__(self, gpio_trig,gpio_echo):
      self.trig = gpio_trig
      self.echo = gpio_echo
  def pulseIn(self,PIN, start=1, end=0):
      if start==0: end = 1
      t_start = 0
      t_end = 0
      # ECHO_PIN が HIGH である時間を計測
      while GPIO.input(PIN) == end:
          t_start = time.time()
      while GPIO.input(PIN) == start:
          t_end = time.time()
      return t_end - t_start
  def calc_distance(self,TRIG_PIN, ECHO_PIN, num,
v=34000):
      list = None
      for i in range(num):
          # TRIG ピンを 0.3[s]だけ LOW
          GPIO.output(TRIG_PIN, GPIO.LOW)
          time.sleep(0.3)
          # TRIG ピンを 0.00001[s]だけ出力(超音波発射)
          GPIO.output(TRIG_PIN, True)
          time.sleep(0.00001)
          GPIO.output(TRIG_PIN, False)
          # HIGH の時間計測
          t = self.pulseIn(ECHO_PIN)
          # 距離[cm] = 音速[cm/s] * 時間[s]/2
          d = v * t/2
          distance = math.floor(d *100) /100
          #print(distance, "cm")
          if distance > 0 and distance < 1000:
              break
      # ピン設定解除
      GPIO.cleanup()
      return distance
  def result(self,temperature):
          # 音速[cm/s]
          v = 33150 + 60*temperature
          # ピン番号を GPIO で指定
          GPIO.setwarnings(False)
          GPIO.setmode(GPIO.BCM)
          # TRIG_PIN を出力, ECHO_PIN を入力
          GPIO.setup(self.trig,GPIO.OUT)
          GPIO.setup(self.echo,GPIO.IN)
          GPIO.setwarnings(False)
          # 距離計測(TRIG ピン番号, ECHO_PIN 番号, 計測回
数, 音速[cm/s])
          d = self.calc_distance(self.trig, self.echo, 10, v)
          return d
if __name__ == '__main__':
      TRIG = 17
      ECHO = 27
      hcsr04 = HCSR04(TRIG,ECHO)
      print hcsr04.result(20)
```

Google スプレッドシートのプログラム

```
# -*-coding: utf-8 -*-
import httplib2
import numpy as np
from apiclient import discovery
from oauth2client.service_account import
ServiceAccountCredentials
SCOPES = 'https://www.googleapis.com/auth/spreadsheets'
APPEND_RANGE = 'Sheet1!A1:B1'
class SpreadSheet(object):
  def __init__(self, sheet_id):
    self.sheetId = sheet_id
    self.append_length = length
credentials=ServiceAccountCredentials.from_json_keyfile_na
me(gs.json,scopes=SCOPES)
    http_auth = credentials.authorize(httplib2.Http())
discoveryUrl=('https://sheets.googleapis.com/$discovery/rest?"
version=v4')
    self.service = discovery.build('sheets', 'v4', http=http_auth,
discoveryServiceUrl=discoveryUrl)
  def append(self, values):
    assert np.array(values).shape==(self.append_length,) ,
"The shape of value %s must be 3" %
(np.array(values).shape,self.append_length)
    value_range_body = {'values':[values]}
result=self.service.spreadsheets().values().append(spreadshee
tId=self.sheetId, range=APPEND_RANGE,
valueInputOption='USER_ENTERED',
body=value_range_body).execute()
    #print(result)
if __name__ == '__main__':
  sheet = SpreadSheet("Google スプレッドシートのシート ID")
  sheet.append(["test", "test", 3,])
```

```
#service.spreadsheets().values().append()
 #spreadsheetId='シートの ID'
 #range='データを追加する場所 例）Data1!A1:C1 は Data1 とう名前のシートの A 列から C 列'
 #valueInputOption='データ入力オプション　RAW：そのまま入力、USER_ENTERED：ユーザ指定の書式に従う？'
 #body='入力するデータ'
#).execute()
```

Gmail のプログラム

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import os.path
import datetime
import smtplib
from email import Encoders
from email.Utils import formatdate
from email.MIMEBase import MIMEBase
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText
#Gmail アカウント
ADDRESS = "自身のアカウント"
PASSWARD = "自身のパスワード"
#SMTP サーバの設定(Gmail 用)
SMTP = "smtp.gmail.com"
PORT = 587
def　create_message(from_addr,　to_addr,　subject,　body, mime=None, attach_file=None):
    """
    ＃メッセージを作成する
    @:param from_addr 差出人
    @:param to_addr 宛先
    @:param subject 件名
    @:param body 本文
    @:param mime MIME
    @:param attach_file 添付ファイル
    @:return メッセージ
    """
    msg = MIMEMultipart()
    msg["From"] = from_addr
    msg["To"] = to_addr
    msg["Date"] = formatdate()
    msg["Subject"] = subject
    body = MIMEText(body)
    msg.attach(body)
    # 添付ファイル
    if mime != None and attach_file != None:
        attachment=MIMEBase(mime['type'],mime['subtype'])
          file = open(attach_file['path'])
          attachment.set_payload(file.read())
          file.close()
          Encoders.encode_base64(attachment)
          msg.attach(attachment)
          attachment.add_header("Content-Disposition","attachment", filename=attach_file['name'])
    return msg
def send(from_addr, to_addrs, msg):
    """
    ＃メールを送信する
    @:param from_addr 差出人
    @:param to_addr 宛先(list)
    @:param msg メッセージ
    """
    smtpobj = smtplib.SMTP(SMTP, PORT)
    smtpobj.ehlo()
    smtpobj.starttls()
    smtpobj.ehlo()
    smtpobj.login(ADDRESS, PASSWARD)
    smtpobj.sendmail(from_addr, to_addrs, msg.as_string())
    smtpobj.close()
if __name__ == '__main__':
    #宛先アドレス
    to_addr = "dreamer10969@gmail.com"
    #件名と本文
    subject = "氷量"
    body = "氷の残量が少ないです。"
    #添付ファイル設定(text.txt ファイルを添付)
    mime={'type':'text', 'subtype':'comma-separated-values'}
    attach_file={'name':'test.txt', 'path':'./text.txt'}
    #メッセージの作成(添付ファイルあり)
    #msg = create_message(ADDRESS, to_addr, subject, body, mime, attach_file)
    #メッセージ作成(添付ファイルなし)
    msg = create_message(ADDRESS, to_addr, subject, body)
    #送信
    send(ADDRESS, [to_addr], msg)
```