令和元年度　特別研究報告書

電子レンジ自然発火防止システムの検討

龍谷大学　理工学部　情報メディア学科

T160426　八木 直樹

指導教員　三好 力　教授

内容概要

核家族化や生涯未婚率の増加などにより単身世帯数が増加している。電化製品も従来の大型の電化製品と違い、小型で安価な単身世帯に向けた電化製品が増えている。単身世帯に向けた電化製品は価格を抑えるため必要最低限の機能しか搭載しておらず思わぬ事故につながることがある。電子レンジを例に挙げると様々な機能を搭載している高機能電子レンジでは複数のセンサによって食材の過加熱を防ぐ安全装置があるが単機能電子レンジではそのような機能を搭載していない。本研究では単機能電子レンジに向けた安価で搭載可能な食材の自然発火防止システムを検討する。

第1章 はじめに

1.1 研究背景

近年、核家族化の影響や若年層が地方から都心の方へ流出または生涯未婚率の増加などの要因により単身世帯が増えている。(図 1.1.1)単身世帯が増えたことにより単身世帯向けの家庭用電化製品が数多く売られている。総務省統計局の調べによると 30 歳未満の単身世帯の電子レンジの普及率は男性71．6%、女性89．7%と男女ともに洗濯機の普及率を上回っていることがわかった[1]。近年では冷凍食品のバリエーションが増えており日本冷凍食品協会の調べによると国内消費量が増加していることも電子レンジ普及率が高い要因だと考えられる。(図 1.1.2)また、電子レンジを使用した一人暮らし向けの時短レシピなどが料理本やレシピ掲載サイトなどで多く見受けられる。

今日の電子レンジは食材を温める機能だけでなく庫内にスチームを噴射し加熱するスチーム料理や100℃以上の水蒸気を発生させて表面を焼く過熱水蒸気調理など様々な機能が搭載され従来の電子レンジではできなかった料理もできる電子レンジが開発されている。

しかし、東京都消防庁の調べによると電子レンジによる誤った使用方法による火災事故は増加しているとわかった。(図 1.1.3、1.1.4)[2][3]電子レンジの出火原因としては必要以上の過熱が最も多かった。電子レンジの機能はマイクロ波と呼ばれる電磁波の振動により食材に含まれる水分子を振動させて、水分子を振動させたときに発生する熱によって加熱が行われている[4]。しかしマイクロ波は食材の水だけでなく、マイクロ波を吸収する物質ならば何でも加熱される。それにより加熱をしすぎてしまい食材から水分が失われてしまう。水分が失われた食材の温度は上昇してしまう。可燃性の物質には発火点があり発火点を超えてしまうことにより食材が自然発火を起こす。

本研究では単身世帯に普及している単機能電子レンジにサーモグラフィカメラを用いた自然発火防止システムを検討する[5]。また、本研究では単機能電子レンジに向けたサーモグラフィカメラを用いた自然発火防止システムを検討している、そのためコストが高いシステムでは普及しづらいと考え、高性能のサーモグラフィカメラでなく安価に手に入ることができる温度測定範囲が狭いサーモグラフィカメラで実現を図る。

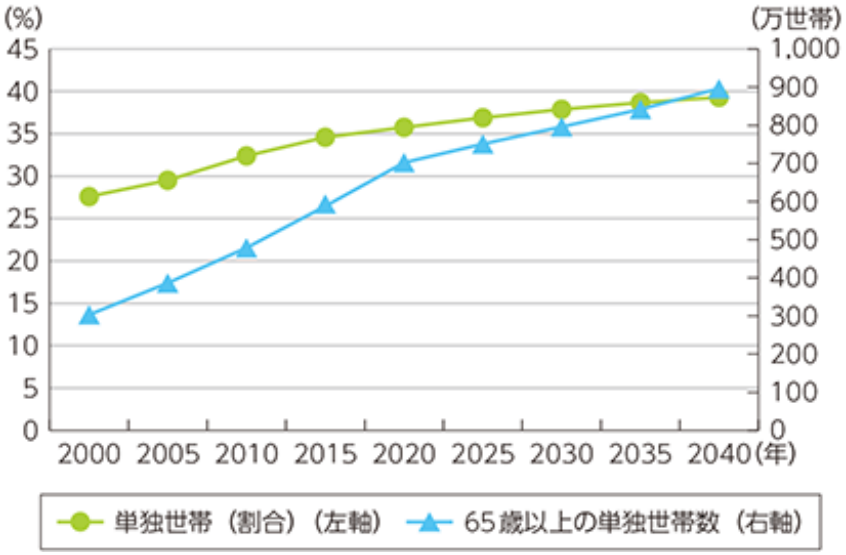


図 1.1.1 日本の世帯数の将来推計（全国推計）

図 1.1.2 冷凍食品国内消費量推移

図 1.1.3 電子レンジ火災件数

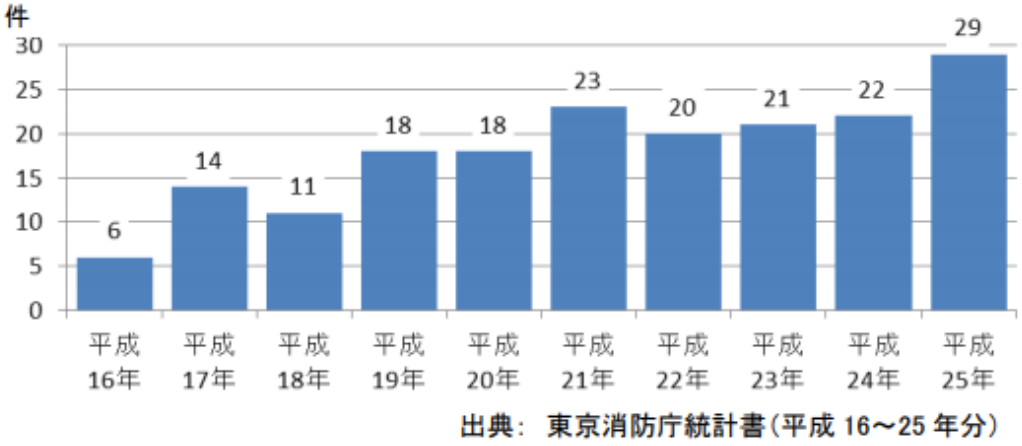


図 1.1.4 電子レンジ火災

第2章　提案手法

本研究では単機能電子レンジにサーモグラフィカメラを用いた電子レンジ庫内の自然発火防止システムを検討する。

本研究では比較的安価な「Conta™ サーモグラフィ AMG8833 搭載」のサーモグラフィカメラを用いた自然発火防止システムの実現を目的としている。そのため検出温度範囲が約 0～80℃までしか測定ができず、一般的な食材の自然発火を起こす危険がある約300℃の温度を測定することができない。そこで、検出温度範囲が約 0～80℃であるサーモグラフィカメラでも自然発火を起こす危険がある温度を検知するため電子レンジが食材を加熱する仕組みに注目した。電子レンジは食材の水分子を激しく動かすことにより食材を加熱しているそのため水分子を持たない食器などは過熱することができず、食材の温度が伝わることにより温度が上昇する[6]。食器の温度を測定することで検出温度範囲が約 0～80℃のサーモグラフィカメラを使用しても食材が 300℃の温度が検知できると考えた。

第3章　関連技術

3.1　電子レンジ

電子レンジにはマイクロ波を発生させる装置マグネトロンが備わっている。マグネトロンは陽極が円筒形で，その中心軸に線状の陰極が設けられて，陰極は高電圧をかけられると陽極に向かって熱電子を放出する。この空間に中心軸方向の磁場をかけると，粒子である熱電 子にはローレンツ力という力が作用するため熱電子の軌道が曲げられてしまう。この磁界を強くして熱電子の軌道を曲げていくと，熱電子が陽極に到達できずに陰極に戻るようになり，円筒内で振動しながらグルグルと周回するようになる。この熱電子の振動と電界の周期的変化のタイミングをうまく合わせることで，出力アンテナからマイクロ波を発生させることができる。マイクロ波を食品に当てると，食品に含まれている水分子を振動させることができ、振動した水分子同士がぶつかり合うと，そこに摩擦熱が発生して食品の温度が上がる。ドアガラスには電子レンジから発生するマイクロ波を遮断するための網目状の金属が張り巡らされている。

図 3.1.1 電子レンジ

3.2　サーモグラフィカメラ

赤外線サーモグラフィは対象物から放射される赤外線エネルギーを、赤外線カメラのレンズで結像することで、対象の温度を可視化している。赤外線カメラにはマイクロボロメータと呼ばれる赤外線エネルギーを電気信号に変換する仕組みがあり、物質から放出された赤外線エネルギーを測定することで物質の温度を測定することができる。

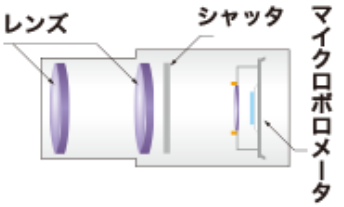


図 3.2.1 サーモカメラ原理

3.3 AMG8833

Panasonic 社の 8x8 の赤外線アレイセンサ「AMG8833」を搭載したセンサモジュール、各素子の温度測定範囲は約 0℃～80℃。測定エリアはセンサ正面（上下左右約 60 度）の四角錘で、このエリアを 8x8 ピクセルに分割した 2 次元画像が得られる。

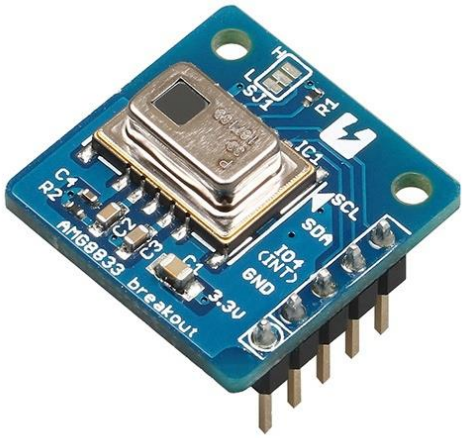


図 3.3.1 AMG8833

3.4 MLX80640

Raspberry Pi や Arduino で使用できる、高品質サーマルカメラ ブレークアウト、MLX90640 カメラには、約-40℃～+300℃の温度を約 1℃の精度、最大 64FPS で検出できる 768（32 x 24）IR ピクセルの配列がある。

図 3.4.1MLX80640

3.5 はんだごて

はんだに熱を加えて金属同士を合金接合する「はんだ付け」を行うための道具である。主に電子工作などに用いられる。ヒーターの構造から「ニクロムヒータータイプ」「セラミックヒータータイプ」の 2 種類のものがあり「ニクロムヒータータイプ」はこて先にニクロム線を巻いて外側から加熱する安価であるため学習キットや家庭用として使用される。「セラミックヒータータイプ」はタングステンで作ったヒーターをセラミックで包んで、こて先を内側から加熱する。絶縁性に優れ、IC などのデリケートな電子部品のはんだ付けに適している。

本研究では白光株式会社の「FX600」を使用する。「FX600」はこて先の温度が約 200℃～400℃に変化させることができる「セラミックヒータータイプ」のはんだごてである。

3.6 Raspberry Pi

Raspberry Pi は英国の「Raspberry Pi Foundation」が提供する小型のコンピューターである。USB や HDMI などの端末が搭載されており周辺機器を用意せずともディスプレイやキーボードが使える。GPIO（General Purpose Input / Output）というインターフェイスを使うことで Raspberry Pi で電球やモーターの制御、温度や明るさなどの情報を取得といった電子工作が手軽に行うことができる。

図 3.6.1 Raspberry Pi 3 model B

第4章　実験

4.1　実験目的

単機能電子レンジに向けた自然発火防止システムの提案のため低価格のサーモグラフィカメラと高機能のサーモグラフィカメラで比較して低価格のサーモグラフィカメラでの自然発火防止システムが実現可能か調べる。

4.2　実験方法

本研究では Raspberry Pi 3 model B を使用し、サーモグラフィカメラを作成する2種類の AMG8833 を搭載したセンサモジュールと MLX90640 を搭載したセンサモジュールをそれぞれの検出を調べることにより低価格の AMG8833 のセンサモジュールでも庫内の食材の自然発火を検知できるのか調べる。

4.2.1　Raspberry Pi 3 初期設定

Raspberry Pi 3 を使用するため Raspbian をダウンロードする[7]。microSD カードのフォーマットし microSD カードにイメージを書き込み Raspberry Pi に microSD カードを入れる[9][10]。

4.2.2　Raspberry Pi 3 配線

サーモグラフィカメラ AMG8833 と MLX90640 の GPIO 端子配線を以下に示す[11]。使用したサンプルコードは付録に記載する。

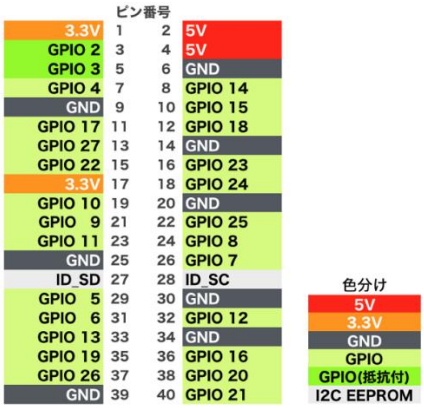


図 4.2.2.1Raspberry Pi ピン番号

AMG8833	ピン番号	MLX90640	ピン番号
SCL	5	3-6V	1
SDA	3	SDA	3
GND	6	SCL	5
3.3V	1	GND	6

図 4.2.2.2 GPIO 端子とサーモグラフィカメラの配線

4.2.3　実験

AMG8833 は検出温度範囲が約 0～80℃のセンサモジュールであるため直接電子レンジ庫内の食材温度を検出するのではなく食材の過熱により食器に伝わる熱により自然発火を検知する。

食器の材質による食材の過熱により食器に伝わる熱に違いがないのかを調べるため使用する食器は電子レンジ使用可能な耐熱性の食器を使用する。磁器、ストーンウェア、プラスチックの 3 種類の材質の食器で実験する。(図 4.2.3.3 図 4.2.3.4 図 4.2.3.5)家庭で使用されることが多いガラス製の食器は急激な温度変化に弱く破損の原因になり電子レンジ利用に不向きのため実験に加えないこととする。

まず初めに電子レンジのドアガラスからサーモグラフィカメラを使用して温度を測定しようと実験を試みたが電子レンジはマイクロ波を遮断するために網状の金属がドアガラスに張り巡らされておりサーモグラフィカメラが使用する赤外線も遮断されてしまい測定ができなかった。そこで、本研究では食材の代わりにはんだごてを使用し食材に温度を伝える。はんだごては白光株式会社の「FX600」を使用する。「FX600」のはんだごてはこて先の温度を約 200℃～500℃まで温度調節が可能である。

図 4.2.3.3 磁器製食器

図 4.2.3.4 ストーンウェア製食器

図 4.2.3.5 プラスチック製食器

図 4.2.3.6FX600

4.3　実験結果

プラスチック製食器ははんだごてで過熱中に破損してしまったため温度変化の確認ができなかった。

図 4.2.3.1 破損したプラスチック製食器

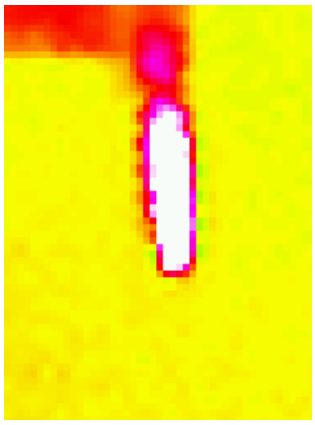


図 4.2.3.2 破損時の測定画像

4.3.1　MLX90640

まず高性能のサーモグラフィカメラ「MLX90640」を搭載したものを使用し、磁器製(白色)、ストーンウェア製(黒色)の 2 種類の食器をはんだごてを使用して 200℃と 350℃の温度で加熱し温度変化を確認した結果を以下に示す。

食器の素材による違いを比べる。図 4.3.1.5 と図 4.3.1.15、図 4.3.1.10 と図 4.3.1.20 の画像を比べたところ食器の素材の変化で食器に伝わる温度に大きな変化がないことが分かった。

次に温度の違いを比べる。はんだごてが約 200℃の場合の図 4.3.1.1 と図 4.3.1.5 よりはんだごてに近い箇所では約 100℃近い温度に変化しており、はんだごてが約 350℃の場合の図 4.3.1.6 と図 4.3.1.10 よりはんだごてに近い箇所では約 200℃～300℃の温度に変化している箇所が増えていることが分かった。

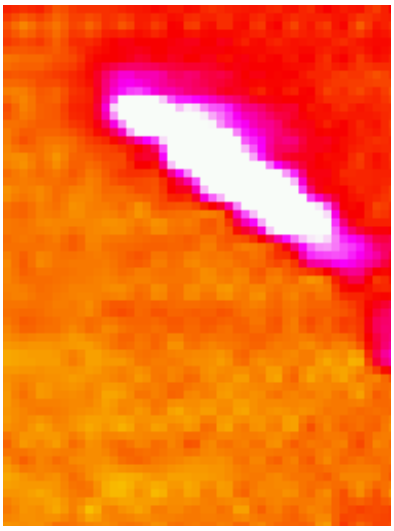


図 4.3.1.1 磁器 200℃(2 分)

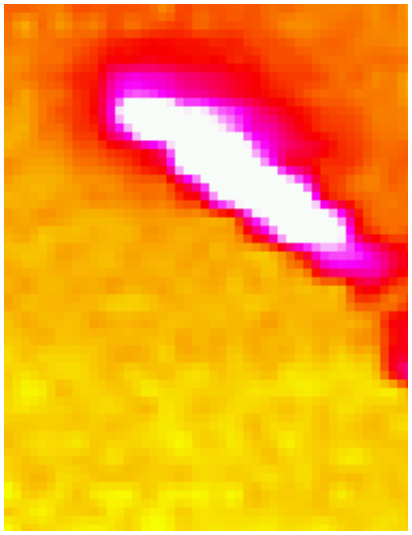


図 4.3.1.2 磁器 200℃(4 分)

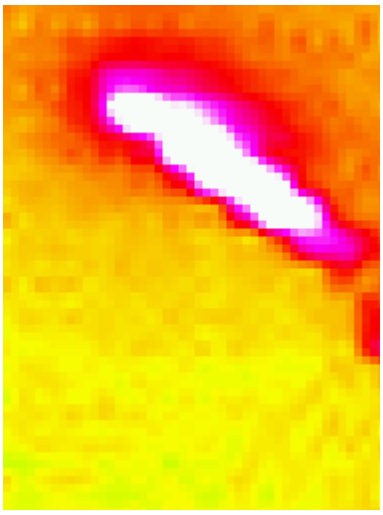


図 4.3.1.3 磁器 200℃(6 分)

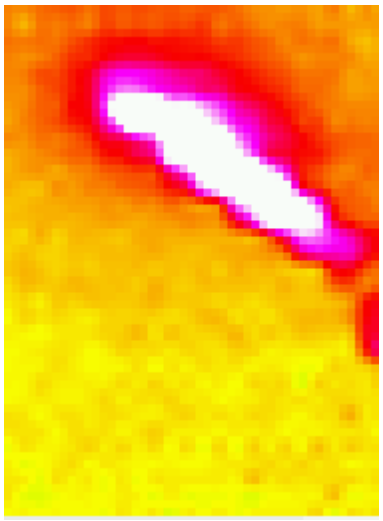


図 4.3.1.4 磁器 200℃(8 分)

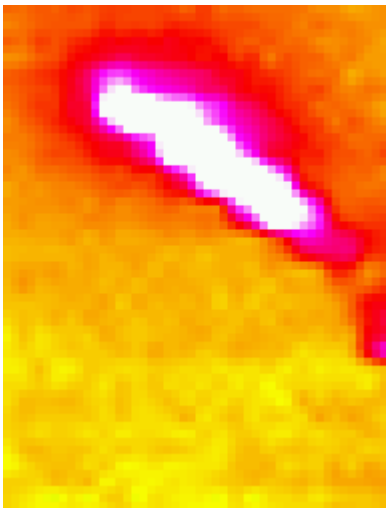


図 4.3.1.5 磁器 200℃(10 分)

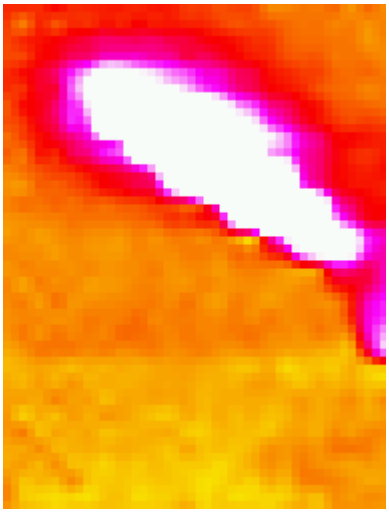


図 4.3.1.6 磁器 350℃(2 分)

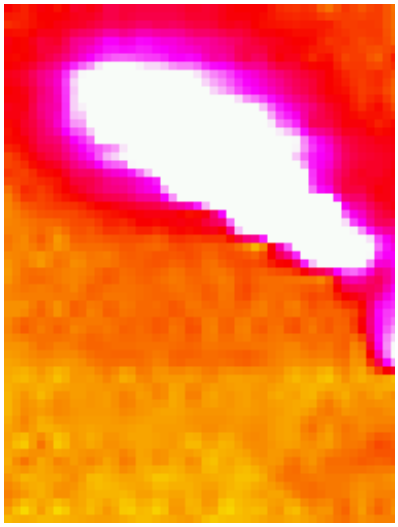


図 4.3.1.7 磁器 350℃(4 分)

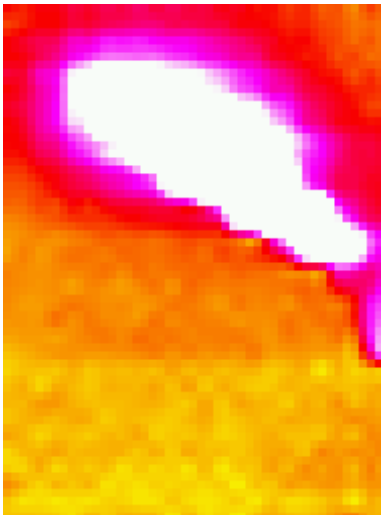


図 4.3.1.8 磁器 350℃(6 分)

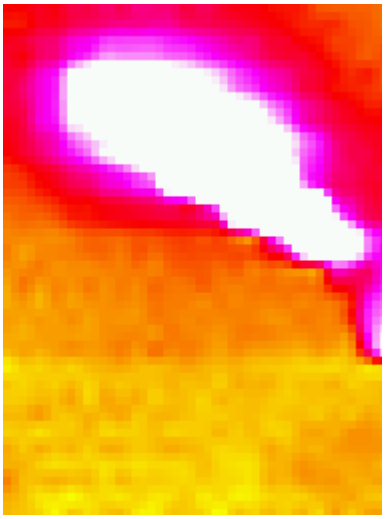


図 4.3.1.9 磁器 350℃(8 分)

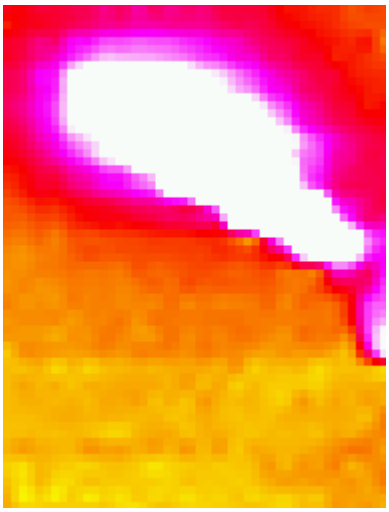


図 4.3.1.10 磁器 350℃(10 分)

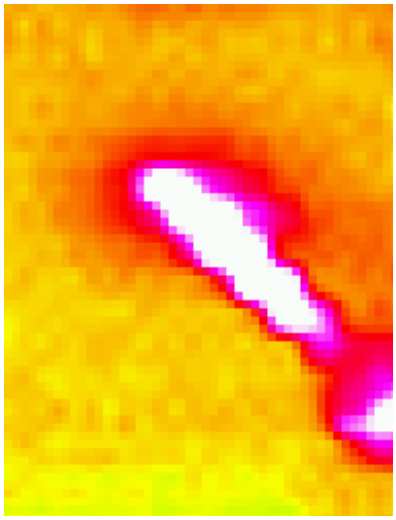


図 4.3.1.11 ストーンウェア製 200℃(2 分)

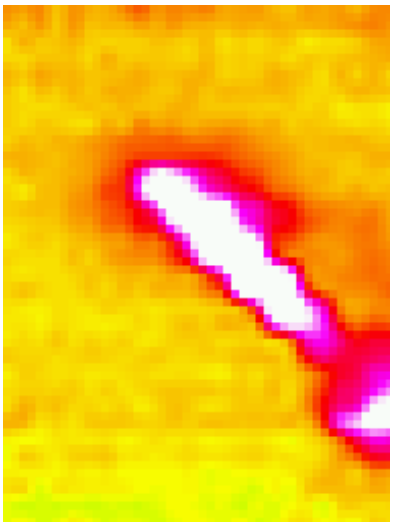


図 4.3.1.12 ストーンウェア製 200℃(4 分)

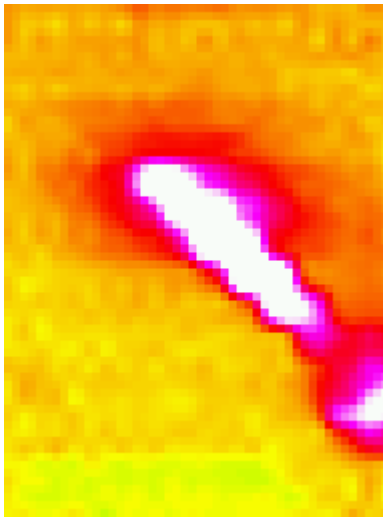


図 4.3.1.13 ストーンウェア製 200℃(6 分)

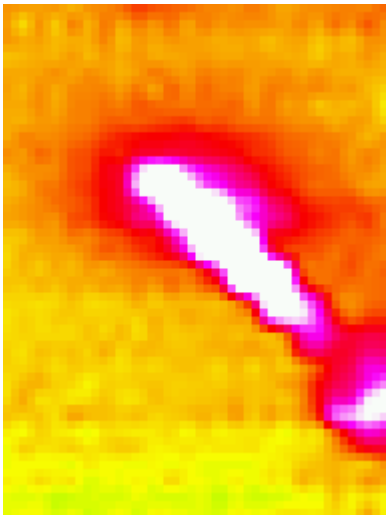


図 4.3.1.14 ストーンウェア製 200℃(8 分)

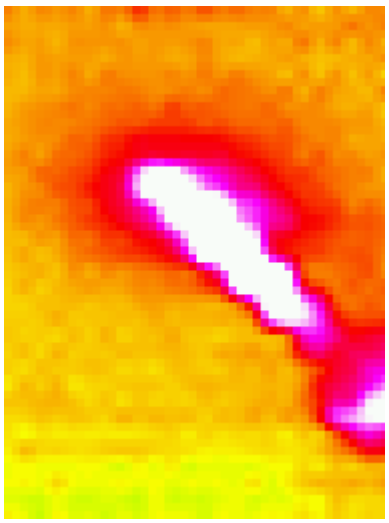


図 4.3.1.15 ストーンウェア製 200℃(10 分)

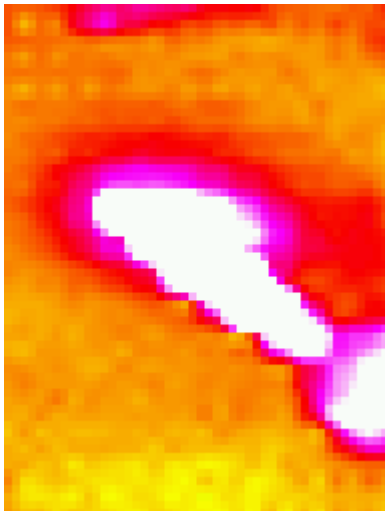


図 4.3.1.16 ストーンウェア製 350℃(2 分)

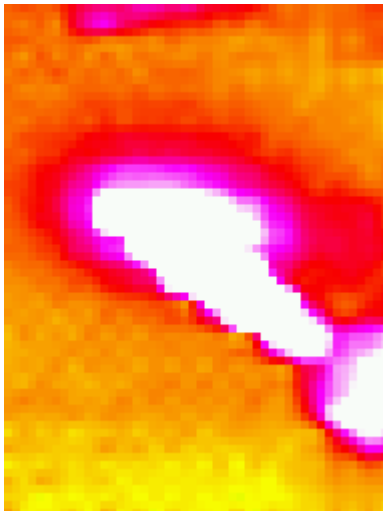


図 4.3.1.17 ストーンウェア製 350℃(4 分)

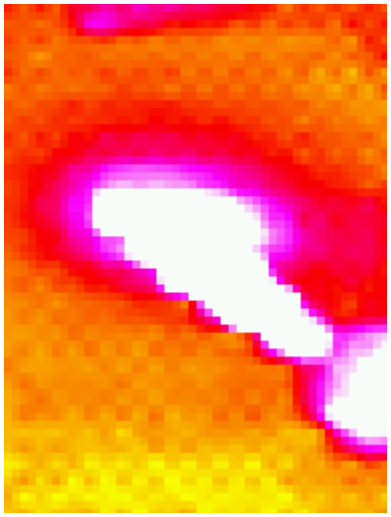


図 4.3.1.18 ストーンウェア製 350℃(6 分)

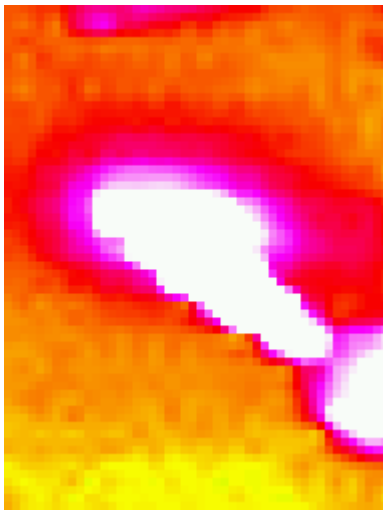


図 4.3.1.19 ストーンウェア製 350℃(8 分)

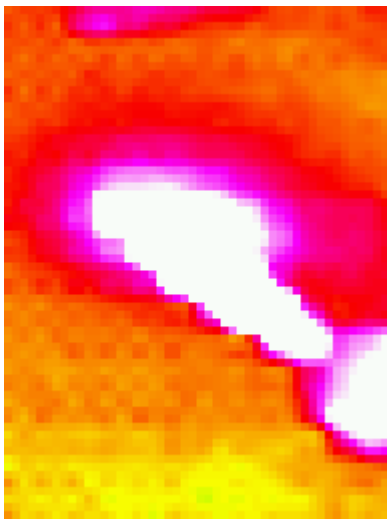


図 4.3.1.20 ストーンウェア製 350℃(10 分)

4.3.2 AMG8833

次に温度測定範囲が約 0℃～80℃のサーモグラフィカメラ「AMG8833」を使用し、磁器製、ストーンウェア製の 2 種類の食器をはんだごてを使用し温め、温度変化を確認した結果を以下に示す。

サーモグラフィカメラ「AMG8833」を使用し磁器製、ストーンウェア製の温度変化を比べたところ図 4.3.2.5 と図 4.3.2.15、図 4.3.2.10 と図 4.3.2.20 より「MLX90640」と同様に食器の素材で食器に伝わる温度に大きな変化がないことが分かった。

図 4.3.2.1 と図 4.3.2.5 より 200℃の場合で時間経過による温度変化に大きな違いが現れず、図 4.3.2.6 と図 4.3.2.10 より 350℃の場合、実験開始 2 分後の画像でははんだごての箇所が測定範囲の最大値である赤色に表示されていたが時間が経過するごとにはんだごて周辺部分にわたって赤色に表示されていた。

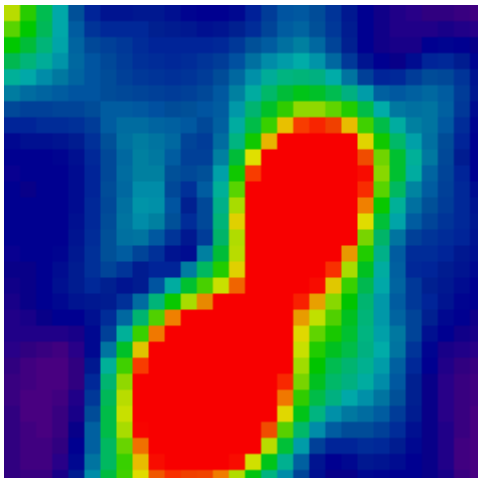


図 4.3.2.1 磁器 200℃(2 分)

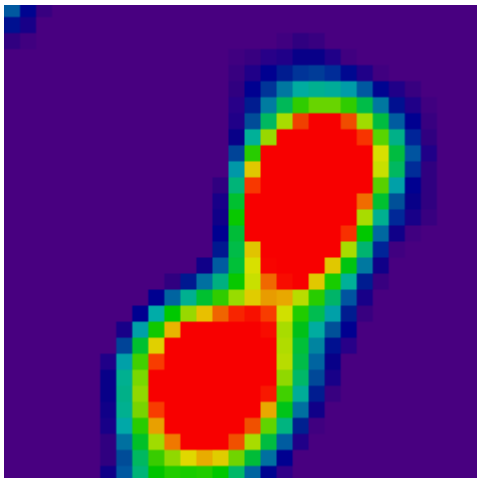


図 4.3.2.2 磁器 200℃(4 分)

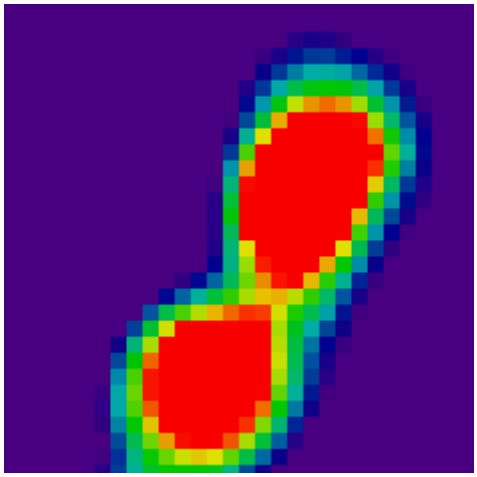


図 4.3.2.3 磁器 200℃(6 分)

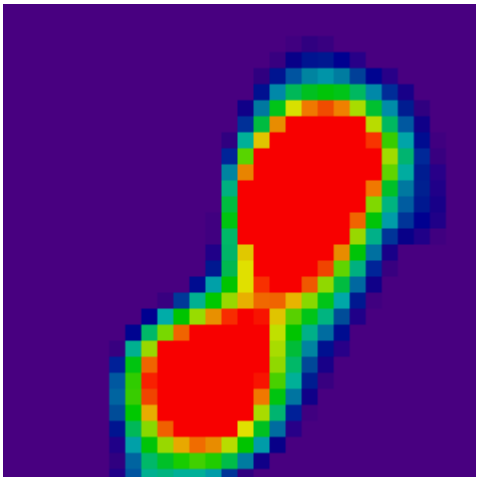


図 4.3.2.4 磁器 200℃(8 分)

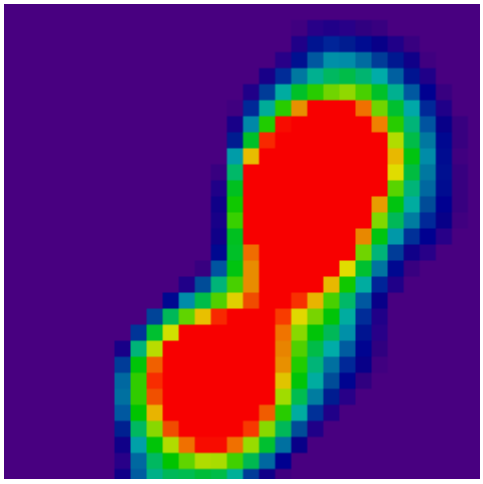


図 4.3.2.5 磁器 200℃(10 分)

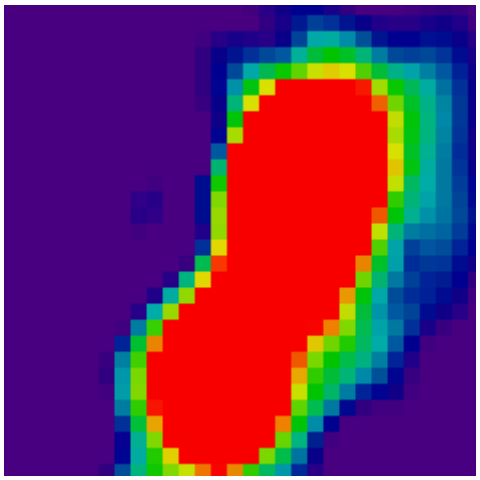


図 4.3.2.6 磁器 350℃(2 分)

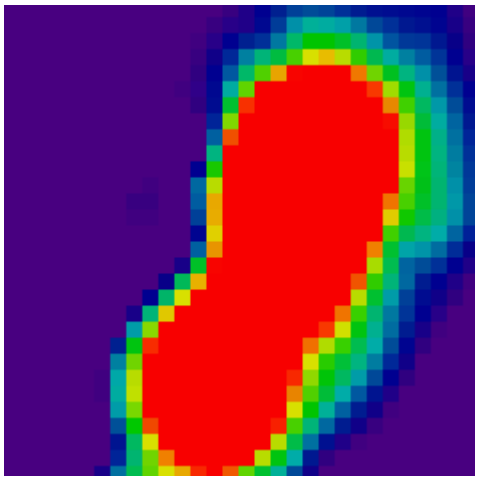


図 4.3.2.7 磁器 350℃(4 分)

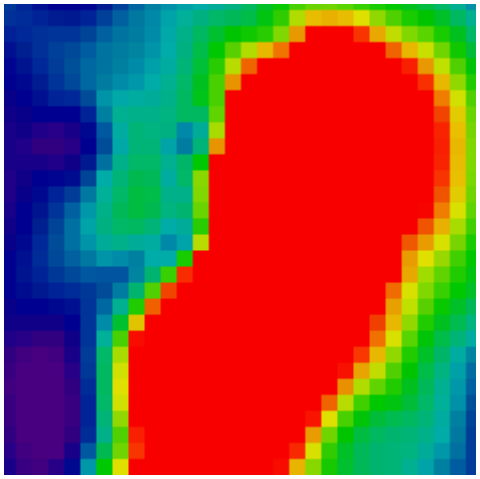


図 4.3.2.8 磁器 350℃(6 分)

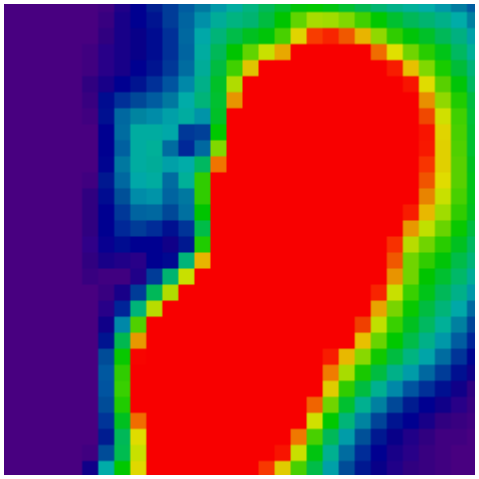


図 4.3.2.9 磁器 350℃(8 分)

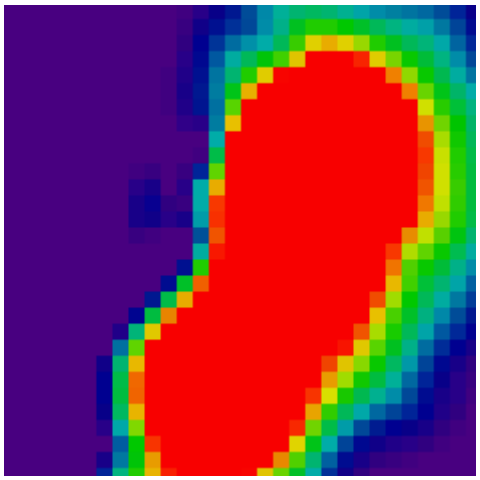


図 4.3.2.10 磁器 350℃(10 分)

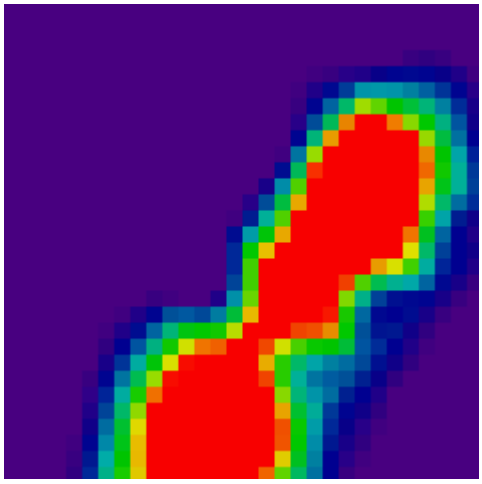


図 4.3.2.11 ストーンウェア製 200℃(2 分)

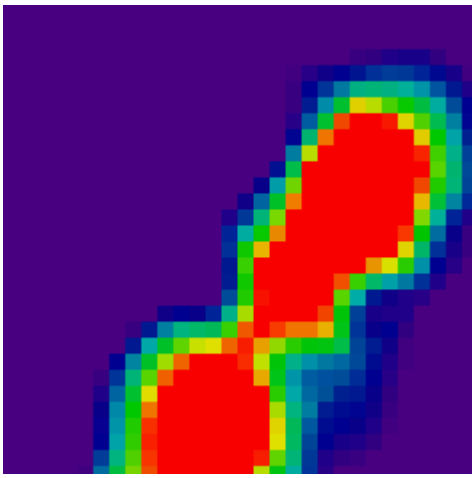


図 4.3.2.12 ストーンウェア製 200℃(4 分)

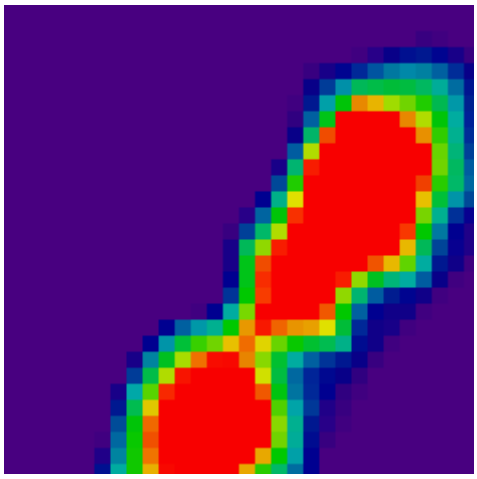


図 4.3.2.13 ストーンウェア製 200℃(6 分)

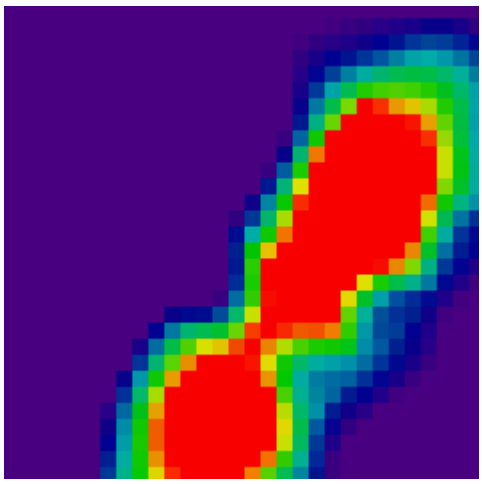


図 4.3.2.14 ストーンウェア製 200℃(8 分)

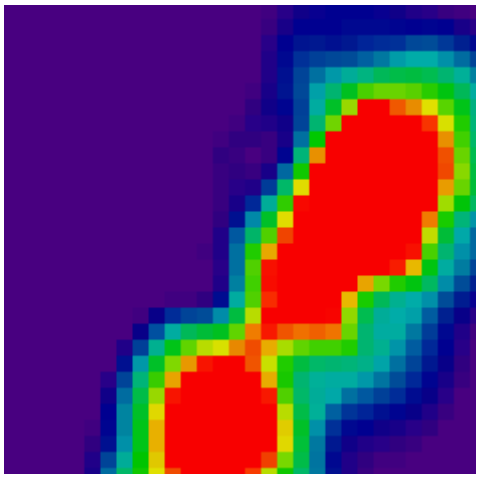


図 4.3.2.15 ストーンウェア製 200℃(10 分)

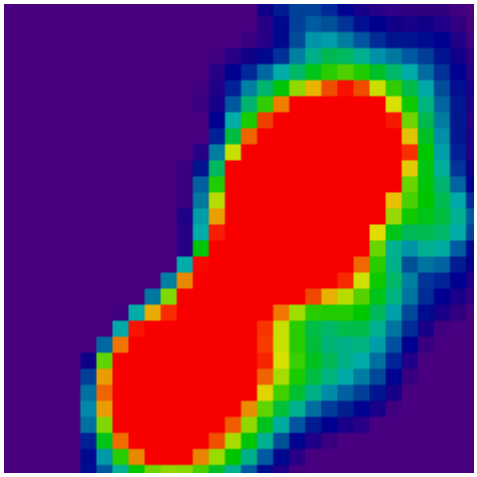


図 4.3.2.16 ストーンウェア製 350℃(2 分)

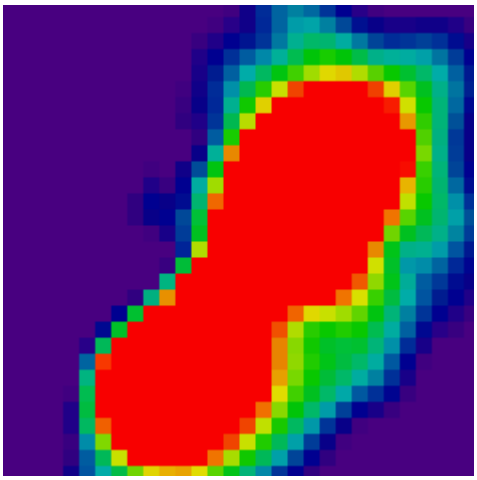


図 4.3.2.17 ストーンウェア製 350℃(4 分)

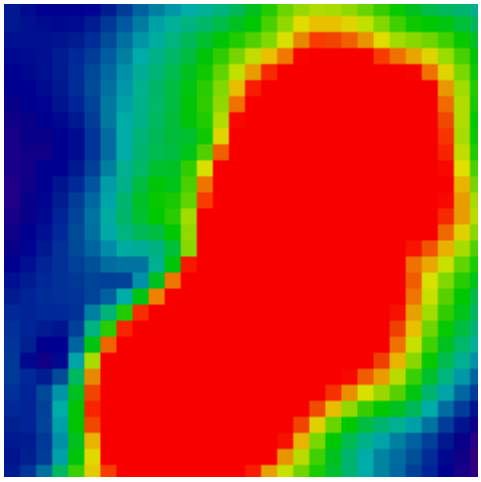


図 4.3.2.18 ストーンウェア製 350℃(6 分)

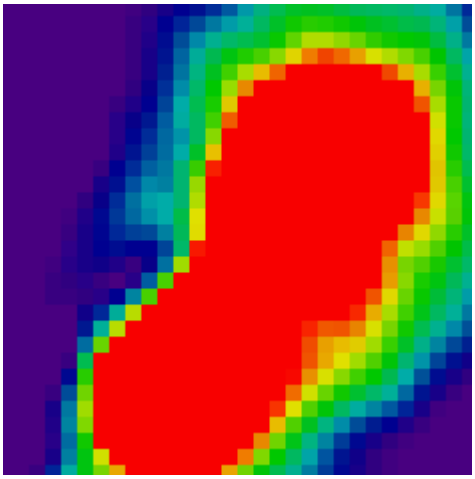


図 4.3.2.19 ストーンウェア製 350℃(8 分)

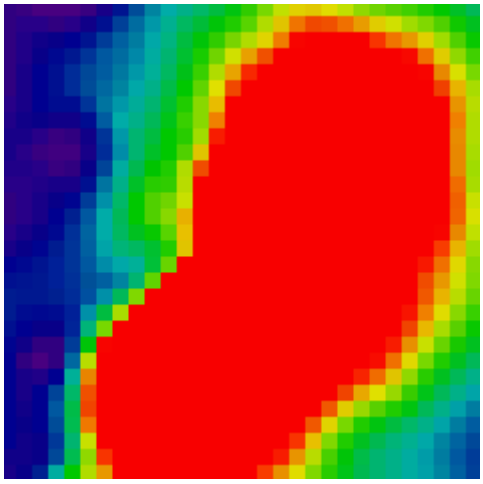


図 4.3.2.20 ストーンウェア製 350℃(10 分)

4.3.3　配色の割合

サーモグラフィ「AMG8833」で測定した画像の赤色で表示されている割合を調べるため配色の見本帳というWebサイトを使用した[8]。配色割合と時間経過の関係を以下のグラフに示す。

表 4.3.3.1 磁器 200℃

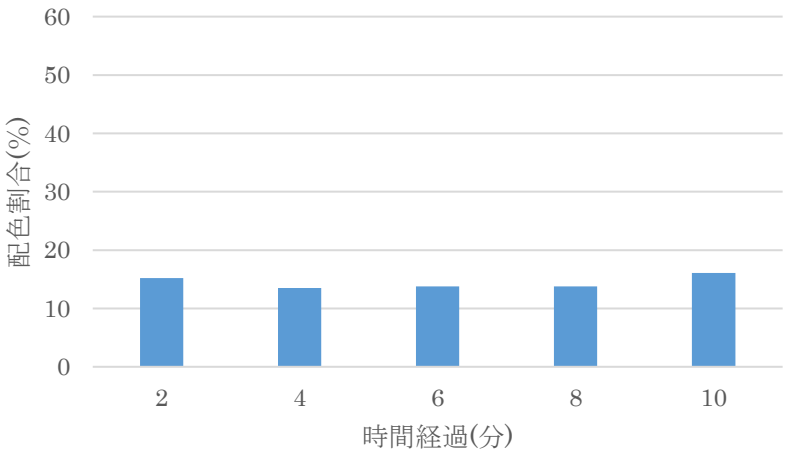


表 4.3.3.2 磁器 350℃

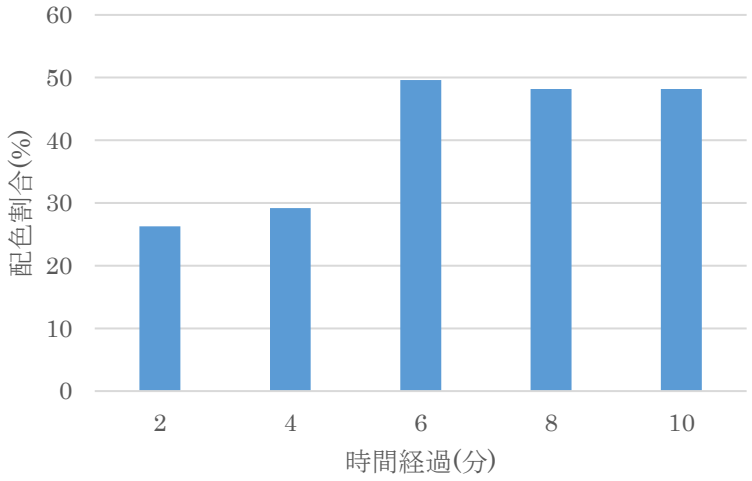


表 4.3.3.3 ストーンウェア製 200℃

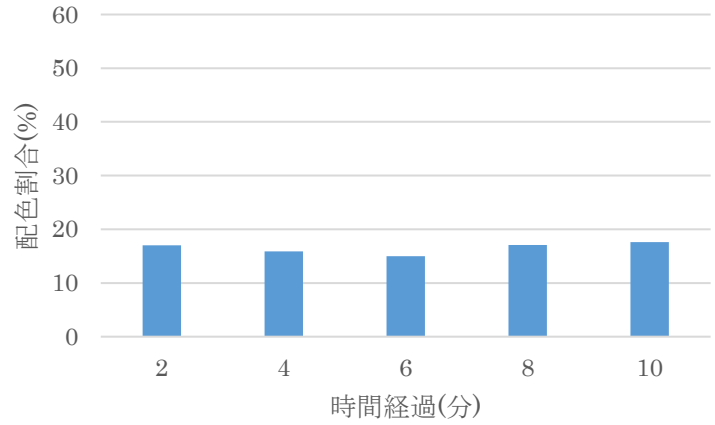
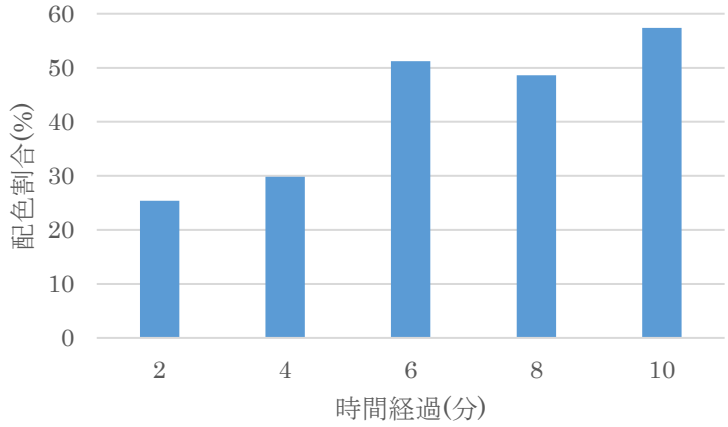


表 4.3.3.4 ストーンウェア製 350℃



配色割合と時間経過の関係を調べると磁器製、ストーンウェア製ともに 200℃の場合では約 15%になっており大きな違いがないと分かる。しかし、自然発火の危険がある 350℃では 2 分経過後では約 25%だったが 6 分後では約 50%になり、2 倍近く赤色の表示箇所が増加していることが分かった。

第5章 考察

「MLX90640」と「AMG8833」それぞれの検知画像を比べると磁器製、ストーンウェア製の食器の温度変化を比べると食器の素材による変化が起こらないと分かった。

グラフで示したとおり自然発火の危険がある 350℃の場合では 6 分後に測定範囲の最大温度である赤色で表示される箇所が約 2 倍に増加しその後も広い範囲で赤色の表示になっている。このことから配色割合と時間経過を調べることにより「AMG8833」のような測定範囲が約 80℃までのサーモグラフィカメラでも自然発火の危険がある温度を検知することができると考えられる。

第6章 まとめ

本研究では単機能電子レンジ向け、電子レンジ庫内の食材の自然発火防止システムについて検討した。Raspberry Pi 3 model B を使用しサーモグラフィカメラを作成した。サーモグラフィカメラ「AMG8833」、「MLX90640」を使用し温度変化の違いを比べ安価なサーモグラフィカメラ「AMG8833」で自然発火の危険があるあるのか検知できるか調べた。はんだごてを使用し電子レンジ庫内の食材を再現し食器を加熱した。はんだごてが 200℃の温度で加熱した場合サーモグラフィカメラ「AMG8833」では時間の経過による温度変化が得られない結果になった。しかし、はんだごてが 350℃の温度で加熱した場合サーモグラフィカメラ「AMG8833」を使用してもはんだごて付近の食器の温度が急激に上昇していることが分かった。

本研究により、温度測定範囲は約 0℃～80℃このエリアを 8x8 ピクセルに分割した 2 次元画像を使用し、食器の温度変化を検知すると約 200℃のはんだごての温度変化では時間経過による変化を検知することができなかったが食材の自然発火を起こす危険性がある約 350℃では加熱 2 分後と 10 分後を比べると温度変化に大きな違いがあることが分かった。この結果により時間経過による食器の温度上昇を検知することで高機能サーモグラフィカメラを使用せずとも温度測定範囲約 0℃～80℃のサーモグラフィカメラを使用することで低価格の単機能電子レンジ自然発火防止システムが実現可能であると分かった。

電子レンジには庫内で発生させたマイクロ波を遮断するため網目状の金属が張り巡らされている。また、複数の電子レンジにはマイクロ波を使用した温め機能の他に赤外線センサを使用したオーブン機能を搭載している、そのため電子レンジのドアガラスには赤外線を遮断する加工がされており単機能電子レンジにも赤外線を遮断するドアガラスが使用されている。ドアガラスを赤外線遮断のものからマイクロ波のみ遮断するものに変えることができればあらゆる単機能電子レンジに後付け可能な自然発火防止センサになり電子レンジの誤操作による事故が減少すると考えられる。

謝辞

本研究を進めるにあたり、ご指導を頂いた卒業論文指導教員の三好力教授に深く感謝致します。また三好研究室の皆様には、多くのご指摘を下さり感謝致します。

参考文献

[1] 総務省　第１部　特集　人口減少時代による ICT による持続的成長
http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h30/html/nd141110.html

[2] ちょっと注目電子レンジから発火！？
https://www.nikkakyo.org/system/files/chumoku255.pdf

[3] 東京くらし WEB
https://www.shouhiseikatu.metro.tokyo.jp/anzen/test/documents/densirange_cms_houkokusho.pdf

[4] 知って納得！メカランド電子レンジ編
https://www.jsme.or.jp/mechalife/jp/student/mechaland/0306.pdf

[5] アピステテクニカルノート
https://www.apiste.co.jp/contents/technical_note/technical/infrared_light/about.html

[6] 日本調理科学会誌　加熱調理と熱物性
https://www.jstage.jst.go.jp/article/cookeryscience/46/4/46_299/_pdf/-char/ja

[7] Raspbian のダウンロード
https://www.raspberrypi.org/downloads/

[8] 配色の見本帳
https://ironodata.info/

[9] 福田和宏,ラズベリー・パイ超入門,ソーテック社,2017

[10] 林和孝,ラズベリー・パイで遊ぼう！,ラトルズ,2017

[11] 河村雅人,大塚紘史,小林佑輔,小山武士,宮崎智也,石黒祐樹,小島康平
IOT/センサの仕組みと活用,翔泳社,2015

付録

AMG8833

```
“”“This example is for ラズベリー Pi（Linux）only!
   It will not work on microcontrollers running CircuitPython!"""

import os
import math
import time

import RPi.GPIO as GPIO

import busio
import board

import numpy as np
import pygame
from scipy.interpolate import griddata

from colour import Color

import adafruit_amg88xx

i2c_bus = busio.I2C(board.SCL, board.SDA)

#low range of the sensor (this will be blue on the screen)
MINTEMP = 26.

#high range of the sensor (this will be red on the screen)
MAXTEMP = 32.

#how many color values we can have
COLORDEPTH = 1024

os.putenv('SDL_FBDEV', '/dev/fb1')
pygame.init()

#initialize the sensor
sensor = adafruit_amg88xx.AMG88XX(i2c_bus, 0x68)

# pylint: disable=invalid-slice-index
points = [(math.floor(ix / 8), (ix % 8)) for ix in range(0, 64)]
grid_x, grid_y = np.mgrid[0:7:32j, 0:7:32j]
# pylint: enable=invalid-slice-index

#sensor is an 8x8 grid so lets do a square
height = 240
width = 240

#the list of colors we can choose from
blue = Color("indigo")
colors = list(blue.range_to(Color("red"), COLORDEPTH))

#create the array of colors
colors = [(int(c.red * 255), int(c.green * 255), int(c.blue * 255)) for c in colors]

displayPixelWidth = width / 30
displayPixelHeight = height / 30

lcd = pygame.display.set_mode((width, height))

lcd.fill((255, 0, 0))

pygame.display.update()
pygame.mouse.set_visible(False)

lcd.fill((0, 0, 0))
pygame.display.update()

#some utility functions
def constrain(val, min_val, max_val):
    return min(max_val, max(min_val, val))

def map_value(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

#let the sensor initialize
time.sleep(.1)

while True:

    #read the pixels
    pixels = []
    for row in sensor.pixels:
        pixels = pixels + row
    pixels = [map_value(p, MINTEMP, MAXTEMP, 0, COLORDEPTH - 1) for p
in pixels]

    #perform interpolation
    bicubic = griddata(points, pixels, (grid_x, grid_y), method='cubic')

    #draw everything
    for ix, row in enumerate(bicubic):
        for jx, pixel in enumerate(row):
            pygame.draw.rect(lcd, colors[constrain(int(pixel), 0,
COLORDEPTH- 1)],
                             (displayPixelHeight * ix, displayPixelWidth * jx,
                              displayPixelHeight, displayPixelWidth))

    pygame.display.update()
```

MLX90640

```
#include <stdint.h>
#include <iostream>
#include <cstring>
#include <fstream>
#include <chrono>
#include <thread>
#include <math.h>
#include <stdlib.h>
#include <errno.h>
#include "headers/MLX90640_API.h"

/*
 * rawrgb
 * ======
 * outputs raw false-color 24bit RGB stream of 24x32 pixels to
 * stdout
 *
 * streaming to remote host with GStreamer tools
 * ---------------------------------------------
 *
 * This is modified example code, which outputs a raw data stream of
 * false-colour thermal images from the sensor to STDOUT.
 * Each image is encoded in RGB (24bit) and has a width of 24 / height
 * of 32 pixels. Thus a single image consumes 2304 bytes and is written
 * at once to stdout.
 *
 * The data could be streamed with gstreamer to visualize it on a remote
 * host, without utilizing framebuffers. This allows to move the CPU
 * intensive encoding to the remote host (nice for headless setups with
 * Pi Zero W).
 *
 * A valid receiver in gstreamer could be started like this on a
 * receiving remote host:
 *
 * $ gst-launch-1.0 udpsrc blocksize=2304 port=5000 ! rawvideoparse use-sink-
caps=false width=32 height=24 format=rgb framerate=16/1 ! videoconvert !
videoscale ! video/x-raw,width=640,height=480 ! autovideosink
 *
 * The command above reads 2304 byte blockes from UDP port 5000,
 * interprets the data as raw video with "width=32 height=24 format=rgb",
 * scales it up to 640x480 pixels (with default interpolation) and
 * renders it to default output. So most of the processing is done on the
 * remote host.
 *
 * The sender, which is running on the device with mlx90640 connected,
 * could use the following command (assuming the receiver IP is
 * 172.16.0.2):
 *
 * $ ./rawrgb | gst-launch-1.0 fdsrc blocksize=2304 ! udpsink host=172.16.0.2
port=5000
 *
 * The sender reads the output generated by the `rawrgb` binary on stdin
 * and forwards it to the remote host 172.16.0.2 on port 5000. There's
 * no further processing done on the sender's end (source host).
 *
 * Note1:
 * For the example above, the receiver should be started first.
 * As the stream isn't encoded, data loss (during UDP transfer) or frame
 * starts couldn't be detected by the receiver. Both would lead to
 * offseted images, if the stream to the receiver doesn't start with a
 * valid frame.
 *
 * Note2:
 * The code was tested on a Raspberry Pi 0 W, the bcm2835 driver
 * was used for I2C communication. Unfortunately, the
 * `MLX90640_GetFrameData` command produces CPU load >60% in this setup.
 * The Linux system used, wasn't a Raspbian but another Debian derivate,
 * which isn't compiled with hard float support. Anyways, the method
 * `MLX90640_GetFrameData` doesn't involve float calculations.
 * The float based false-colour calculations and temprature translation
 * don't impact performance too much on soft float. Seems the limiting
 * factor is the I2C access.
 *
 */


#define MLX_I2C_ADDR 0x33

#define IMAGE_SCALE 5

// Valid frame rates are 1, 2, 4, 8, 16, 32 and 64
// The i2c baudrate is set to 1mhz to support these
#define FPS 16
#define FRAME_TIME_MICROS (1000000/FPS)

// Despite the framerate being ostensibly FPS hz
// The frame is often not ready in tim
// This offset is added to the FRAME_TIME_MICROS
// to account for this.
#define OFFSET_MICROS 850

#define PIXEL_SIZE_BYTES 3
#define IMAGE_SIZE 768*PIXEL_SIZE_BYTES

void put_pixel_false_colour(char *image, int x, int y, double v) {
    // Heatmap code borrowed from: http://www.andrewnoske.com/wiki/Code_-
_heatmaps_and_color_gradients
    const int NUM_COLORS = 7;
    static float color[NUM_COLORS][3] = { {0,0,0}, {0,0,1}, {0,1,0}, {1,1,0}, {1,0,0},
{1,0,1}, {1,1,1} };
    int idx1, idx2;
    float fractBetween = 0;
    float vmin = 5.0;
    float vmax = 50.0;
    float vrange = vmax-vmin;
    int offset = (y*32+x) * PIXEL_SIZE_BYTES;

    v -= vmin;
    v /= vrange;
    if(v <= 0) {idx1=idx2=0;}
    else if(v >= 1) {idx1=idx2=NUM_COLORS-1;}
    else
    {
        v *= (NUM_COLORS-1);
        idx1 = floor(v);
        idx2 = idx1+1;
        fractBetween = v - float(idx1);
    }

    int ir, ig, ib;


    ir = (int)((((color[idx2][0] - color[idx1][0]) * fractBetween) + color[idx1][0]) *
255.0);
    ig = (int)((((color[idx2][1] - color[idx1][1]) * fractBetween) + color[idx1][1]) *
255.0);
    ib = (int)((((color[idx2][2] - color[idx1][2]) * fractBetween) + color[idx1][2]) *
255.0);

    //put calculated RGB values into image map
    image[offset] = ir;
    image[offset + 1] = ig;
    image[offset + 2] = ib;


}

int main(int argc, char *argv[]){

    static uint16_t eeMLX90640[832];
    float emissivity = 0.8;
    uint16_t frame[834];
    static char image[IMAGE_SIZE];
    static float mlx90640To[768];
    float eTa;
    static uint16_t data[768*sizeof(float)];
    static int fps = FPS;
    static long frame_time_micros = FRAME_TIME_MICROS;
    char *p;

    if(argc > 1){
        fps = strtol(argv[1], &p, 0);
        if (errno !=0 || *p != '¥0') {
            fprintf(stderr, "Invalid framerate¥n");
            return 1;
        }
        frame_time_micros = 1000000/fps;
    }

    auto frame_time = std::chrono::microseconds(frame_time_micros +
OFFSET_MICROS);

    MLX90640_SetDeviceMode(MLX_I2C_ADDR, 0);
    MLX90640_SetSubPageRepeat(MLX_I2C_ADDR, 0);
    switch(fps){
        case 1:
            MLX90640_SetRefreshRate(MLX_I2C_ADDR, 0b001);
            break;
        case 2:
            MLX90640_SetRefreshRate(MLX_I2C_ADDR, 0b010);
            break;
        case 4:
            MLX90640_SetRefreshRate(MLX_I2C_ADDR, 0b011);
            break;
        case 8:
            MLX90640_SetRefreshRate(MLX_I2C_ADDR, 0b100);
            break;
```

```
            case 16:
                MLX90640_SetRefreshRate(MLX_I2C_ADDR, 0b101);
                break;
            case 32:
                MLX90640_SetRefreshRate(MLX_I2C_ADDR, 0b110);
                break;
            case 64:
                MLX90640_SetRefreshRate(MLX_I2C_ADDR, 0b111);
                break;
            default:
                fprintf(stderr, "Unsupported framerate: %d¥n", fps);
                return 1;
        }
        MLX90640_SetChessMode(MLX_I2C_ADDR);

        paramsMLX90640 mlx90640;
        MLX90640_DumpEE(MLX_I2C_ADDR, eeMLX90640);
        MLX90640_SetResolution(MLX_I2C_ADDR, 0x03);
        MLX90640_ExtractParameters(eeMLX90640, &mlx90640);

        while (1){
            auto start = std::chrono::system_clock::now();
            MLX90640_GetFrameData(MLX_I2C_ADDR, frame);
            MLX90640_InterpolateOutliers(frame, eeMLX90640);

            eTa = MLX90640_GetTa(frame, &mlx90640); // Sensor ambient
temprature
            MLX90640_CalculateTo(frame, &mlx90640, emissivity, eTa, mlx90640To);
//calculate temprature of all pixels, base on emissivity of object

            //Fill image array with false-colour data (raw RGB image with 24 x 32 x
24bit per pixel)
            for(int y = 0; y < 24; y++){
                for(int x = 0; x < 32; x++){
                    float val = mlx90640To[32 * (23-y) + x];
                    put_pixel_false_colour(image, x, y, val);
                }
            }

            //wite RGB image to stdout
            fwrite(&image, 1, IMAGE_SIZE, stdout);
            fflush(stdout); // push to stdout now

            auto end = std::chrono::system_clock::now();
            auto elapsed = std::chrono::duration_cast<std::chrono::microseconds>(end
- start);
            std::this_thread::sleep_for(std::chrono::microseconds(frame_time -
elapsed));
        }

        return 0;
}
```