2014 年度（平成 26 年度）

ECPⅢ

Final Report

車載用広角レンズを用いた室内見守りシステムの開発

～車載用レンズをこんなものにつけたら～

チーフアドバイザー　：　金丸　隆志　准教授

サブアドバイザー　：　桂　　晃洋　教授

サブアドバイザー　：　新井　敏夫　教授

チームメンバー

G1-11022　加藤　　諒

G1-11039　司馬　聖大

G1-11046　高橋　勇磨

G1-11050　津田　郁也

G1-11062　原田　真吾

G1-11065　福山　陽太

目次

第 1 章　緒言　（福山担当）.....1
1.1　研究背景　（福山担当）.....1
1.1.1　研究テーマ.....1
1.1.2　車載用カメラについて.....2
1.1.3　車載用広角レンズについて.....3
1.2　研究の概要　（福山担当）.....5
1.2.1　作製物の決定.....5
1.2.2　スマートフォン・タブレットの普及率.....7
1.2.3　ロボット掃除機について.....9
1.2.4　研究の目標、目的.....12
1.2.5　研究の進行について.....13
第 2 章　機構　（高橋担当）.....14
2.1　目的　（高橋担当）.....14
2.1.1　動作原理.....14
2.1.2　要求機能・制約条件.....14
2.2　機構の作製　（高橋担当）.....15
2.2.1　センサの選定.....15
2.2.2　カメラを実現するためのアイデア.....16
2.2.3　第 1 カメラ.....17
2.2.4　第 2 カメラ.....22
2.2.5　新しいセンサの選定.....26
2.2.6　新しいカメラを実現するためのアイデア.....27
2.2.7　第 3 カメラ.....29
2.3　モータについて　（高橋担当）.....34
2.3.1　モータの選定.....34
2.3.2　カメラとサーボモータの組み立て方.....35
2.4　機構の作製　（高橋担当）.....38
2.4.1　台座.....38
2.4.2　フロア.....40
2.5　機構の成果　（高橋担当）.....42
2.5.1　動作確認.....42
第 3 章　回路とプログラム　（加藤担当）.....43
3.1　目標　（加藤担当）.....43
3.1.1　概要.....43

3.1.2　要求機能 43
3.2　モータ制御　（加藤担当） 44
3.2.1　Arduino について 44
3.2.2　サーボモータについて 45
3.3　ルンバの制御　（加藤担当） 48
3.3.1　ルンバオープンインタフェース 48
3.3.2　Arduino によるルンバの制御 49
3.4　基板への実装　（加藤担当） 55
3.4.1　ユニバーサル基板での回路の作製 55
第 4 章　映像と命令の無線化（津田担当） 58
4.1　概要　（津田担当） 58
4.1.1　目標 58
4.1.2　要求機能 58
4.2　映像の無線化　（津田担当） 59
4.2.1　通信手段 59
4.2.2　映像の無線化の仕組み 59
4.2.3　映像配信（MJPEG streamer）の説明 60
4.3　シングルボードコンピュータによる映像配信　（津田担当） 61
4.3.1　用いたシングルボードコンピュータ 61
4.3.2　PandaBoard による映像配信 62
4.3.3　PandaBoard での映像の無線化の検証 64
4.3.4　BeagleBone Black による映像配信 65
4.3.5　BeagleBone Black での映像の無線化の検証 67
4.3.6　ODROID U3 による映像配信 68
4.3.7　ODROID U3 を用いた映像の無線化の検証 69
4.4　パノラマビュー　（原田担当） 70
4.4.1　パノラマビューの実現方法 70
4.4.2　カメラ台数の選定 71
4.4.3　試作パノラマビューの作製 71
4.4.4　試作パノラマビューの作製結果 72
4.4.6　カメラ 3 台によるブラウザでのパノラマビューの作製 73
4.4.7　カメラの接続方式 73
4.4.8　ブラウザでのパノラマビュー 74
4.5　命令の無線化　（津田担当） 75
4.5.1　有線での映像配信と命令の構造 75
4.5.2　命令の無線化の構造 76

4.5.3　操作用のインターフェースの概要 77
4.5.4　HTML のインターフェースのボタンのデザインと詳細 78
4.6　考察 80
第 5 章　動作確認テスト　（司馬担当） 81
5.1　評価目標　（司馬担当） 81
5.2　テスト環境　（司馬担当） 82
5.3　テスト方法　（司馬担当） 83
5.4　テスト結果　（司馬担当） 85
5.5　考察　（司馬担当） 87
第 6 章　問題点とその解決法 88
6.1　本システムの問題点 88
6.2　問題の解決法 88
6.2.1　問題 1.の解決法 88
6.2.2　問題 2.の解決法 89
6.2.3　問題 3.の解決法 89
6.2.4　問題 4.の解決法 90
第 7 章　結論　（原田担当） 92
7.1　目標　（原田担当） 92
7.2　目標への達成度　（原田担当） 92
7.3　総合評価　（原田担当） 93
第 8 章　参考文献・URL 94

第１章　　緒言　（福山担当）

1.1　　研究背景　（福山担当）

1.1.1　　研究テーマ

我々は K 社から提供された“車載用レンズをこんなものにつけたら”というテーマに取り組む。このテーマは、車載用カメラに搭載されている広角レンズを他の用途に使うことを考え、実際に形にしていくものである。

車載用レンズは主に通常の画角のレンズを使用したものと、広角のレンズを使用したものの 2 種類がある。通常の画角のレンズは自動車のドライブレコーダーなど、目標物をはっきり確認するためのカメラに使用され、広角レンズは自動車のバックビューモニターなど、広い範囲を確認するためのカメラに使用される。

テーマ遂行にあたり、私たちは図 1-1 の 2 種類の車載用広角レンズを頂いた。この広角レンズを使用して、まだ実用化されていない新しい車載用広角レンズの使用方法を考える。

図 1-1　2 種類の車載用広角レンズ

1.1.2　車載用カメラについて

本節では車載用レンズを搭載した車載用カメラについて説明する。車載用カメラとは、自動車や電車などの乗り物に設置されるカメラ［1-1］のことである。多くは車体の前方を映すものを指すが、車内の様子を記録する監視カメラや、周囲の安全確認のためのカメラも車載用カメラに含まれることがある。

主に自動車でバックビューモニター<後方認識>（図 1-2）や SUBARU 社のアイサイト<前方認識>（図 1-3）、アラウンドビューモニター<周辺認識>（図 1-4）などに利用されているが、それぞれの目的に合わせて車載用カメラに使われているレンズの種類は異なる。

今回私たちが使用する車載用レンズは、バックビューモニターなどに使われている広角レンズである。

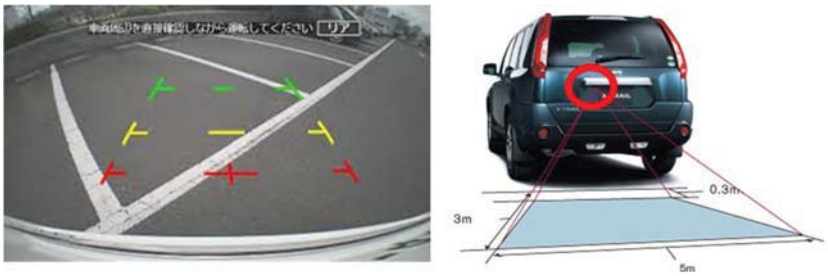

図 1-2 バックビューモニター［1-2］[1-3]

図 1-3 SUBARU 社のアイサイト［1-4］

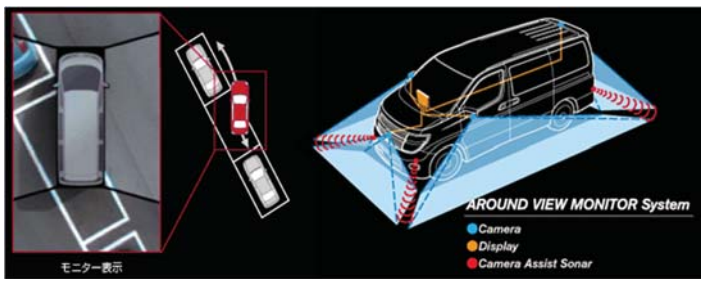

図 1-4 アラウンドビュー［1-5］

1.1.3　車載用広角レンズについて

提供された車載用広角レンズは図 1-5 のように 2 種類ある。厚みが異なるだけではなく、センサへの接続方式も異なる。厚い方をレンズ A、薄い方をレンズ B と呼ぶことにする。それぞれのレンズの規格を下記の表 1-1 に示す。

図 1-5 レンズ A（左）とレンズ B（右）

表 1-1 レンズ A とレンズ B の規格

	画角		寸法	
	垂直(°)	水平(°)	縦横(mm)	高さ(mm)
レンズA	102.5	132.4	24.6	11.55
レンズB	103.5	133.6	23.05	10.082

表からわかるように、レンズ A とレンズ B の性能はほぼ同等である。

これらのレンズからの映像を映せるようにするため、センサの載った基板を組み合わせてカメラを作製する必要がある。

そのために、レンズ A とレンズ B のどちらがセンサと組み合わせやすいかを比較したところ、厚みのあるレンズ A の構造は、センサの載っている基板と合わせるときに干渉してカメラの作製が困難になることがわかった。そのため、本研究では図 1-6 のレンズ B を用いることにした。以後「レンズ」とは「レンズ B」を表すものとする。

図 1-6 レンズ B

1.2　　研究の概要　（福山担当）

1.2.1　　作製物の決定

我々が頂いた車載用広角レンズは、通常のレンズよりも広い範囲を映すことができる。例えば、留守中の部屋の様子を見るための監視カメラとして部屋の中に設置しておけば、室内を広く観察できる。さらに、その映像を外出先からスマートフォンやタブレットで見ることができれば、ペットの様子を観察することができ、高い防犯性が期待できるのではないかと考えた（図 1-7)。

図 1-7 外出先から部屋の映像を見る様子 [1-6] [1-7]

しかし、1 台のカメラだと部屋の様子全体をカバーすることは難しい。そこで、カメラ複数台を用いて部屋の様子を 360°カバーすることにより、死角を極力少なくすることを目指すことにした。

そこで私たちが提案するものは、「室内見守りシステム」というものである。

これは複数のカメラを使用して部屋の様子をパノラマビューで映し出し、外出先から見ることができるというシステムである。さらに、カメラにモータを取り付け、上下左右に動くことを可能にする。これにより低い場所や高い場所に観察対象が位置する場合にも、映像の中心に映すことができる。室内見守りシステムの利用方法として、以下の用途が考えられる。

1. ペットの様子を観察する。
2. 鍵の閉め忘れなどをチェックする。
3. 侵入者がいないか、映像で確認する。
4. 外出先で忘れ物に気付いた際、部屋に置き忘れていないかを確認をする。

このシステムを iRobot 社の自走式掃除機「ルンバ 527J」（図 1-8）に取り付けることを考えた。理由は 2 点あり、まず 1 点目はパノラマビューを映すために部屋の中心付近にあっても不自然ではない点である。2 点目の理由は、ルンバは外部から制御可能な通信プロトコルが公開されているので、カメラに取り付けられているモータの操作と同時にルンバ本体を操ることが可能なことである。ルンバ本体を操作することで、他の部屋の様子を見ることも可能になる。また、モータとルンバの操作はスマートフォンやタブレット上で行うこととする。

図 1-8 ルンバ 527J [1-8]

1.2.2　　スマートフォン・タブレットの普及率

前節でスマートフォンやタブレット上でモータとルンバを操作すると述べた。現在、スマートフォンやタブレットが国内でどれだけ普及しているかを示したものが、図 1-9 である。

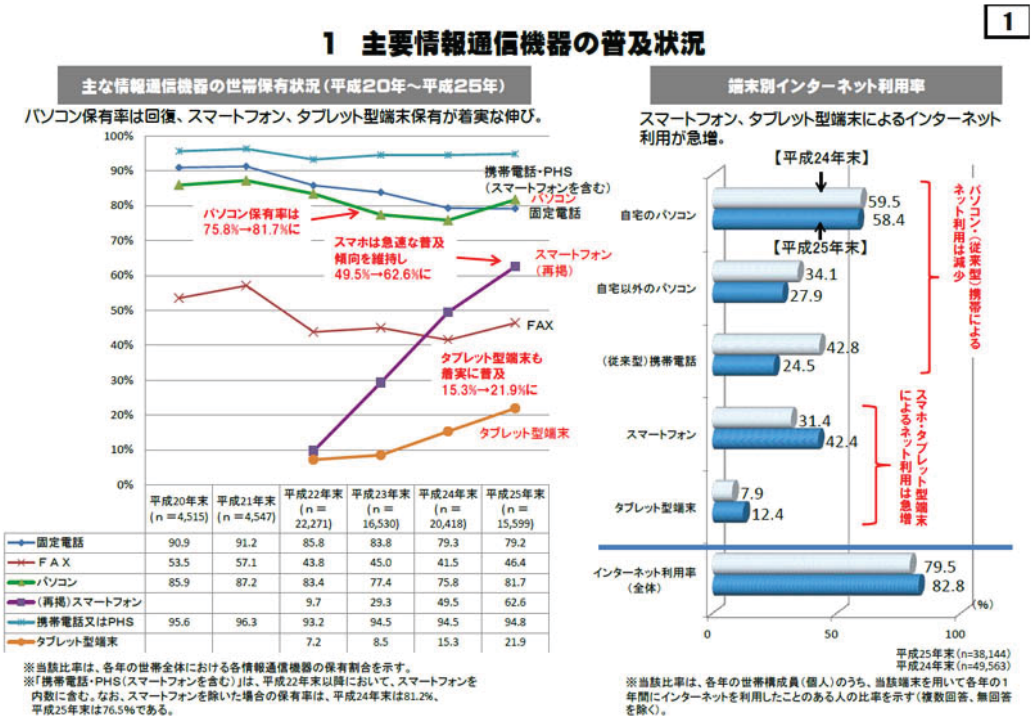

図 1-9 主要情報通信機器の普及状況 [1-9]

スマートフォンとタブレットの世帯保有状況（左のグラフ）に注目すると、平成 24 年から平成 25 年にかけて、スマートフォンの保有率は 49.5%から 62.6%に上昇している。タブレットの保有率も 15.3%から 21.9%に上昇していることがわかる。利用率（右のグラフ）についても同様で、平成 24 年から平成 25 年にかけて、スマートフォンでのインターネット利用率は 31.4%から 42.4%に上昇し、タブレットでのインターネット利用率は 7.9%から 12.4%に上昇していることがわかる。

そして図 1-10 がタブレット端末の国内出荷台数予測についてのグラフである。

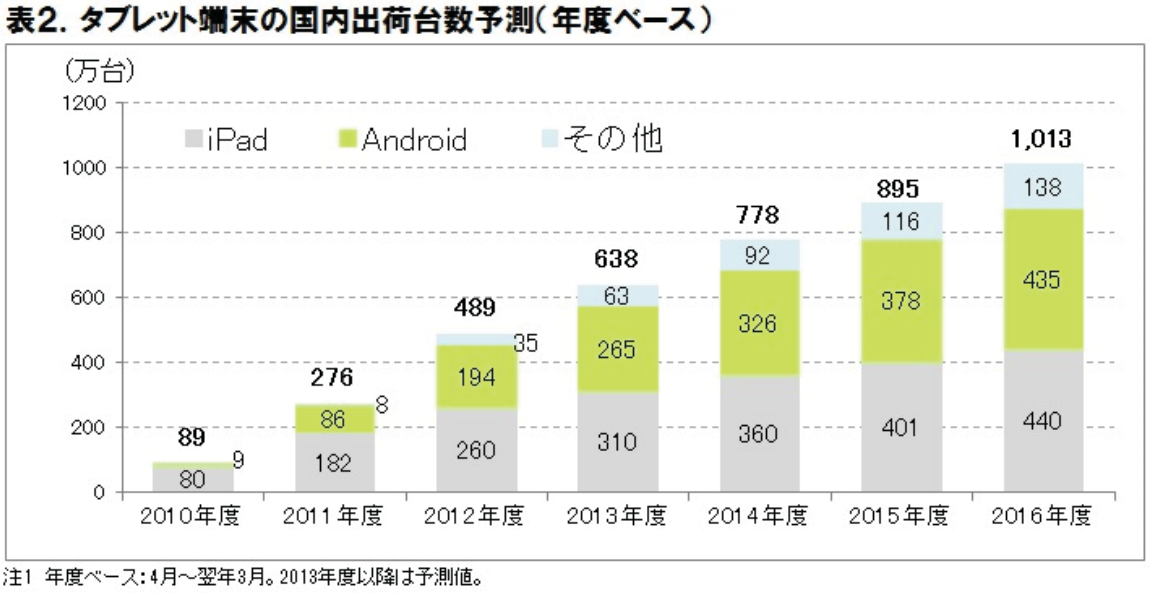

図 1-10 タブレット端末の国内出荷台数予測［1-10］

年々出荷台数が上昇し、市場規模が拡大してきていることが分かる。この 2 つの図から、今後は今よりも多くのスマートフォン、タブレットの利用者が増える見込みなので、本研究で作製する室内見守りシステムの需要の向上が見込める。

1.2.3　　ロボット掃除機について

ロボット掃除機とは、1.2.1 節で挙げた「ルンバ」のように、自走して自動的に掃除をする機械のことである。ロボット掃除機にはルンバ以外にもシャープ社の「COCOROBO（ココロボ）」や、東芝社の「トルネオ　ロボ」、パナソニック社の「ルーロ」、などがある。

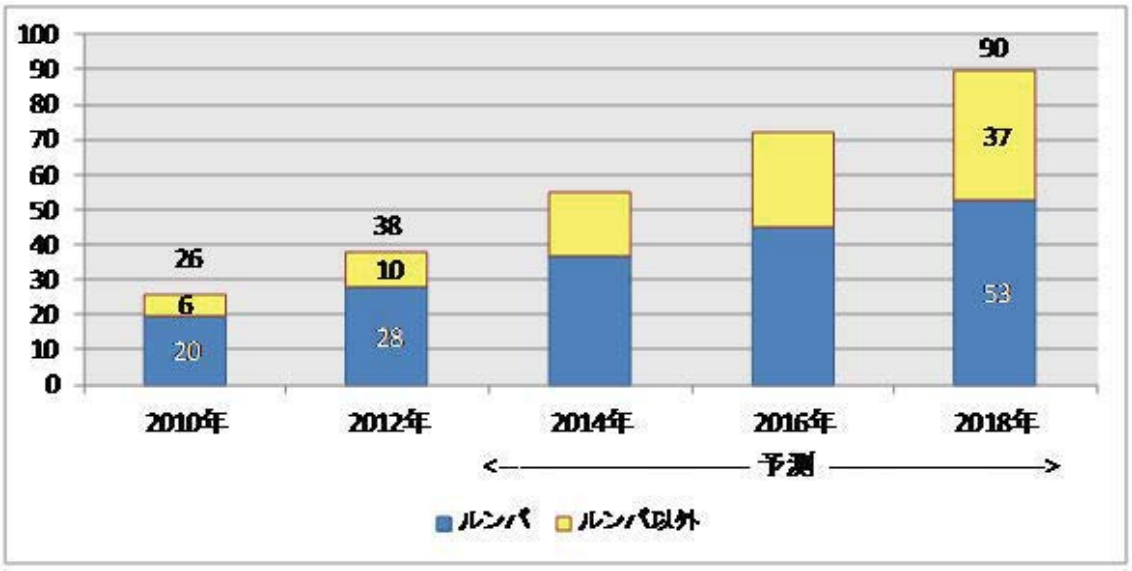

図 1-11 国内のロボット掃除機市場規模予測（単年販売台数、単位：万台）　[1-11]

図 1-11 のように、国内のロボット掃除機の市場規模は 2018 年には 2010 年の約 3.5 倍になると予測されている。このように今後はロボット掃除機の利用者が増える見込みなので、試験的に室内見守りシステムを取り付ける媒体としては適しているといえる。

ロボット掃除機の例として、シャープ社の「COCOROBO RX-V200」（図 1-12）を紹介する。

図 1-12 COCOROBO RX-V200 [1-12]

ロボット掃除機「COCOROBO」シリーズの特徴は、本体に 1 台のカメラが搭載されているところである。COCOROBO とスマートフォンが外出中など異なる通信環境下にある時に「おそとモード」（図 1-13）という機能が使用できる。この機能を使うと、COCOROBO が置いてある部屋の室温、湿度などがわかる上に、COCOROBO に搭載されているカメラが撮影した静止画などを確認することができる。「おそとモード」の外出先から部屋の様子を確認できる点では、既に防犯性を伴った機能を持っているといえる。

図 1-13 おそとモードでの操作画面 [1-13]

しかしこの「おそとモード」にも欠点がある。「おそとモード」では静止画を撮るのみで、動画を撮影することができない点である。これでは外出先からリアルタイムで部屋の様子を観察できるとは言えない。さらに、COCOROBO にはカメラが 1 台のみが搭載されているので、1 枚の写真で部屋全体の様子を見ることができない。静止画を 1 枚撮影するごとに、COCOROBO 本体が 90° 回転しながら計 4 枚の静止画を撮影することで 360° をカバーしている。

その他にも「おそとモード」では、外出先から COCOROBO 本体を移動させる機能が搭載されている。しかし自由に操ることができるわけではない。「おそとモード」の画面から「間取りマップ」(図 1-14) というページにあらかじめ部屋の間取りや家具の配置を入力し、COCOROBO を指定した場所へ移動させるという機能である。

図 1-14 間取りマップの設定画面 [1-14]

このように、上記の「おそとモード」による部屋の様子を撮影する機能と、「間取りマップ」を利用した本体を移動させる機能には一定の制限があることがわかる。これに対して私たちが提案する「室内見守りシステム」を搭載したルンバならば、一度に 360° を映した映像をリアルタイムで観察でき、外出先からルンバを思い通りに操作することが可能になる。

1.2.4　研究の目標、目的

私たちが開発する室内見守りシステムは、スマートフォンやタブレット上で外出先から部屋の様子を観察できるシステムである。

このシステムをロボット掃除機に搭載し、部屋の様子を動きながら観察することができるようにする。

このシステムの利点は下記の通りである。

1. ペットや赤ん坊、老人の様子を外出先から観察することで防犯性を高めることができる。
2. 鍵の閉め忘れなどをチェックすることができる。
3. 侵入者がいた場合、映像で確認することができる。
4. 外出先で物を忘れたとき、部屋に置き忘れていないか確認できる。
5. 複数の部屋の観察を、ロボット掃除機本体を移動させることによって実現できる。

なお、本研究ではルンバを操作するために用いるデバイスをスマートフォンではなくタブレットに統一する。理由は、スマートフォンの画面は小さくパノラマビューの映像を確認しにくいからである。

1.2.5　研究の進行について

我々は研究を進めていく上で、3 つの班に分かれて作業をした。

1 つ目の班は機構班である。機構班の役割は以下に 3 つ示す。

1. カメラのフレームを作製すること
2. 次に、ルンバの上にカメラを載せるための台座を作製すること
3. 最後に、ルンバの上にバッテリーや周辺回路を置くためのフロアを作製すること

2 つ目の班は Arduino 班である。Arduino 班の役割は以下に 2 つ示す。

1. Arduino を用いてサーボモータを操作可能にすること
2. Arduino とルンバを接続して、ルンバの操作を可能にすること

3 つ目の班はイメージ班である。イメージ班の役割は以下に 3 つ示す。

1. ルンバ上に載せた複数台のカメラで撮影した映像を無線でタブレットに配信すること
2. 複数台のカメラで撮影した映像をパノラマビューで表示するシステムを作製すること
3. ルンバをタブレット上でコントロールするページを HTML 形式で作製すること

上記の 3 班がそれぞれ作業を進行し、室内見守りシステムの完成を目指す。

第 2 章　　機構　（高橋担当）

2.1　　目的　（高橋担当）

2.1.1　　動作原理

本章では本研究で作製する室内見守りシステムの機構について解説する。

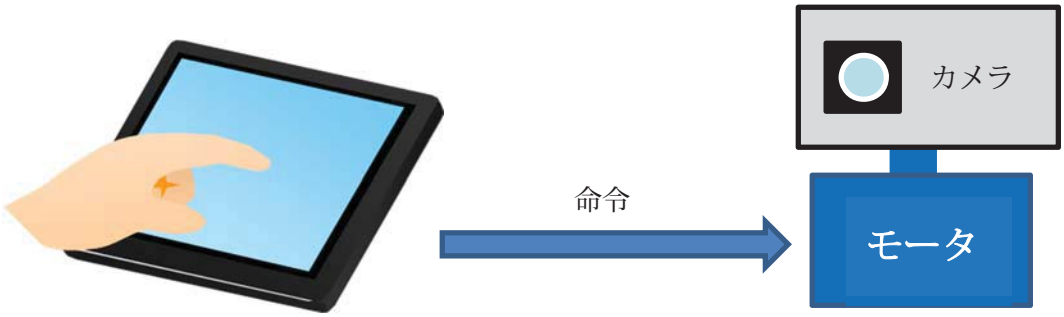

図 2-1 動作原理

この室内見守りシステムを実現するためには、図 2-1 のようにタブレットをタッチすることで、プログラムを通してモータを稼働させカメラを上下左右に動かすことが必要である。また、本研究ではタブレット上でルンバの操作も可能にする。機構班は、このカメラと動力部分の機構を作製していく。

2.1.2　　要求機能・制約条件

室内見守りシステムの作製にあたって必要な機構の要求機能を以下に示す。

1. 360°映すことのできる台数のカメラのフレームを作製すること
2. カメラの上下左右方向の運動を可能にすること

次に機構を動作させるにあたり、モータと周辺回路とバッテリーを取り付ける必要がある。さらに想定する取り付け位置を考慮した制約条件を以下に示す。

1. モータを取り付けることを考慮したカメラのフレームを設計すること
2. ルンバにモータの付いたカメラの取り付け方と 360°映るような配置を考えた台座の作製を行うこと
3. バッテリーと周辺回路の取り付け位置を考慮したフロアを作製すること

これらを考慮し作製に取り組んでいく。

2.2　　機構の作製　（高橋担当）

2.2.1　　センサの選定

1.1.3 節で述べたとおり、カメラとはレンズとセンサを組み合わせたものである。本研究で提供されたのはレンズのみであるので、機構班はセンサを選定するところから始めた。

数種類のセンサを探したところ、まず小さくて扱いやすいという理由から BUFFALO 社の WEB カメラ BSW32KM03（図 2-2）の中に使われているセンサ（図 2-3）を第 1 カメラと第 2 カメラで使用することにした。

図 2-2 BSW32KM03 [2-1]

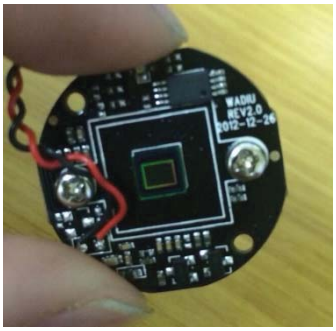

図 2-3 BSW32KM03 のセンサ

しかし、後述の理由により BSW32KM03 のセンサではレンズとのスペックが合わなかったため、Logicool 社の WEB カメラ C310（図 2-4）の中に使われているセンサ（図 2-5）を第 3 カメラのタイプ A、タイプ B で使用することにした。

図 2-4 C310 [2-2]

図 2-5 C310 のセンサ

上記の第 1 カメラ、第 2 カメラ、第 3 カメラタイプ A、第 3 カメラタイプ B を 2.2.3 節以降で順に解説していく。

2.2.2　　カメラを実現するためのアイデア

既存の WEB カメラのセンサを使用して作製する、2.1.2 の要求機能・制約条件を満たすカメラのフレームについて機構班は様々な案を出した。

その結果、3 つのパーツを使用して組み立てるカメラのフレームの設計をすることにした。

そこで各部品の図面を 3 次元 CAD ソフトウェアである Solidworks で作製した。このソフトウェアを用いることで設計・アセンブリ・図面の修正が容易にでき、また正確な図面を簡単に作製することができるので作製時間も短縮することができる。

2.2.3　　第 1 カメラ

第 1 カメラ（図 2-6）とは BSW32KM03 のセンサを用いて作製したカメラである。第 1 カメラはレンズとセンサのピントを手動で合わせられるよう、レンズパーツ、センサパーツ、外枠パーツの 3 つのパーツから組み立てられている。

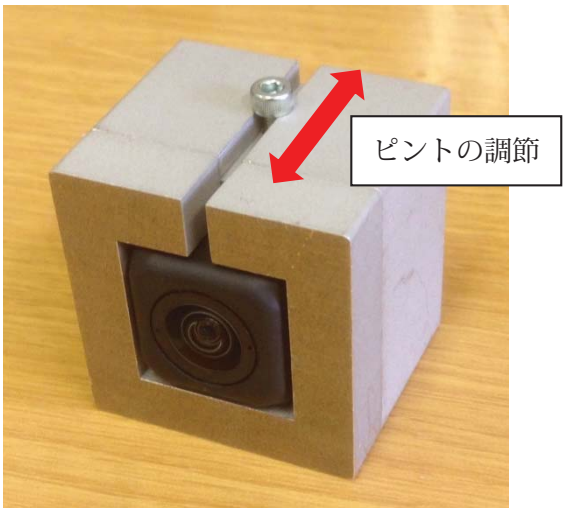

図 2-6 第 1 カメラの写真

図 2-6 のようにカメラの上面にねじを取り付け、ねじの頭をフレームよりも外に出しそれをつまみとする。そして、手動でつまみを前後に動かしてピントを合わせる仕様である。

第 1 カメラのそれぞれのパーツを以下に示す。

図 2-7 はレンズパーツである。レンズパーツはレンズの外周に合わせて四角くくり抜かれている部分にレンズをはめ込み固定する。このパーツはカメラの前面部分に当たるパーツである。

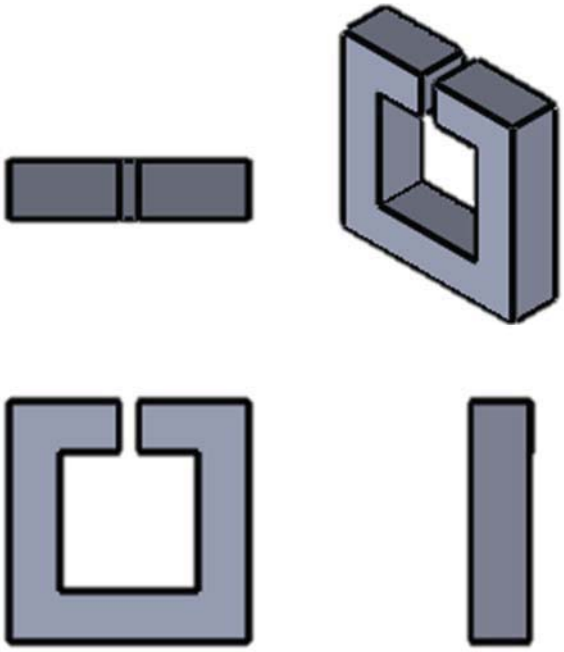

図 2-7 第 1 カメラのレンズパーツ

図 2-8 はセンサパーツである。センサパーツの前面部分に空いている 2 つのねじ穴とセンサに空いている穴を重ねて、ねじで留める。このパーツは外枠パーツの内部に含まれるパーツである。図 2-9 は実際にセンサパーツにセンサを取り付けた様子である。

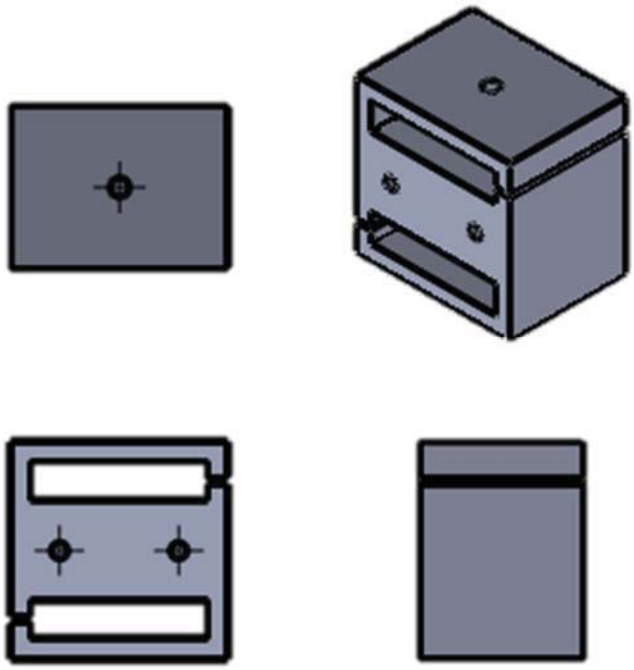

図 2-8 第 1 カメラのセンサパーツ

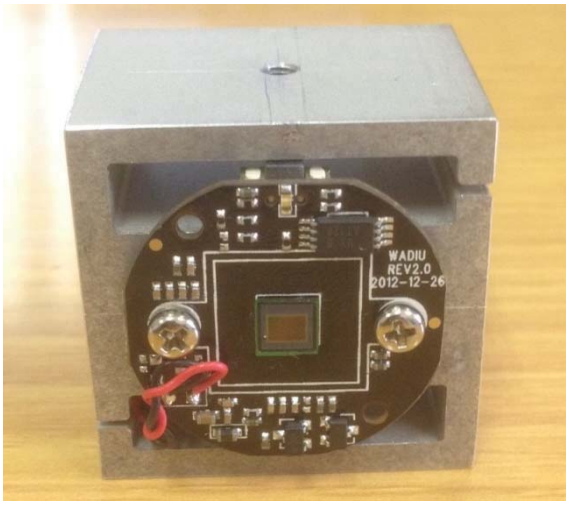

図 2-9 第 1 カメラのセンサパーツと BSW32KM03 のセンサ

図 2-10 は外枠パーツである。外枠パーツはレンズパーツの後面に当たるパーツである。

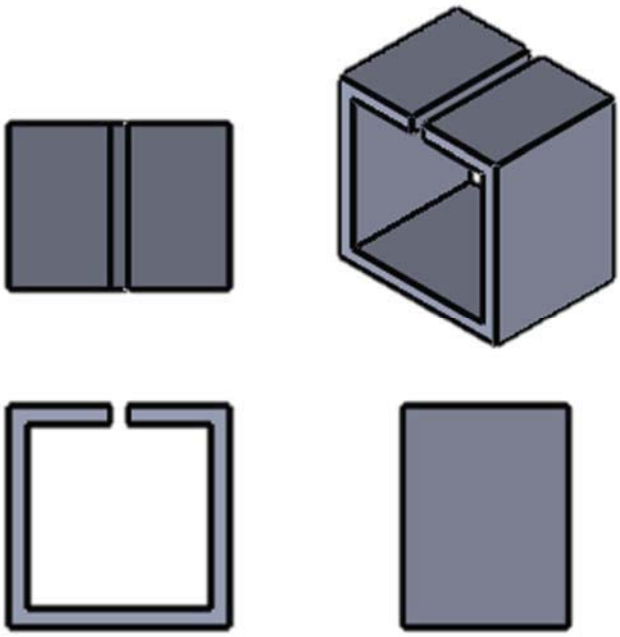

図 2-10 第 1 カメラの外枠パーツ

組立図を下記に示す。第 1 カメラではレンズパーツと外枠パーツは接着剤で固定している。

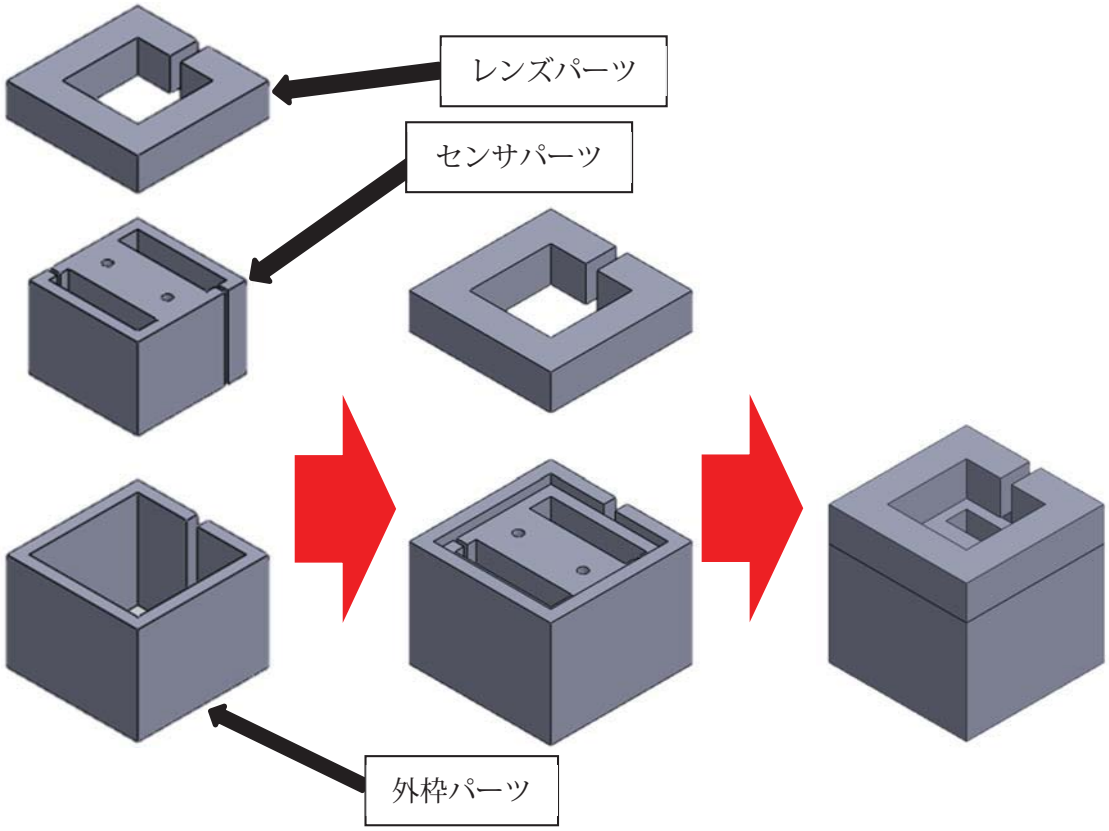

図 2-11 第 1 カメラの組立図

考察

思い通りにカメラを組み上げられ、ピントも手動で合わせることができた。しかし、以下の点から改良が必要であると判断した。

1. モータの取り付けができなかったこと
2. レンズはレンズパーツにはめ込んでいるだけなのでずれやすいこと
3. レンズパーツと外枠パーツが固定されているのでレンズがずれた時に直すのに手間がかかるということ

2.2.4　　第 2 カメラ

第 2 カメラ（図 2-12）とは BSW32KM03 のセンサで作製した第 1 カメラの改良版のカメラである。第 1 カメラと同様にレンズとセンサのピントを手動で合わせられるよう、レンズパーツ、センサパーツ、外枠パーツの 3 つのパーツから組み立てられている。基本的な作りは変わらないが、モータの取り付けやパーツの固定方法など改良点がいくつかある。

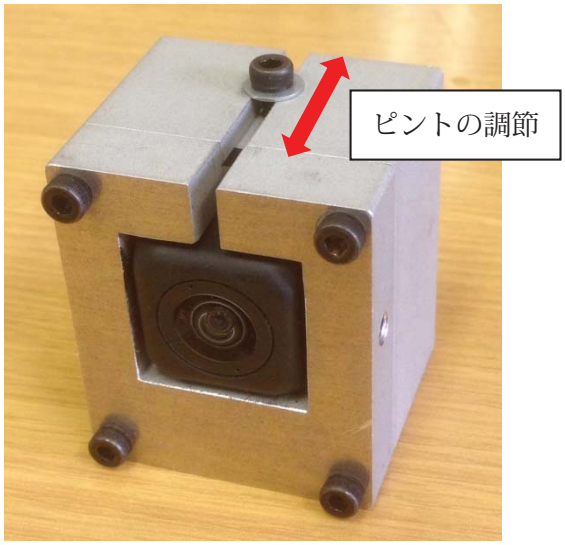

図 2-12 第 2 カメラの写真

このカメラでは第 1 カメラと同様に、センサパーツの上面にねじを取り付け、ねじの頭を外枠パーツよりも外に出しそれをつまみとする。そして、手動でつまみを前後に動かしてピントを合わせる仕様である。

第 2 カメラのそれぞれのパーツを以下に示す。

図 2-13 はレンズパーツである。レンズパーツはレンズの外周に合わせて四角くくり抜かれている部分にレンズをはめ込み固定する。このパーツはカメラの前面部分に当たるパーツである。

第 1 カメラとの変更点を以下に示す。

1. カメラの下側にモータを取り付けるため、底面の厚みを増やした。
2. レンズをずれにくくするためにレンズの左右 2 か所からいもねじで固定させる機構にした。
3. 外枠パーツとねじで固定させるために四隅にねじ穴を切った。

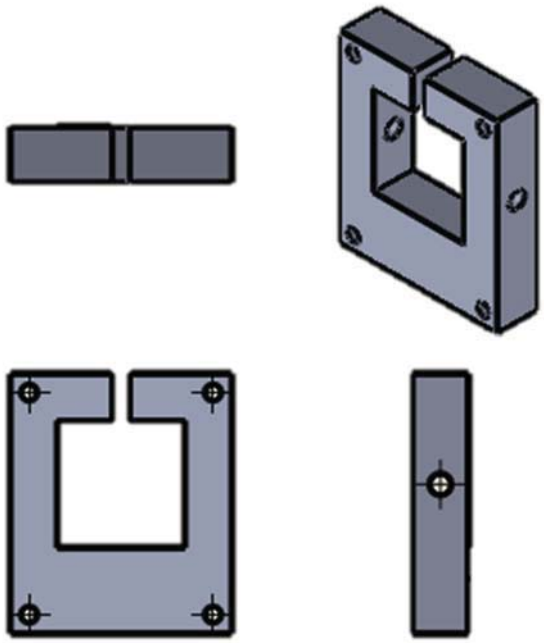

図 2-13 第 2 カメラのレンズパーツ

図 2-14 はセンサパーツである。センサパーツの前面部分に空いている 2 つのねじ穴とセンサに空いている穴を重ねて、ねじで留める。このパーツは外枠パーツの内部に含まれるパーツである。

第 1 カメラとの変更点を以下に示す。

1. 軽量化のためセンサを取り付ける部分の体積を減らした。
2. レンズパーツと外枠パーツをねじで固定させる機構にした。

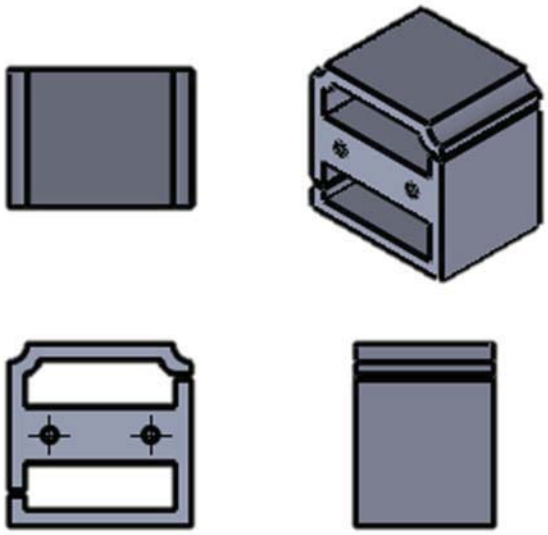

図 2-14 第 2 カメラのセンサパーツ

図 2-15 は外枠パーツである。外枠パーツはレンズパーツの後面に当たるパーツである。

第 1 カメラとの変更点を以下に示す。

1. カメラの下側にモータを取り付けるためのねじ穴を切るために、底面の厚みを増やした。
2. レンズパーツと外枠パーツをねじで固定させる機構にした。

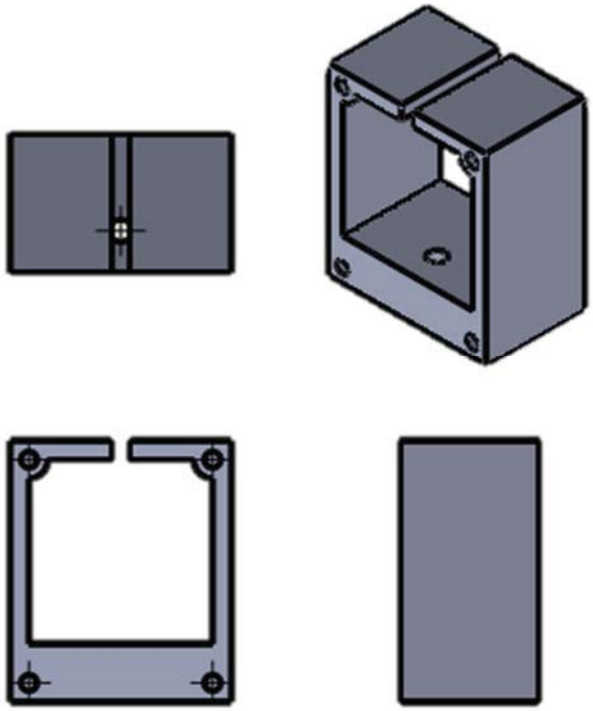

図 2-15 第 2 カメラの外枠パーツ

考察

第 1 カメラに加えて、下記に示す長所が得られた。

1. モータの取り付けができたこと
2. レンズとレンズパーツがずれにくいこと
3. レンズがずれたとしても簡単に直せること

2.2.5　新しいセンサの選定

第 1 カメラ、第 2 カメラと BSW32KM03 のセンサを使用してカメラを作製したが、映し出される範囲がレンズの仕様書に記載されているスペックよりも狭いという問題が生じた。

K 社のリエゾンに尋ねたところ、センサのスペックがレンズと合っていないのではないかというご指摘を頂いた。調べたところ、本来 1/4 インチのセンサを用いるべきところ、BSW32KM03 のセンサは 1/5 インチであった。そこで本来の画角で映し出すために 1/4 インチのセンサを選定しなおすことにした。

1/4 インチのセンサを数種類探したところ、センサに付いている突起物が少なくカメラのフレームと干渉しないという理由から前述の通り Logicool 社の WEB カメラ C310（図 2-4）の中に使われているセンサ（図 2-5）を第 3 カメラのタイプ A、タイプ B で使用することにした。

2.2.6　新しいカメラを実現するためのアイデア

既存の WEB カメラのセンサを使用し、2.1.2 の要求機能・制約条件を満たすカメラのフレームについて案を出した。その結果、2 つの案で設計することにした。

1 つは第 3 カメラタイプ A(図 2-16)である。このカメラは C310 の本来のレンズを外し、提供されたレンズを付けたふたパーツ（図 2-17）を設計し取り付けるという案である。

図 2-16 第 3 カメラタイプ A

このふたパーツを下記に示す。

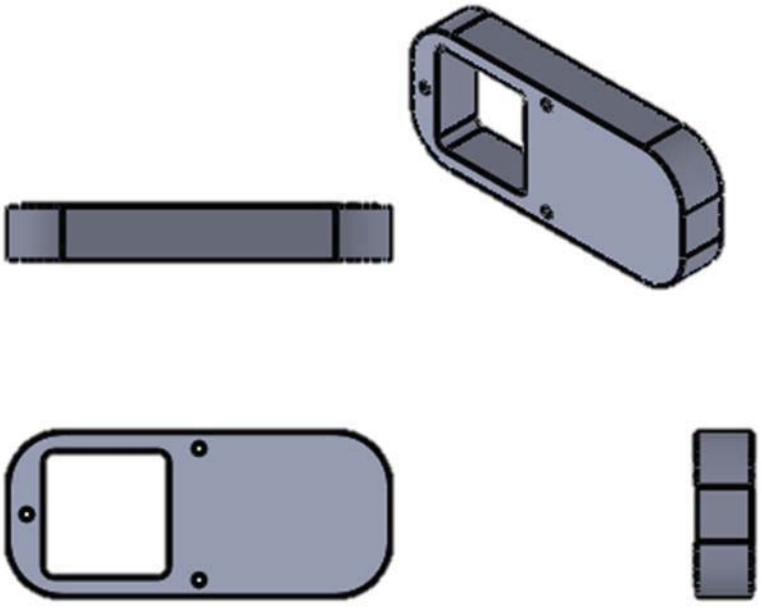

図 2-17 第 3 カメラタイプ A のふたパーツ

もう 1 つは次節で述べる第 3 カメラタイプ B である。このカメラは第 2 カメラと同じ仕組みで C310 のセンサに合わせた 3 つのパーツを用いてカメラのフレームを組み立てるという案である。

2 つの案を比較したところ、以下の理由から第 3 カメラタイプ A を廃案とした。

1. カメラのフレームの一部が加工パーツではないのでモータの取り付けが難しいこと
2. レンズがずれた際に直すのに多くの手間がかかること

そのため本研究では最終的に第 3 カメラタイプ B を使用することにした。以後「第 3 カメラ」とは「第 3 カメラタイプ B」を表すものとする。

2.2.7　　第 3 カメラ

第 3 カメラ（図 2-18）とは C310 のセンサで作製したカメラである。第 3 カメラと同様にレンズとセンサのピントを手動で合わせられるよう、レンズパーツ、センサパーツ、外枠パーツの 3 つのパーツから組み立てられている。

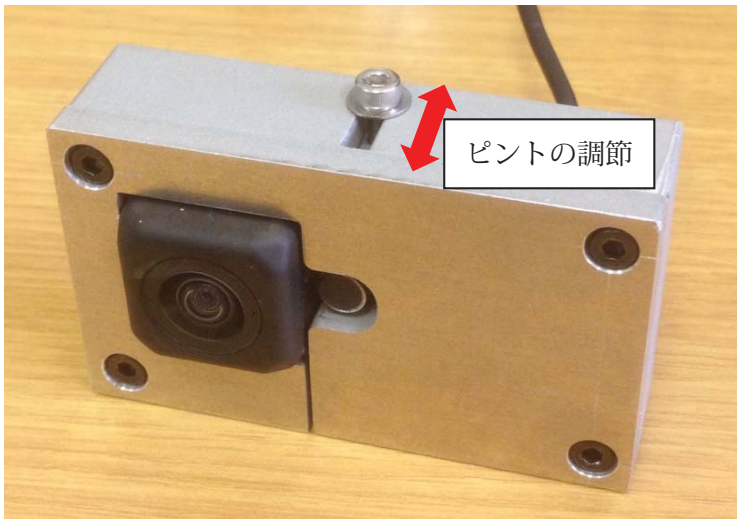

図 2-18 第 3 カメラの写真

このカメラでは第 3 カメラと同様に、センサパーツの上面にねじを取り付け、ねじの頭を外枠パーツよりも外に出しそれをつまみとする。そして、手動でつまみを前後に動かしてピントを合わせる仕様である。

第 3 カメラのそれぞれのパーツを以下に示す。

図 2-19 はレンズパーツである。レンズパーツは大きく四角にくり抜かれている部分にレンズをはめ込み、左からいもねじ 1 つで固定する。その右側の小さくくり抜かれている部分はセンサに付いているマイクを通すための穴である。このパーツはカメラの前面部分に当たるパーツである。

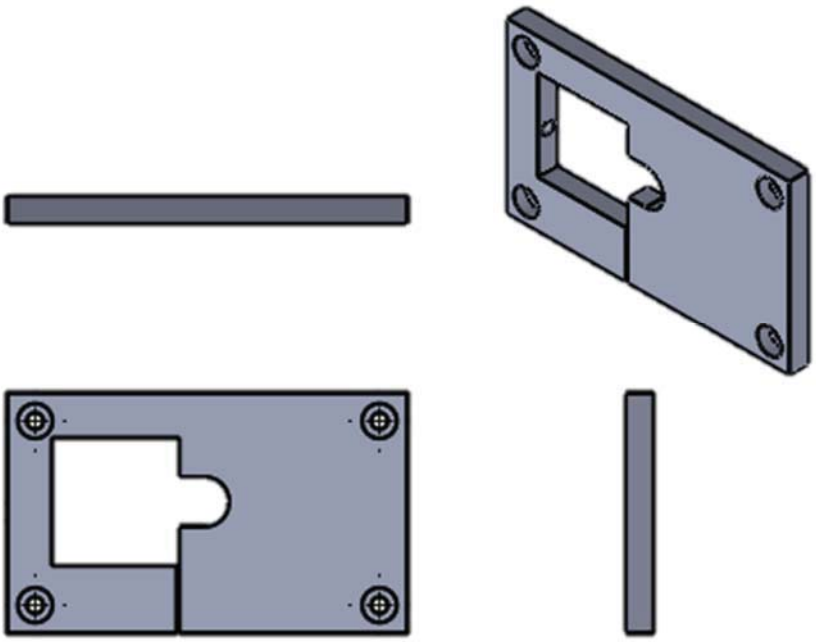

図 2-19 第 3 カメラのレンズパーツ

図 2-20 はセンサパーツである。センサパーツの左下と右上の 2 か所に空いているねじ穴とセンサに空いている穴を重ねて、ねじで留める。このパーツは外枠パーツの内部に含まれるパーツである。図 2-21 は実際にセンサパーツにセンサを取り付けた様子である。

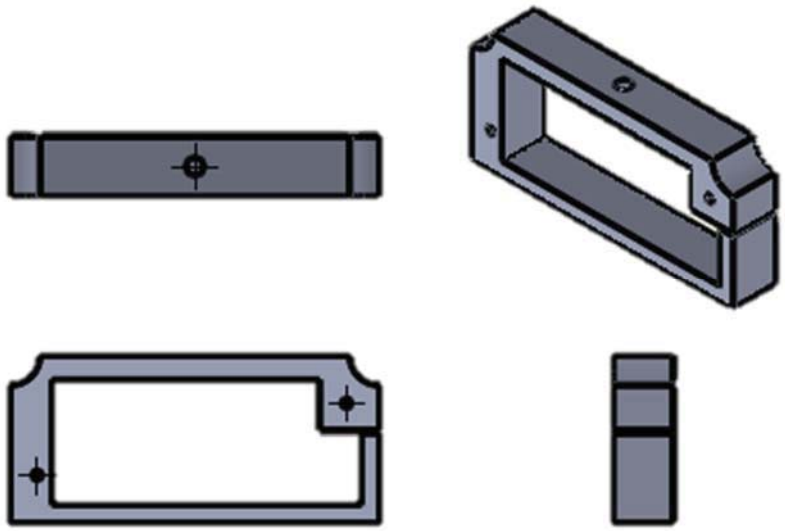

図 2-20 第 3 カメラのセンサパーツ

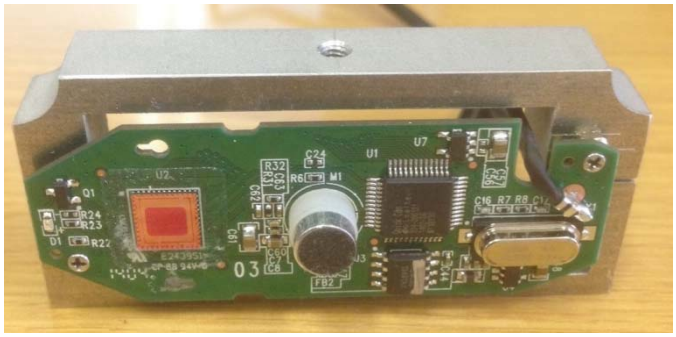

図 2-21 第 3 カメラのセンサパーツと C310 のセンサ

図 2-22 は外枠パーツである。外枠パーツはレンズパーツの後面に当たるパーツである。

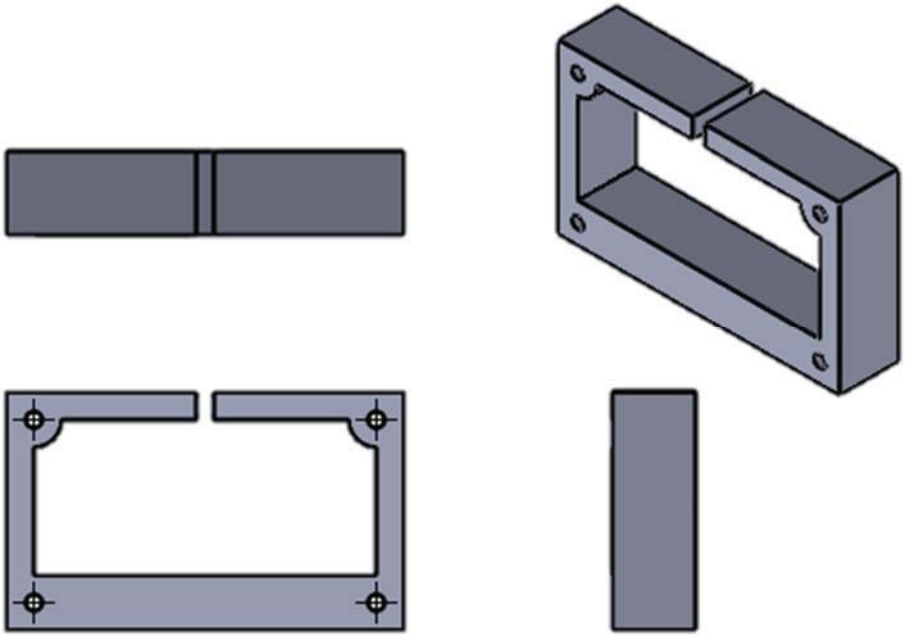

図 2-22 第 3 カメラの外枠パーツ

以上の 3 つのパーツを第 1 カメラの図 2-11 と同様に組み立ててカメラを作製した。

カメラのフレームを作製し終えたので、WEB カメラの通常レンズと、広角レンズで画角の比較を行う。図 2-23 は C310 の WEB カメラで撮った写真であり、図 2-24 は第 3 カメラで撮った写真である。比較してみると図 2-24 の方が広い範囲を映し出していることがわかる。

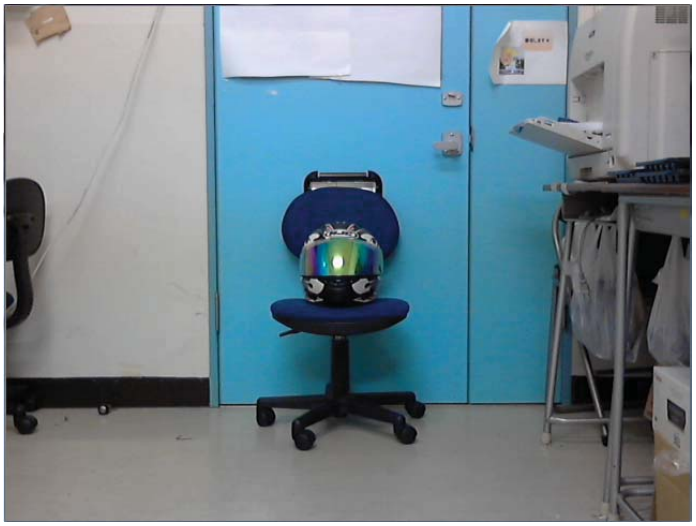

図 2-23 WEB カメラ C310 で撮った写真

図 2-24 第 3 カメラで撮った写真

考察

思い通りに組み立てられ、問題点も無かった。さらにセンサを変えたことにより、レンズのスペックとほぼ同じ 133°の広範囲を映すことができた。つまり 360°映すためのカメラは 3 台で足りるので、機構班はこの第 3 カメラを 3 台作製することにした。

2.3　モータについて　（高橋担当）

2.3.1　モータの選定

カメラのフレームが完成したので次にこれにモータを取り付け、カメラを上下左右方向に動作できるようにする。カメラ 1 台につき上下方向に 1 つ、左右方向に 1 つ、計 2 つのモータが必要となる。その際に使用するモータは近藤科学社のサーボモータ KRS-788HV ICS Red Version（図 2-25）を採用した。

カメラを動かすために必要なトルクを T＝重量×距離の公式に当てはめて計算する。カメラの重量は約 0.1kg である。また、サーボモータには後述するサーボアームを装着するので重心までの距離は約 2.5cm となる。よって、

$$
\begin{aligned}
T &= \text{重量} \times \text{距離} \\
&= 0.1 \times 0.25 \\
&= 0.25 \ (\text{kgf} \cdot \text{cm})
\end{aligned}
$$

と必要なトルクが計算できる。我々が用いる KRS-788HV ICS Red Version は最大トルク 10.0kgf・cm なのでカメラを動かすには十分な性能を持っていることがわかった。

2.3.2　　カメラとサーボモータの組み立て方

本節ではカメラとサーボモータの取り付け方法を解説する。

はじめに、左右方向の機構から解説していく。まず KRS-788HV ICS Red Version に同じ近藤科学社のサーボアーム 700A（図 2-26）を取り付ける。そうすると図 2-27 のようになる。

図 2-25 KRS-788HV ICS Red Version [2-3]

図 2-26 サーボアーム 700A [2-4]

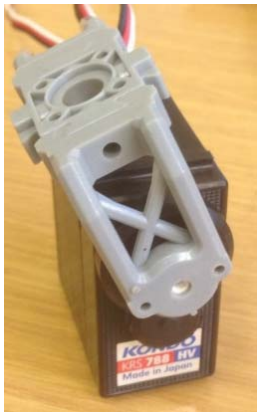

図 2-27 サーボモータとサーボアームを組み立てた様子

サーボモータとサーボアームを組み合わせたものに図 2-28 のように第 3 カメラを組み合わせる。

図 2-28 第 3 カメラの左右方向の組み立てた様子

次に上下方向の機構を解説する。左右方向に動かすサーボモータに自作のブラケット（図2-29）を取り付ける。その様子が図2-30である。

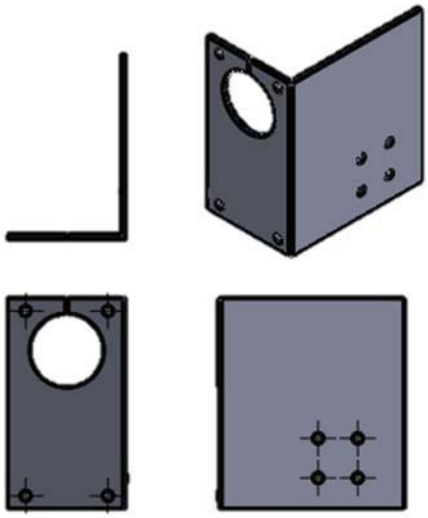

図 2-29 ブラケット

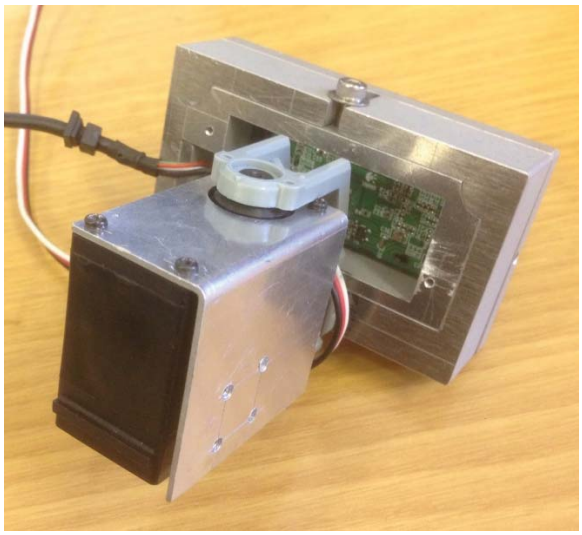

図 2-30 第3カメラと左右方向用のサーボモータとブラケット

次に上下方向用のサーボモータを図 2-31 の様に取り付ける。

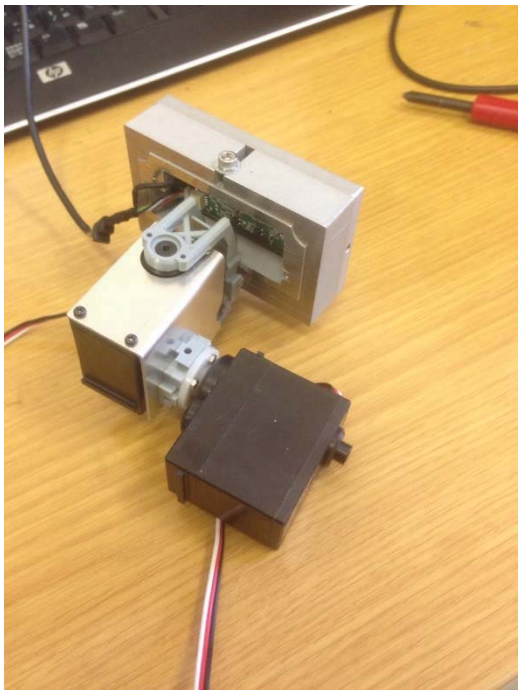

図 2-31 第 3 カメラの上下左右方向の組立図

このままでは不安定だが次の節で解説する台座に取り付けるとサーボモータは安定に固定される。

考察

思い通りに組み立てられた。上下左右方向の動作も問題ない。しかし、どうしてもサーボモータ 2 つとカメラを一直線に組み立てることができなかった。

2.4　機構の作製　（高橋担当）

2.4.1　台座

カメラ 3 台をルンバに取り付けるために台座（図 2-32）を作製することにした。

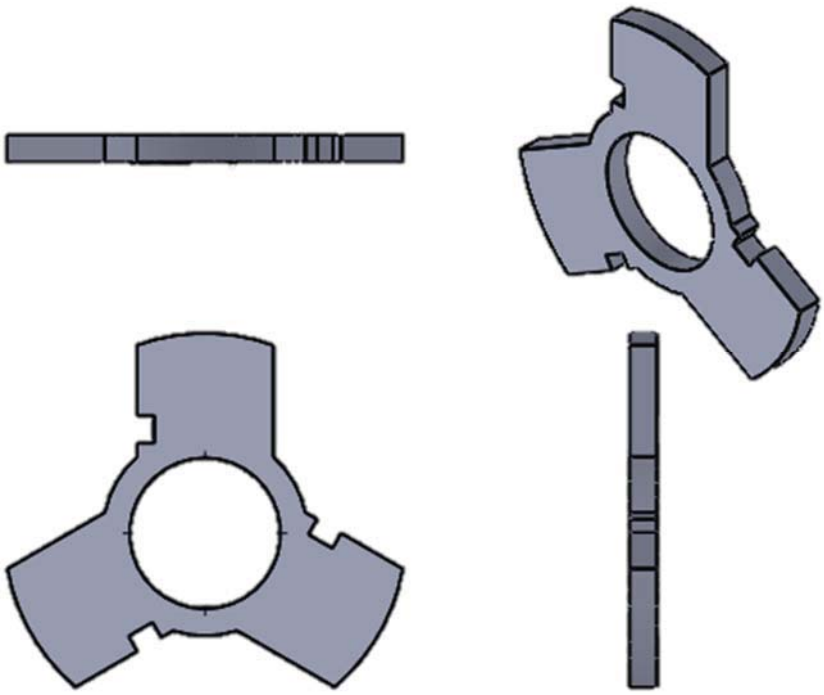

図 2-32 台座

この台座では 3 枚の羽のような部分にそれぞれ上下方向用のサーボモータを取り付ける。取り付けの際、羽の先に左右方向用のサーボモータが台座から出るように上下方向用のサーボモータを置き、留め具（図 2-33）を上から被せ 2 か所からねじで留める。また、この台座はルンバ用に作製するため、ルンバのスイッチを押せるように中心に穴をあける。

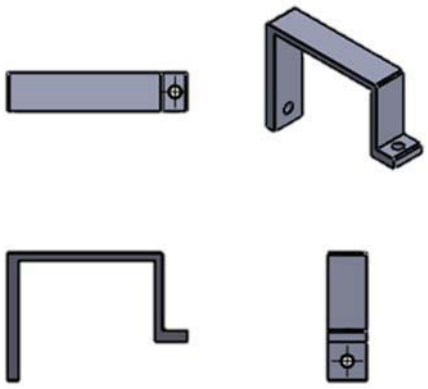

図 2-33 留め具

既製品であるルンバを加工することを避け、台座とルンバは耐震マットで吸着させる。耐震マットはフリーカット用を使用して台座の裏側に図 2-34 のように張り付けた。

台座にカメラ 3 台を取り付け、それをルンバに取り付けた様子が図 2-35 である。

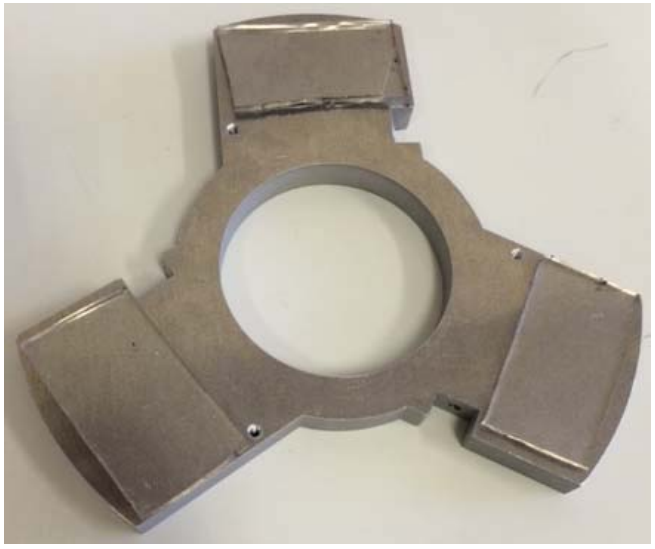

図 2-34　台座に耐震マットを貼りつけた様子

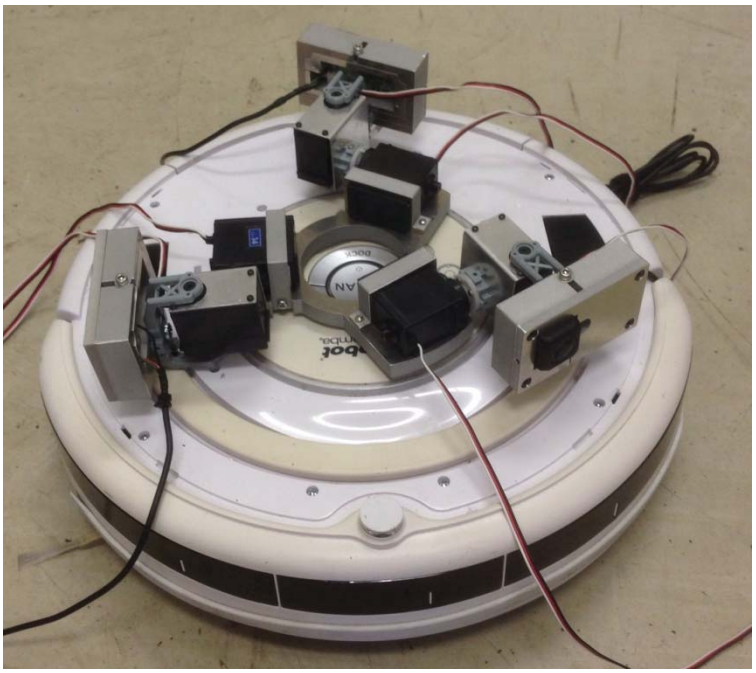

図 2-35 ルンバにカメラ 3 台を取り付けた様子

考察

カメラの必要な動作に台座が干渉することもなく、思い通りに仕上げられた。

2.4.2　フロア

3 章以降で解説するように、ルンバの上に周辺回路を 3 つとバッテリーを 1 つ設置しなければならない。しかし、ルンバの上に直接設置するとカメラの動作に干渉したり、映像に映り込んだりしてしまう問題があるので、2 階建てのようにフロアを作製することにした。

フロアは 1 枚の円形のプレート（図 2-36）と 3 本の足（図 2-37）で作製した。足にねじ穴を切り、ねじで固定する。足はカメラが上を向いても接触しない範囲で可能な限り短く設計した。プレートの中心の穴はカメラと周辺回路を接続するコードを通すために開けた。

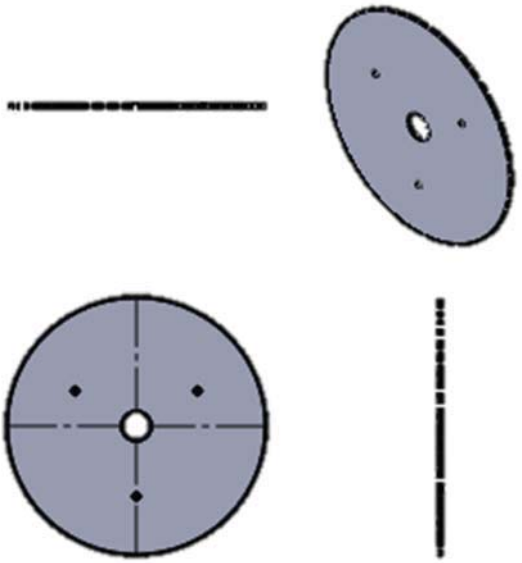

図 2-36 プレート

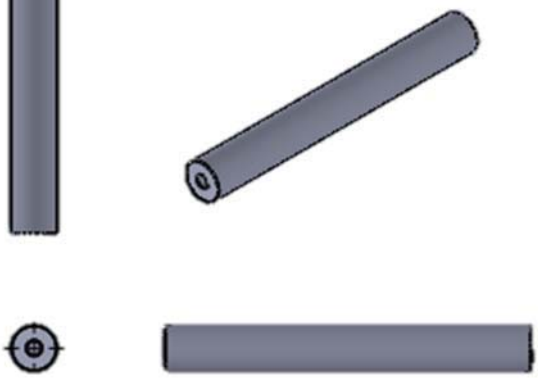

図 2-37 足

フロアはアルミで加工しているため、このままでは周辺回路の導通部分と接触してしまう。そこで周辺回路の下にスペーサーを入れて周辺回路に開いている穴に通し、ナットで締めて固定した。スペーサーはプレートの裏側からねじで固定してある。バッテリーは取り外しができるようにマジックテープで固定した。図 2-38 はフロアに周辺回路とバッテリーを取り付けた様子である。

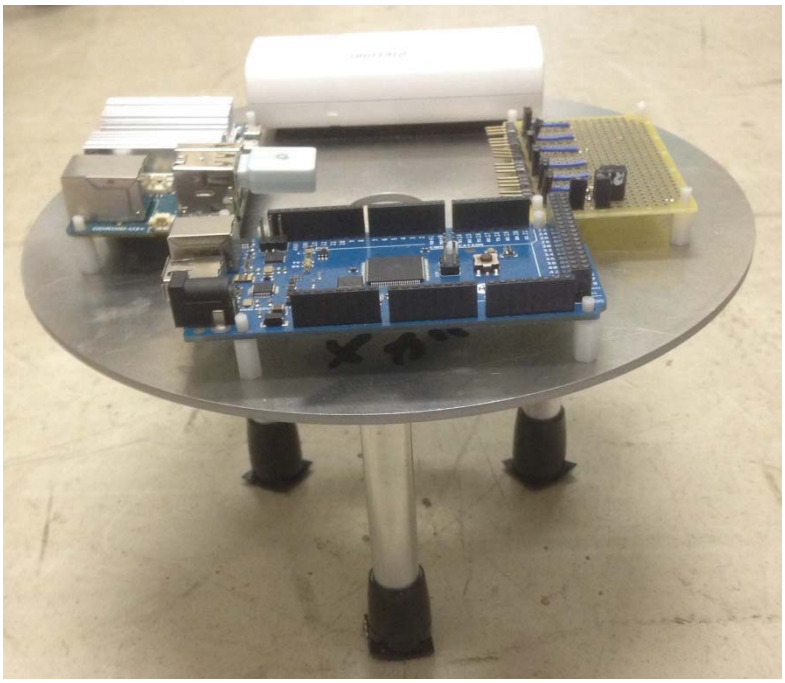

図 2-38 フロアに周辺回路とバッテリーを取り付けた様子

また、フロアの足の先にキャップを付け、ルンバとの固定方法はマジックテープにした。

考察

思い通りの仕上がりにできた。フロアをマジックテープで固定したことによって簡単に取り外しができて、カメラの取り外しや周辺回路の配線をスムーズに行うことができた。

2.5　機構の成果　（高橋担当）

2.5.1　動作確認

図 2-39 は機構を完成させた全体の写真である。なお、この写真では配線は接続していない。これで必要な全ての機能を機構に搭載することができた。モータを上下左右に必要な動作をさせても台座やフロアと干渉することはなかった。

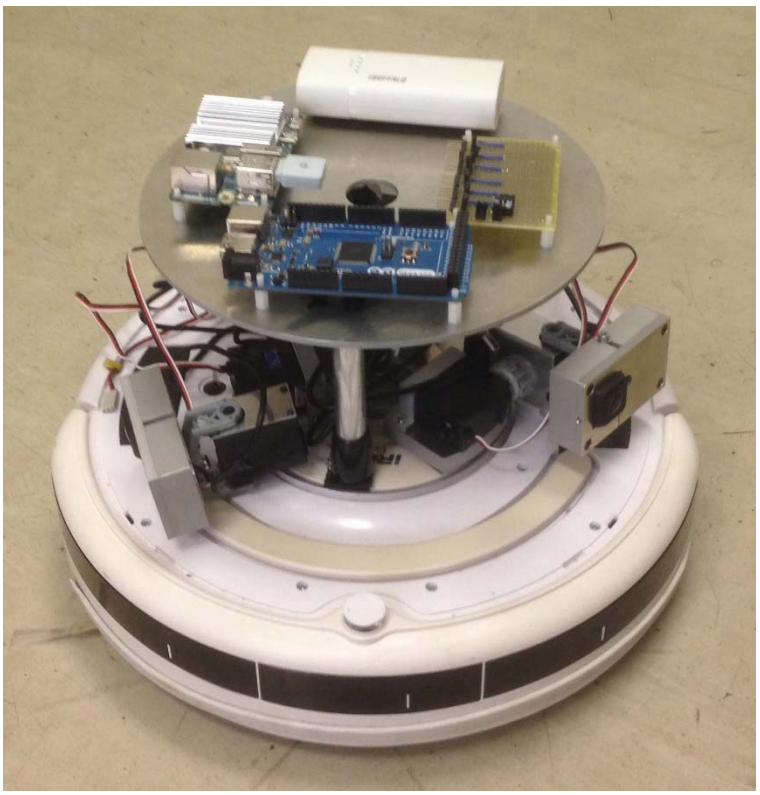

図 2-39 機構の完成写真

2.5.2　考察

機構班としての全ての工程を終了することができた。作製した機構は 2.1.2 の要求動作・制約条件を全て満たすことができた。

第 3 章　　回路とプログラム　（加藤担当）

3.1　　目標　（加藤担当）

3.1.1　　概要

本章では、ルンバとカメラの制御を行うための回路とプログラムについて解説する。

3.1.2　　要求機能

本章での室内見守りシステムを実現するための要求機能を以下に示す。

1. カメラの上下左右方向のモータ動作の制御をすること
2. ルンバの前後動作、右回転左回転動作の制御をすること
3. 基板への回路の実装をすること

3.2　　モータ制御　（加藤担当）

3.2.1　　Arduino について

本研究ではサーボモータとルンバの制御に Arduino というマイコンボードを使用する。大きさはルンバ上部の面積と比べて小さく載せやすく、プログラム初心者でも簡単にプログラムを記述できるという利点がある。試作時には図 3-1 の Arduino UNO を使用していたが、実際にルンバに搭載する際はシリアル通信を 2 系統使うために Arduino MEGA（図 3-2）に変更した。

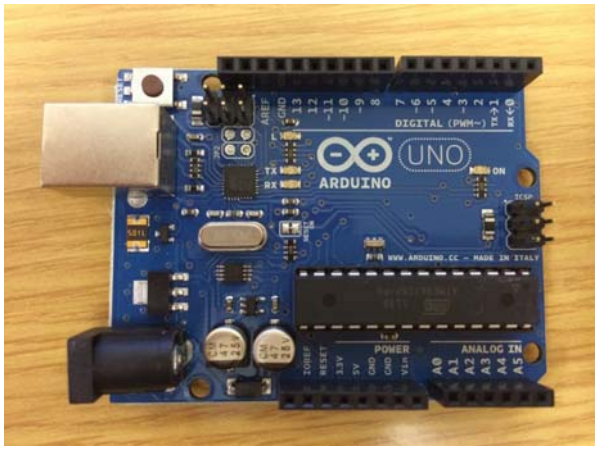

図 3-1 Arduino UNO

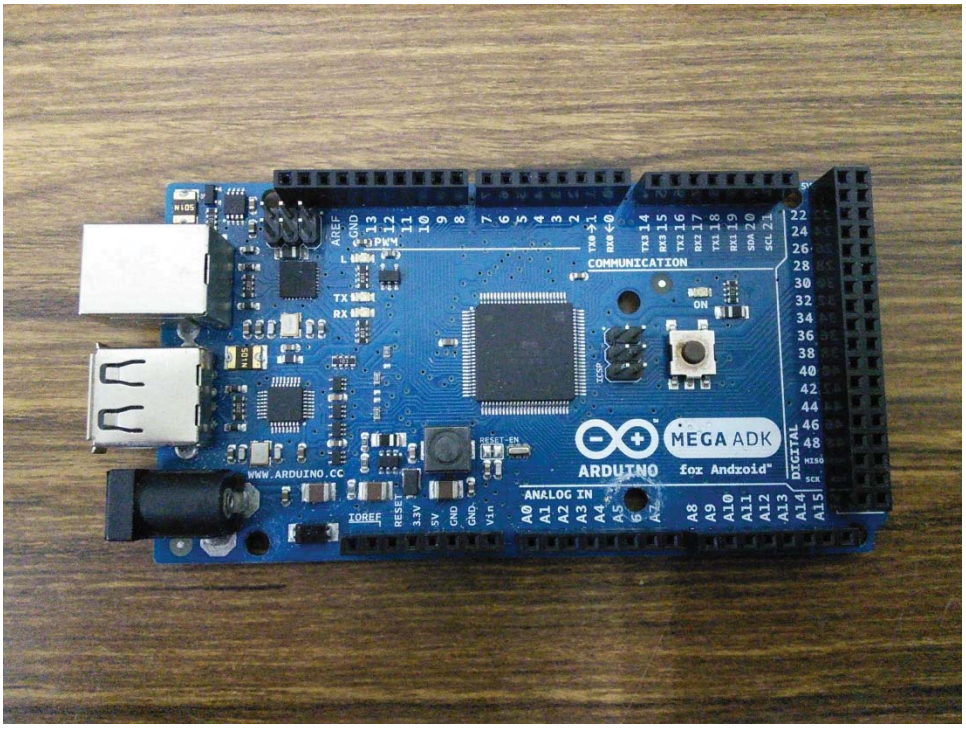

図 3-2 Arduino MEGA

3.2.2　　サーボモータについて

本研究で用いるサーボモータとは、回転角度を指定し制御できるモータである。信号を送ると 0～180°程度の範囲で指定した角度まで回転するので、ロボットの関節部分にしばしば用いられている。5V の電源、GND、そして指令用の 3 ピンを接続して制御する。本研究で用いるサーボモータは図 3-3 に示した近藤科学社の KRS -788HV ICS Red Version で、仕様は最大トルク 10.0kgf・cm、最大動作角度 180°である。図 3-4 は Arduino UNO とサーボモータの簡単な接続例である。

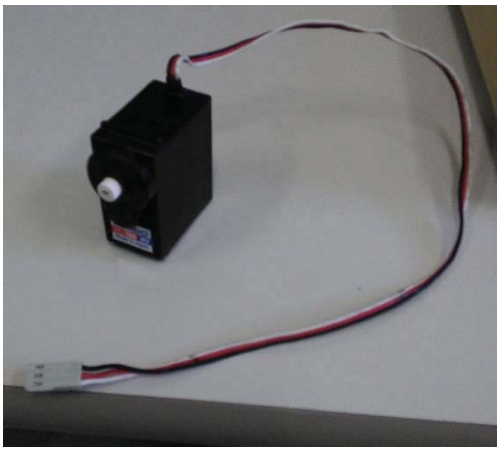

図 3-3 KRS -788HV ICS Red Version

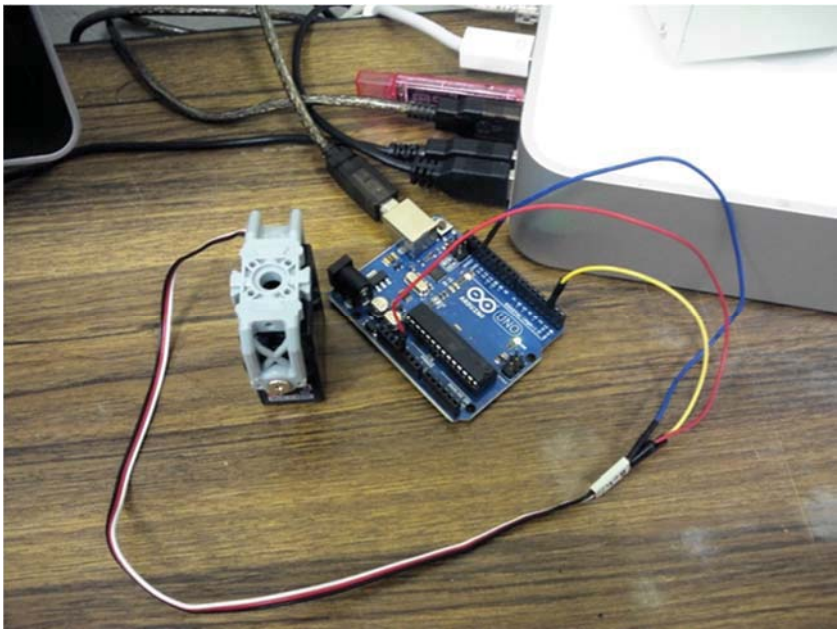

図 3-4 Arduino とサーボモータの接続例

本研究では、カメラの上下左右方向の制御を行うために、カメラ 1 つにつき 2 個のサーボモータが必要である。ルンバに乗せるカメラは 3 つであるため、6 個のサーボモータを動かさなければならない。さらに、パノラマビューを作製するため、3 つのカメラの上下方向の制御を同時に行う機能が必要である。この 2 つの機能を実現するため、「RC サーボモータを Arduino と Processing で GUI 制御（その 1）【Arduino 編】」[3-1]というサイトのプログラムを参考に、6 つの角度を、Arduino に送り、サーボモータ 6 個を制御することにした。このときの通信コマンドのフォーマットは、図 3-5 の通りである。

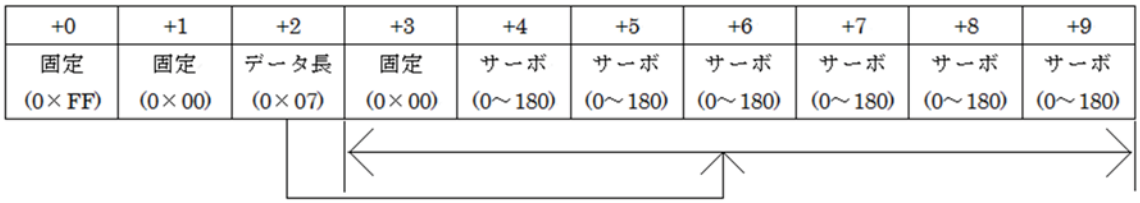

図 3-5　通信コマンドフォーマット

図 3-5 の+0 と+1 の 2 バイトは Arduino との通信のヘッダである。+2 は+3～+9 までのデータのバイト数で、+4～+9 はサーボモータの角度指定に割り当てられている。このときの Arduino のプログラムについて簡単に解説する。下記プログラムの配列「buff」の添字はサーボモータ制御に割り当てられた+4～+9 の数字に対応する。与えられた数字が 0 以上 180 以下であれば、各サーボモータをその角度に動かす。

```
void ServoCmdExec（）
{
  if（(buff[4] >= 0）&&（buff[4] <= 180)）
    servo0.write（buff[4]）；
  if（(buff[5] >= 0）&&（buff[5] <= 180)）
    servo1.write（buff[5]）；
  if（(buff[6] >= 0）&&（buff[6] <= 180)）
    servo2.write（buff[6]）；
  if（(buff[7] >= 0）&&（buff[7] <= 180)）
    servo3.write（buff[7]）；
  if（(buff[8] >= 0）&&（buff[8] <= 180)）
    servo4.write（buff[8]）；
  if（(buff[9] >= 0）&&（buff[9] <= 180)）
    servo5.write（buff[9]）；
}
```

Arduino へのコマンド送信は PC 上で Processing と呼ばれる言語を用いて行う。
Processing で書かれた下記の部分で、10 バイトの命令を順に送っている。

```
void servoCmd（）｛
  if（!init） return;
  for（int i = 0; i < 10; i++）｛
    myPort.write（(byte）cmd1[i]）；
  }
}
```

実際に Processing から Arduino へコマンドを送り、サーボモータを制御している様子が図 3-6 である。

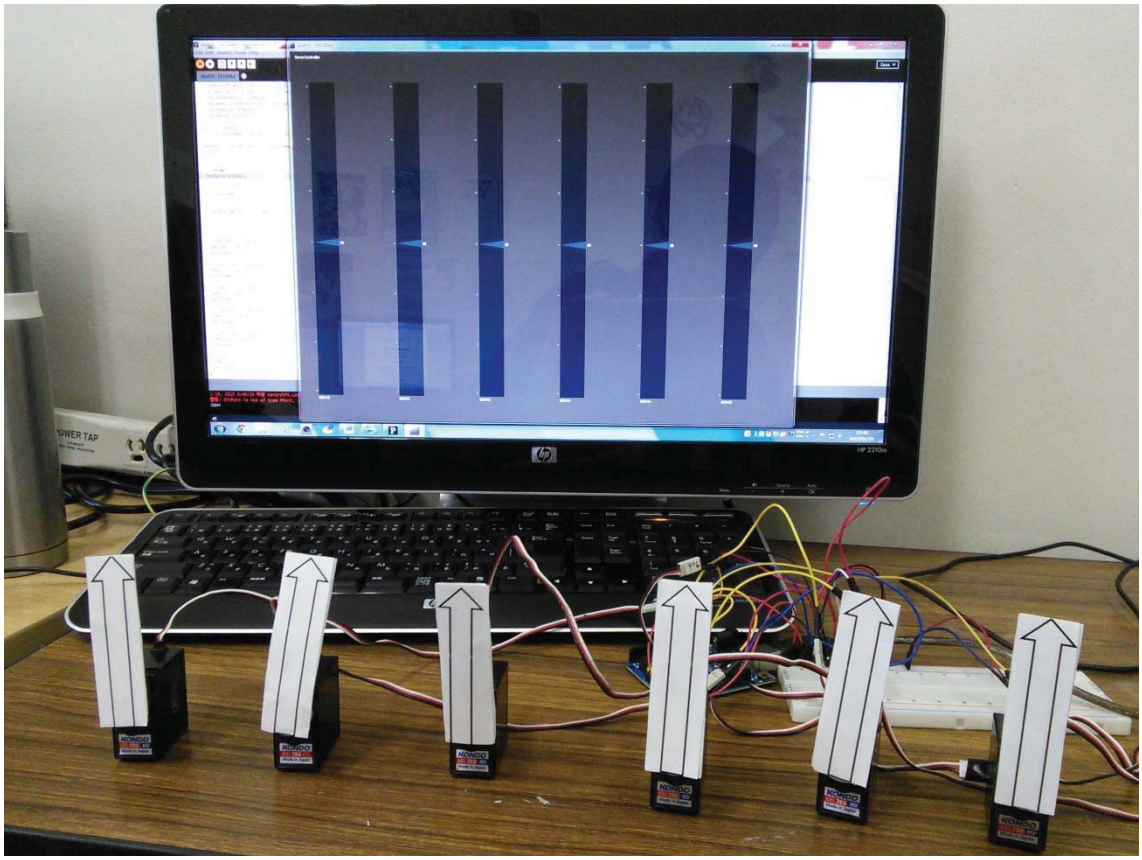

図 3-6　6 個のサーボモータの制御

3.3　　ルンバの制御　（加藤担当）

3.3.1　　ルンバオープンインタフェース

360°のパノラマビューを移動しながら閲覧するため、ルンバを制御する必要がある。ルンバには iRobot 社が公開しているルンバオープンインタフェース（ROI）というものが存在し、外部からシリアル通信でコマンドを送ることでルンバを制御できる。図 3-7 がそのインタフェースに接続するためのコネクタ miniDIN の 7Pin である。

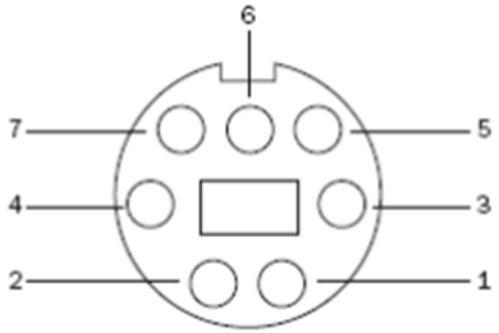

図 3-7 コネクタの形 [3-2]

図 3-8 実際のルンバのコネクタ

3.3.2　Arduino によるルンバの制御

ルンバオープンインタフェースを利用して、ルンバの制御を行うプログラムを「ルンバの ROI を使っての制御Ⅱ（Arduino ソフトウェア編）」というサイト［3-3］を参考に実現した。プログラムへのコマンドは、「Arduino 開発環境」というソフトウェアのシリアルモニタという機能で Arduino に送り、それに応じた指示をルンバに出すというものである。図 3-9 はシリアルモニタの画面で、図 3-10 はルンバ制御の様子である。

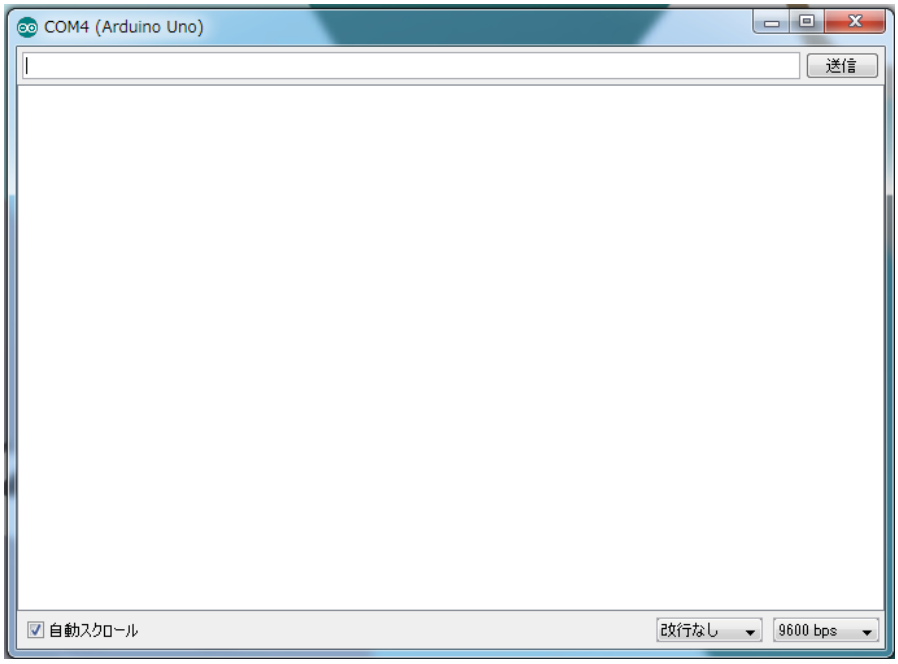

図 3-9 シリアルモニタ

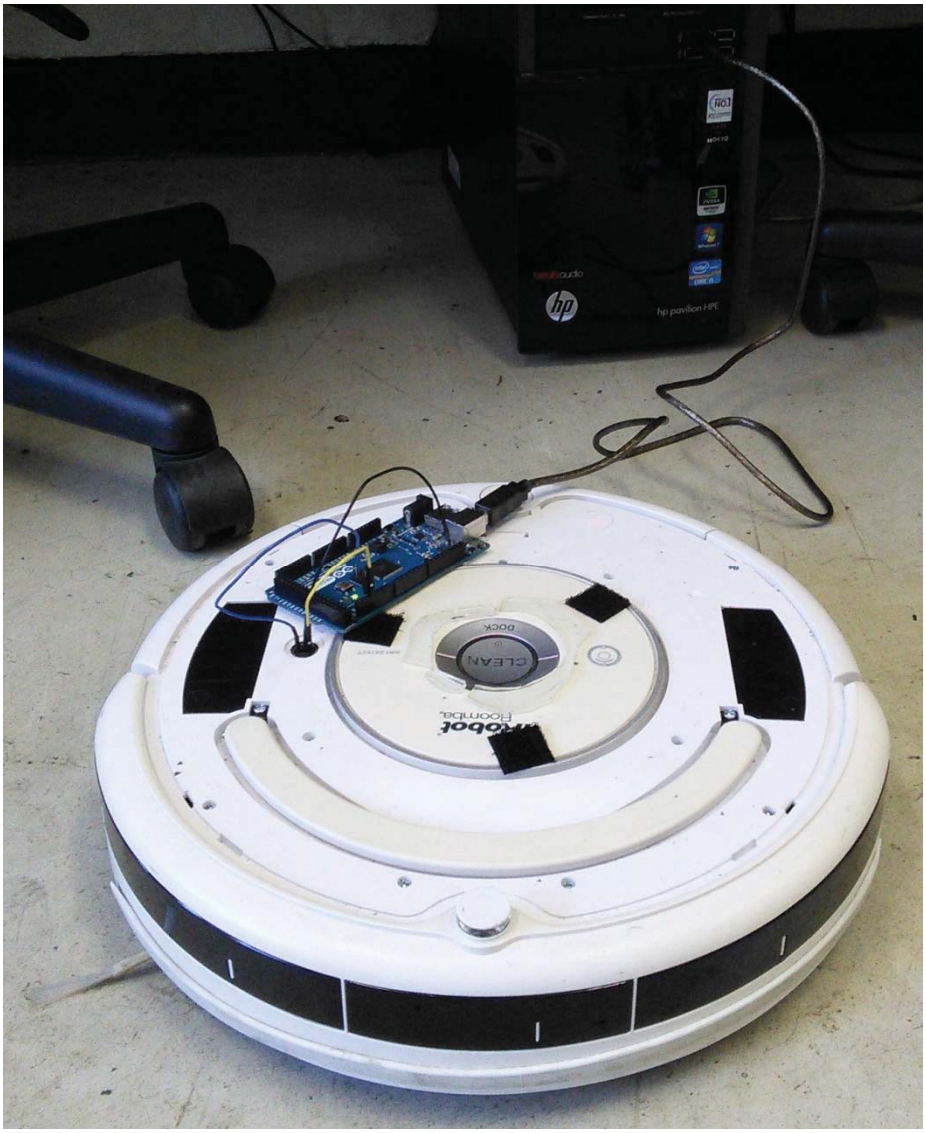

図 3-10 ルンバの制御プログラムのテスト

表 3-1 は、ルンバ制御用のコマンドをまとめたものである。

表 3-1　ルンバ制御コマンド

ルンバの行動	行動コード	指定子	文字数
停止	s		1
前進	w	回転速度-500～+500 一文字目は必ず+が 0 -が 1 とする	5
後退	x	回転速度-500～+500 一文字目は必ず+が 0 -が 1 とする	5
その場右回転	d	回転速度-500～+500 一文字目は必ず+が 0 -が 1 とする	5
その場左回転	a	回転速度-500～+500 一文字目は必ず+が 0 -が 1 とする	5

このコマンドを Arduino のシリアルモニタに入力することでルンバを制御できる。入力例として、「モータの回転速度 200 で前進」のコマンドを入力するときは、「w0200」と入力する。以下にこのときに使用した Arduino のプログラムの簡単な解説を記載する。

```
//停止
case 's':stopRoomba（）；                break;
//前進
case 'w':
                speedNum = serialReadSpeed（）;//速度を受信する
                //Serial.print（"速度："）；
                //Serial.print（speedNum）；
                //Serial.print（"¥n"）；
                foward（speedNum）；       break;
//後退
case 'x':
                speedNum = serialReadSpeed（）;//速度を受信する
                backward（speedNum）；       break;
//その場右旋回
case 'd':
                speedNum = serialReadSpeed（）;//速度を受信する
                turnRight（speedNum）；  break;
//その場左旋回
case 'a':
                speedNum = serialReadSpeed（）;//速度を受信する
                turnLeft（speedNum）；    break;
```

これは表 3-1 の行動コードに対する場合分けを行っている部分である。上記プログラムの「case」と書かれた後ろに行動コードが書かれている。

下記プログラムは、表 3-1 の指定子にあたる速度を取得するプログラムである。下記プログラムの「case 0、case 1、case 2、case 3」がそれぞれ 1 文字目、2 文字目、3 文字目、4 文字目の処理を行う。まず入力された 4 つの文字がそれぞれ 0~9 までの数字であるかどうかを判断する。次に、case 0 で 1 文字目が 0 なら正、1 なら負と定める。そして 2 文字目を case 1 で、3 文字目を case 2 で、4 文字目を case 3 で処理し、速度に変換する。

```
int serialReadSpeed（）
{
    int speedNum = 0 //戻り値となる
        , readByte = 0;
    boolean signFlag;//符号がどっち　true ならプラス、false ならマイナス
    int temp;

    while（ true ）
    {
        if（Serial.available（） > 0）
        {
            temp = Serial.read（）；
            //数字かどうかチェック
            //数字だったら処理する
            if（　（'0' <= temp）　&&　（temp <= '9'）　）
            {
                switch（ readByte ）
                {
                case 0://Serial.print（ temp-'0' ）; Serial.print（"¥n"）；
                        if（ （temp - '0'） == 0 ）　signFlag = true;
                        else signFlag = false;
                        break;
                case 1://Serial.print（ temp-'0' ）; Serial.print（"¥n"）；
                        speedNum +=　（temp-'0'）　* 100;
                        break;
                case 2://Serial.print（ temp-'0' ）; Serial.print（"¥n"）；
                        speedNum +=　（temp-'0'）　* 10;
                        break;
```

```
                case 3://Serial.print（ temp-'0' ）; Serial.print（"¥n"）;
                    speedNum +=  （temp-'0'）  ;
                    break;
                }
                readByte++;
            }
        }
        //4 文字受信したならばループ終了
        if（ readByte >= 4 ）  break;
    }
    //負数だったならば
    if（ !signFlag ）  speedNum *=  （-1 ）;
    //速度上限と下限の設定
    if（ speedNum > 500 ）  speedNum = 500;
    else if（ speedNum <  （-500）  ）  speedNum = -500;

    return speedNum;
}
```

3.4　基板への実装　（加藤担当）

3.4.1　ユニバーサル基板での回路の作製

ここまでで実現した Arduino によるモータおよびルンバ制御に必要な機器をルンバに搭載する。その際、図 3-11 のような配線を行った。

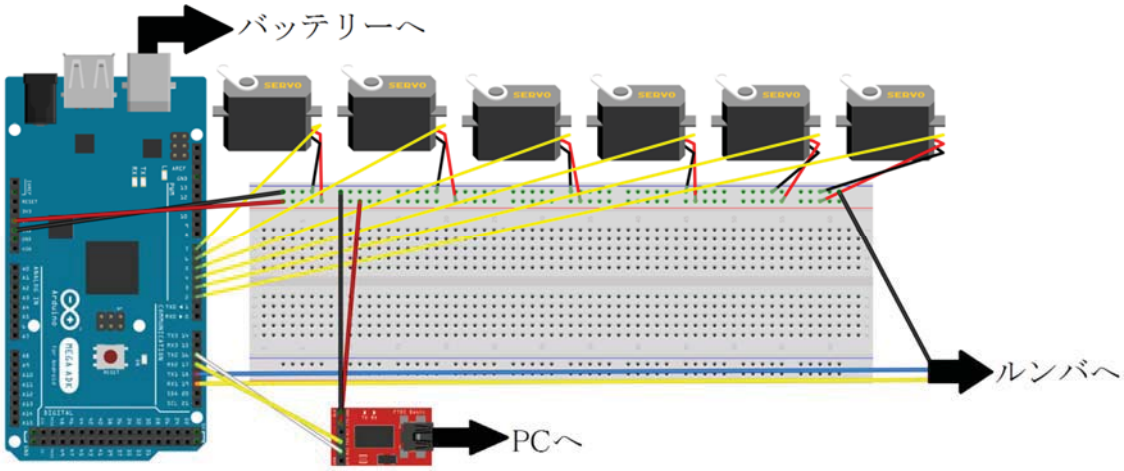

図 3-11 配線図

これをルンバに乗せる際、省スペース化と、配線ミスの防止に配慮する必要があるため、図 3-12 のユニバーサル基板に接続用の回路を作製することにした。すでに機構班が Arduino や ODROID U3 などを乗せるフロアを設計していたため、それに収まるように作製を行う（図 3-13）。完成したプログラムを使用して動作確認を行った（詳しくは 5 章で説明する）。また、完成したプログラムは別紙付録に記載する。

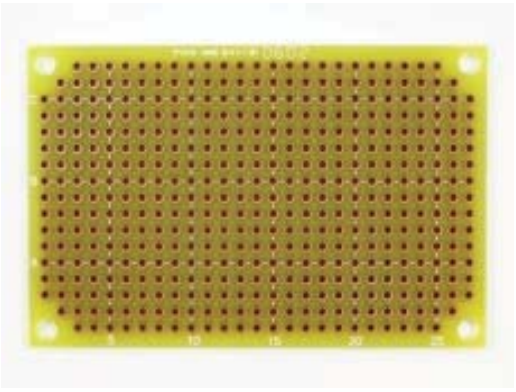

図 3-12 ユニバーサル基板

図 3-13 基板を乗せるフロア

完成した基板が図 3-14 である。

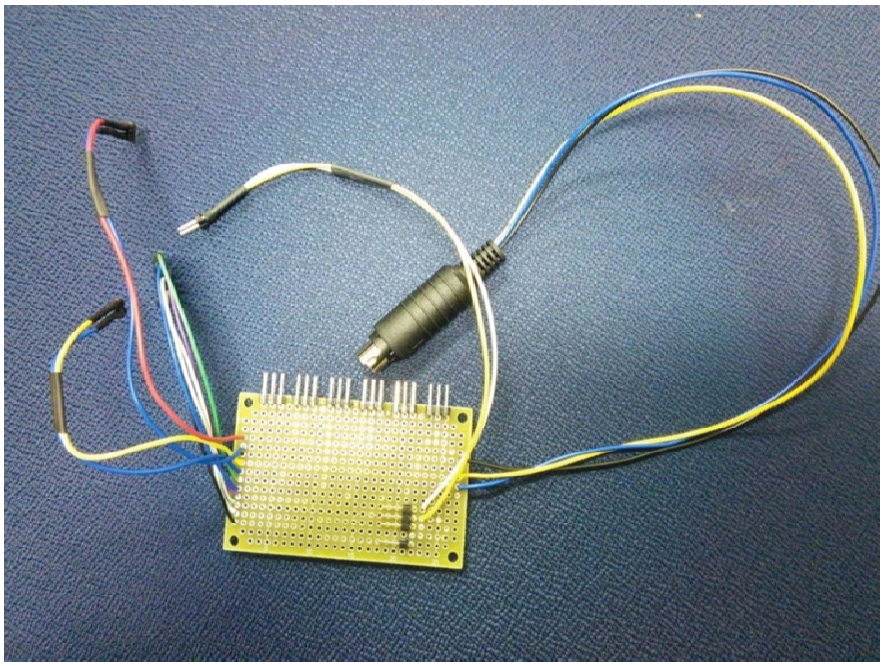

図 3-14 作製した基板

図 3-14 の基板を用いて、実際にルンバ上で接続したものが図 3-15 である。ルンバを遠隔操作するために、PC の代わりに ODROID U3 が搭載されている。ODROID U3 については第 4 章で解説する。

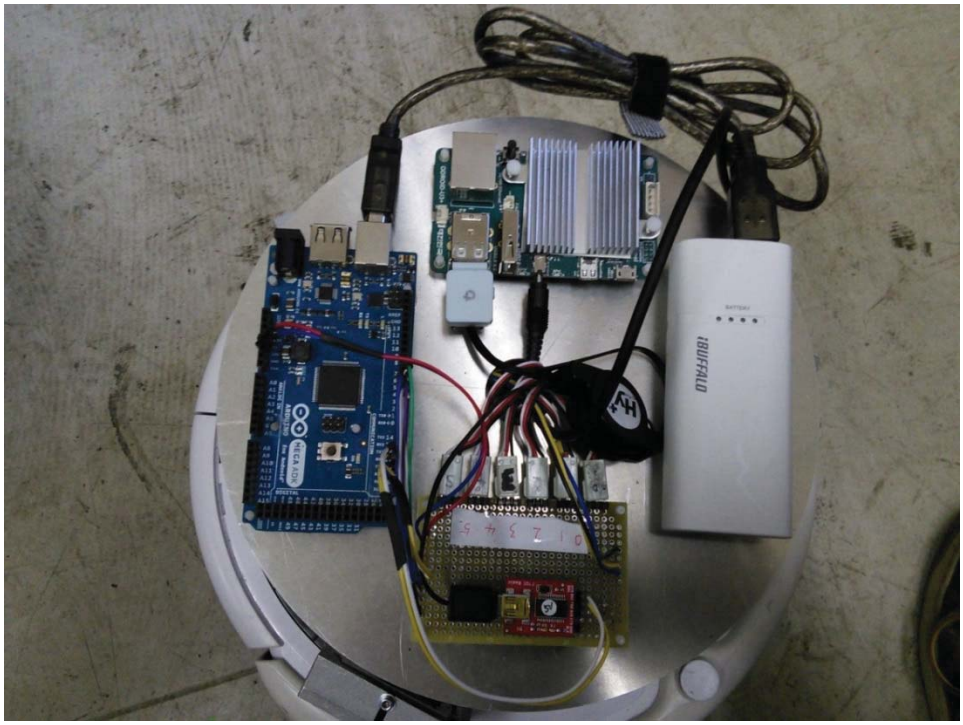

図 3-15 接続時の様子

電源は、Arduino と ODROID U3 の電源にモバイルバッテリーを 1 つ使用した。6 個のサーボモータは、Arduino から電源を供給している。Arduino の使用しているピンとそれに対応する回路を表 3-2 にまとめた。

表 3-2　Arduino ピンと回路対応表

Arduino ピン番号	対応する回路
2~7	サーボモータの命令ピン
16、17	USB シリアル変換モジュールの RX、TX ピン
18、19	ルンバの RX、TX ピン
5V 出力ピン（Arduino）	サーボモータの電源
GND ピン	サーボモータの GND ルンバの GND USB シリアル変換モジュールの GND

第 4 章　　映像と命令の無線化（津田担当）

4.1　　概要　（津田担当）

4.1.1　　目標

本章では映像の無線化と命令の無線化について解説する。本研究ではカメラ搭載のルンバが室内で自由に移動できることを目標に研究を進めている。そのため、カメラからの映像や制御命令を有線接続ではなく、無線で送受信することが必要になる。

4.1.2　　要求機能

本章での室内見守りシステムを実現するための要求機能を以下に示す。

1. 有線接続ではない方法で 360°の映像を見られること
2. 遠隔でルンバを操作できること
3. 遠隔でサーボモータを操作できること

4.2　　映像の無線化　（津田担当）

4.2.1　　通信手段

映像の無線化を行うために、Wi-Fi を介したインターネット回線を用いる。

4.2.2　　映像の無線化の仕組み

図 4-1 が無線化の概要図となる。カメラをシングルボードコンピュータにつなぎ、タブレットやパソコンなどのブラウザでカメラの映像を見る。シングルボードコンピュータとは 1 枚の回路基板に単体のコンピュータとして動作するのに必要なすべての機能・要素が搭載されたものである。産業機器の制御用組み込みコンピュータなどとして利用されることが多い。今回は映像を配信する役割とモータへの指令を Arduino へ伝える役割を担っている。

作製するアプリは OS に左右されず、Windows/Android/iOS を利用できるように HTML5 と JavaScript を用いて記述し、ブラウザを用いて操作を行う。

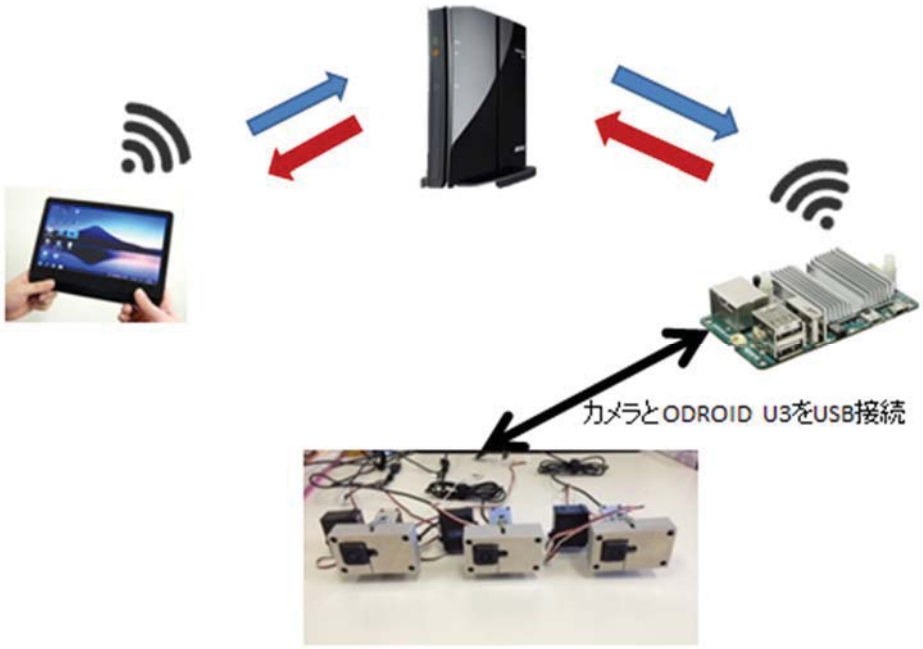

図 4-1 無線化の仕組みの概要図 [4-1] [4-2] [4-3]

4.2.3　　映像配信（MJPEG streamer）の説明

映像を無線で配信するために MJPEG streamer というソフトウェアを用いる。このソフトウェアをシングルボードコンピュータ上で動作させることにより、「http:///パソコンの IP アドレス:8080」をブラウザ上で指定することでカメラの映像を見ることができる。

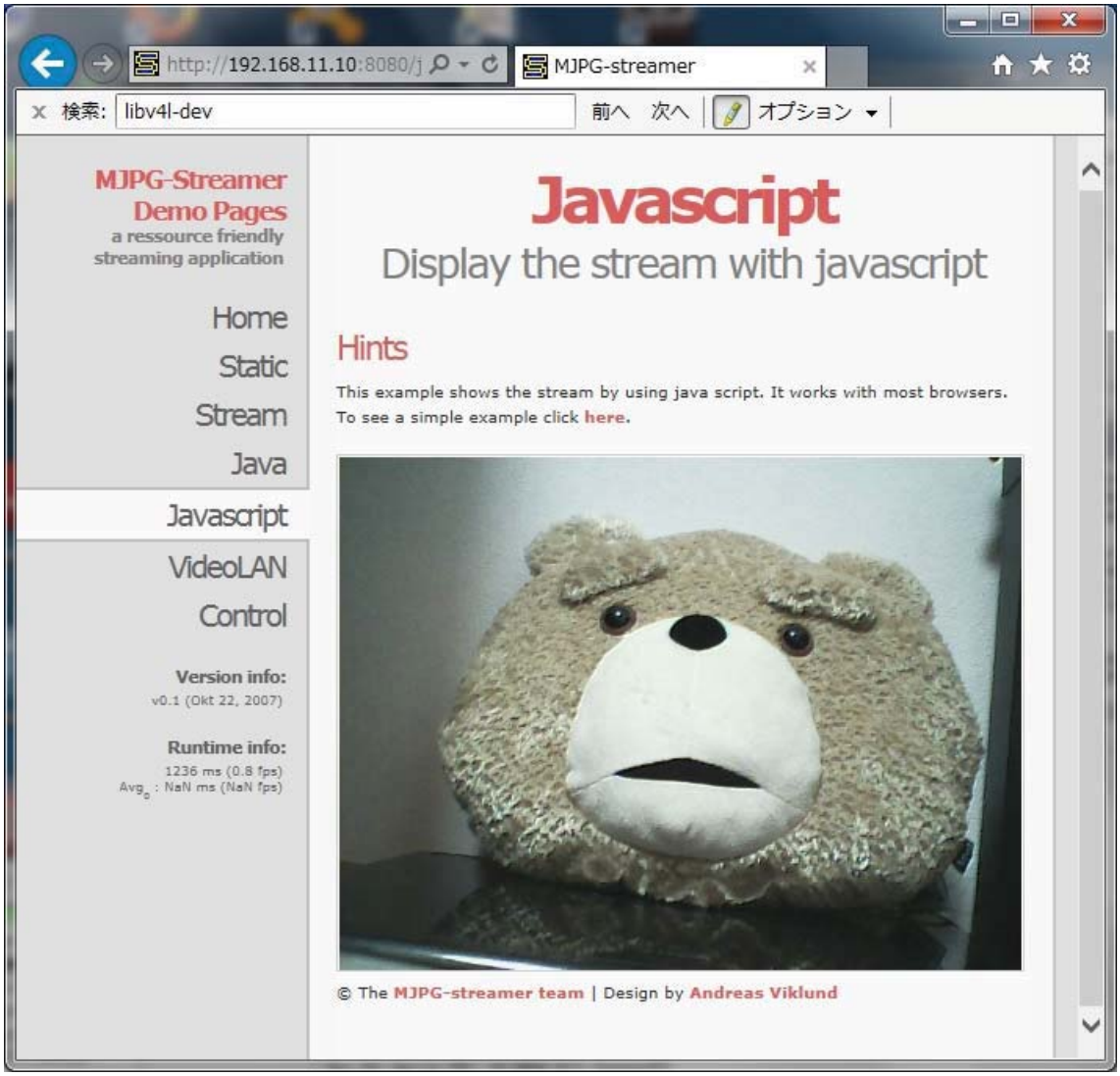

図 4-2 MJPEG streamer のトップ画像[4-4]

4.3　シングルボードコンピュータによる映像配信

（津田担当）

4.3.1　用いたシングルボードコンピュータ

今回映像配信を行うために用いたシングルボードコンピュータを紹介する。

1. PandaBoard
2. BeagleBone Black
3. ODROID U3

カメラ 1 台の際は PandaBoard や BeagleBone Black を用いたが、カメラ 3 台の映像を配信するためには ODROID U3 の計算能力が必要だったので、最終的には ODROID U3 を用いた。

次にそれぞれの特徴を順に紹介する。

4.3.2　PandaBoard による映像配信

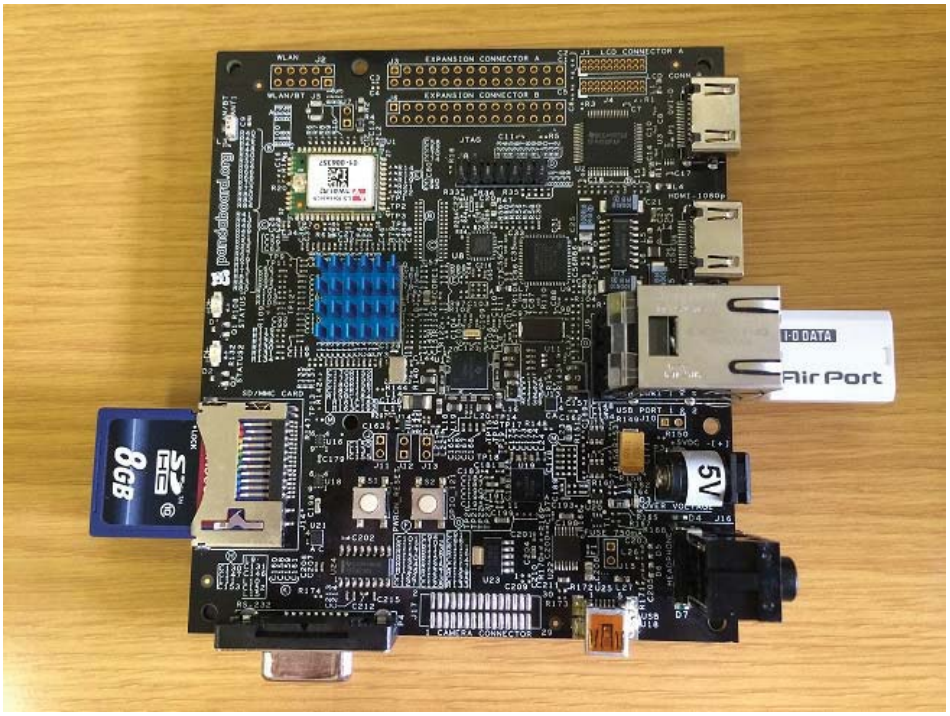

図 4-3 PandaBoard

PandaBoard の仕様は以下の通りである。

表 4-1　PandaBoard のスペック

大きさ	114.3mm×101.6mm
電源	5V
CPU	1.2GHz デュアルコア ARM Cortex-A9 MPCore
GPU	384MHz SGX540
メモリ	LPDDR2 SDRAM 1GB
ストレージ	SD カード（最高 32GB）
無線 LAN	802.11 b/g/n
USB ホスト	USB 2.0 ホスト 2 個、USB On-The-Go 2.0　1 個

PandaBoard には SD カードがささっており、これに MJPEG streamer がインストールされている。また、Wi-Fi への接続には基板上の Wi-Fi デバイスを用いず、IO DATA 社製の無線 LAN アダプタ（型番 WN-G150U）を用いる。

図 4-4 IO DATA 社製の無線 LAN アダプタ（型番 WN-G150U）[4-5]

実際に PandaBoard と作製したカメラとモバイルバッテリーを繋いでルンバの上に配置をした様子が図 4-5 である。この時点では第 2 章の第 2 カメラを用いた。

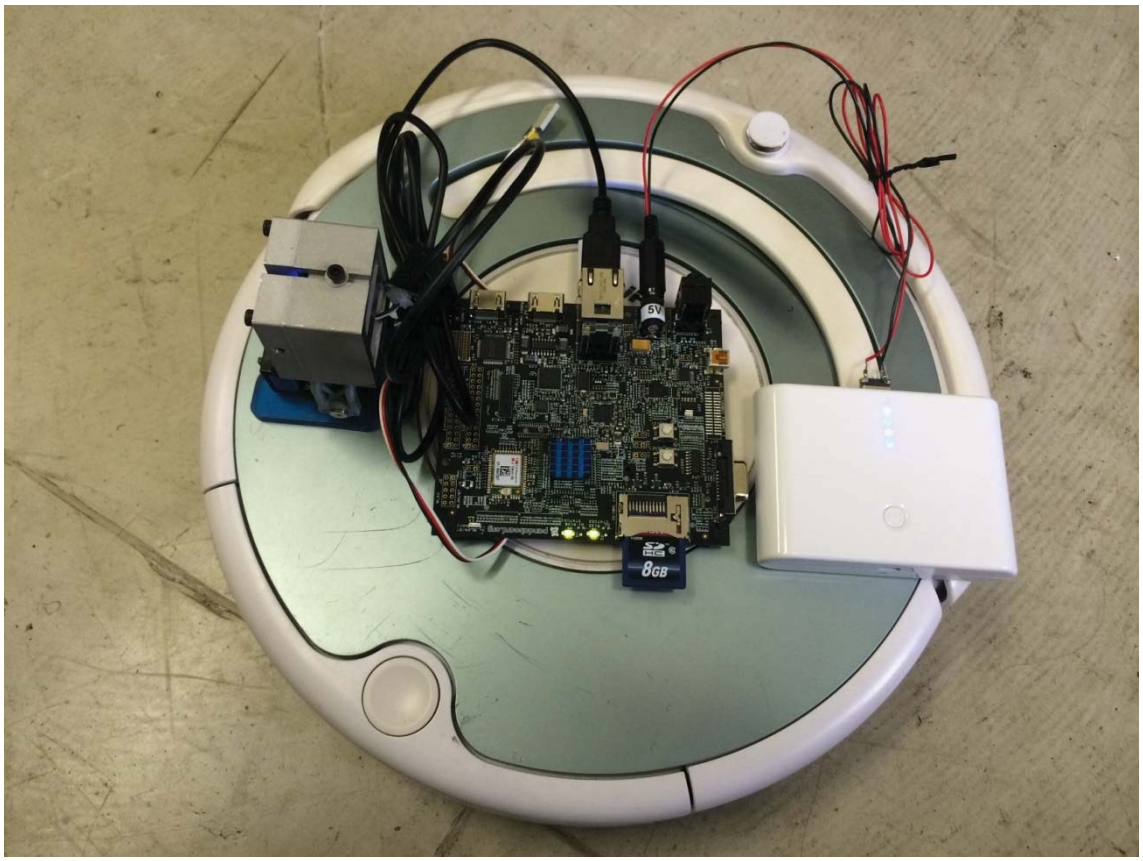

図 4-5 PandaBoard を用いた映像取得の様子

4.3.3　PandaBoard での映像の無線化の検証

実際に PandaBoard で映像の無線化の検証を行った。図 4-6 のようにボールを何度も往復させたところ、ブラウザ上では映像のラグや詰まりがあることを確認できた。

これを改善するため、無線 LAN デバイスとシングルボードコンピュータを変更することにした。

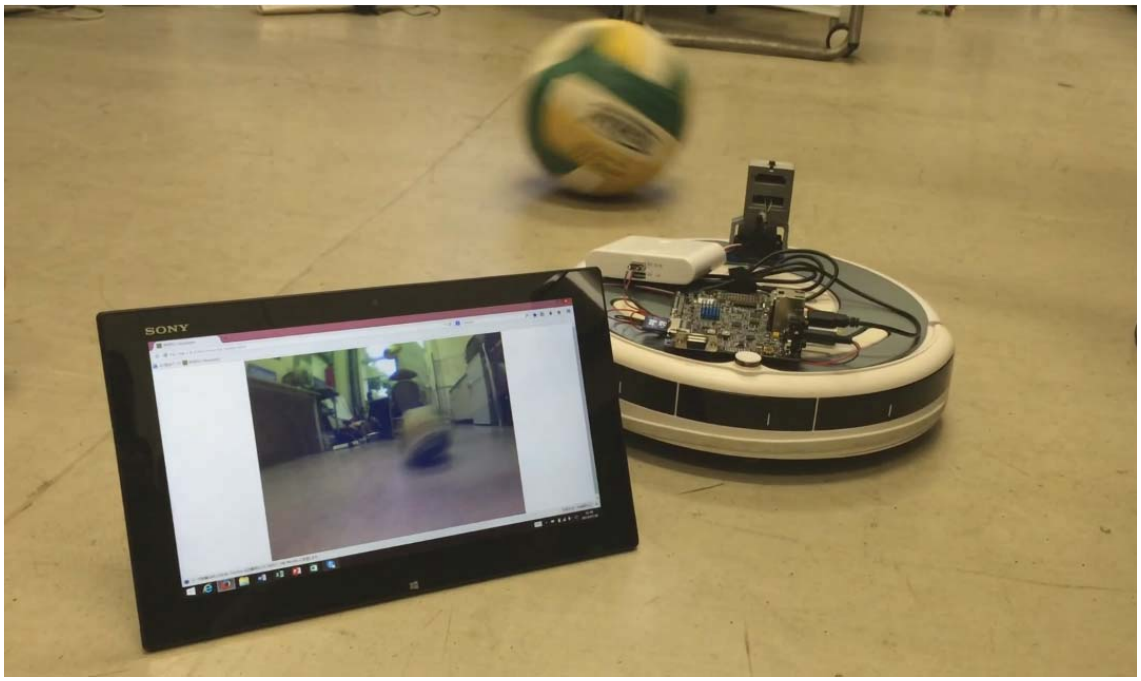

図 4-6 PandaBoard を用いた映像検証

4.3.4　BeagleBone Black による映像配信

前回の検証で PandaBoard では映像のラグや詰まりがあったので、BeagleBone Black というシングルボードコンピュータに変更する。

図 4-7 BeagleBone Black

BeagleBone Black の仕様は以下の通りである。

表 4-2　BeagleBone Black のスペック

大きさ	86.4mm×53.4mm
電源	ミニ USB または DC ジャック
CPU	Sitara AM3359AZCZ100, 1GHz, 2000 MIPS（Cortex-A8）
GPU	SGX530 3d, 20M Polygons/S
メモリ	DDR3L 512MB
ストレージ	eMMC 2GB + microSD
無線 LAN	RJ45x1 （100Mbps）
USB ホスト	USB ホスト 2 個、Mini USB クライアント 1 個 10/100 Mbps Ethernet　1 個

また、シングルボードコンピュータの変更と同時に、無線 LAN アダプタも変更した。

図 4- 8 プラネックスコミュニケーションズ社製の無線 LAN アダプタ（型番 GW-450D）[4-6]

図 4-8 に示したプラネックスコミュニケーションズ社製の無線 LAN アダプタ（型番 GW-450D）を用いる。前節で用いたものは通信速度が 150MBps に対し、このアダプタは 5GHz 帯のネットワークに接続でき、通信速度も 433MBps と非常に速い。前節までに用いたものと比べて、約 3 倍の通信速度なのでスムーズに映像配信ができるのではないかと考えた。

4.3.5　BeagleBone Black での映像の無線化の検証

BeagleBone Black で映像の無線化の検証を行った。なお、ここから第 2 章の第 3 カメラを使用している。

前回と同様にボールを何度も往復させたところ、映像の詰まりやラグはなく、スムーズに映像が映ることを確認できた。

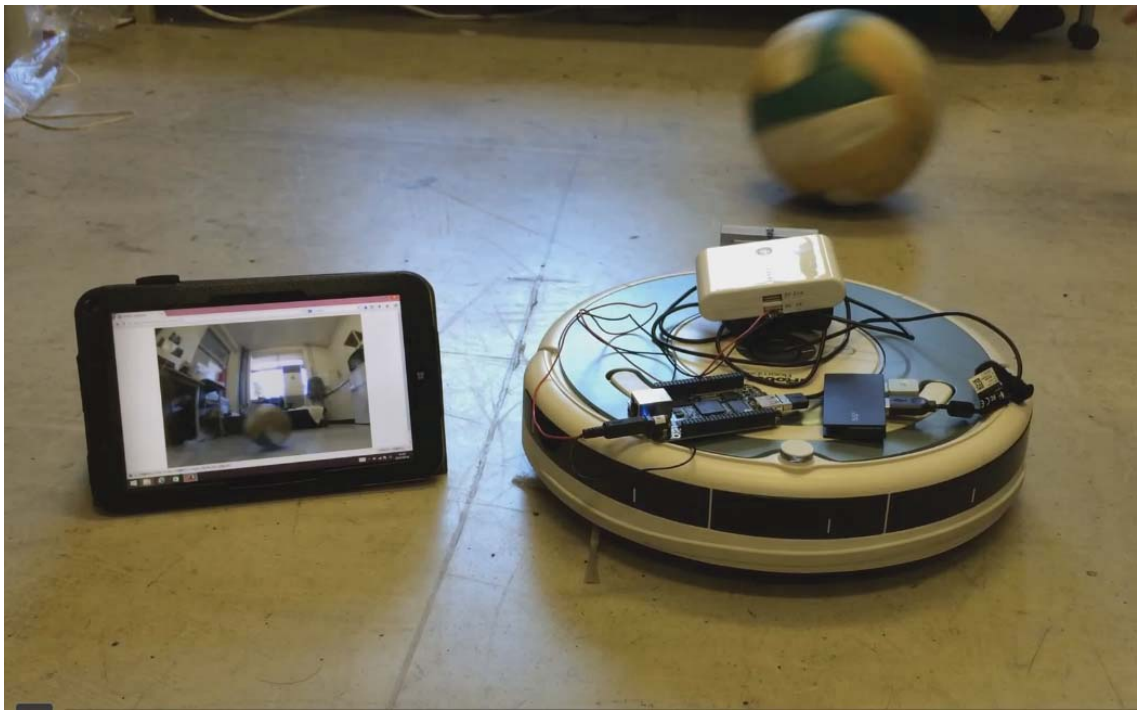

図 4-9 BeagleBone Black を用いた映像検証

検証が終了した後に、カメラ 3 台を搭載してパノラマビューを実現しようと試みた。しかし、BeagleBone Black の計算能力が足りず、3 つの映像を映すことはできなかった。

4.3.6　ODROID U3 による映像配信

4.3.5 節に用いた BeagleBone Black ではカメラ 1 台の映像はスムーズに見ることができたが、カメラ 3 台では計算能力が足りず映像を見ることができなかった。ここで新たなシングルボードコンピュータ、ODROID U3 を用いる。BeagleBone Black がシングルコアに比べこれはクアッドコアなので処理能力が格段に上昇し、カメラ 3 台の映像が配信可能になるではないかと考えた。

図 4-10 ODROID U3

仕様は以下の通りである。

表 4-3　ODROID U3 のスペック

大きさ	48mm × 83mm
電源	5V 2A
CPU	サムスン電子 4412 Prime Cortex-A9 Quad Core 1.7Ghz
GPU	Mali-400 4 コア 440MHz
メモリ	2GB LPDDR2 880MHz
ストレージ	microSD メモリーカード eMMC
USB ホスト	USB2.0 ホスト　3 個

ここでも同様に無線 LAN アダプタ GW-450D を用いる。

4.3.7　　ODROID U3 を用いた映像の無線化の検証

今回は ODROID U3 を用いてカメラ 3 台の映像配信を行った。前回と同様に、ボールを何度も往復させたところ、映像の詰まりやラグはなく、スムーズにカメラ 3 台の映像が映ることがわかった。

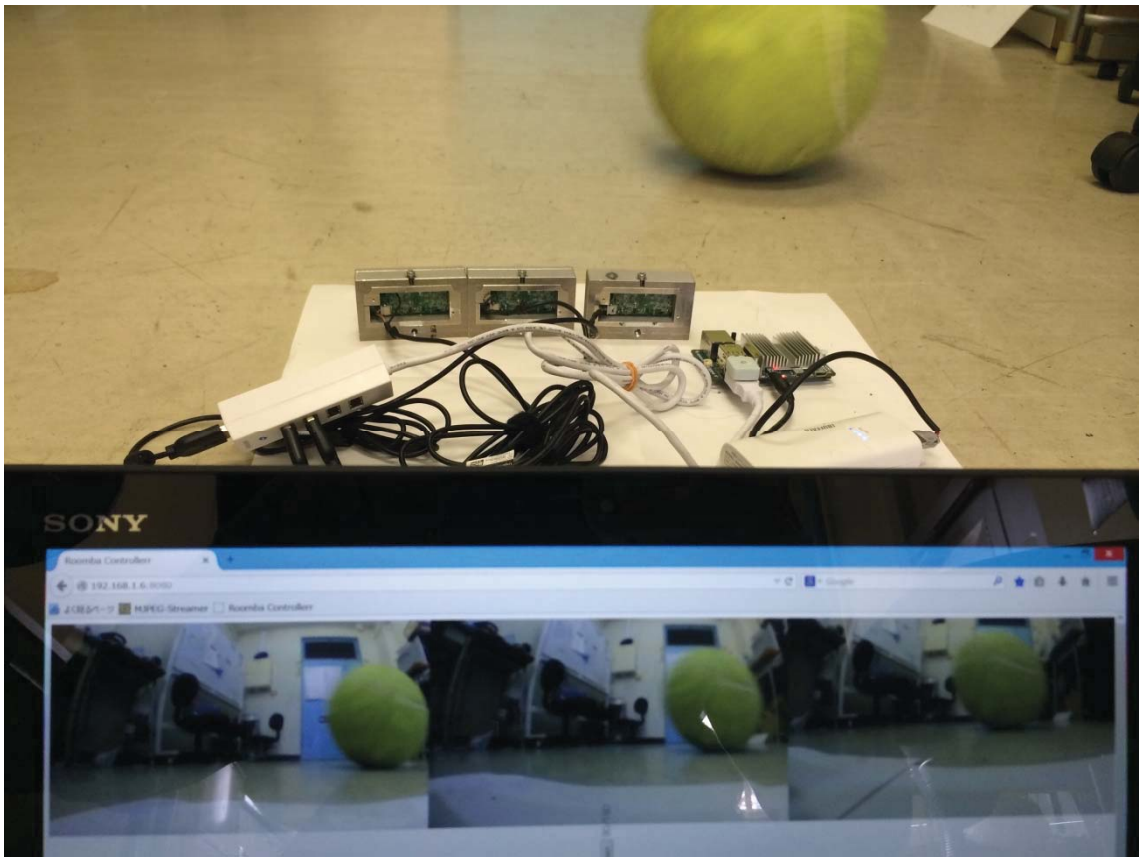

図 4-11 ODROID U3 を用いた映像検証

この検証を踏まえたうえで、パノラマビューを実現する。

4.4　パノラマビュー　（原田担当）

4.4.1　パノラマビューの実現方法

我々が用いる広角レンズは、水平方向の画角が 133.6°であるため、図 4-12 のように 3 台のカメラを円周上に配置することで 360°見渡すことができる。3 台のカメラからの映像を図 4-13 のようにタブレットのブラウザに並べて配置する。

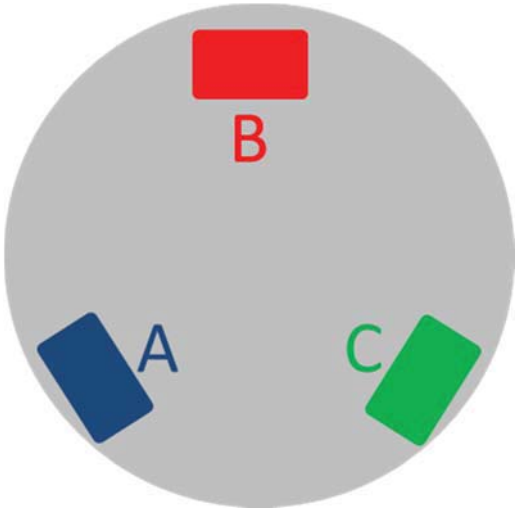

図 4-12　3 台のカメラを円周上に配置するイメージ図

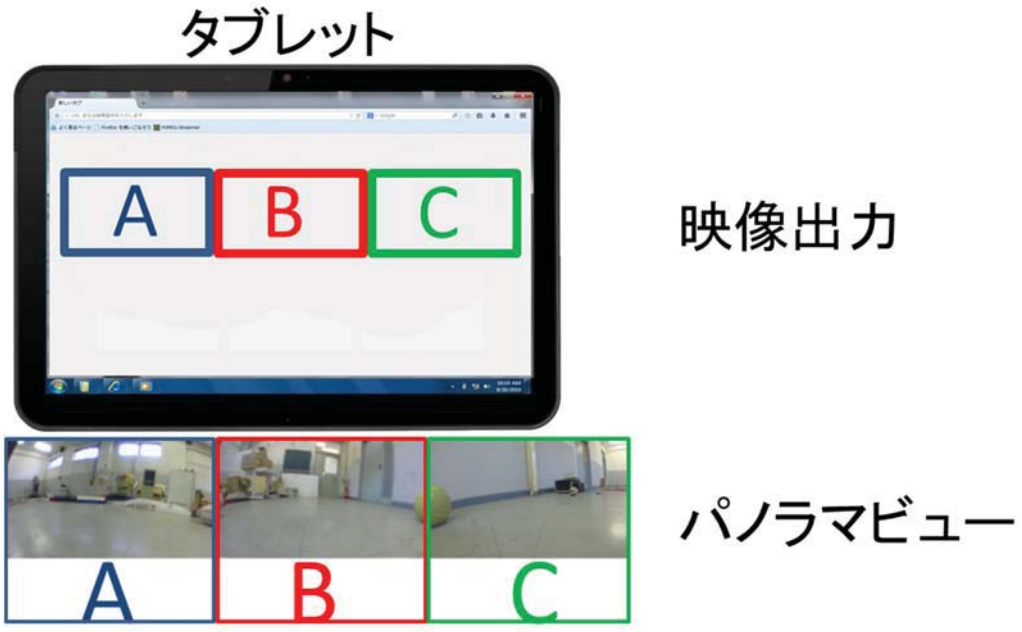

図 4-13　パノラマビューの映像出力

4.4.2　カメラ台数の選定

パノラマビューの作製にあたり、実際にカメラ 3 台で 360°映せるか検証する必要がある。検証には第 2 章で作製した第 2 カメラ、第 3 カメラを用いた。検証するために 1 台のカメラをルンバに乗せローカルでタブレットに USB 接続する。タブレットに映し出された映像をスクリーンショットで画像として保存する。一回ごとに位置をずらし 360°の範囲をカバーできているか確認を行った。

4.4.3　試作パノラマビューの作製

実際のパノラマビューは動画だが試作パノラマビューでは静止画像で検証を行う。我々が実現したいパノラマビュー同様に静止画像を横並びにして検証する。

4.4.4　　試作パノラマビューの作製結果

第 2 カメラと第 3 カメラの画像を手動で横に並べた試作パノラマビューが以下の図 4-14 と図 4-15 である。

図 4-14 第 2 カメラでの作製結果

図 4-15 第 3 カメラでの作製結果

4.4.5　　試作パノラマビューの考察

第 2 カメラでは、360 ° 映すために 4 枚の画像が必要であった。これは第 2 カメラに用いた WEB カメラのセンサが 1/5 インチであり、広角レンズの範囲を全て映し出すことができていなかったからである。一方、第 3 カメラでは、3 枚で 360° 見る事ができた。第 3 カメラでは、WEB カメラのセンサは 1/4 インチであったためである。結果、第 3 カメラを 3 台使用することで 360° のパノラマビューが可能であることが分かった。

4.4.6　　カメラ 3 台によるブラウザでのパノラマビューの作製

我々が実現したいパノラマビューはブラウザ上にパノラマビューの映像を出力することで実現される。ブラウザ上にパノラマビューの映像を出力するために MJPEG streamer（4.2.3 節を参照）を 3 台のカメラ分だけ起動した。

4.4.7　　カメラの接続方式

図 4-16 がカメラの接続方式の概要図である。ODOROID U3 とカメラの接続は USB 接続である。ODROID U3 の USB ポート数は 3 個であり、内 1 個を無線 LAN アダプタに使用している。このままだと 3 台のカメラと ODROID U3 が接続できないため ODROID U3 と 3 台のカメラは USB ハブを経由して接続する。

図 4-16 カメラの接続方式の図 [4-1] [4-7]

4.4.8　ブラウザでのパノラマビュー

3 台のカメラの映像をブラウザに出力した様子が図 4-17 である。パノラマビューが実現できていることが確認できた。

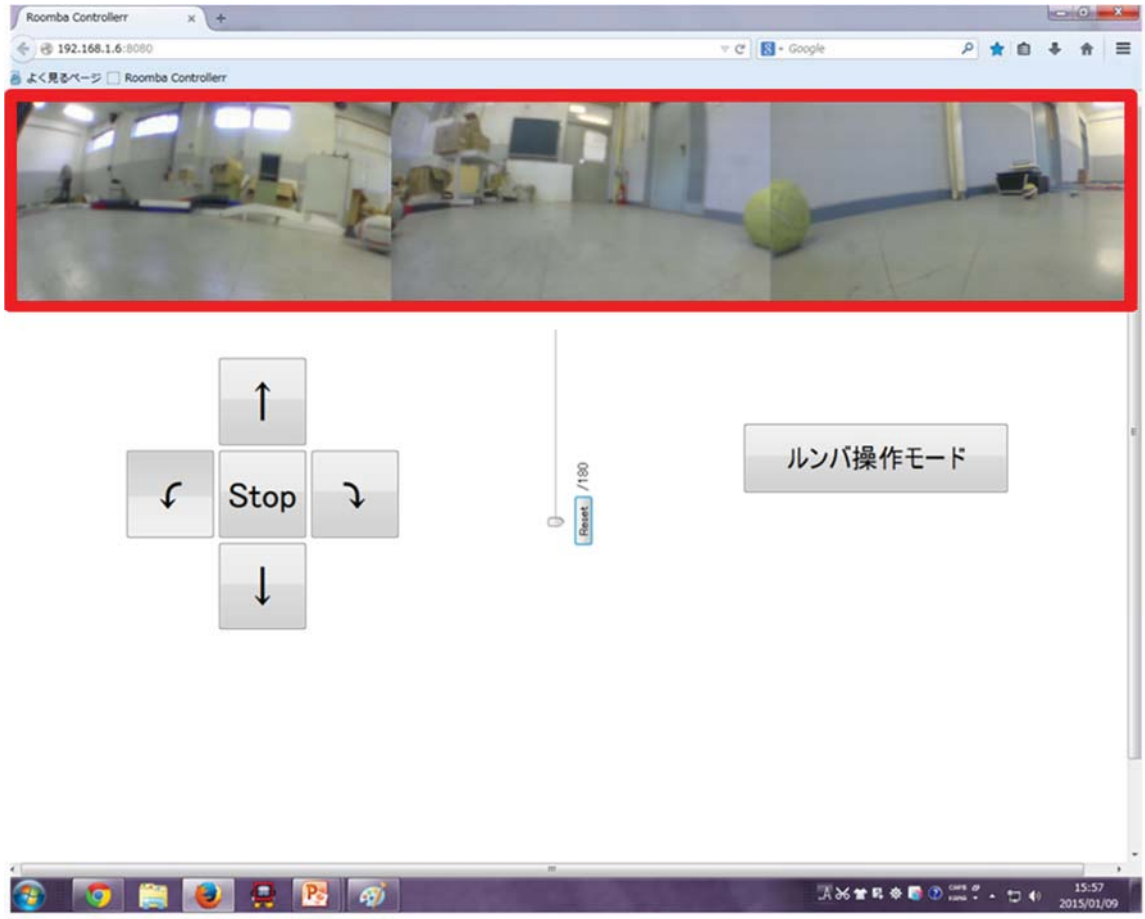

図 4-17 撮影したパノラマビュー

4.5　命令の無線化　（津田担当）

4.5.1　有線での映像配信と命令の構造

映像の無線化は実現できたので、本節では命令の無線化を行う。無線化を行う以前の有線接続での概要図は以下のようになる（図 4-18）。

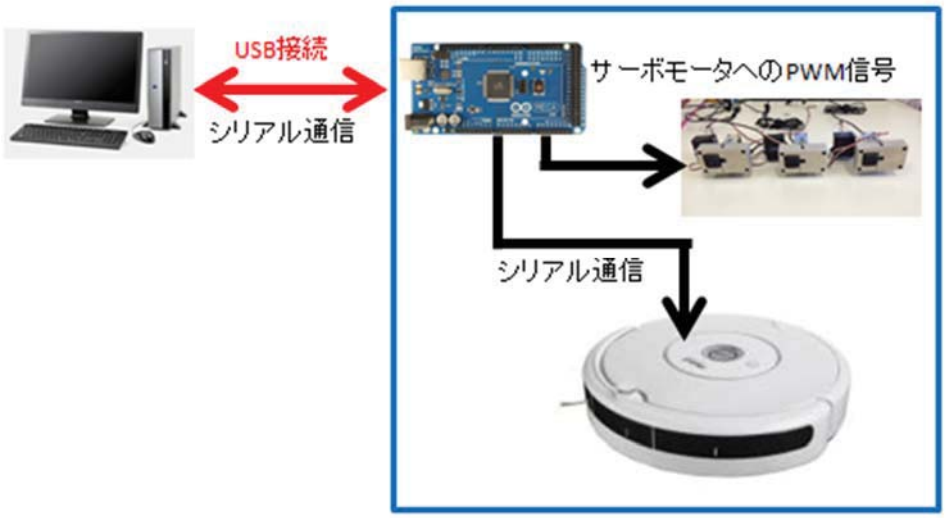

図 4-18 有線接続による映像配信と命令の概要図 [4-8] [4-9] [4-10]

しかしながらこのままでは、パソコンからルンバへ有線命令を送るためルンバの行動範囲が制限されてしまう。そのために図 4-18 の赤で記されたパソコンと Arduino 間の USB 接続の無線化を行う必要がある。

4.5.2　　命令の無線化の構造

図 4-19 が無線接続の概要図になる。Windows タブレットから Wi-Fi ルーターを経由し、無線で ODROID U3 に命令を送り、Arduino からそれぞれルンバとモータに命令を送る。

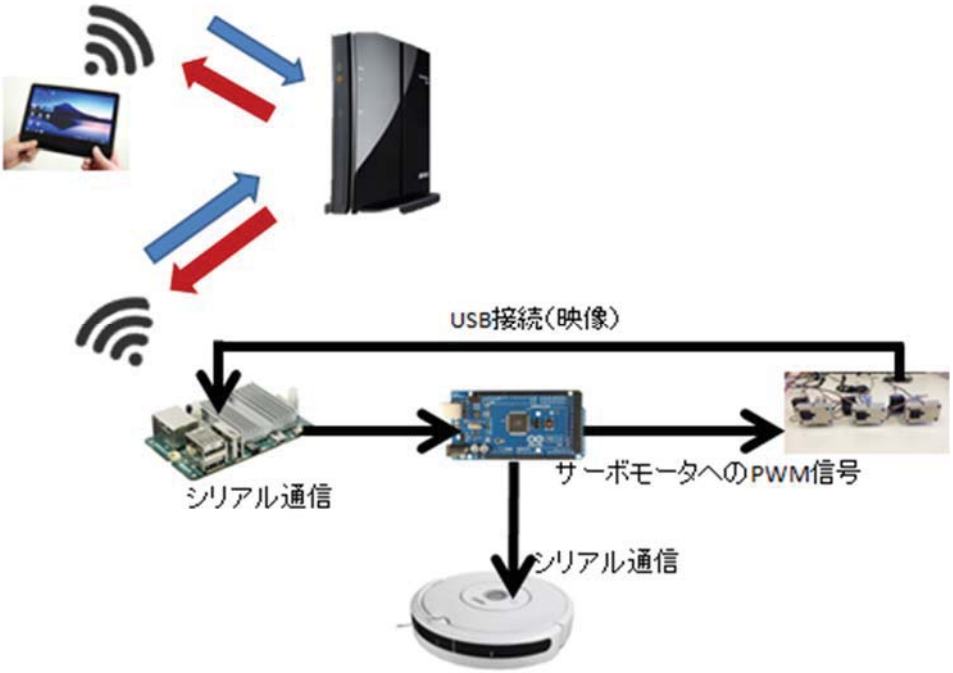

図 4-19 無線による命令の概要図

この概要図を実現することにより、ルンバの行動範囲が広がる。つまり、インターネットの環境があればどこからでもルンバを自由に操ることが可能になる。

4.5.3　操作用のインターフェースの概要

図 4-20 は操作用のタブレットの画面である。この画面の上部がカメラ 3 台の映像で、下部がルンバのコントロール、サーボモータの上下駆動、ルンバ操作モードから構成されている。

コントロール用ボタンは前進、左回転、右回転、後退、停止からなる。移動用のボタンを押したらストップボタンを押すまで動き続ける。サーボモータは、15°刻みで調節することができる。ルンバ操作モードボタンは、ルンバの車輪が浮いて操作が不能になったとき、再び操作可能にするためのリセットボタンである。

次のページでは HTML によるボタンの記述と、コマンドの内容を詳しく解説する。また、ページ全体のプログラムは別紙付録に記載する。

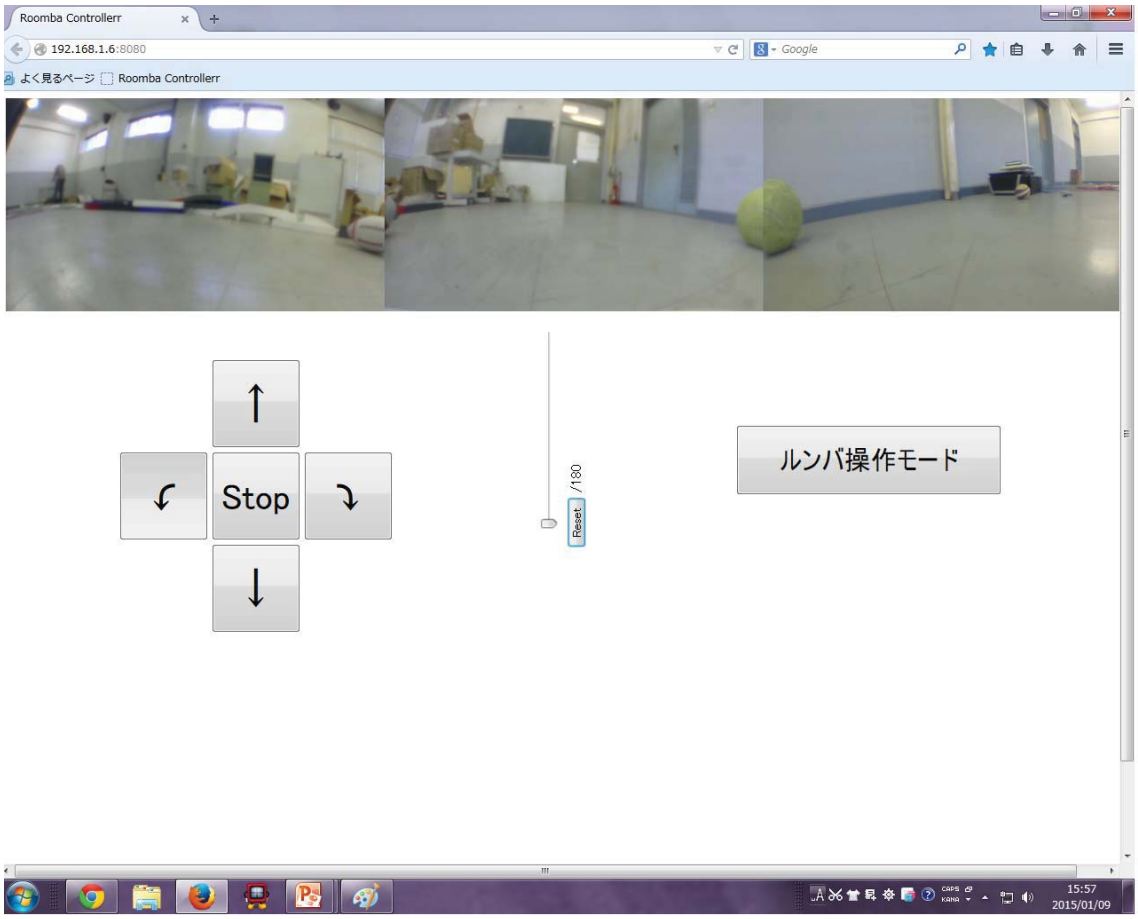

図 4-20 ブラウザのインターフェース

4.5.4　HTML のインターフェースのボタンのデザインと詳細

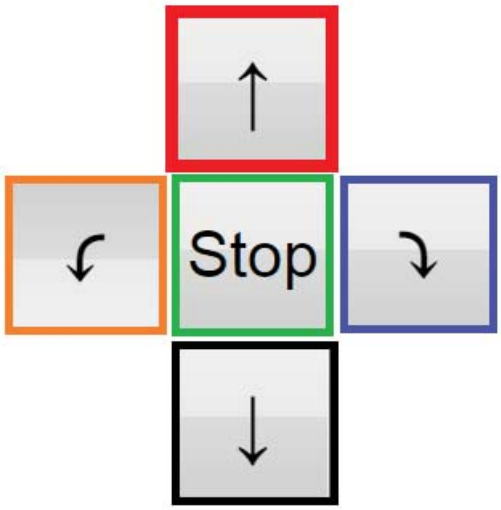

図 4-21 ルンバの操作ボタン

図 4-21 の赤色の枠はルンバの前進用のボタンである。以下のタグで実現する。

<input type="button" value="↑" onclick="roombaForward（）" style="WIDTH: 100px; HEIGHT: 100px ;font-size:50px;"/>

図 4-21 のオレンジ色の枠はルンバの左回転用のボタンである。以下のタグで実現する。

<input type="button" value="⤴" onclick="roombaLeft（）" style="WIDTH: 100px; HEIGHT: 100px ; font-size:50px; " />

図 4-21 の緑色の枠はルンバの停止用のボタンである。以下のタグで実現する。

<input type="button" value="Stop" onclick="roombaStop（）" style="WIDTH: 100px; HEIGHT: 100px ;font-size:40px;" />

図 4-21 の青色の枠はルンバの右回転用のボタンである。以下のタグで実現する。

<input type="button" value="⤵" onclick="roombaRight（）" style="WIDTH: 100px; HEIGHT: 100px ; font-size:50px;" />

図 4-21 の黒色の枠はルンバの後退用のボタンである。以下のタグで実現する。

<input type="button" value="↓" onclick="roombaBackward（）" style="WIDTH: 100px; HEIGHT: 100px ; font-size:50px; " />

図 4-22 サーボモータ操作用のスライダー

図 4-22 はサーボモータを上に向かせるスライダーである。初期位置はニュートラル位置の 90 に設定してある。また、リセットを押すと初期位置である 90、つまり正面に戻る。以下のタグで実現する。

```
<div style="margin-left:-15%;margin-right:40%;">
<form onsubmit="return false" oninput="level.value = flevel.valueAsNumber">
<input name="flevel" id="flying" type="range" oninput="servoMove（this.value）"
onchange="servoMove（this.value）" style="width:30%;padding:0px;font-size:30px;"
min="90" max="180" step="15" value="0"/>
<input type="button" value="Reset" onclick="reset（）; servoMove（90）">
<output for="flying" name="level">90</output>/180</form></div>
```

図 4-23　ルンバ操作モードのボタン

図 4-23 はルンバの操作が不能になった時に用いるリセットボタンである。以下のタグで実現する。

```
<input type="button" style="width:30%;padding:20px;font-size:30px;"
onClick="roombaActivate（）" value="ルンバ操作モード" />
```

4.6　　考察

ODROID U3 を用いカメラ 3 台の映像を無線で映し 360°の範囲を見ることができた。また、ブラウザのボタンとスライダーは、問題なく表示することができた。これにより、パノラマビュー機能が完成したので、本研究の目的である死角を少なくするということを達成できた。

映像の検証結果をふまえ、本章で要求される機能は満たした。次の章で遠隔でルンバとサーボモータの操作の検証を行う。

第 5 章　　動作確認テスト　（司馬担当）

5.1　　評価目標　（司馬担当）

本研究で作製した機構・回路・ブラウザを連携して動作させるテストを行う。評価は機構班・Arduino 班・イメージ班、の 3 つの班の要求機能をもとに行うことにする。
要求機能を表 5-1 にまとめる。

表 5-1 要求機能一覧

作製した班	要求機能
機構班	カメラのフレームを製作すること ルンバの上にカメラを載せるための台座を製作すること ルンバの上にバッテリーや種変回路を置くためのフロアを製作すること
Arduino 班	サーボモータを操作してカメラの上下左右方向の動作を可能にすること ルンバの操作を可能にすること ユニバーサル基板への回路の実装をすること
イメージ班	360°のパノラマ映像を映しだすこと 遠隔でサーボモータの操作を可能にすること 遠隔でルンバの操作を可能にすること

図 5-1 車載用広角レンズを用いた室内見守りシステムの完成図

5.2　テスト環境　（司馬担当）

動作確認は廊下と研究室で行なった（図 5-2）。研究室の部屋の大きさは奥行 700cm、幅 350cm で、障害物としては机と椅子がある。無線通信は研究室内に配備されている Wi-Fi を用いて行うとする。被写体として目立ちやすいタカラトミー社のファービーを用意した（図 5-3）。テストで使用した端末は SONY 社の Windows タブレット VAIO Tap11 である。

図 5-2 実験を行う研究室

図 5-3 被写体（タカラトミー社のファービー）

5.3　テスト方法　（司馬担当）

本システムを動かすテストコースを以下の図 5-4 のように設定する。スタート地点から赤バツ印の指定位置まで動かす。指定位置に被写体であるファービーを置く。ファービーはカメラの向きの調整による上下の映像範囲を確認しやすくするため、床から約 40cm の高さに配置する。（図 5-5）

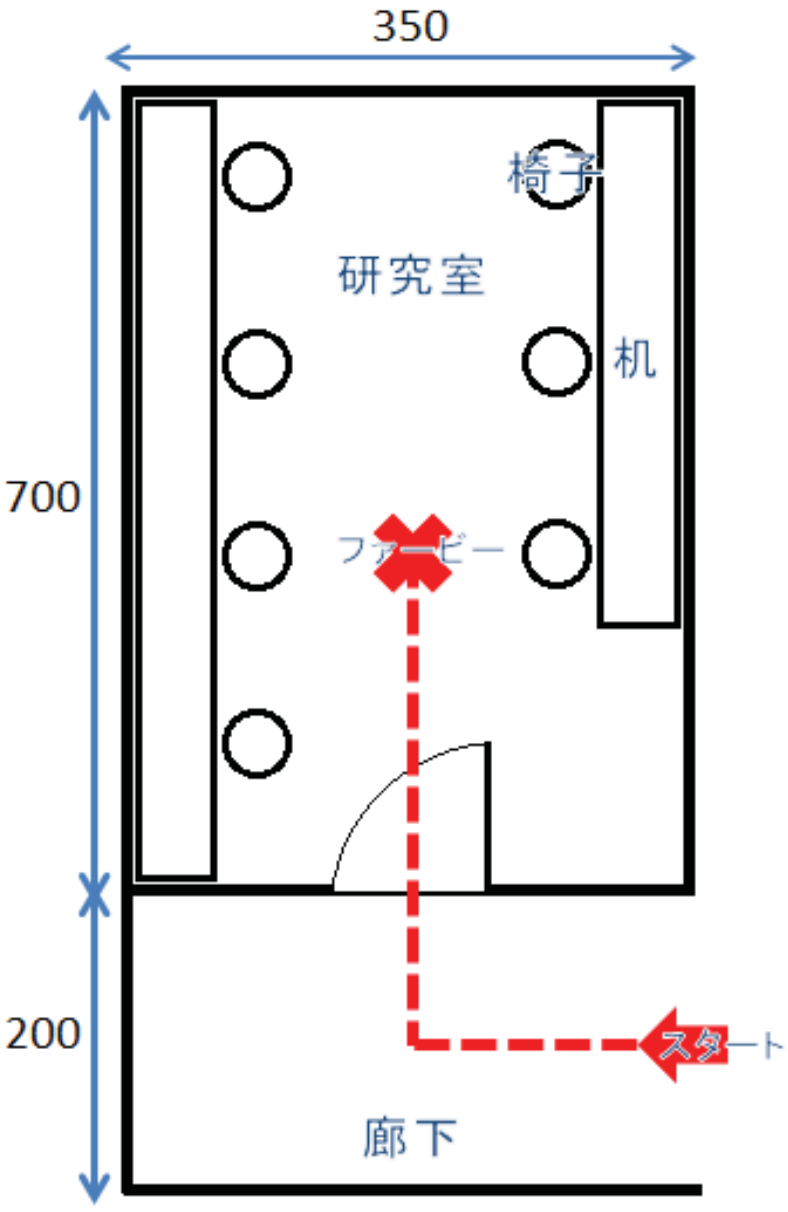

図 5-4 テストコース（cm）

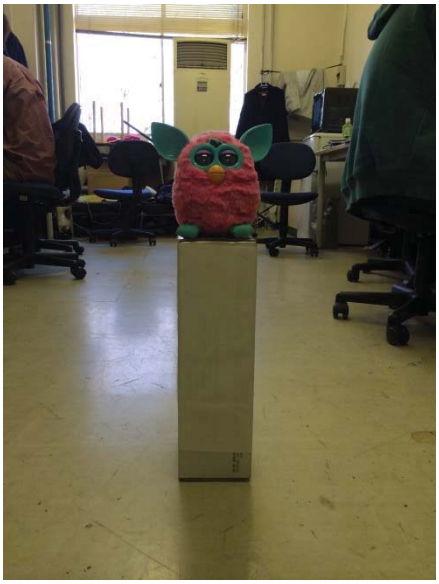

図 5-5 ファービーの配置位置（床から約 40 ㎝）

テストは以下の手順で行う。

1. 本システムのスイッチを入れ､ルンバと Arduino が正常に接続できているかを確認する。接続されているとルンバが操作可能なモード、すなわちルンバのスイッチランプが消えるのだが、その確認である。
2. タブレットでブラウザを起動して、カメラの映像が映ること、ルンバの基本動作の操作が可能であることを確認する。
3. 操作する人は部屋の中が見えない廊下を定位置とし、タブレットに映るカメラの映像のみをもとに、本システムをスタート地点から指定位置まで操作する。
4. 本システムが指定位置まで到達し、映像が 360°のパノラマをカバーできているか確認する。
5. カメラの角度を調節し、被写体を捉える。
6. 本システムの機構及び配線に異常がないか確認する。

5.4　テスト結果　（司馬担当）

動作確認テストの結果を、実際のテスト時の画像とともに 5.3 章の手順通りに説明する。

1. 手順 1 を行い、手順 2 のタブレットにカメラの映像が映ること、タブレットからルンバの基本動作、前進、後退、右回転、左回転の操作が可能であることを確認することができた。（図 5-6）

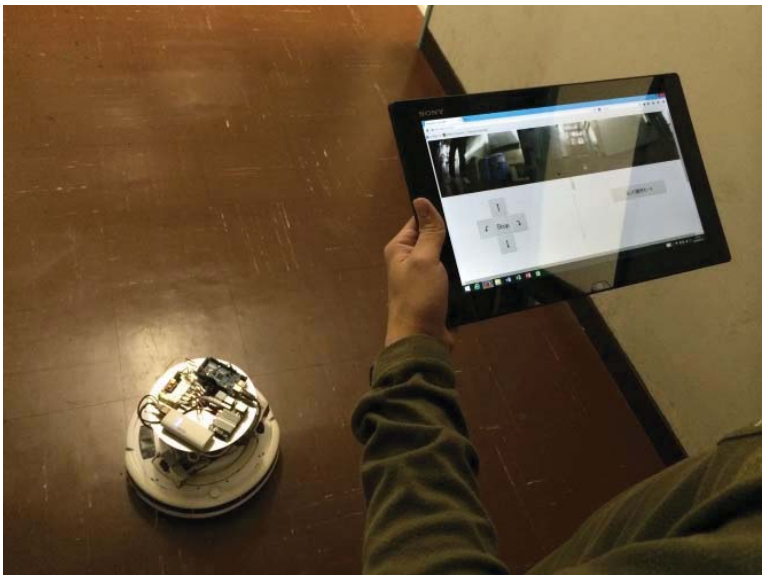

図 5-6 タブレットで確認

2. 手順 3 を行い、手順 4 の本システムが指定位置まで到達したところ。その時のカメラの映像（図 5-7）を確認すると 360°のパノラマをカバーできていることが分かる。

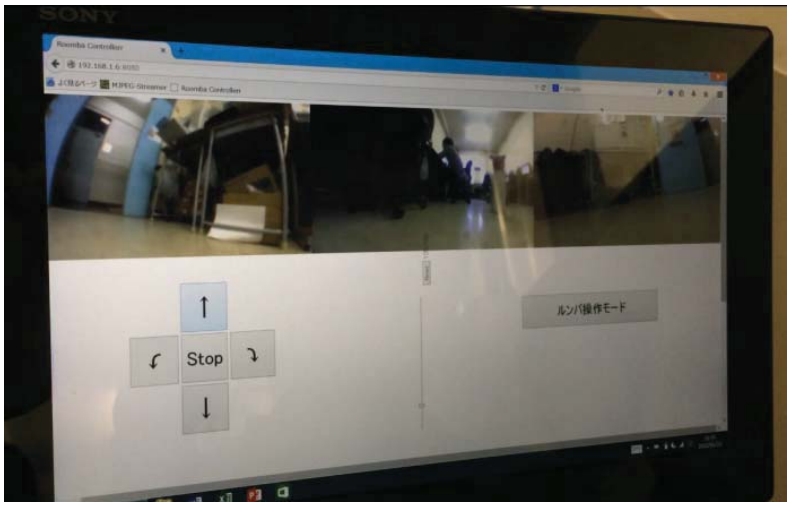

図 5-7　360°の部屋の様子

3. 手順 5 のカメラの向きを調整し、被写体であるファービーを映像で捉えたところ（図 5-8)。図の赤丸で囲んだ部分にファービーが映っていることが確認できる。

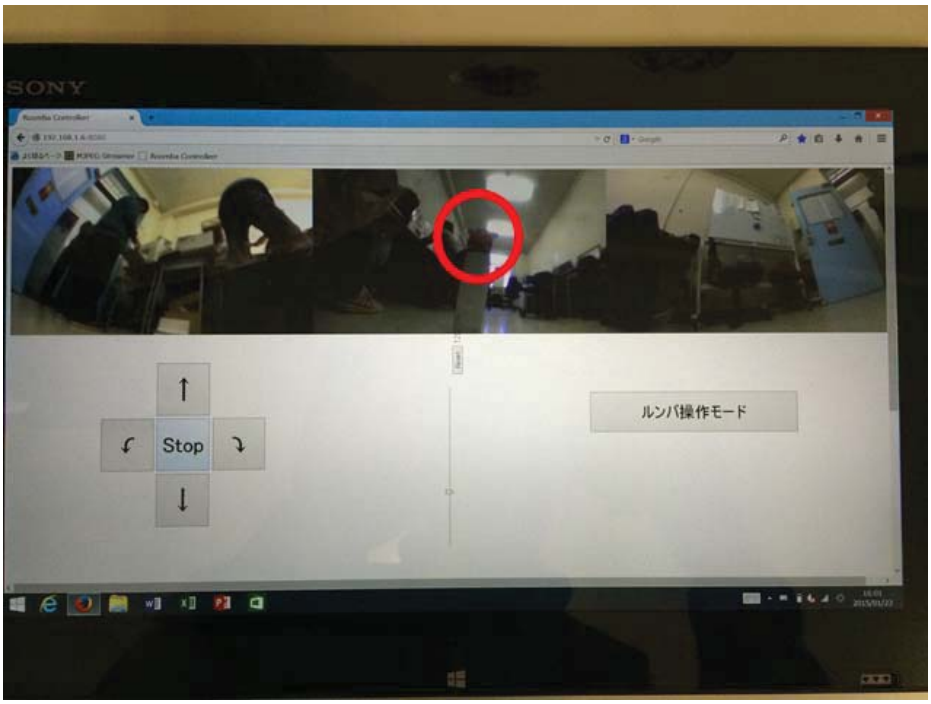

図 5-8 タブレットの映像

4. 手順 6、本システムの異常は確認されなかった。

5.5　　考察　（司馬担当）

動作確認の結果から表 5-1 に示した「機構」、「Arduino」、「イメージ」に関する要求機能が全て満たされていることが確認できた。それぞれの班の成果がうまく連動できたといえる。

カメラの向きの調整、ルンバの操作、それぞれタブレット上からスムーズに行うことができた。Arduino の機能 2 つと映像の無線化もうまくいったと言える。

テスト後の本システムの異常は確認されなかったが、テスト中、カメラの向きを調整する際にカメラのコードやモータのコードが障害になる可能性があった。

何回かテストを行うと、3 つのカメラの映像のうち何台か映らない時があった。原因として供給電力の不足が考えられる。

今回のテストの結果や考察を踏まえ、次章にて本システムの問題点と改善法について印、最終章にて結論を述べる。

第 6 章　　問題点とその解決法

6.1　　本システムの問題点

第 5 章にて我々が作成した見守りシステムが期待通りに動作することが確認されたが、下記のような問題点がある。本章ではこれらの問題の解決法を示す。

問題 1. ルンバの速度が固定されていること

ルンバの速度は 0mm/s から 500mm/s まで可変である。しかし、第 5 章までのシステムはボタンのオンオフによるコントロールを行うため、速度が 100mm/s に固定されている

問題 2. 操作画面を構成する HTML が特定ブラウザでしか適切に表示されない

HTML、特に垂直方向のスクロールバーの挙動と配置はブラウザ依存があるため、第 5 章までのシステムが最適に動作するのは Windows タブレットの Firefox ブラウザのみである

問題 3. パノラマ映像の 3 画面を同時表示すると個々の画面が小さくなる

パノラマ映像の特性上 3 つの画面を横に並べる必要があり、映像が小さく表示されてしまう

問題 4. カメラ 3 つの向きがルンバの中心を向いていないため利用場面が限定される

この問題は次節で図を用いて解説する

6.2　　問題の解決法

6.2.1　　問題 1.の解決法

ルンバの速度を可変にするため、ボタンではなくタブレットのタッチによる操作のインターフェースに変更した（図 6-1）[6-1]。映像の下にタッチエリアを配し、タッチエリアの中心付近をタッチすると低速で移動し、中心から離れたエリアをタッチするほど高速で移動できる。以上により、問題 1.を解決できた。

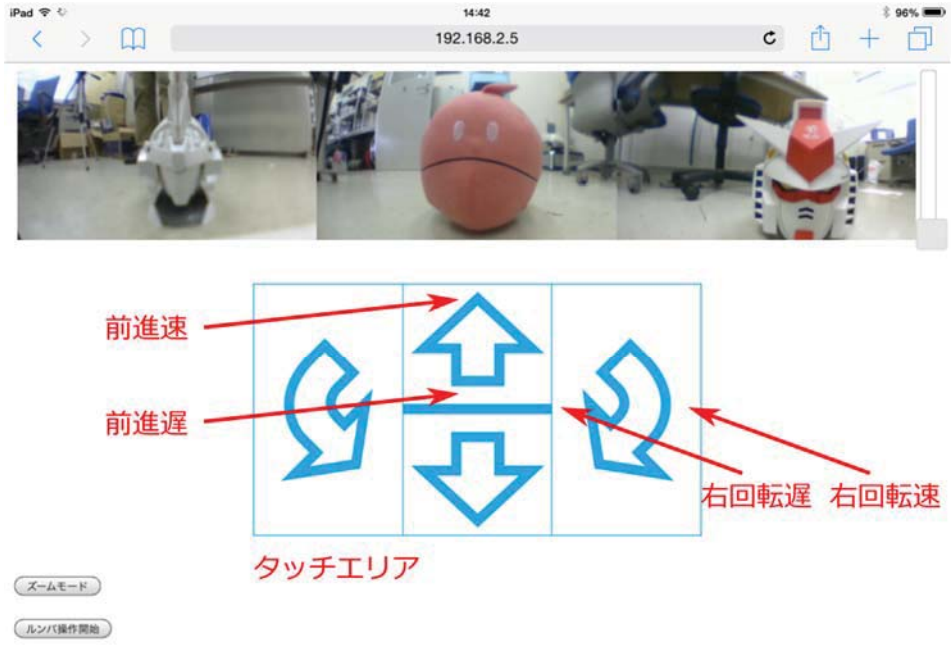

図 6-1 タッチによるコントローラー

6.2.2 問題 2.の解決法

カメラの高さを変更するためのスライダーを、HTML の標準部品から、jQuery UI のスライダーに変更し、映像の横に縦に配置した（図 6-1）[6-1][6-2]。これにより、このコントローラーは Windows/Android/iOS で動作するようになった。図 6-1 は iPad mini 上でのコントローラーを示している。

6.2.3 問題 3.の解決法

映像のパノラマ表示のために個々の映像が小さくなってしまう問題は、図 6-1 のパノラマモードに加え、図 6-2 のようなズームモードを追加することで解決した。このモードは、3 つの映像のうちの中心の映像（進行方向の映像）のみを画面に表示し、その映像をそのままタッチエリアとするというモードである。図 6-1 と図 6-2 の左下にあるボタンにて相互のモードを行き来することができる。

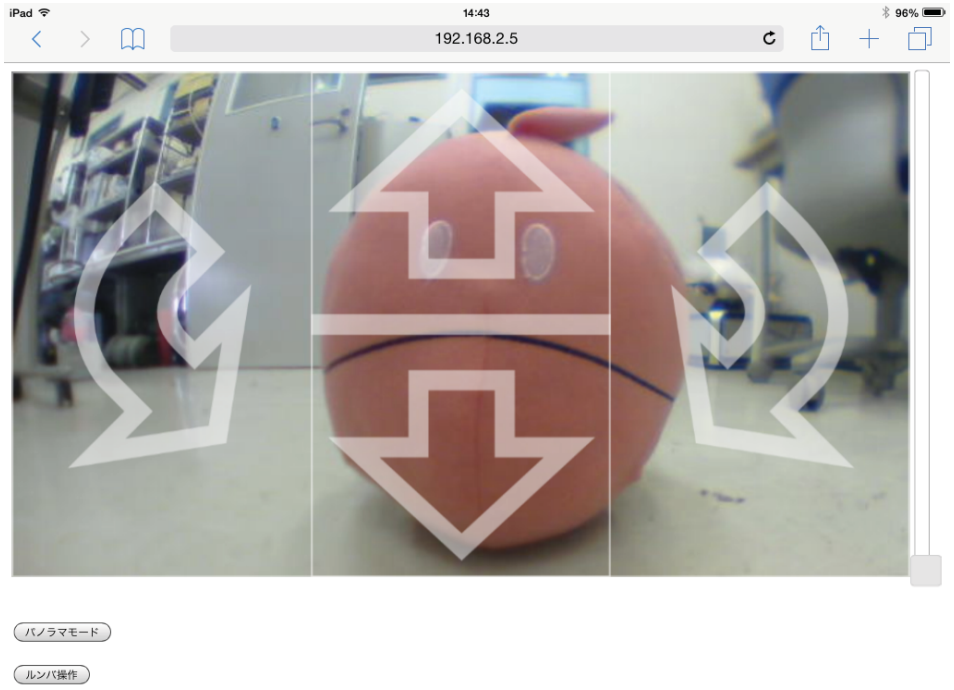

図 6-2 ズームモード

6.2.4 問題 4.の解決法

カメラの配置の問題を図 6-3 により解説する。カメラは第 2 章で解説したように、サーボモータに接続された上で、台座に配置されている。この構造上の問題により、カメラの向きがルンバの中心を向いていない。そのため、カメラの配置が後方で左右対称ではないという問題がある。

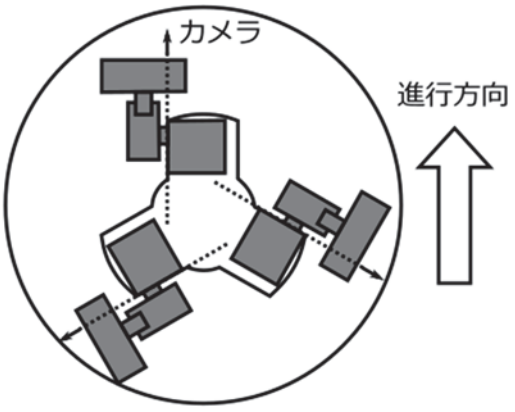

図 6-3 カメラの向きがルンバ中心を向いていない問題

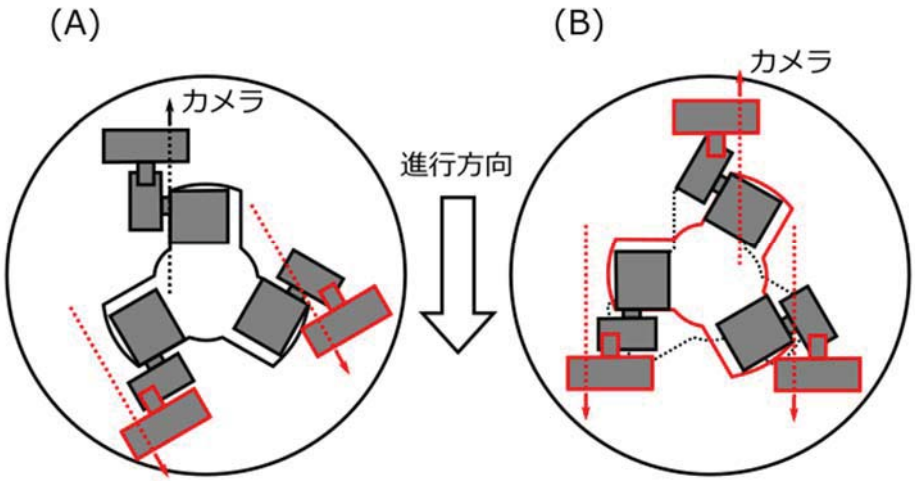

図 6-4 後方のカメラをステレオカメラとして用いるためには

そのため、このカメラの応用には制限がかかってしまう。例えば、ルンバ後方の 2 つのカメラを 3 次元の立体視を実現するステレオカメラとして利用することを考えよう。まず進行方向は図 6-4 のように図 6-3 とは逆になる。進行方向の逆転はソフトウェアにより容易に実現可能である。しかし、後方 2 つのカメラを同じ向きに向けようとすると、図 6-4(A)のように進行方向とは 30 度ずれた方向に向いてしまう。

このずれを解消するためには、図 6-4(B)のようにカメラの台座自体の向きを変更する必要がある。台座はルンバと耐震マットで固定されているため、これを変更するにはシステムを一度停止して台座とカメラを配置し直す必要がある。

言い換えると、図 6-3 から図 6-4(A)はソフトウェア的に変更できるが、図 6-4(B)へはハードウェア的な変更が必要だということである。この事実が、このシステムの応用範囲を狭めている。

この問題を解決するには、台座の再設計の必要があり、本研究では実現できなかった。

第 7 章　　結論　（原田担当）

7.1　　目標　（原田担当）

我々が作製する車載用広角レンズを用いた室内見守りシステムの要求機能は、タブレットで 360°のパノラマビューの映像を見ることができ、カメラの上下駆動とルンバの操作ができることであった。

7.2　　目標への達成度　（原田担当）

機構班は車載用広角レンズを搭載したカメラ、カメラ 3 台を取り付けるための台座、バッテリー周辺回路を設置するフロアを作製した。

Arduino 班は Arduino MEGA でサーボモータとルンバを動かすためのプログラムと基板を作製した。サーボモータのプログラムは、6 個のサーボモータを上下駆動させるものである。ルンバのプログラムは前後の移動と左右の回転のプログラムである。基板はルンバ、サーボモータ、Arduino MEGA、ODROID U3 を接続するためのものである。

イメージ班は 360°のパノラマビュー（カメラ 3 台）の映像出力、通信制御の無線化、ブラウザのインターフェースを作製した。通信制御の無線化は ODROID U3 を使用し Wi-Fi を介したインターネット回線を用いてタブレットとの通信を可能にした。インターフェースは HTML5 で開発を行った。画面上部にパノラマビューの映像、下部にルンバのコントロールボタンとサーボモータの上下駆動のスライダーを配置した。

第 5 章の動作確認テストの考察で述べた、テスト中カメラの向きを調整する際にカメラのコードやモータのコードが障害になる問題は、コードの長さが原因だと考えられるので、必要な長さに作製することで解決する。映像出力が不安定な状態は、電力供給の不足と考えられるので、供給電力を上げることで解決できると考えられる。

7.3　　総合評価　（原田担当）

本研究テーマである「車載用広角レンズを用いた室内見守りシステムの開発」について、目標を達成するために機構班、Arduino 班、イメージ班の 3 つに分かれ活動した。

製作したカメラとルンバを組み合わせ実際にタブレットで操作したところ、室内見守りシステムのすべての機能は問題なく動作した。無線を介して映像を見ることやルンバを操作することにおいては目標を達成できたといえる。車載用広角レンズを用いた室内見守りシステムを作製し、車載用広角レンズの新たな用途の提案ということにおいて要求を満たすことができた。

第 8 章　　参考文献・URL

[1-1] 車載用カメラとは－日本語表現辞典 Weblio 辞書
http://www.weblio.jp/content/%E8%BB%8A%E8%BC%89%E7%94%A8%E3%82%AB%E3%83%A1%E3%83%A9

[1-2] [拡大画像]Car Watch アルパイン、車種専用ステアリング連動バックビューモニターカメラなど 18 機種
http://car.watch.impress.co.jp/img/car/docs/531/827/html/06.jpg.html

[1-3] 【楽天市場】NISSAN ニッサン/X-TRAIL エクストレイル日産純正バックビューモニター【対応年式 2010.07～次モデル】【desir de vivre】：desir de vivre [1-3]
http://item.rakuten.co.jp/desir-de-vivre/xtrail201007-087/

[1-4] SUBARU：スバル アイサイト オーナーズサポートサイト アイサイトの仕組みを知ろう
http://www.subaru.jp/eyesightowner/about/

[1-5] アラウンドビューモニター｜日産｜技術開発の取り組み
http://www.nissan-global.com/JP/TECHNOLOGY/OVERVIEW/avm.html

[1-6] 猫部屋探検－まこブログ－
http://makocchan.hatenablog.com/entry/2012/09/30/111519

[1-7] ワンルームマンション投資のデメリットと利回り｜はじめての個人年金保険
http://www.paci-nenkin.com/sisan/626/

[1-8] ルンバ 500 シリーズ｜iRobot ロボット掃除機ルンバ 公式サイト
http://www.irobot-jp.com/storeproduct/500series/

[1-9] 2014 年国内タブレット世帯普及率 21.9%(by 総務省)－BTO パソコン.jp
http://bto-pc.jp/btopc-com/select/tablet-soumu-go-jp-2014.html

[1-10] タブレット端末の出荷台数(日本国内.海外)まとめ：2013 年度版｜アプリマーケティング研究所
http://appmarketinglabo.net/tablet-market-japan-world/

[1-11] 国内のロボット掃除機市場は 2018 年に 90 万台に－シードプランニング調査｜マイナビニュース
http://news.mynavi.jp/news/2013/07/22/204/

[1-12] ロボット家電 COCOROBO：シャープ
http://www.sharp.co.jp/cocorobo/

[1-13] COCOROBO SQUARE iOS 版機能紹介｜COCOROBO SQUARE｜ソフトウェア・ダウンロード｜ロボット家電 COCOROBO：シャープ
http://www.sharp.co.jp/cocorobo/manual/ios.html

[1-14] COCOROBO ナビ 間取りマップの作製｜ソフトウェア・ダウンロード｜ロボット家電 COCOROBO：シャープ http://www.sharp.co.jp/cocorobo/manual/navi3.html

[2-1] マイク内蔵 320 万画素 WEB カメラ F2.2 ガラスレンズ搭載モデル
http://buffalo.jp/product/multimedia/web-camera/bsw32km03/

[2-2] HD Webcam C310
http://www.logicool.co.jp/ja-jp/product/7076?crid=34

[2-3] KRS-788HV ICS Red Version
http://kondo-robot.com/product/krs-788hv-ics-red-version

[2-4] サーボアーム 700A
http://kondo-robot.com/product/01122

[3-1] RC サーボモータを Arduino と Processing で GUI 制御（その 1）【Arduino 編】
http://blog.goo.ne.jp/roboz80/e/a74c391ed238ad702196b1579f9e71c6

[3-2] Arduino から Roomba を Software Serial で動かす
http://qiita.com/DUxCA/items/36a13e68722c51c72927

[3-3] ルンバの ROI を使っての制御Ⅱ（Arduino ソフトウェア編）
http://flanker711.blog49.fc2.com/blog-entry-98.html

[3-4] RC サーボモータを Arduino と Processing で GUI 制御（その２） 【Processing 編】
http://blog.goo.ne.jp/roboz80/e/6d309257fd23f2cbee97254df460fb56

[4-1] Joshin ネットショッピング｜まだまだある、タブレット PC
http://joshinweb.jp/pc/tablet3.html?ACK=TOKU&CKV=111111

[4-2] 価格.com｜バッファロー、VPN 機能搭載ギガビット有線 LAN ルーター
http://news.kakaku.com/prdnews/cd=pc/ctcd=0075/id=15989/imageno=0/

[4-3] gpad.tv｜Hardkernel ODROID-U3
http://gpad.tv/develop/hardkernel-odroid-u3/

[4-4] MJPEG streamer のトップ画像
http://blog-imgs-63.fc2.com/r/a/s/raspuser/Mjpeg-streamer.jpg

[4-5] IO DATA 社製の無線 LAN アダプタ（型番 WN-G150U）
http://img1.kakaku.k-img.com/images/productimage/fullscale/K0000100577.jpg

[4-6] プラネックスコミュニケーションズ社製の無線 LAN アダプタ(型番 GW-450D)
http://www.planex.co.jp/product/product_photo/300/gw-450d.jpg

[4-7] バッファロー｜画像ダウンロード｜BSH5AD0310WH（ホワイト/1m）
http://buffalo.jp/download/photo/supply/bsh5ad0310.html

[4-8] オンキヨー社製｜デスクトップパソコン｜S721・S720・S517 シリーズ
http://www.jp.onkyo.com/pc/desktop/

[4-9] FabxFab｜Arduino Mega2560
http://www.fabxfab.com/?pid=53448359

[4-10] amazon.co.jp｜iRobot Roomba 自動掃除機　ルンバ 530 白色
http://www.amazon.co.jp/iRobot-Roomba-%E8%87%AA%E5%8B%95%E6%8E%83%E9%99%A4%E6%A9%9F-%E3%83%AB%E3%83%B3%E3%83%90-530-%E7%99%BD%E8%89%B2/dp/B000UU7TZE

[6-1] 金丸隆志｜Raspberry Pi で学ぶ電子工作 講談社（2014）

[6-2] Takashi Kanamaru｜（追加コンテンツ）サーボモーターを PC やスマートフォンから角度制御する
http://raspibb.blogspot.jp/2015/01/pc.html

謝辞

最後に、この場をお借りして本研究を進めるにあたり、研究テーマや車載用広角レンズの提供等のご支援いただいた K 社、並びに、2 年間ご指導ご鞭撻を頂きました金丸隆志准教授、新井敏夫教授、桂晃洋教授、機構の設計や金属加工において快く対応していただいた花野井利之様、研究において親身になり多くの助言を頂いた修士の先輩方、協力していただいた皆様に心より感謝致します。これを持って謝辞とかえさせていただきます。

2014 年度（平成 26 年度）

ECPⅢ

Final Report

Arduino プログラム、HTML のプログラム

JavaScript のプログラムの付録

チーフアドバイザー　：　金丸　隆志　准教授

サブアドバイザー　：　桂　　晃洋　教授

サブアドバイザー　：　新井　敏夫　教授

チームメンバー

G1-11022　加藤　　諒

G1-11039　司馬　聖大

G1-11046　高橋　勇磨

G1-11050　津田　郁也

G1-11062　原田　慎吾

G1-11065　福山　陽太

目次

第 1 章　最終的な Arduino のプログラム

1~18

第 2 章　最終的な HTML のプログラム

P19~21

第 3 章　JavaScript のプログラム

P22~26

第 1 章　　最終的な Arduino のプログラム

本章では、実際に本研究で使用した最終的なプログラム(Arduino)を以下に載せる。

```
//ルンバ　ヘッダ
#include <math.h>
#include <Servo.h>//サーボ　ヘッダ

//無線シールドのピン配置になっているので define コメントにしては残しておく
//#define TX_PIN 12
//#define RX_PIN 13
#define RoombaSerial Serial1

static float g_odometry_x = 0;     //オドメトリの X 座標
static float g_odometry_y = 0;     //オドメトリの Y 座標
static float g_odometry_Angl = 0;//オドメトリの角度

//オドメトリの更新時間
static const int UPDATE_TIME_MSEC = 50;

//**********定数を定義**********
static const float ENCODER_MM_PAR_PULSE = 0.448;
//static const int     L = 250;//車輪間距離(中央-中央):235 (外-外):250
static  const  float     L  =  238.0952;//車輪間距離(中央-中央):235  (外-外):250    770  は
238.0952 らしい

static const int RIGHT = 0;
static const int LEFT = 1;
//バッテリなどで使用
static const int CURRENT = 0;
static const int MAX = 1;

//前回のエンコーダー値
static int g_beforEncoderR;
static int g_beforEncoderL;
```

```
//サーボ変数
unsigned char incomingByte = 0; // for incoming serial data
unsigned char buff[30];
int RecvCount = 0;
int ParamLen = 0;
int ParamCount = 0;


Servo servo0;
Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;
Servo servo5;

//ルンバ独自関数
int odometryInit()
{
  g_beforEncoderR = encoderOder(0);//右エンコーダー初期化
  g_beforEncoderL = encoderOder(1);//左エンコーダー初期化
}


void setup()
{
  RoombaSerial.begin(19200);
  Serial.begin(9600);

  // Start
  Serial.println("Start");
  RoombaSerial.write((byte)128);     // Start -&gt; Passive Mode
  delay(50);
  RoombaSerial.write((byte)130);     // Safe Mode
  delay(50);
  // Passive Mode
```

```
  //RoombaSerial.write((byte)128);     // Start -&gt; Passive Mode

  servo0.attach(2);
  servo1.attach(3);
  servo2.attach(4);
  servo3.attach(5);
  servo4.attach(6);
  servo5.attach(7);

  odometryInit();
}

//戻り値はエンコーダーの値
//引数：0 は右車輪 1 は左車輪
int encoderOder(int encoderNum)
{
  int bufSize;//バッファに貯まっているサイズ
  int temp;//シリアル通信できたもの
  unsigned int count;//エンコーダのカウンタ
  bufSize = RoombaSerial.available();

  RoombaSerial.write((byte)142);
  if( encoderNum == RIGHT ) RoombaSerial.write((byte)43);     //右モーターの番号
  else                      RoombaSerial.write((byte)44);     //左モーターの番号

  while( RoombaSerial.available() < 2 )//2byte 来るまで無限ループ
  {
  }

  if( RoombaSerial.available() >= 2 )//
  {
    //unsigned char temp;
    temp = RoombaSerial.read();
    count = temp * 256;
    //Serial.print( temp );
    //Serial.print( " , " );
```

```
    temp = RoombaSerial.read();
    count += temp;
    //Serial.print( temp );
    //Serial.print( " , " );
    //Serial.print( count );
    //Serial.print( "¥n" );
  }
  return count;

}


//戻り値は更新時間
unsigned long odometry()
{
  //変化量
  float dX , //⊿X
  dY ,//⊿Y
  dAngle ,//⊿Y
  dL ,//左車輪の移動量
  dR;//右車輪の移動量

  int tempR,tempL , counterR , counterL;
  tempR = encoderOder(0);//右エンコーダー
  tempL = encoderOder(1);//左エンコーダー
  unsigned long updateTime = millis();

  //前回の計測からカウンタが一瞬で半分以上(32768)変化したらオーバーフローしている
  if(       tempR - g_beforEncoderR < -32768 ) counterR = ((long)tempR + (long)65536 - (long)g_beforEncoderR); //オーバーフローした場合
  else if( tempR - g_beforEncoderR >  32768 ) counterR = ((long)tempR - (long)65536 - (long)g_beforEncoderR); //マイナスになった場合
  else  counterR = tempR - g_beforEncoderR;

  if(       tempL - g_beforEncoderL < -32768 ) counterL = ((long)tempL + (long)65536 - (long)g_beforEncoderL); //オーバーフローした場合
```

```
  else if( tempL - g_beforEncoderL >  32768 ) counterL = ((long)tempL - (long)65536 - (long)g_beforEncoderL); //マイナスになった場合
  else  counterL = tempL - g_beforEncoderL;

  //  Serial.print(tempL - g_beforEncoderL);
  //  Serial.print(" , ");
  //  Serial.print(tempR - g_beforEncoderR);
  //  Serial.print("¥n");
  g_beforEncoderR = tempR;//右エンコーダーの情報を更新
  g_beforEncoderL = tempL;//左エンコーダーの情報を更新

  dR = counterR * ENCODER_MM_PAR_PULSE;
  dL = counterL * ENCODER_MM_PAR_PULSE;

  dAngle = (double)(dL-dR)/L;
  g_odometry_Angl += dAngle/M_PI*180.0;



  dX = ( dL + dR )/2.0 * cos( (g_odometry_Angl/180)*M_PI );//⊿X
  dY = ( dL + dR )/2.0 * sin( (g_odometry_Angl/180)*M_PI );//⊿Y

  g_odometry_x += dX;
  g_odometry_y += dY;
  return updateTime;
}



void loop()
{
  int command;
  static boolean odometryMode = false;

  if( Serial.available() > 0 )//データが来ていたら
  {
    static int speedNum = 100;//速度
    int radiusNum = 0;
```

```
command = Serial.read();

switch( command )
{
case 's':
   stopRoomba();
   break;
   //前進
case 'x':
   speedNum = serialReadSpeed();//速度を受信する
   //Serial.print("速度：");
   //Serial.print(speedNum);
   //Serial.print("¥n");
   foward( speedNum );
   break;
   //後退
case 'w':
   speedNum = serialReadSpeed();//速度を受信する
   backward(speedNum);
   break;
   //右ターン
case 'd':
   speedNum = serialReadSpeed();//速度を受信する
   turnRight( speedNum );
   break;
   //左ターン
case 'a':
   speedNum = serialReadSpeed();//速度を受信する
   turnLeft( speedNum );
   break;
   //左コーナリング
case 'q':
   speedNum = serialReadSpeed();//速度を受信する
   radiusNum = serialReadRadius();//旋回半径を設定
   corneringLeft( speedNum , 100 );
   break;
```

```
    //右コーナリング
case 'e':
    speedNum = serialReadSpeed();//速度を受信する
    radiusNum = serialReadRadius();//旋回半径を設定
    corneringRight( speedNum , 100 );
    break;


case 'b':
    motorsPWMControl( true );
    break;
case 'B':
    motorsPWMControl( false );
    break;


case 'c':
    Serial.print( encoderOder(RIGHT) );
    Serial.print( "¥n" );
    break;
case 'C':
    Serial.print( encoderOder(LEFT) );
    Serial.print( "¥n" );
    break;


    //バッテリ容量
case 'p':
    Serial.print( getBattery( CURRENT ) );
    Serial.print( "¥n" );
    break;
case 'P':
    Serial.print( getBattery( MAX ) );
    Serial.print( "¥n" );
    break;
```

```
case 'o'://小文字 o ならオドメトリを送り始める
  odometryMode = true;
  break;
case 'O': //大文字 O ならオドメトリを送るのをやめる
  odometryMode = false;
  break;
case 'r'://座標リセットコマンド
  g_odometry_x = 0;     //オドメトリの X 座標
  g_odometry_y = 0;     //オドメトリの Y 座標
  g_odometry_Angl = 0;//オドメトリの角度
  break;

case 'S'://サーボ命令のとき
  // Servo commands
  while(true)
  {
    if(Serial.available() > 0)
    {
      int exitServoCommand = 0;
      // read the incoming byte:
      incomingByte = Serial.read();

      switch(RecvCount)
      {
      case 0:
        ParamLen = 0;
        ParamCount = 0;
        if(incomingByte == 0xff) {   // Headder
          buff[RecvCount++] = incomingByte;
        }
        break;
      case 1:
        if(incomingByte == 0) {   // ID
          buff[RecvCount++] = incomingByte;
        }
        break;
```

```
            case 2:                 // LENGTH
              buff[RecvCount++] = incomingByte;
              ParamLen = incomingByte;
              break;
            default:
              if(ParamLen > 0) {
                buff[RecvCount++] = incomingByte;
                ParamCount++;
                if(ParamCount >= ParamLen) {
                  ServoCmdExec();
                  RecvCount = 0;
                  exitServoCommand = 1;
                }
              }
              else {
                RecvCount = 0;
              }
              break;
            } // switch

            if(exitServoCommand == 1){
              break;
            }
          }
        } // while true

        break;
    }
}


static unsigned long odometryUpdateTime = millis();//時間初期化
//前回の時間から 30msec 超えていたら
if( ( millis() >= (odometryUpdateTime + 30)) )
{
   odometryUpdateTime = odometry();
```

```
      ///odometryUpdateTime //= millis();

      //もしオドメトリ表示モードだったら
      if( odometryMode )
      {
         Serial.print( odometryUpdateTime );
         Serial.print(",");
         Serial.print( g_odometry_x );
         Serial.print(",");
         Serial.print( g_odometry_y );
         Serial.print(",");
         Serial.print( g_odometry_Angl );
         Serial.print("¥n");
      }

   }

} // loop


//**********************以下通信関係**********************
//速度設定
int serialReadSpeed()
{
   int speedNum = 0 //戻り値となる
   , readByte = 0;
   boolean signFlag;//符号がどっち　true ならプラス、false ならマイナス
   int temp;

   while( true )
   {
      if(Serial.available() > 0)
      {
         temp = Serial.read();
         //数字かどうかチェック
         //数字だったら処理する
```

```
        if(   ('0' <= temp) && (temp <= '9') )
        {
          switch( readByte )
          {
          case 0://Serial.print( temp-'0' ); Serial.print("¥n");
            if( (temp - '0') == 0 ) signFlag = true;
            else signFlag = false;
            break;
          case 1://Serial.print( temp-'0' ); Serial.print("¥n");
            speedNum += (temp-'0') * 100;
            break;
          case 2://Serial.print( temp-'0' ); Serial.print("¥n");
            speedNum += (temp-'0') * 10;
            break;
          case 3://Serial.print( temp-'0' ); Serial.print("¥n");
            speedNum += (temp-'0') ;
            break;
          }
          readByte++;
        }
      }
      //4 文字受信したならばループ終了
      if( readByte >= 4 ) break;
    }
    //負数だったならば
    if( !signFlag ) speedNum *= (-1 );
    //速度上限と下限の設定
    if( speedNum > 500 ) speedNum = 500;
    else if( speedNum < (-500) ) speedNum = -500;

    return speedNum;
}
//旋回半径設定
int serialReadRadius()
{
    int radiusNum = 0 //戻り値となる
```

```
, readByte = 0;
boolean signFlag;//符号がどっち　true ならプラス、false ならマイナス
int temp;

while( true )
{
  if(Serial.available() > 0)
  {
    temp = Serial.read();
    //数字かどうかチェック
    //数字だったら処理する
    if(  ('0' <= temp) && (temp <= '9') )
    {
      switch( readByte )
      {
      case 0://Serial.print( temp-'0' ); Serial.print("¥n");
        if( (temp - '0') == 0 ) signFlag = true;
        else signFlag = false;
        break;
      case 1://Serial.print( temp-'0' ); Serial.print("¥n");
        radiusNum += (temp-'0') * 1000;
        break;
      case 2://Serial.print( temp-'0' ); Serial.print("¥n");
        radiusNum += (temp-'0') * 100;
        break;
      case 3://Serial.print( temp-'0' ); Serial.print("¥n");
        radiusNum += (temp-'0') * 10;
        break;
      case 4://Serial.print( temp-'0' ); Serial.print("¥n");
        radiusNum += (temp-'0') ;
        break;
      }
      readByte++;
    }
  }
  //4 文字受信したならばループ終了
```

```
    if( readByte >= 5 ) break;
  }
  //負数だったならば
  if( !signFlag ) radiusNum *= (-1 );
  //速度上限と下限の設定
  if( radiusNum > 2000 ) radiusNum = 500;
  else if( radiusNum < (-2000) ) radiusNum = -500;

  return radiusNum;
}

void motorsPWMControl( boolean mode)
{

  RoombaSerial.write((byte)144);     // Drive

  if( mode )
  {
    RoombaSerial.write((byte)127);   // Velocity: 0x00c8 = 200
    RoombaSerial.write((byte)127);
    RoombaSerial.write((byte)127);   // Radius: 0x8000 = Straight
  }
  else
  {
    RoombaSerial.write((byte)0);   // Velocity: 0x00c8 = 200
    RoombaSerial.write((byte)0);
    RoombaSerial.write((byte)0);   // Radius: 0x8000 = Straight
  }
}

//前進命令
void foward(int speedNum)
{
  // Forward
  //Serial.println("Forward");
```

```
  //試しに byte 型配列を使ってみる
  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];


  RoombaSerial.write((byte)137);     // Drive
  //速度指定
  //配列の後ろから送信する
  RoombaSerial.write(sendSpeedData[1]);     // Speed
  RoombaSerial.write(sendSpeedData[0]);     // Speed
  RoombaSerial.write((byte)0x00);   // Velocity: 0x00c8 = 200
  RoombaSerial.write((byte)0x00);
}


//後退命令
void backward(int speedNum)
{
  speedNum*=(-1);//バックなので速度をマイナスにする
  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];

  // Backward
  //Serial.println("Backward");
  RoombaSerial.write((byte)137);     // Drive
  //速度指定
  //配列の後ろから送信する
  RoombaSerial.write(sendSpeedData[1]);     // Speed
  RoombaSerial.write(sendSpeedData[0]);     // Speed

  RoombaSerial.write((byte)0x80);   // Radius: 0x8000 = Straight
  RoombaSerial.write((byte)0x00);
}
```

```
//左その場回転
void turnLeft(int speedNum)
{
   byte sendSpeedData[2];
   sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
   sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];
   //Serial.println("Turn");
   RoombaSerial.write((byte)137);     // Drive
   //速度指定
   //配列の後ろから送信する
   RoombaSerial.write(sendSpeedData[1]);     // Speed
   RoombaSerial.write(sendSpeedData[0]);     // Speed
   RoombaSerial.write((byte)0x00);   // Radius: 0x0001 = Turn in place counter-clockwise
   RoombaSerial.write((byte)0x01);
}

//右その場回転
void turnRight(int speedNum)
{
   byte sendSpeedData[2];
   sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
   sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];
   // Turn in place clockwise
   //Serial.println("Turn");
   RoombaSerial.write((byte)137);     // Drive
   //速度指定
   //配列の後ろから送信する
   RoombaSerial.write(sendSpeedData[1]);     // Speed
   RoombaSerial.write(sendSpeedData[0]);     // Speed
   RoombaSerial.write((byte)0xff);   // Radius: 0x0001 = Turn in place clockwise
   RoombaSerial.write((byte)0xff);
}
//停止
void stopRoomba()
{
   // Stop
```

```
  //Serial.println("Stop");
  RoombaSerial.write((byte)137);     // Drive
  RoombaSerial.write((byte)0x00);
  RoombaSerial.write((byte)0x00);
  RoombaSerial.write((byte)0x00);
  RoombaSerial.write((byte)0x00);
}
//右コーナリング旋回
void corneringRight(int speedNum , int radius)
{
  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];

  radius *= (-1 );//右旋回は半径が負数
  byte sendRadiusData[2];
  sendRadiusData[0] = ( (byte *) ( (void*)(&radius) ) )[0];
  sendRadiusData[1] = ( (byte *) ( (void*)(&radius) ) )[1];

  // Turn in place clockwise
  //Serial.println("Turn");
  RoombaSerial.write((byte)137);     // Drive
  //速度指定
  //配列の後ろから送信する
  RoombaSerial.write(sendSpeedData[1]);     // Speed
  RoombaSerial.write(sendSpeedData[0]);     // Speed
  RoombaSerial.write(sendRadiusData[1]);     // Speed
  RoombaSerial.write(sendRadiusData[0]);     // Speed
  //RoombaSerial.write((byte)0xff);   // Radius: 0x0001 = Turn in place clockwise
  //RoombaSerial.write((byte)0xff);
}
//左コーナリング旋回
void corneringLeft(int speedNum , int radius)
{
  byte sendSpeedData[2];
  sendSpeedData[0] = ( (byte *) ( (void*)(&speedNum) ) )[0];
```

```
  sendSpeedData[1] = ( (byte *) ( (void*)(&speedNum) ) )[1];

  byte sendRadiusData[2];
  sendRadiusData[0] = ( (byte *) ( (void*)(&radius) ) )[0];
  sendRadiusData[1] = ( (byte *) ( (void*)(&radius) ) )[1];

  // Turn in place clockwise
  //Serial.println("Turn");
  RoombaSerial.write((byte)137);     // Drive
  //速度指定
  //配列の後ろから送信する
  RoombaSerial.write(sendSpeedData[1]);     // Speed
  RoombaSerial.write(sendSpeedData[0]);     // Speed
  RoombaSerial.write(sendRadiusData[1]);     // Speed
  RoombaSerial.write(sendRadiusData[0]);     // Speed
}

unsigned int getBattery(int mode)
{
  int bufSize;//バッファに貯まっているサイズ
  int temp;//シリアル通信できたもの
  unsigned int capacity;//バッテリ容量
  bufSize = RoombaSerial.available();

  RoombaSerial.write((byte)142);
  if( mode == CURRENT   ) RoombaSerial.write((byte)25);     //右モーターの番号
  else                    RoombaSerial.write((byte)26);     //左モーターの番号

  while( RoombaSerial.available() < 2 )//2byte 来るまで無限ループ
  {
  }

  if( RoombaSerial.available() >= 2 )//
  {
    temp = RoombaSerial.read();
    capacity = temp * 256;
```

```
    temp = RoombaSerial.read();
    capacity += temp;
  }

  return capacity;
}

//サーボ独自関数
void ServoCmdExec()
{
  //   buff[3]        //CMD (not used)

  if((buff[4] >= 0)&&(buff[4] <= 180))
    servo0.write(buff[4]);
  if((buff[5] >= 0)&&(buff[5] <= 180))
    servo1.write(buff[5]);
  if((buff[6] >= 0)&&(buff[6] <= 180))
    servo2.write(buff[6]);
  if((buff[7] >= 0)&&(buff[7] <= 180))
    servo3.write(buff[7]);
  if((buff[8] >= 0)&&(buff[8] <= 180))
    servo4.write(buff[8]);
  if((buff[9] >= 0)&&(buff[9] <= 180))
    servo5.write(buff[9]);
}
```

第２章　　最終的な HTML のプログラム

本章では、実際に本研究で使用した最終的なプログラム(HTML)を以下に載せる。

```
<html>
<head>
<title>Roomba Controllerr</title>
<meta charset="utf-8">
<script src="js/socket.io.js">
</script>
<script src="js/roomba.js">
</script>
<style>
        .button {
    transform: rotate(180deg);
}
.sample {
    transform: rotate(-90deg);
}

.reset {
    transform: rotate(-90deg);
}



</style>
</head>


<body onload="init()">
<div align="center">
<canvas id="canvas">
</canvas>
</div>
<br>
<br>
```

```
<br>

<table border="0" style="float:left;">
<tr>
<td> </td>
<td><input type="button" value="↑" onclick="roombaForward()" style="WIDTH:
100px; HEIGHT: 100px ;font-size:50px;"/>
<td> </td>
</tr>
<tr>
<td><div class="button"><input type="button" value="&#x2934;"
onclick="roombaLeft()" style="WIDTH: 100px; HEIGHT: 100px ; font-size:50px; " />
</div></td>
<td><input type="button" value="Stop" onclick="roombaStop()" style="WIDTH:
100px; HEIGHT: 100px ;font-size:40px;" />
</td>
<td><input type="button" value="&#x2935;" onclick="roombaRight()" style="WIDTH:
100px; HEIGHT: 100px ; font-size:50px;" />
</td>
</tr>
<tr>
<td> </td>
<td><input type="button" value="↓" onclick="roombaBackward()" style="WIDTH:
100px; HEIGHT: 100px ; font-size:50px; " />
 </td><td> </td>
 </tr>
</table>
<div class="sample">
<div style="margin-left:-15%;margin-right:40%;">
<form onsubmit="return false" oninput="level.value = flevel.valueAsNumber">
<input name="flevel" id="flying" type="range" oninput="servoMove(this.value)"
onchange="servoMove(this.value)" style="width:30%;padding:0px;font-size:30px;"
min="90" max="180" step="15" value="0"/>
<input type="button" value="Reset" onclick="reset(); servoMove(90)">
<output for="flying" name="level">90</output>/180
</form>
```

```
  </div>
  </div>
<div align="right">
<input type="button" style="width:30%;padding:20px;font-size:30px;"
onClick="roombaActivate()" value="ルンバ操作モード" />
</div>
</div>
</body>
</html>
```

第３章　　Java Script のプログラム

本章では、実際に本研究で使用した Java Script のプログラムを以下に載せる。

```
var mCount = 0;
var mCanvas;
var mCtx;

var mImg0_1;
var mImg0_2;

var mImg1_1;
var mImg1_2;

var mImg2_1;
var mImg2_2;

var mWidth = 640;
var mHeight = 360;

var mPortrait = true;

var host = location.host;
var hostname = host.split(":")[0];
var socketURL = 'http://' + hostname + ':' + '8888';
var socket = io.connect(socketURL);



var URL0;
var URL1;
var URL2;



// for iPhone (?)
window.onorientationchange = function()
{
   var iR = Math.abs( window.orientation );
```

```
    if ( iR == 0 || iR == 90 ){
        resize_canvas();
    }
}

function init(){
    mCanvas = document.getElementById("canvas");
    mCtx = mCanvas.getContext('2d');

    resize_canvas();

    var mqOrientation = window.matchMedia("(orientation: portrait)");

    // for Firefox
    mqOrientation.addListener(function() {
        resize_canvas();
    });
}

function imageSetup(){

    mImg0_1 = new Image();
    mImg0_2 = new Image();

    mImg1_1 = new Image();
    mImg1_2 = new Image();

    mImg2_1 = new Image();
    mImg2_2 = new Image();

    var host = location.host;
    var hostname = host.split(":")[0];

    var port0= 9000;
    var port1= 9001;
    var port2= 9002;
```

```
URL0 = 'http://' + hostname + ':' + port0 + '/?action=snapshot';
URL1 = 'http://' + hostname + ':' + port1 + '/?action=snapshot';
URL2 = 'http://' + hostname + ':' + port2 + '/?action=snapshot';


mImg0_1.src = URL0 +'&'+(mCount++);
mImg0_1.onload = function() {
   mImg0_2.src = URL0 +'&'+(mCount++);
   mCtx.drawImage(mImg0_1, 0, 0, mWidth/3, mHeight);
};

mImg0_2.onload = function() {
   mImg0_1.src =URL0+'&'+(mCount++);
   mCtx.drawImage(mImg0_2, 0, 0, mWidth/3, mHeight);
};


mImg1_1.src = URL1 +'&'+(mCount++);
mImg1_1.onload = function() {
   mImg1_2.src = URL1 +'&'+(mCount++);
   mCtx.drawImage(mImg1_1, mWidth/3, 0, mWidth/3, mHeight);
};

mImg1_2.onload = function() {
   mImg1_1.src =URL1+'&'+(mCount++);
   mCtx.drawImage(mImg1_2, mWidth/3, 0, mWidth/3, mHeight);
};


mImg2_1.src = URL2 +'&'+(mCount++);
mImg2_1.onload = function() {
   mImg2_2.src = URL2 +'&'+(mCount++);
   mCtx.drawImage(mImg2_1, 2*mWidth/3, 0, mWidth/3, mHeight);
};
```

```
    mImg2_2.onload = function() {
        mImg2_1.src =URL2+'&'+(mCount++);
        mCtx.drawImage(mImg2_2, 2*mWidth/3, 0, mWidth/3, mHeight);
    };

}

function resize_canvas(){

    if(window.innerWidth < window.innerHeight){
        isPortrait = true;
    }else{
        isPortrait = false;
    }

    mWidth = window.innerWidth;
    mHeight = 9*mWidth/16/3;

    mCanvas.width = mWidth;
    mCanvas.height = mHeight;

    imageSetup();
}


function roombaForward(){
        socket.send("w0100");
}

function roombaBackward(){
        socket.send("x0100");
}

function roombaRight(){
        socket.send("d0100");
}
```

```
function roombaLeft(){
	socket.send("a0100");
}

function roombaStop(){
	socket.send("s");
}

function roombaActivate(){
	socket.send("A");
}

function servoMove(val){
	socket.send("S"+val);
}
```

車載用広角レンズを用いた室内見守りシステムの開発

指導教員　金丸　隆志　准教授　　副指導員　新井　敏夫　教授　　桂　晃洋　教授
G1-11022 加藤　諒　G1-11039 司馬　聖大　G1-11046 高橋　勇磨
G1-11050 津田　郁也 G1-11062 原田　真吾　G1-11065 福山　陽太

1.　緒言

近年スマートフォンやタブレットの普及が進んでおり、スマートフォンと連動している製品が目立つようになってきた。

市販されているネットワークカメラは、決められた範囲しか見られないので、死角が発生してしまう。そこで、ロボット掃除機ルンバに車載用広角レンズを用いたカメラを取り付ければ、より広いエリアをカバーできるのではないかと考え、今回の研究に至った。

2.　カメラの作製

分解した市販品のウェブカメラのセンサと、本研究で用いる車載用広角レンズを組み合わせ、カメラを作製した。

図 1-1 作製したカメラ

3.　室内見守りシステムの開発

室内見守りシステムの開発にあたり、要求機能としてルンバを操作することと、カメラの角度を調節することの 2 つが上げられる。また、360°のパノラマビューを作製することで、防犯性が向上し、目標物を探す手間が省けると考えた。

4.　室内見守りシステムの概要

図 1- 1 は室内見守りシステムの概要図である。タブレットからインターネットの回線を使いルンバを遠隔操作する。

図 1-2 イメージ図

4.　実験

我々はルンバの上に 3 台のカメラを取り付けた台座とフロアを載せタブレットから無線で遠隔操作する実験を行った。結果、ブラウザから無線でルンバのコントロール、サーボモータを利用したカメラの上下の駆動、パノラマビューの映像出力に成功した。

図 1-3 実験

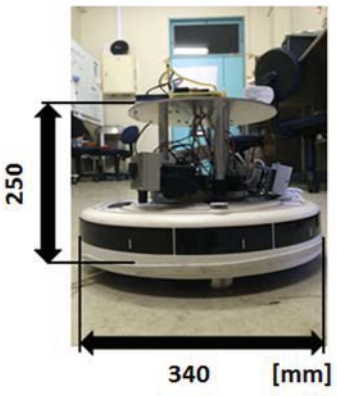

図 1-4 機構

5.　結言

(1)　ルンバに取り付けるカメラの台座とフロアを作製した。

(2)　ルンバとサーボモータを制御するプログラムを作製した。

(3)　ブラウザから操作する通信システムを作製した。

(4)　3 台のカメラの映像でパノラマビューシステムを作製した。