コンピュータ理工学特別研究報告書

題目

Raspberry Pi を用いた模型自動車の制御

-光切断法による障害物検知と自動運転-

学生証番号　244659

氏名　竹内貴哉

提出日　平成 29 年 1 月 31 日

指導教員　蚊野　浩

京都産業大学
コンピュータ理工学部

要約

本研究では，模型自動車と Raspberry Pi，カメラモジュール，レーザスリット光を照射する光源，を組み合わせ，自動運転を模擬する車とした．レーザ光を車体のおよそ 15cm 前方に照射し，その反射をカメラで観察することで障害物を検出する．検出した状態に応じて模型自動車を自動運転させた．その結果，14 号館 2 階の廊下を一周させることができた．

レーザスリット光を使った障害物検知では，まず，反射光の赤色成分を二値化する．このとき，明るい赤，中間の赤，暗い赤の 3 つに分けて検出し，それらの論理和をレーザ光の反射として検出した．その結果，路面の状態や環境光の変化に対応できるようになった．しかし，環境光が強すぎる場合には，肉眼でも反射光を確認することが難しくなり，検出できなかった．

障害物の検知は，レーザ光の検出結果を画像の左・中央・右の 3 領域に分けて判断した．判断の方法は，レーザ光を検出した位置をレーザ光の初期位置と比較することで行った．また，3 つの領域の検知結果の組み合わせから，障害物の場所と大きさを特定した．

運転アルゴリズムは，障害物検出の結果を用いて，次のようにした．

(1) 左（右）に障害物がある場合，右（左）方向にステアリングを動かし，回避する．
(2) 左（右）に大きな障害物がある場合，停止し，後退する．その後，およそ 30 度右（左）方向に回避する．
(3) 前方中央のみに障害物がある場合，右に大きな障害物がある場合と同じ動きをする
(4) 前方すべてに障害物がある場合も同様であるが，およそ 60 度右折して回避する．
(5) 3 回連続で右左折を繰り返した場合，左方向に後退することで車体を 90° 右に向ける．

実験の結果，廊下を一周させることができた．しかし，毎回確実に一周するわけではなく，無駄な動きも多い．右折を行うべき場所で左折したり，環境光によって動作が不安定になるという問題が残った．

目次

1章 序論

自動車の自動運転技術が注目されている．自動運転のレベルを大きく分けると，ドライバーの運転支援にとどまるものと，完全な自律走行がある．運転支援は，ドライバーの認知・判断・操作といった運転操作を補助する機能である．一例は，前方車両との車間距離に応じて自動的にブレーキ操作を行う衝突被害軽減ブレーキである．自律走行は，運転支援を極限まで高度化したもので，ドライバーを必要とせず，車だけで自動的に走行する．

自動運転の処理を認知・判断・操作に分けることができる．認知とは，LiDAR(Light Detection and Ranging，レーザレーダ)等の様々なセンサを用いて，歩行者や車両，路面状況等を的確に検出し，周囲の状況を理解することである．判断とは，認知の結果から次に実行すべき行動を決定することである．操作とは，認知の結果から，自分が走行すべき軌道を算出し，その軌道に沿って自車を走行させることである．

本研究は，模型自動車を自動車に見立て，自作LiDARセンサを用いて，自律走行させるものである．本研究の目的は，障害物検出とそれに基づく自動運転の基本的な技術を自作することによって，画像処理技術，組み込みシステム技術，自動運転に利用されるコンピュータ技術の基礎を取得することである．その中でも，安定して障害物を検出することができ，本学の14号館２階の廊下を一周することができる画像処理技術を開発することが，本研究の主たる目標である．

今回の研究のために実車を使うことは容易でない．そこでRaspberry Piにカメラモジュールを接続したものを，車載カメラと車載コンピュータに見立てた．実車のモデルとして，TAMIYA社のラジコンカーに自作の加工を加えたものを利用した．LiDARセンサとして，模型自動車の前方上部に横向き一直線に赤色の光を照射するレーザポインタを利用した．そして，車載カメラとLiDARセンサを用いて，障害物を検出し，自動的に回避するプログラムを開発した．

本論文は，２章で電動模型自動車の接続について説明し，３章で障害物検知手法について説明する．次いで，４章では運転アルゴリズムと実際に障害物を回避するかの実験と，その結果と考察を述べる．５章で結論を述べる．

2 章　自動運転技術の現状

自動運転技術に関する研究として，1986 年頃から米国カーネギメロン大学で進められた Navlab が有名である．Navlab は 1995 年に自動運転による米国横断を試み，3,000 マイルの行程の 98%以上の自動運転に成功した．この時の自動車は，ポンティアックのミニバンにコンピュータとして Sparc LX クラスのワークステーションを搭載していた．自動運転のためのセンサとしては，カラーカメラ，GPS センサ，ジャイロセンサを利用した．

2004 年から 2007 年には，米国で DARPA グランドチャレンジというロボットカーレースが実施された．そして，この成果を引き継ぐような形で Google が自動運転技術の研究開発と実用化を進めた．Google はいくつかの試作車を開発した．2014 年に発表した図 1 のものは，ハンドルもブレーキもアクセルもない電気自動車である．車体の上に設置されているのはレーザスキャナー（あるいは LIDAR）と呼ばれる距離画像センサで，車体の近くにある障害物を検出することができる．Google は 7 年間にわたり自動運転の開発プロジェクトを進めたが，ごく最近（2016 年 12 月），自社では自動運転カーを事業化しないことを発表した．

図 1 Google の自動運転カー

グーグルが進めた完全自動運転は，技術的な課題の解決や法整備が必要であり，実用化の時期がはっきりしない．一方，自動運転のための要素技術を見ると，自動車メーカーが着実に実用化を進めている．すでに広く実用化されている衝突被害軽減ブレーキは，前方車両との車間距離を計測し，その変化に基づいてブレーキを制御するシステムである．また，2016 年から日産が自動運転技術を大きく宣伝している．これは，高速道路で同じ車線内を走る場合において，自動車がハンドル，ブレーキ，アクセルを自動的に

制御するものである．ただし，ドライバはハンドルから手を離すことはできない．

米国運輸省道路交通安全局（National Highway Traffic Safety Administration: NHTSA）は自動運転のレベルを表１のように定義している．これによると，衝突被害軽減ブレーキはレベル１，日産が製品化した技術はレベル２である．なお，日産の技術では，環境を認識するためのセンサとしてカラーカメラを利用し，その映像を画像処理することで自動車の制御情報を得ている．

表 1 NHTSA による自動化のレベルの定義

レベル	概要
1	操舵，制動，加速等の機能が独立に自動化されている．
2	操舵，制動，加速等の2つ以上の機能が協調的に自動化されている．
3	操舵，制動，加速等の全てを自動的に行うが，緊急時のみドライバが対応する．
4	操舵，制動，加速等の全てを自動的に行い，ドライバは関与しない．

自動運転を実現するためには，人間が行っている認識，判断，操作をコンピュータに代替させる必要がある．自動運転を行うためにはセンサが必須である．代表的な環境センサとその概要を表2に示す．代表的な環境センサは4つあり，それぞれに役割がある．

カメラは，色や文字を認識でき，標識や道路表示などを読み取ることができる．単眼カメラとは，１台のカメラで連続的な静止画を観察し，その図形の変化から対象の位置を検出するものである．ステレオカメラは，単眼カメラよりコストがかかるが，２台のカメラの画像を重ねて，位相差を検出することで，対象の正確な位置を検出する．

LIDAR はレーザ光を物体に照射して，方向と跳ね返ってくる時間を計算し，空間上の物体の位置や形状を特定するものである．周辺の物体との相対的な距離や位置関係が把握可能になり，周囲の自動車や歩行者などの物体を３次元形状として捉え，人間や自動車などの障害物情報が正確である．

ミリ波レーダは，発せられた電波の反射波を測定し，物体の位置と速度を検出するものである．他のセンサに比べ，雨や霧，逆光などの影響を受けにくいことから，視界の効かない夜間や悪天候時に強いという特徴がある．

このようにそれぞれのセンサには，得意・不得意があり，お互いを補い合っている．

表 2 自動運転に利用する代表的な環境センサ

センサ	原理
単眼カメラ	1台のカメラで撮影した映像を処理することで，車線や障害物，前方車両までの距離などを認識する．
ステレオカメラ	2台のカメラで撮影した映像を処理することで，車線や障害物，前方車両までの距離などを認識する．
LIDAR	LIDAR: Light Detection and Ranging．レーザービームを照射し，その反射時間から距離を得る．
ミリ波レーダー	波長が1～10mmのミリ波を照射し，その反射時間から前方までの距離を得る．

レベル 1・2 の自動化はそれぞれのセンサを単独で利用することで実用化が可能である．レベル 3 以上の自動化は，これらのセンサを複合的に利用することが必要と考えられる．特に，重要になるのは LIDAR である．なぜならば，このセンサを用いることで，車両周囲の 3 次元的な情報を詳細に得ることが可能になるからである．

3 章　研究に用いた模型自動車システム

本研究では，自作の組み込みシステムで市販の模型自動車を制御することで，自動運転の実験を行った．組み込みコンピュータとして PIC マイコンと Raspberry Pi を利用した．模型自動車はタミヤ RC カーのシャーシ MF-01X（図 2）を利用した．これを通常の RC カーとして利用する場合には，電波受信機から前輪用サーボモータ（前輪の方向角度を変える）と ESC（Electric Speed Controller）に PWM 信号を送り，車体の動きを制御する．今回は，前輪用サーボモータと，前進・後退速度を制御する ESC に送る 2 組の信号を PIC マイコンから制御する．そして，Raspberry Pi から PIC マイコンに I2C インタフェースを介して指令を送ることで，模型自動車を動作させる．

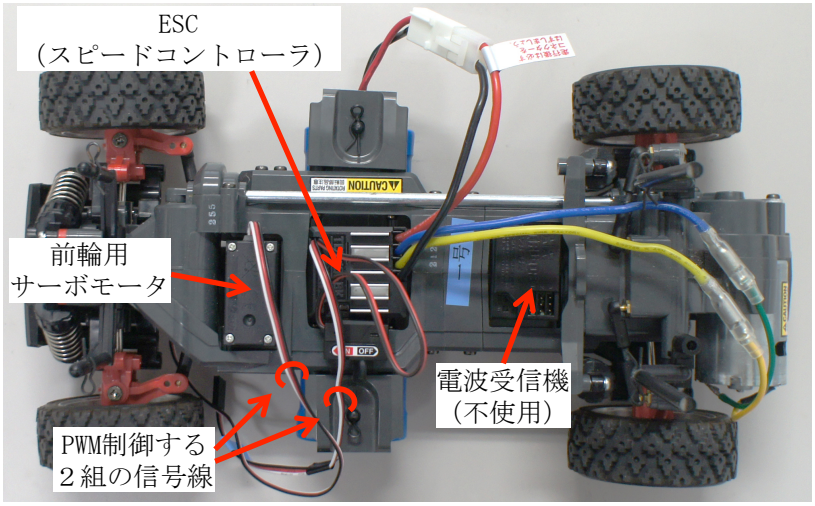

図 2　タミヤ RC カーのシャーシ MF-01X

サーボモータと ESC に加える信号は，いずれも，図 3 のような PWM 制御のための周期的矩形波である．ここで，パルス幅は概ね 1.5msec を中心にして 1.1msec～1.9msec の範囲で制御するようになっている．ステアリング（前輪の方向角度）を制御する場合，1.5msec のパルス幅で前輪の方向角度が中央になる．1.1msec あるいは 1.9msec のパルス幅で左右に最大角度になる．前進・後退の駆動制御に使う場合，1.5msec のパルス幅でニュートラル状態になる．1.5msec よりも短くすると前進，1.1msec で最大の回転数で前進，1.5msec よりも長くすると後退になる．なお，パルス電圧の大きさは 6.0V 程度である．

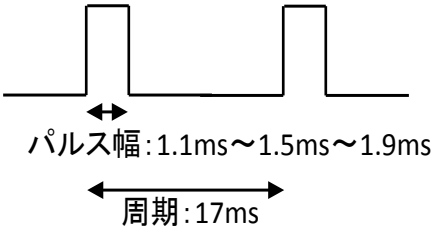

図 3　サーボモータと ESC に加える PWM 信号

図 3 の PWM 信号を生成するために PIC マイコンを利用した．図 4 に PIC マイコンと RC カーの接続を示す．PIC マイコンには PIC16F819 を利用した．PIC マイコンのプログラムを付録 1 に示す．また，図 4 に示すように，PIC マイコンは I2C インタフェースを介して Raspberry Pi とも接続されており，PWM 信号を適切な状態に設定するためのコマンドを入力することができる．このように構成することで，Raspberry Pi で開発したプログラムから模型自動車を制御することが可能になった．

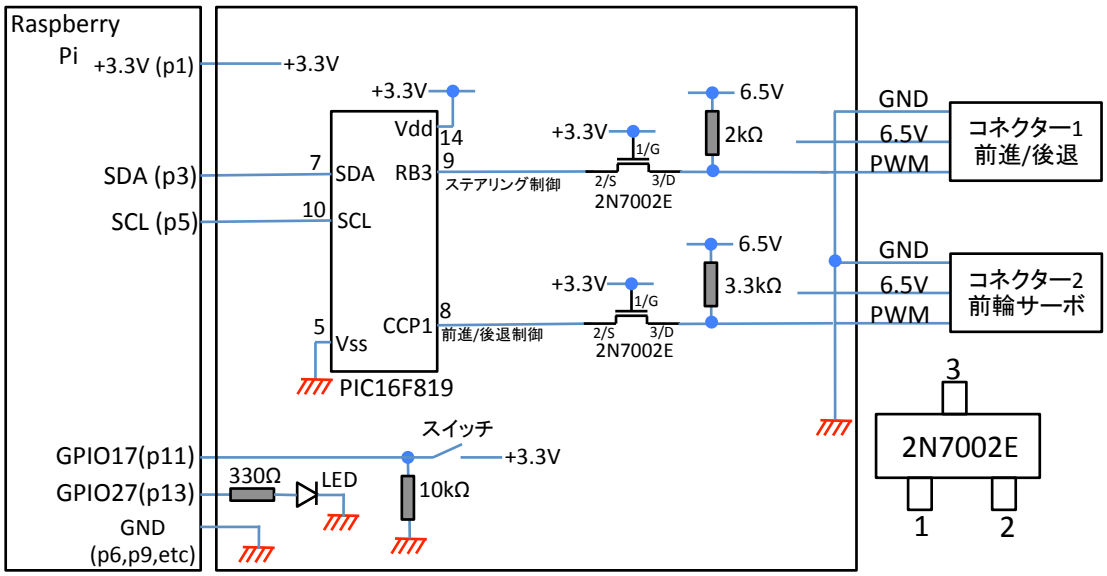

図 4　Raspberry Pi と RC カーの ESC とのインタフェース回路

Raspberry Pi（図 5）は，名刺サイズのコンピュータで，Linux 系の OS が動作する．ディスプレイやネットワークに接続することで，通常のパソコンとして利用可能である．また，汎用入力端子(GPIO)を用いて回路や装置を接続することが容易で，組み込み機器を制御することができる．今回は一組の GPIO 端子を I2C インタフェースの信号線として利用して，PIC マイコンと接続した．

Raspberry Pi は専用のカメラモジュールが販売されている．５メガピクセルの画像センサとレンズから構成されており，2592×1944 画素の静止画と，1920×1080 の動画

を記録することができる．今回の研究では，カメラとシート状のスリット光を発するレーザ光源を組み合わせることで，簡易的な LIDAR センサとした．レーザ光源は出力が 5mW，電圧が 3V，波長が 630～650mm で寸法が直径 12mm，長さ 35mm の赤色レーザを利用した．

図 6 に模型自動車システムを構成する要素の接続関係を示す．図 7 に模型自動車の外観と動作状態の模式図を示す．Raspberry Pi のプログラムは C/C++で開発した．画像処理ライブラリとして OpenCV を利用した．

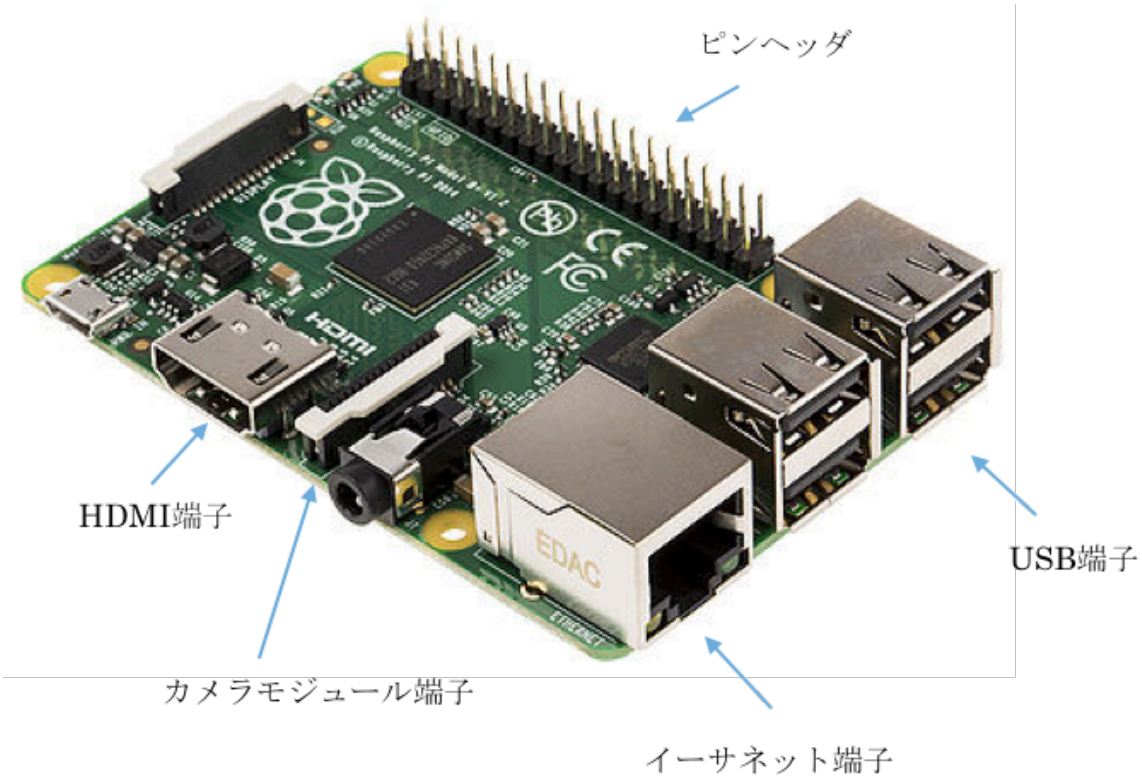

図 5　Raspberry Pi の外観

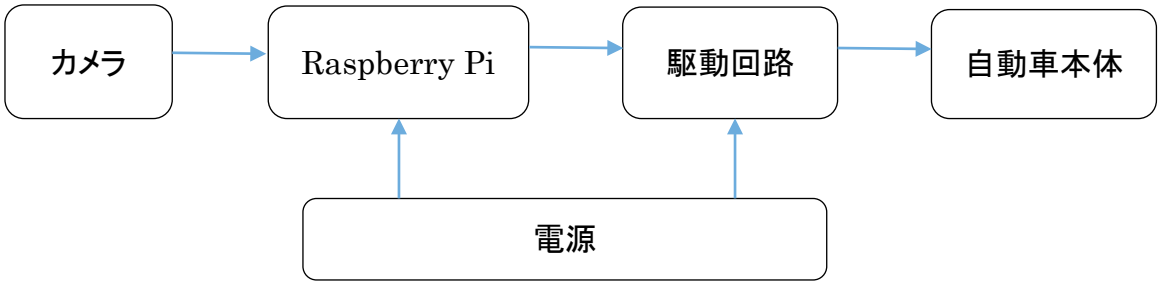

図 6　模型自動車を構成するシステムの要素

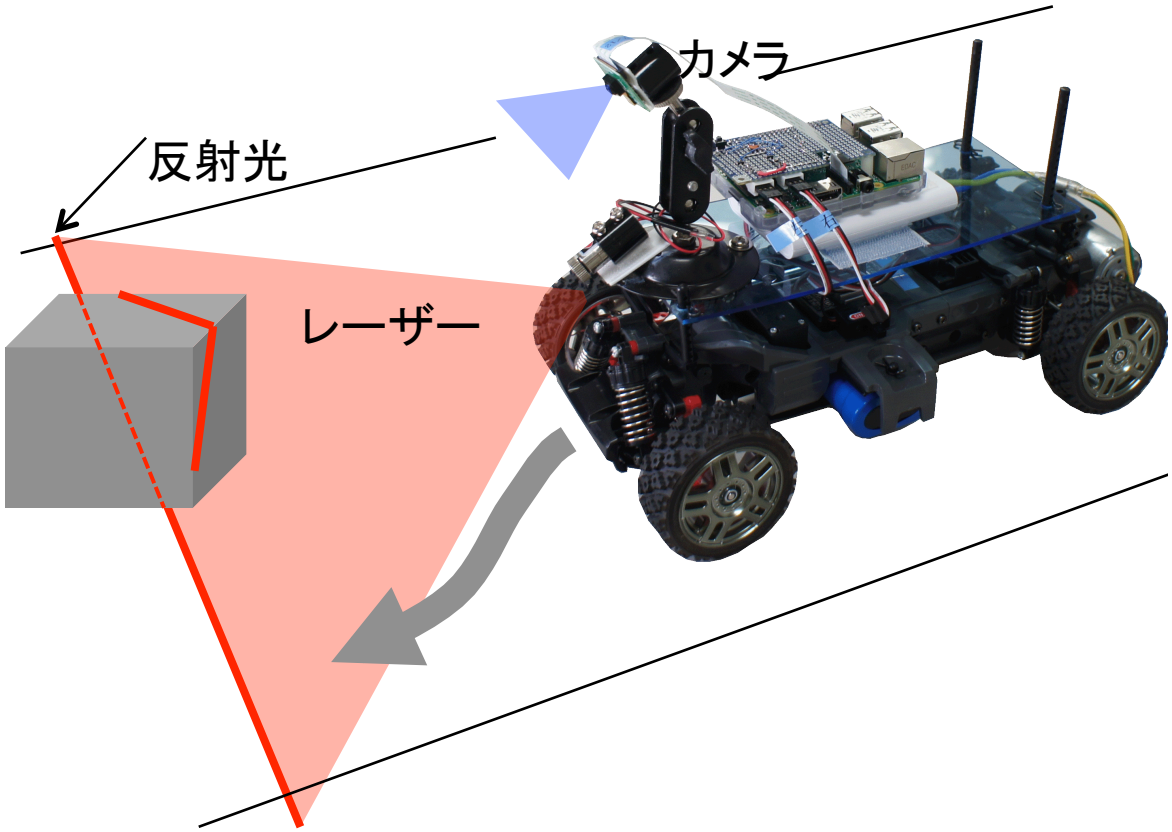

図 7　模型自動車システムの外観と動作状態の模式図

4 章　障害物の検知手法

4.1　提案手法の概要

本研究で開発した障害物の検知手法の概要を説明する．図 7 のように，模型自動車にカメラモジュールとレーザポインタを取り付ける．レーザ光は模型自動車のおよそ 15cm 前方を照射する．カメラモジュールで車両の前方を撮影し，路面に照射されるレーザ光の状態を 640×480 画素の画像として入力する（図 8）．そして，レーザ光が照射した位置を画像処理で抽出する．

画像処理は，あらかじめ指定した範囲の赤色の画素を二値化するものである．その詳細は 4.2 で述べる．この時，ROI(Region of Interest，関心領域)処理を行い，処理の負荷を軽減する．ROI はフィルタ処理やその他の操作の対象として選択する領域で，OpenCV で指定できる領域は短形のみである．今回の ROI を図 8(a)の青枠で示す．

路面に障害物がない場合，レーザ光は横一直線の定まった場所に照射される．その位置を抽出し初期位置として記憶する（図 8(a)の状態）．走行時は，制御のループごとにレーザ光の位置を抽出する（例えば，図 8(b)のように）．そして，現在のレーザ光の位置と初期位置を比較することで，障害物の位置を推定する．推定結果に基づいてステアリングと前進・後退・ストップを制御する．自動運転を実現するプログラムの流れを図 9 に示す．

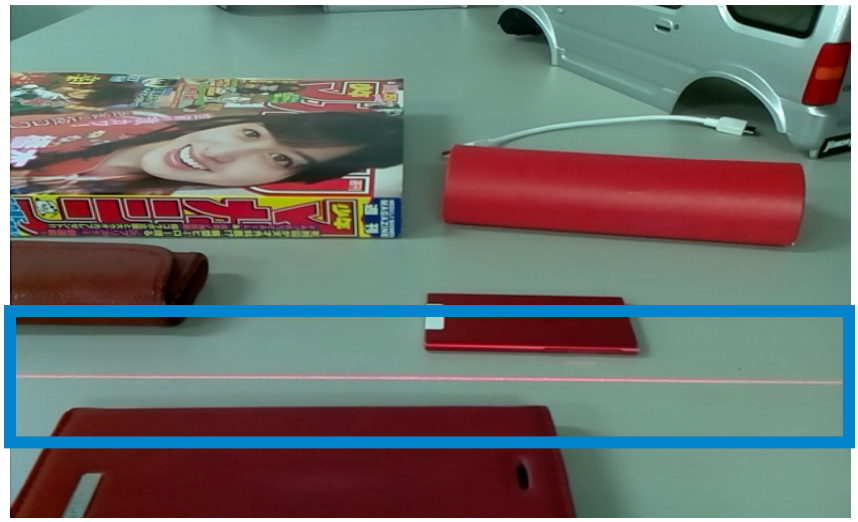

図 8(a)　入力画像の例と ROI(x:0～640，y:340～440)

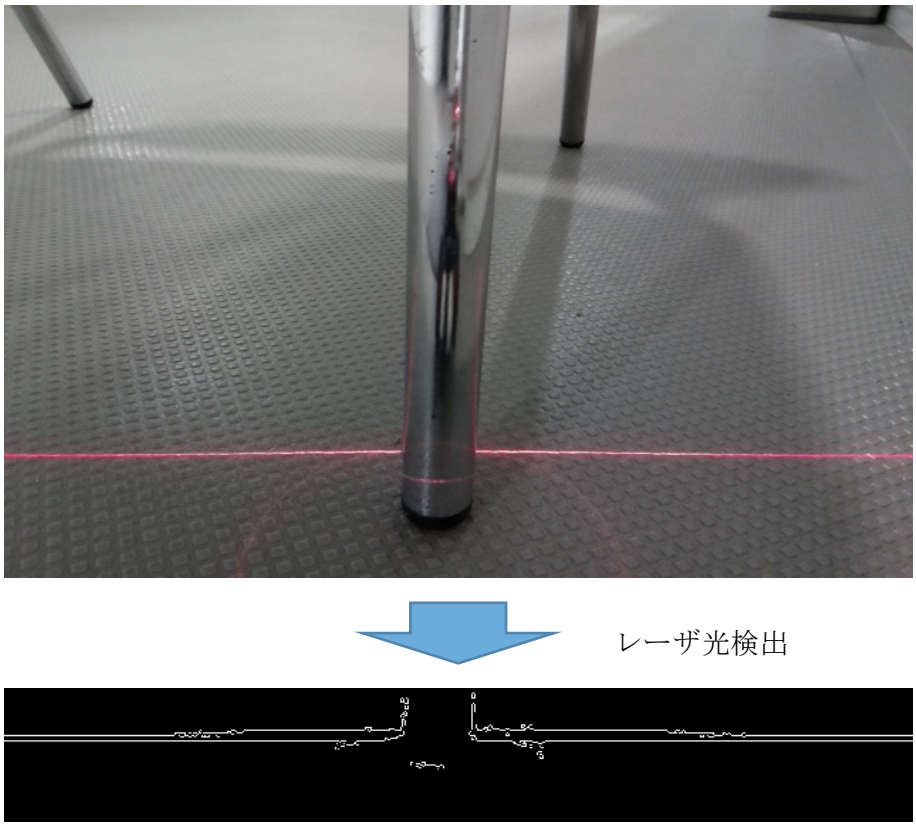

図 8(b)　入力画像の例（障害物有り）

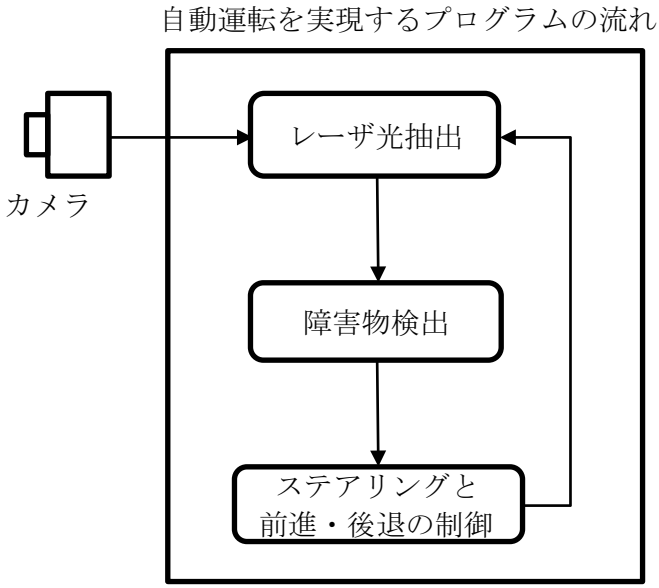

図 9　自動運転を実現するプログラムの流れ

4.2　レーザスリット光の検出

入力画像からレーザスリット光が照射した位置を抽出するための画像処理を説明する．

まず，入力画像の画素値を RGB から HSV に変換する．HSV を使うことで，色の範囲・色の濃さ・色の明るさを独立に指定することができる．

レーザの反射光は見た目には鮮やかな赤色であるが，被写体の色によって HSV 値が変化する．想定した走路におけるさまざまな条件で反射光を観察した結果，レーザ光を精度良く検出するためには，反射光の明るさを３段階に分け，それぞれの明るさごとに H 値（色相）と S 値（彩度）の範囲を指定することが有効であった．定性的に述べると，いずれの場合も色相の範囲は同程度であったが，反射光が明るい場合，彩度は低い傾向があり，反射光が暗い場合，彩度は高い傾向があった．図 10 に，明るい赤を赤色，中間の赤を緑色，暗い赤を青色で示した画像を示す．図 10(a)は，暗闇にレーザ光を照射した場合である．処理後の画像を見ると，中心部の白い部分を明るい赤として検出しており，その外側に中間の赤，さらに外側に暗い赤を検出している．図 10(b)は，壁にレーザ光を照射した場合である．処理後の画像を見ると，中心部が明るい赤，その外側が中間の赤として検出されており，暗い赤はほぼ検出されなかった．

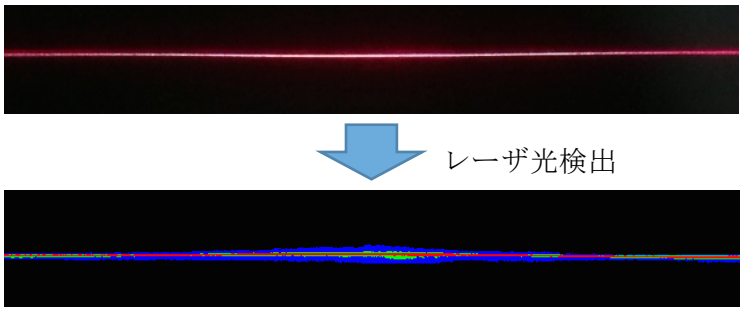

図 10(a)　3 段階のレーザ光検出

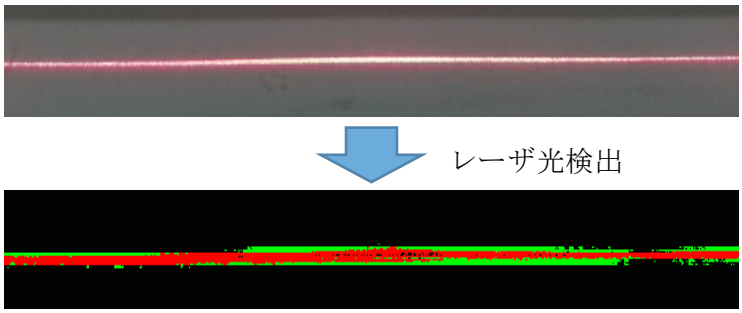

図 10(b)　3 段階のレーザ光検出

最終的に，今回の走行環境に最適と判断した HSV 値の範囲を表 3 に示す．それぞれの範囲の論理和をレーザ光が照射した画素として二値化した．

表 3　各色の HSV 値（H:0～180，S:0～255，V:0～255）

明るい赤	H:144～36，S:25～255，V:153～255
中間の赤	H:144～36，S:51～255，V:102～153
暗い赤	H:144～36，S:102～255，V:51～102

今回の処理で抽出する色の範囲を確認するために，カラーチェッカーを入力画像として処理した．その結果を図 11 に示す．純粋な赤だけでなく，橙，茶，黄，黄緑，等の色も認識している．反射光の色は，物体の色によってある程度広がる．したがって，この結果は妥当であると考えられる．

図 11　カラーチェッカーの画像を二値化した結果

図 11 のままでは抽出される領域が広いので，Canny 法によるエッジ検出を行った．Canny 法は，ノイズを削減しつつ画像の輝度勾配を見つけ，非極大値の抑制とヒステリシス効果を使ったエッジ検出法である．その結果を図 12 に示す．

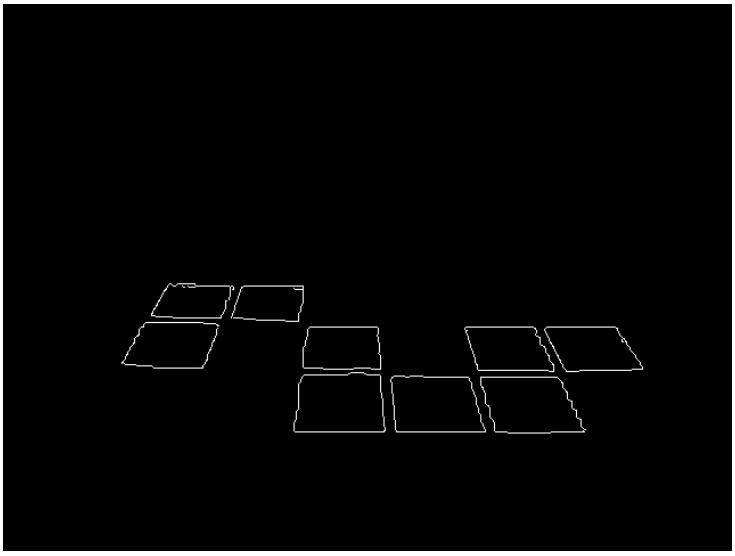

図 12　図 11 をエッジ検出した画像

実際にレーザ光を照射した場合の入力画像とその結果を図 13 に示す. 図 13 を見てわかるように，概ね正しく反射光を検出している.

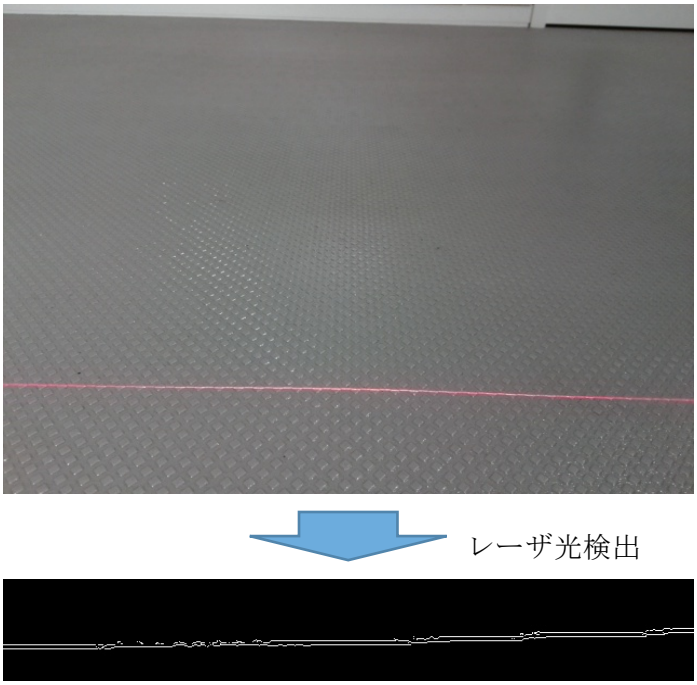

図 13　廊下でレーザ光を照射した入力画像と処理画像

4.3　障害物の判断

4.2 の手法で抽出したレーザ光の位置を，あらかじめ測定した初期位置と比較することで，障害物の位置を推定する．今回は，比較的大きな障害物を回避して走行させることを想定して，図 14 に示すように，ROI 内を 3 つの領域（領域 A・B・C）に分けた．それぞれの領域は，左から x 座標が 10～240，230～410，400～630，y 座標はいずれも 20～70 の範囲の領域となっている（ROI の領域は x 方向が 640 画素，y 方向が 100 画素）．

領域 A の処理について説明する．領域 A 内を，初期位置±12 の領域とそれ以外の領域にわける．これらの領域内の画素値を調べ，その値が 255 の場合カウントする．そのカウント数を比較することにより，障害物の有無を判断する．以後，初期位置±12 の領域を(1)，それ以外の領域を(2)と記す．

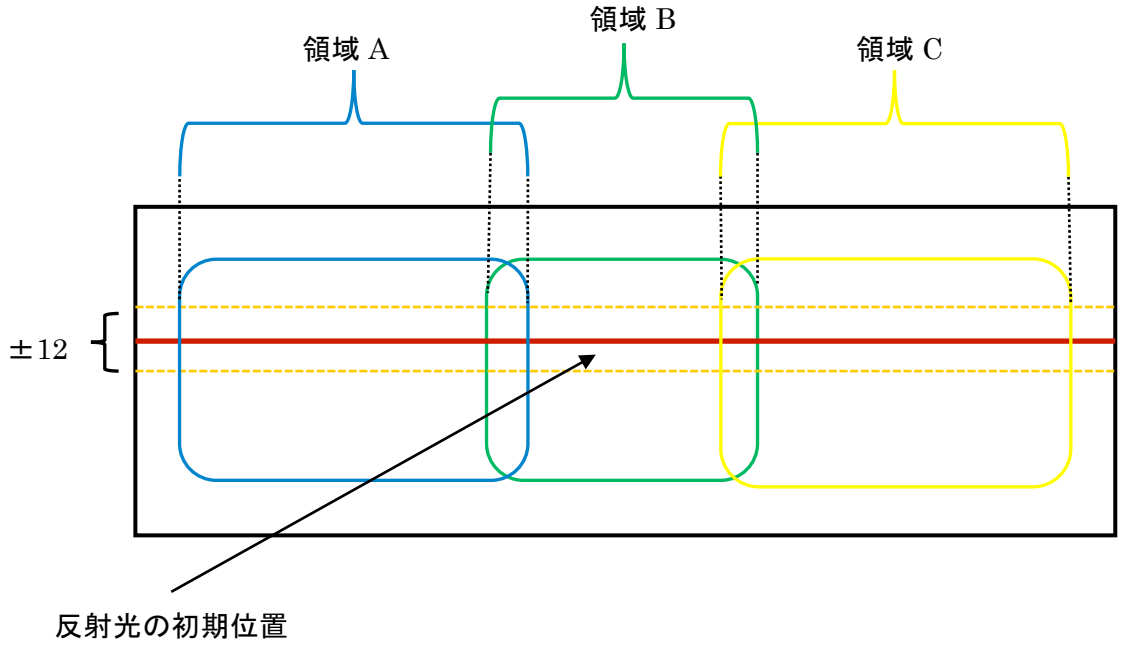

図 14　障害物の検知を行う 3 つの領域

障害物がない場合，反射光は (1)に存在する．したがって，(1)内のカウント数が多く，(2)内のカウント数は少ない．障害物がある場合，原則的には(1)内のカウント数が少なくなり，(2)内のカウント数が増える．しかし，障害物がある場合の反射状態は不安定であり，(1)内のカウント数が十分に小さくならないこともある．

このように，(1)内のカウント数が一定数以上で(2)内のカウント数が一定数以下の場合に障害物がないと判断し，(1)内のカウント数が一定以下，または，(2)内のカウント数が一定数以上の場合に障害物があると判断する．

これらの処理を領域 B，C にも同様に行う．障害物の位置の特定は，3 領域ごとの障害物の有無により判断する．３つの領域内の障害物の有無の組み合わせは８通りある（表 4)．表 4 の 0 が障害物無し，１が有りを示し，(A，B，C)は，領域 A（左)，領域 B（中央)，領域 C（右）を示す．

表 4　障害物の場所の特定

1:(A，B，C)=(0，0，0)	障害物無し
2:(A，B，C)=(0，1，0)	真ん中のみに障害物有り
3:(A，B，C)=(0，1，1)	右側大部分に障害物有り
4:(A，B，C)=(0，0，1)	右側に障害物有り
5:(A，B，C)=(1，0，0)	左側に障害物有り
6:(A，B，C)=(1，1，0)	左側大部分に障害物有り
7:(A，B，C)=(1，1，1)	前方全てに障害物有り
8:(A，B，C)=(1，0，1)	左右に障害物有り

表 4 のように障害物の場所を特定する．ただし今回の研究では，表 4 の 8 は 7 と区別しないで制御した．

5 章　運転アルゴリズムの開発と自動走行実験

5.1　運転アルゴリズム

障害物検知の結果を用いて走行状態を制御する．前輪タイヤのステアリング角度を制御する数値は，中央を 0，向かって左を正の値，向かって右を負の値で表す．稼動域の数値は-15〜15 の範囲である．また，前進後退を制御する数値は，停止を 0，前進を負の値，後退を正の値で表す．稼動域の数値は-30〜-12，12〜30 で，絶対値が大きくなれば高速に動く．-11 から 11 の数値では車体が動かない．なお，本研究で使用した模型自動車は，直進させても少しずつ左に寄っていく傾向がある．

基本的な制御ルールを次のようにした．

表 4 の 1: 障害物がない場合．最も遅い速度（-12）で直進する．

表 4 の 2: 中央だけに障害物がある場合．一旦停止し，200ms 後退させて，停止する．これによって急ブレーキをかける．その後，障害物を避けるのに十分な距離を確保するために後退する．そして，ステアリングを右に最大限(-15)きり，およそ 30 度右折して回避する．（表 4 の 6 の動作と同じ）

表 4 の 3: 中央から右側にかけて障害物がある場合．一旦停止し，200ms 後退させて，停止する．これによって急ブレーキをかける．その後，障害物を避けるのに十分な距離を確保するために後退する．そして，ステアリングを左に最大限（15）きり，およそ 30 度左折して回避する．

表 4 の 4: 右側だけに障害物がある場合．ステアリングを左に最大限（15）に切り，速度を少し上げる（-13）．

表 4 の 5: 左側だけに障害物がある場合．表 4 の 4 の動作を左右対象にしたもの．

表 4 の 6: 中央から左側にかけて障害物がある場合．表 4 の 3 の動作を左右対象にしたもの．

表 4 の 7: 前方全てに障害物がある場合．表 4-6 とほぼ同じであるが，およそ 60 度右折して回避する．

表 4 の 8: 左側と右側に障害物がある場合．表 4-7 と同じ．

上の制御ルールだけでは，廊下の角に追いやられてしまうと脱出できなくなる場合がある（5.3 で詳細を述べる）．その解決方法として，表 4 の 2，3，6，7 のどの組み合わせでも，3 回連続で行われたとき，一旦停止し，ステアリングを左 (15) にきりながら，後退させた．これらのアルゴリズムで，障害物を回避しながら，廊下を走行させる．

5.2　障害物検知の実験結果

提案手法を検証するため,実際に廊下を自動走行させた.廊下の様子を図 15 に示す.

図 15　自動走行させた廊下の様子

図 15 を見てわかるように，主な障害物はゴミ箱，マット，傘立て，パイプ椅子，壁である．提案した制御手法によりゴミ箱，傘立て，パイプ椅子，壁を回避することができた．実際に障害物を検出した例を図 8(b)と図 16 に示す．しかし，マットを回避する動作には若干の問題が生じた.

図 8(a)はパイプ椅子の脚を障害物として検出している．この場合，画像の中央付近に障害物があり,表 4 の 2 に対応したルール(後退し,ステアリングを右にきって前進)適用して制御した．図 16 は廊下の右側にある壁を障害物として検出した例である．画像の右側がレーザ光の初期位置から外れている．この場合，表 4 の 4 に対応したルール（ステアリングを左にきる）を適用して制御した.

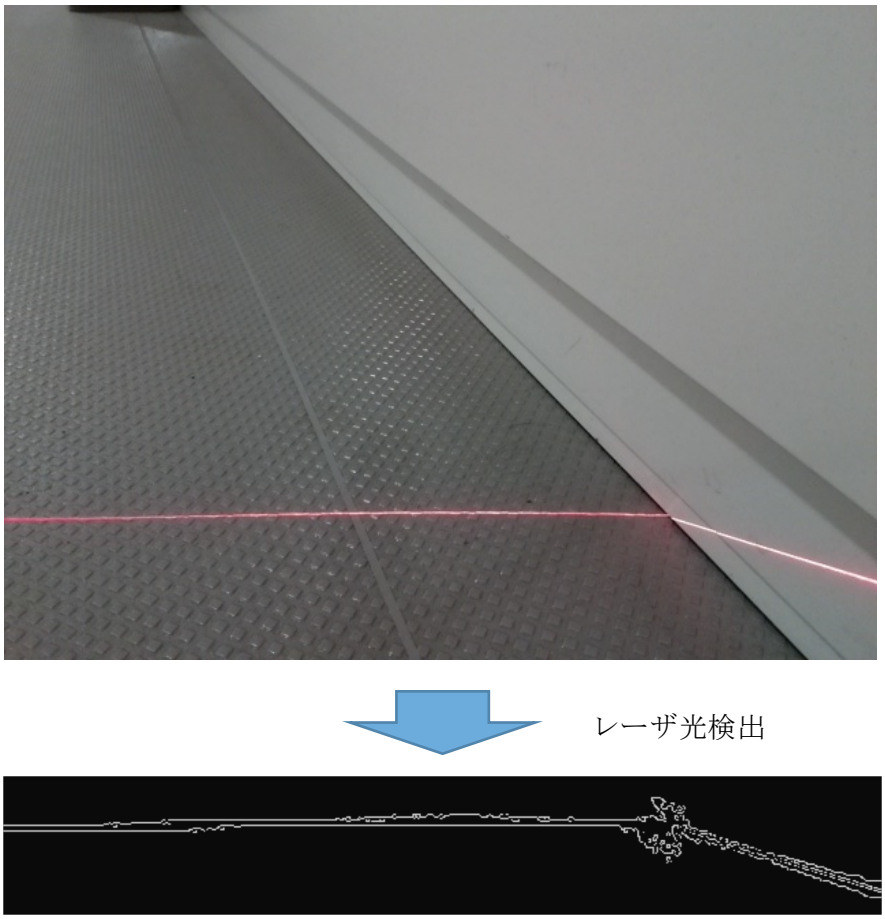

図 16　右側の壁を障害物として検出した例

黒いマットは厚みが薄いので，回避するのではなくそのまま進むことができる．しかし，路面が廊下から黒いマットに変わるところで制御が不安定になる場合があった．

詳しく解析すると，黒いマットの縁部分にレーザ光が当たったとき，廊下表面の凹凸に光が散乱し，レーザ光以外のものを検出していた．その様子を図 17 に示す．また，レーザ光がない場合の様子を図 18 に示す．この現象は，二値化のための彩度と明度の範囲を調整することによって除去することができる．しかし，現在の設定を変更しても，他の障害物が検出できなくなるなど，問題が発生する．このように，２値化の動作が不安定になることはあるが，廊下を一周することはできるので．現在の設定を変更していない．

また，廊下には１枚だけ黄色のマットがある（図 15 の廊下左側奥）．黄色のマットは黒のマットに比べ高さがあるため，正しく障害物として認識できた．

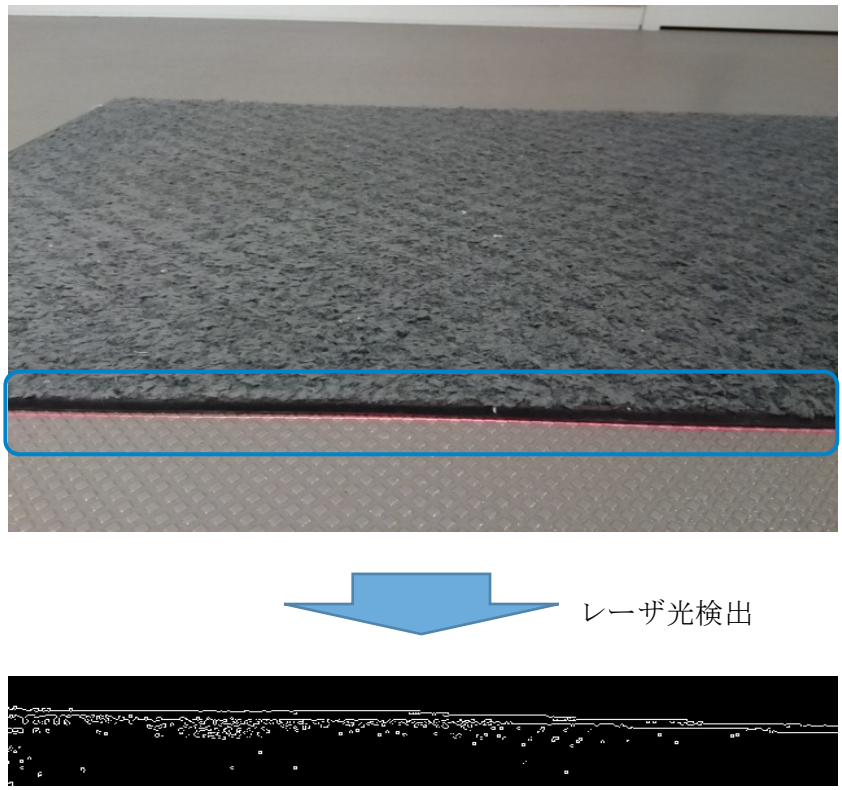

図 17　黒いマットの縁の様子

図 18　図 17 のレーザ光がない場合の様子

環境光が強い場合にはレーザ光を検出できない場合がある．環境光が強いと，レーザ光を肉眼で確認することも難しい．これを改善するには，より強い光を放つレーザポインタを用いる必要がある．

5.3　自動走行に関する実験結果

本研究の目標を，14 号館 2 階の廊下を自動走行で一周させることにした．まず，右回りに走行することを想定して実験を行った．図 19 にそのときの動作の一例を模式的に示す．

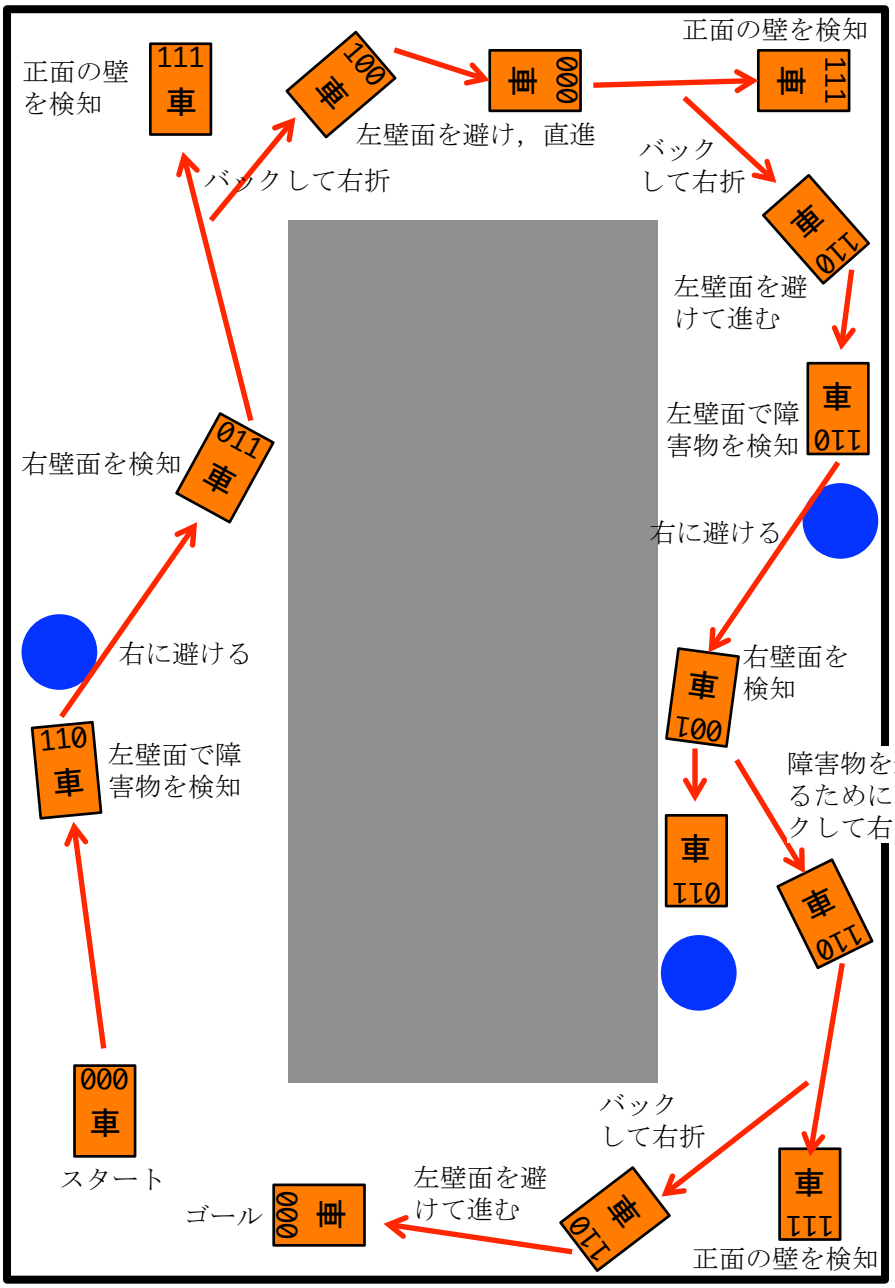

図 19　廊下を一周させた場合の動作の一例

図 19 の橙色の四角が模型自動車，青い丸は障害物である．3 つの数値は障害物の検知結果を，表 4 に基づいて示したものである．これはひとつの例だが，すべてが想定通りに進むときは，このようにして廊下を一周する．

難しい動きをする例を図 20 に示す．図 20(a)は正面と左側に壁があり，自動車がやや右から正面の壁に向かっていった場合である．図 20(a)左の状態における，障害物検知の結果は表 4 の 3（正面と右に障害物）か 7（全面に障害物）になる．正面と右に障害物と検知した場合，バックして 30° 左折する．全面に障害物と検知した場合，バックして 60° 右折する．廊下を右回りする場合には，バックして 60° 右折が正しい制御である．しかし，場合によっては，図 20(a)のようにバックして左折することがある．

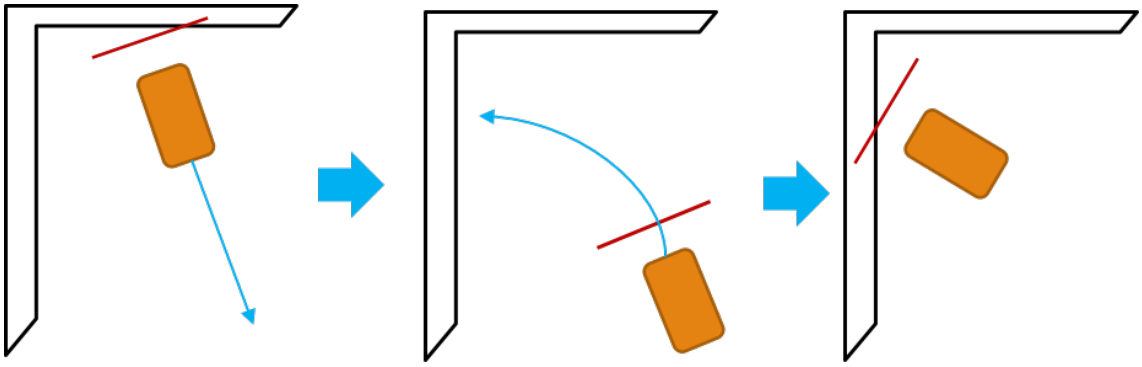

図 20(a)　正面と左に壁があり，自動車がやや右から正面の壁に向かった場合の動作例

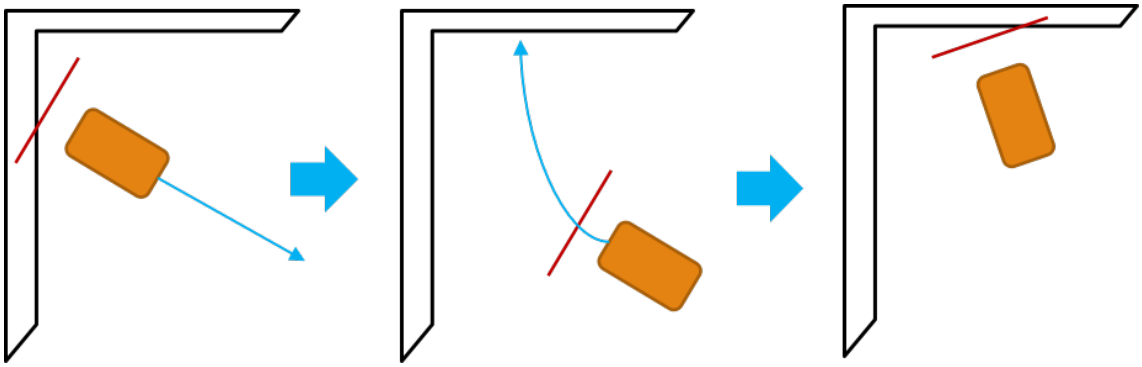

図 20(b)　図 20(a)に続く動作例

図 20(a)に続く動きを予想したものを図 20(b)に示す．図 20(b)左の状態における，障害物検知の結果は表 4 の 6（正面と左に障害物）か 7（全面に障害物）になる．したがって，いずれの場合もバックして右折する．すると，図 20(a)左の状態に戻る．この後は，図 20(a)と図 20(b)の動きを何度も繰り返してしまう可能性がある．

この状態から脱出する方法として，表 4 の 2，3，6，7 による右左折を 3 回連続して行った場合，ステアリングを左にきりながら後退させることにした．これにより，車体をおよそ 90 度右方向に向かせることができる．そうすれば，次に壁を検出しても右折を行い，想定通りの動作に戻ることができる．この様子を図 21 に示す．

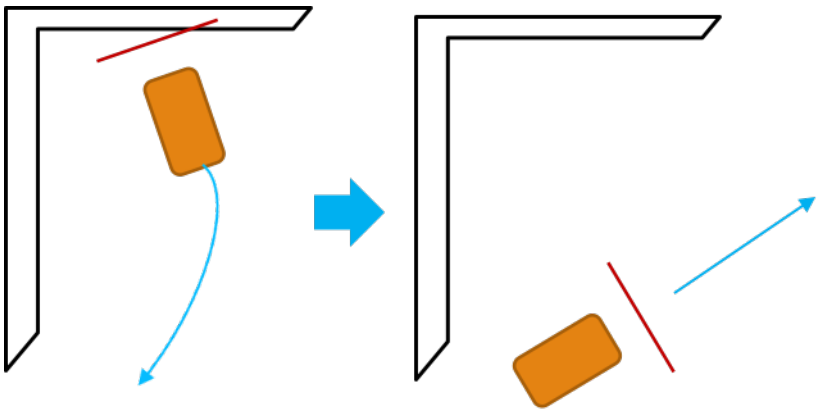

図 21　壁の角から脱出する様子

走行速度は，設定可能な範囲で最も遅くしている．それでも，障害物に衝突してしまう場合がある．これは，レーザが照射している位置が車体から近いため，障害物検知の判断とそれに応じた制御が間に合わないためである．一方，本研究で使用しているレーザ光で遠くを照射すると，カメラで十分に検出できなくなる．両者のバランスを考慮して現在の位置に設定した．強いレーザ光を用いて測定距離を伸ばすことにより，障害物検知と制御のための時間的マージンをとることができ，高速で安定した動作が可能になる．

開発したプログラムの制御ループの周期は 23fps であった．これは，１回の処理が 43msec 程度であることを示す．

5.4　考察

黒いマット以外の，廊下にある障害物を検知して回避することができた．黒いマットに合わせて２値化のパラメータを調整すると，他の障害物が検出できなくなってしまう．現状では，２値化のパラメータを固定して，全ての障害物を検知することは難しい．実際の自動車での自動運転では，あらゆる状況下で，障害物を正しく検出する必要がある．今回の研究によって，そのような技術がいかに高度で困難なものであるか，ということを実感した．

模型自動車の制御に関しては，なんとか廊下を一周させることができた．しかし，常に確実に右回りに走行し続けることはできなかった．人間が自動車を運転しているときのように，次の進路がわかれば，無駄のない制御を行うことができると考えられる．そこで，廊下を一周させる場合でも，模型自動車に廊下の構成と障害物の場所を地図として記憶させ，その情報とカメラからの情報を比較することで，より確実で無駄のない走行を可能にできるのではないかと考えた．

6章　結論

カメラモジュールを接続したRaspberry Piとレーザポインタを模型自動車に搭載し，赤いレーザ光の反射光をカメラで読み取り，その画像から障害物を検出した．検出した障害物情報を元に，自動車に見立てた模型自動車を，自動運転アルゴリズムで制御し，14号館2階の廊下を右回りに一周させることができた．

環境光が強いところでは，反射光を検出することができなかった．また，障害物を検出するタイミングと模型自動車の速度が噛み合っておらず，急停止しても止まりきれずに障害物に衝突してしまう場合があった．その他にも，車体と障害物の位置関係によって，本来右折すべきところで左折してしまう場合があった．これらのことから，最も大きな課題は，より遠くの障害物をより高精度に検出する技術を開発することであると言える．例えば，より強い光を放つレーザポインタを用いることで，障害物を検出する距離を伸ばすことができる．

本研究のようにRaspberry Piや模型自動車を用いることにより，本来なら高コストでリスクの高い実験も，低コストで安易に行えることが分かった．よって，Raspberry Piは，低コストで模擬実験等を行うことができ，さらに組み込み機器の基本技術を学ぶ上での利用価値が非常に大きい．

参考文献

[1] 野辺継男,“自動運転の開発動向,”情報処理, Vol. 57, No.5, pp.436-440, May 2016.

[2] 竹内栄二朗,“環境認識（認知）技術,”情報処理，Vol. 57，No.5，pp.441-445，May 2016.

[3] 菅沼直樹，米陀佳祐,“自動運転自動車のパスプランニング,”情報処理，Vol. 57，No.5，pp.446-450，May 2016.

[4] 加藤真平,“自動運転ソフトウエア　–オープンソースソフトウエアの利活用–,”情報処理，Vol. 57，No.5，pp.456-460，May 2016.

謝辞

本論文を作成にあたり,丁寧な御指導を賜りました蚊野浩教授に深く感謝申し上げます.

付録 1　本研究で作成した自動運転のプログラム

	内容
move.cpp	カメラモジュールで得た画像の中から，レーザ光を抽出する．そのレーザ光の状態を観察し，障害物を検出する．その結果から前進・後退，ステアリングを判断し，走行するプログラム．

付録 2　PIC マイコンのプログラム

図 22 に PIC マイコンのプログラムリストを示す．このプログラムの働きは I2C インタフェースから指令を入力することと，その指令に応じて，ESC への信号を発生することである．詳細はプログラムリストのコメントに記述しているが，92 行目から始まる無限ループで I2C からの指令を受け取る．そして，タイマー割り込みを使って ESC への制御信号を発生する．

```
1   #include <16f819.h>
2
3   // CONFIG
4
5   #fuses NOPROTECT,CCPB2,NODEBUG,NOWRT,NOCPD,NOLVP,NOBROWNOUT,PUT,NOMCLR,NOWDT,INTRC
6   // CONFIG, DEBUG（NODEBUG ではなく），MCLR（NOMCLR ではなく）
7   //#fuses NOPROTECT,CCPB2,DEBUG,NOWRT,NOCPD,NOLVP,NOBROWNOUT,PUT,MCLR,NOWDT,INTRC
8
9   //内部クロック 1MHz で利用する。
10  #use delay(internal = 1MHZ) //(8MHZ,4MHZ,2MHz,1MHZ,500kHz,250kHZ,125kHz,31kHz)
11
12  // I2C の定義。Raspberry Pi 側からみたアドレスは 0x2f（0x5e を 1bit 右シフトした値)。
13  #use i2c(SLAVE, SDA=PIN_B1, SCL=PIN_B4, ADDRESS=0x5e, FAST, FORCE_HW)
14  #use fast_io(B) //固定入出力モード
15
16  //PWM のパルス幅に関する定数、
17  #define Width1_ctr 367 //タイマー1（クロック 4usec）は 365 をセットして、1.5msec 程度
18  #define Width2_ctr 85 //タイマー2 は 85（クロック 16usec）をセットして 1.5msec 程度
19  #define Width1_max  (Width1_ctr + 30) //±50 で±0.2msec
20  #define Width1_min  (Width1_ctr - 30)
21  #define Width2_max  (Width2_ctr + 15) //±12 で±0.192msec
22  #define Width2_min  (Width2_ctr - 15)
23
24  #define ACK 1
25
26  //タイマー1、タイマー2 に設定する値用のメモリ
27  int16 Width1 = Width1_ctr;
28  int8 Width2 = Width2_ctr;
```

```
29
30  /*
31  タイマー0 で 17msec ごとに割込みを発生させ、
32   * 割込み発生後、タイマー1 とタイマー2 に設定した時間幅のパルスを生成する
33   */
34  #INT_TIMER0
35  void intTimer0() {
36      //タイマー0 による割込みが発生していることを確認するために 18 番ピンをトグルする
37      //output_toggle(PIN_A1);
38
39      //次の割込みのためにカウンタをセットする。
40      set_timer0(122);
41
42      //CCP1 に 4us×Width1 のパルスを発生させる
43      setup_ccp1(CCP_OFF);    //CCP1（8 番ピン）を Low にする
44      setup_ccp1(CCP_COMPARE_CLR_ON_MATCH);   //カウント値が WIdth1 にマッチすれば CCP1
45  を Low にする
46      CCP_1 = Width1;
47      set_timer1(0);
48      //タイマー2 のクロックは 4us×4=16us
49      setup_timer_2(T2_DIV_BY_4, Width2, 1);
50      set_timer2(0);
51      enable_interrupts(INT_TIMER2);  //タイマー2 の割込みを許可する。
52      output_high(PIN_B3);            //9 番ピンを High にする。
53  }
54
55  /*
56   * タイマー2 による割込みルーチン。9 番ピンを Low にする。
57   */
58  #INT_TIMER2
59  void intTimer2() {
60      output_low(PIN_B3);
61      setup_timer_2(T2_DISABLED, 0, 1);
62  }
63
64  /**** main 関数 ****/
65  void main() {
66      setup_oscillator(OSC_1MHZ);
67      //output_low(PIN_A1); //動作確認のために 18 ピンを High にする。
68
69      //ポート B の全端子の入出力を設定する。
70      set_tris_b(0xf3);
71
72      //17msec 周期で発生する割込み用にタイマー0 を使用する
73      //タイマー0 のクロックは 4us×32=128us、カウンタは 8bit
74      setup_timer_0(RTCC_INTERNAL | RTCC_DIV_32 | RTCC_8_BIT);
75      //17ms/128=133 である。タイマーには 255-133=122 をセットする・
76      set_timer0(122);
77
78      //タイマー1 のクロックは 4us
79      setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);
80
81      //タイマー0 による割込みを許可する
82      enable_interrupts(INT_TIMER0);
83      enable_interrupts(GLOBAL);
84
85      /* Raspberry Pi 側は wiringPi の I2C 関数 wiringPiI2CWrite(fd,data)を使ってデータを
86       * 送ってくることを想定している。この場合、1 回の wiringPiI2CWrite()の呼び出しで
87       * まず、デバイスアドレスの 1 バイト、続いて data の 1 バイトが送られる。下記のプロ
88  グラム
89       * ではデバイスアドレスは読み飛ばしている。
90       */
91      signed int8 rcvData;
92      while(1) {
93          if (i2c_poll()) {
```

```
94              rcvData = i2c_read(ACK);
95
96              if (rcvData == 0x5e) {
97                  //NOP
98              }
99              //下位2ビットにコマンドの種類、上位6ビットに数値がセットされている
100             else if ((rcvData&0x03) == 0x01) {
101                 Width1 = Width1_ctr + ((rcvData&0xfc)/4);
102                 if (Width1 > Width1_max) Width1 = Width1_max;
103                 else if (Width1 < Width1_min) Width1 = Width1_min;
104             }
105             else if ((rcvData&0x03) == 0x02) {
106                 Width2 = Width2_ctr + ((rcvData&0xfc)/4);
107                 if (Width2 > Width2_max) Width2 = Width2_max;
108                 else if (Width2 < Width2_min) Width2 = Width2_min;
109             }
110         }
111     }
112 }
```

図 22 PIC マイコンのプログラムリスト