令和 3 年度卒業論文

RTK 測位を用いた教材用台車の自律走行の研究

旭川工業高等専門学校
システム制御情報工学科
池神 那京　長谷部 航

指導教員　大柏 哲治
提出日　2022 年　1 月　21 日

目次

第1章　研究背景・目的

農業の自動化において，高精度な位置情報測位技術である RTK **測位**[1]が注目されている．また，学校教育においてプログラミングに取り組む機会が増えている[2]現状がある．

マイコンはプログラムを記述することで，機械に必要な動作をさせるために有用なツールである．そこで，RTK 測位を用いて自律走行する**マイコンカー教材**を作ることで，学生が**自動化技術**や**プログラミング**に対する関心を深める良い機会になると考えた．

そこで本研究では，RTK 測位を用いて**ラジコンカーの自律走行**を目指す．このラジコンカーのことを以降，RTK **台車**と呼ぶことにする．（図 1.1）

本研究で作成した教材は，工業大学や他高専などに活用してもらうことを想定している．

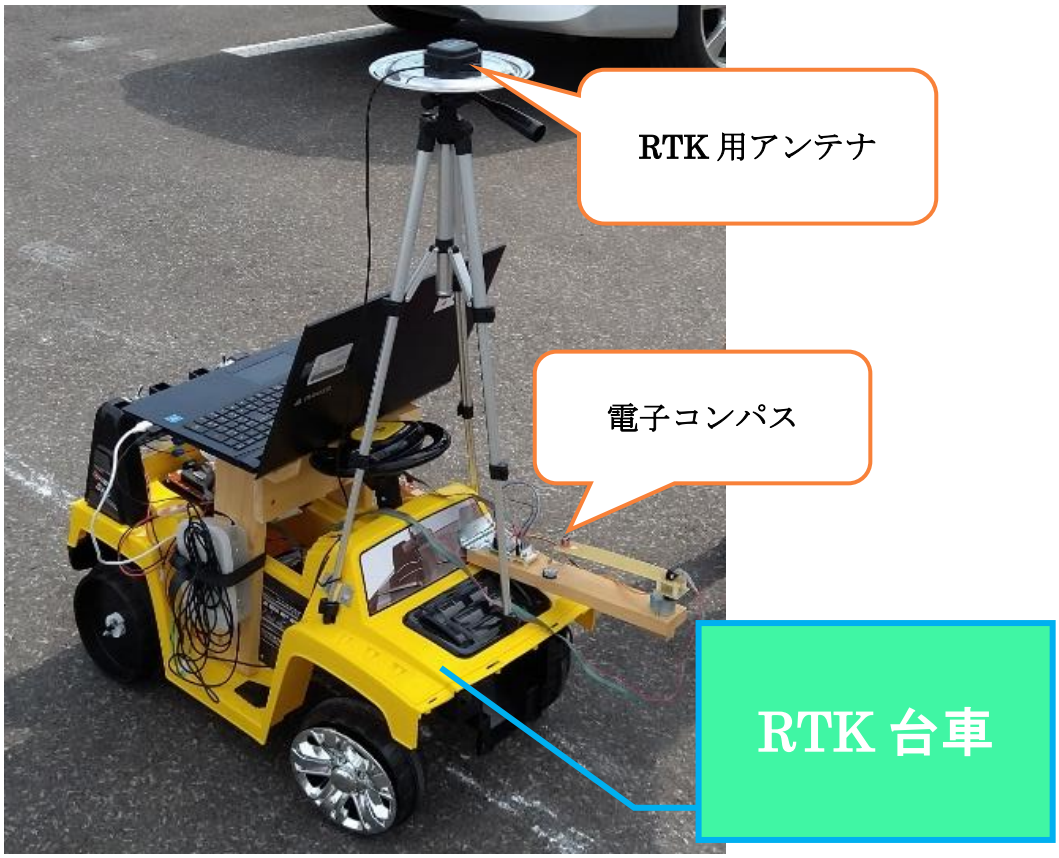


図 1.1　RTK 台車の外観と使用センサ

[1] Real-Time-Kinematic の略であり，地上に設置した「基準局」からの位置情報データによって，高い精度の測位を実現する技術のこと．センチメートル単位で測位が可能である．（文献[1]より）
[2] 学習指導要領の改訂によって，2020 年度から小学校でもプログラミング教育を導入することになった．（文献[2]より）

第2章　先行研究と本研究の違い

先行研究である「RTK 測位による自律走行実験：2020 年度」では，RTK 測位によって得た**位置情報のみ**を用いて直線走行を行っていた．しかし，この方法では直線走行時の**蛇行が大きい**ことが問題として挙げられていた．

そこで本研究では，位置情報に加えて車体の**角度**を用いることで，より安定した直線走行を実現することが目的である．

本研究と先行研究の違いが出てくる状況を図 2.1 に示す．まず，RTK 測位で自律走行をするにあたって，**ずれ量**と**角度**について定義する．

- **ずれ量**：経路から車体が離れている距離のこと
- **角度**：経路に対する車体の傾き

図 2.1 の(a)と(b)ではずれ量がどちらも等しいが，車体の角度が異なっている．どちらの状況も RTK 台車は直線経路に戻るために前輪の操舵を行う．ただし(a)の状況では，先行研究と本研究の手法によって，前輪の切れ角 α の大きさが異なる．

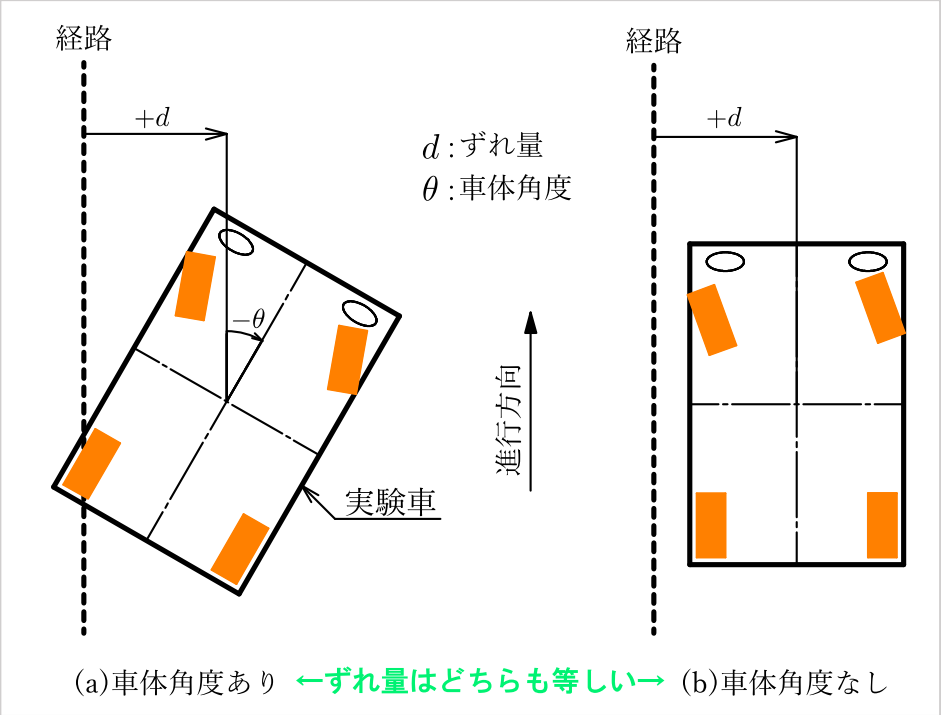


図 2.1　先行研究と本研究の違いが出る状況

ここで，先行研究と本研究によって切れ角が異なる理由について詳しく見ていく．

まず，先行研究の手法では，経路からの**ずれ量のみ**を用いて切れ角を決定している．ただし，この方法には問題点がある．例えば，経路からのずれ量が等しければ，車体の姿勢に関係なく**同じ切れ角**を出力してしまう．

一方で，本研究の手法では，経路からの**ずれ量**と車体の**角度**を用いて，切れ角の決定を行う．車体の角度は電子コンパスを用いて測定する．車体の角度を取得することで，車体の**姿勢に応じて切れ角を変化**させることが出来る．切れ角が車体の姿勢によって変化するので，直線経路に戻るために必要な**走行距離が短くなる**というメリットがある．

次ページの 図 2.2, 図 2.3 にそれぞれ先行研究，本研究の手法による切れ角の算出イメージをまとめる．

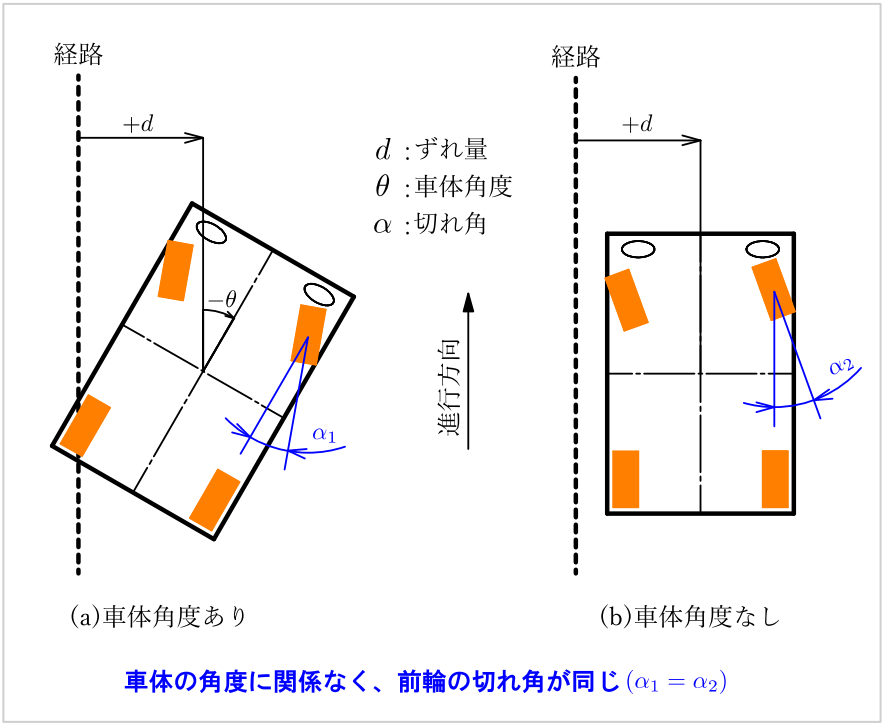


図 2.2　先行研究の手法により算出される切れ角

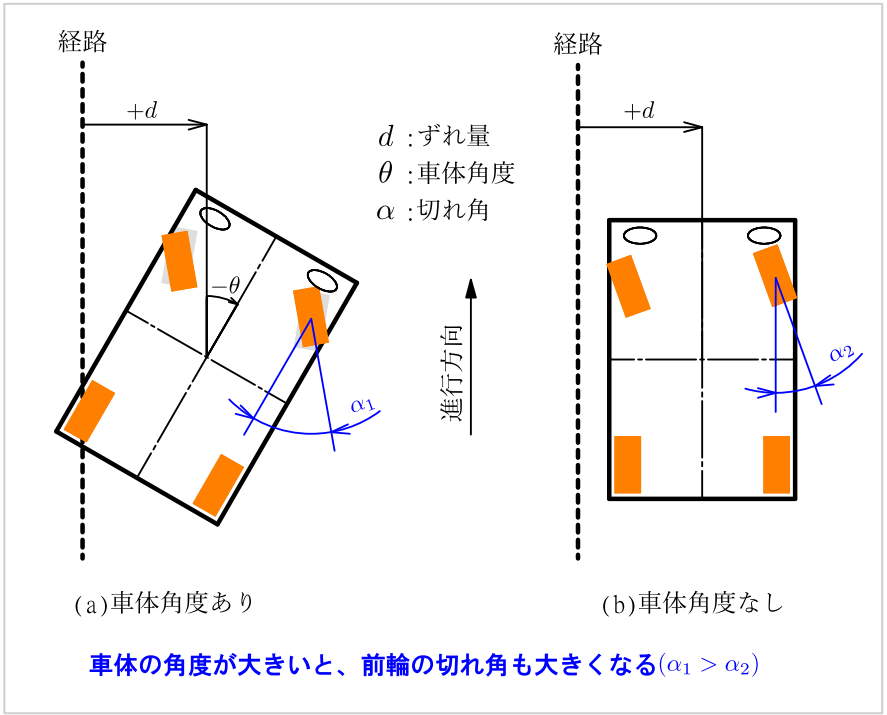


図 2.3　本研究の手法により算出される切れ角

第3章　理論

3.1　超音波センサによる直線走行実験

3.1.1 実験の目的

この実験は，位置情報と車体の角度を用いて自律走行するための予備実験として実施した．この実験目的を整理すると，以下のようになる．

- PC からマイコンへの動作命令を **“文字”** ではなく，**“数値”** に変更したいため[3]
- 走行軌道の修正に **“ずれ量”** と **“角度”** の両方を使いたいため
- 超音波センサを用いた実験結果を，**RTK による直線走行に応用**したい

これらの目的を実現するために，超音波センサを用いて直線走行することを試みた．

3.1.2 RTK による直線走行の概要

超音波センサによる直線走行の原理を示す前に，最終的な目標である RTK 測位を用いた直線走行について改めて整理する．

2 章で示したように，切れ角の制御には**ずれ量**と**角度**を使用する．最終的に，ずれ量と角度はそれぞれ，RTK アンテナと電子コンパスから得る．この 2 つの量を**フィードバック**することで，直線走行を実現するというのが目標である．

そこで，**超音波センサ**を用いて**ずれ量と角度を再現**し，フィードバック制御により走行する実験を行った．この実験で直線走行をすることが出来れば，RTK アンテナと電子コンパスの場合でも，同様の制御方法で自律走行できると考えられる．（図 3.1）

[3] 昨年度の研究では，マイコンが PC から「”a”」という文字を受信したら「直進」，「”b”」を受信した場合には「左折」するというような制御方法を採用していた．本研究では，マイコンが PC からずれ量の数値（整数）を受信し，マイコン側で動作を決定する方式をとっている．

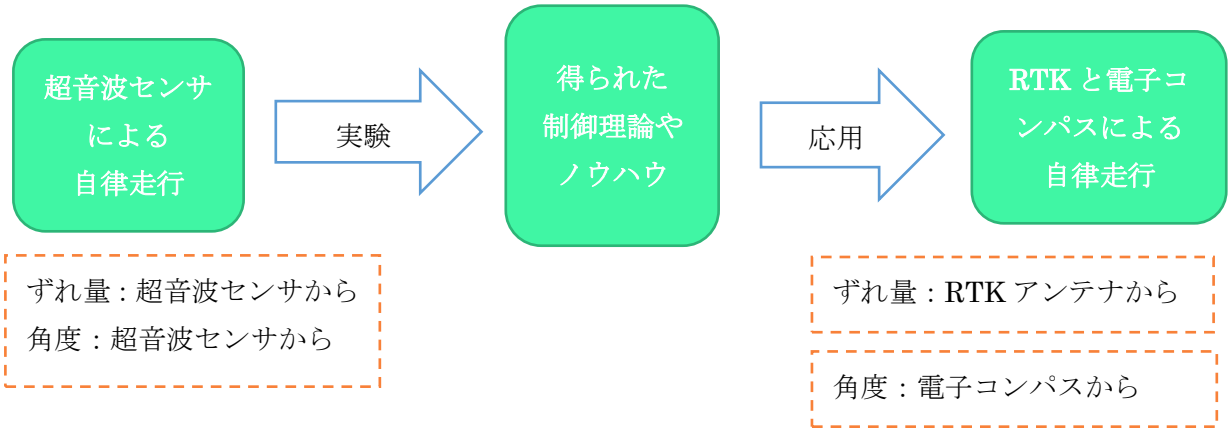


図 3.1　本実験のアイデア

3.1.3 超音波センサによるずれ量と角度の再現

2 つの超音波センサを用いて，**ずれ量**と**角度**を計算するために必要な値を図 3.2 に示す．ずれ量と角度はそれぞれ次のようにして計算する．

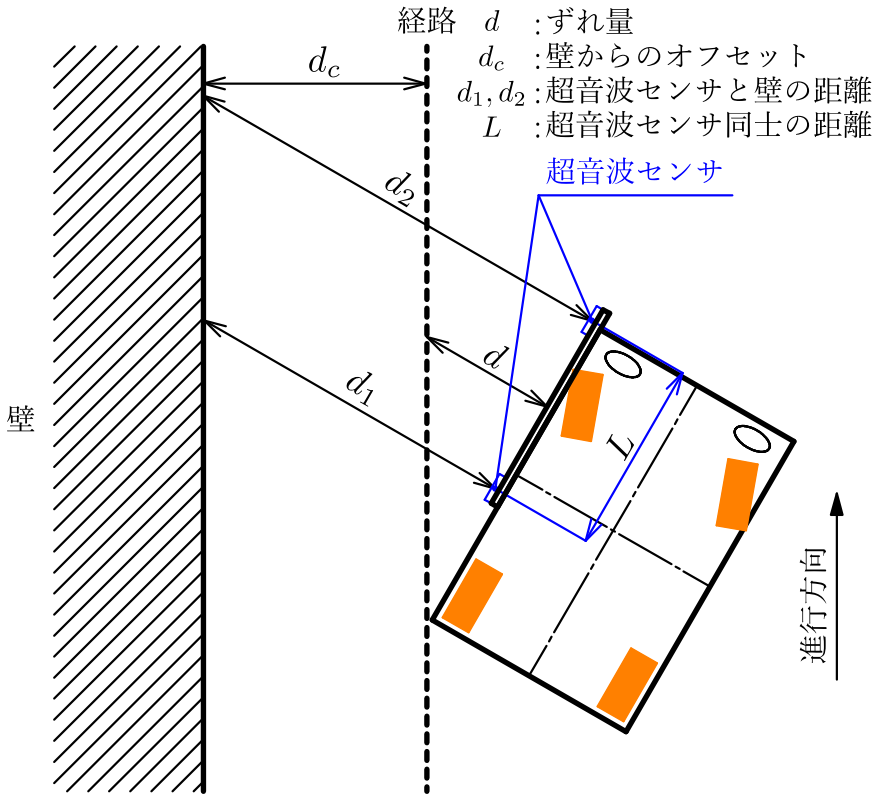


図 3.2　超音波センサを用いた直線走行の原理

◆ ずれ量の計算

壁からの**ずれ量** d_{wall} は図 3.2 のように，2 つの超音波センサと**壁までの距離** d_1, d_2 の**平均値**とする．すなわち，

$$d_{wall} = \frac{d_1 + d_2}{2} \tag{3.1}$$

RTK を用いる時には，このずれ量がゼロになるように走行する．しかし今回の場合，d_{wall} がゼロになると車体が壁に衝突しまうので，プログラム上では少し壁から離れるように d_c ずらしている．

$$d = \frac{d_1 + d_2}{2} - d_c \tag{3.2}$$

◆ 角度の計算

角度 θ は図 3.3 のように，2 つの超音波センサと**壁までの距離** d_1, d_2 と，**超音波センサ同士の距離** L を用いて計算可能である．逆三角関数を用いると，角度 θ は次のように表現できる．

$$\theta = \tan^{-1} \frac{d_1 - d_2}{L} \tag{3.3}$$

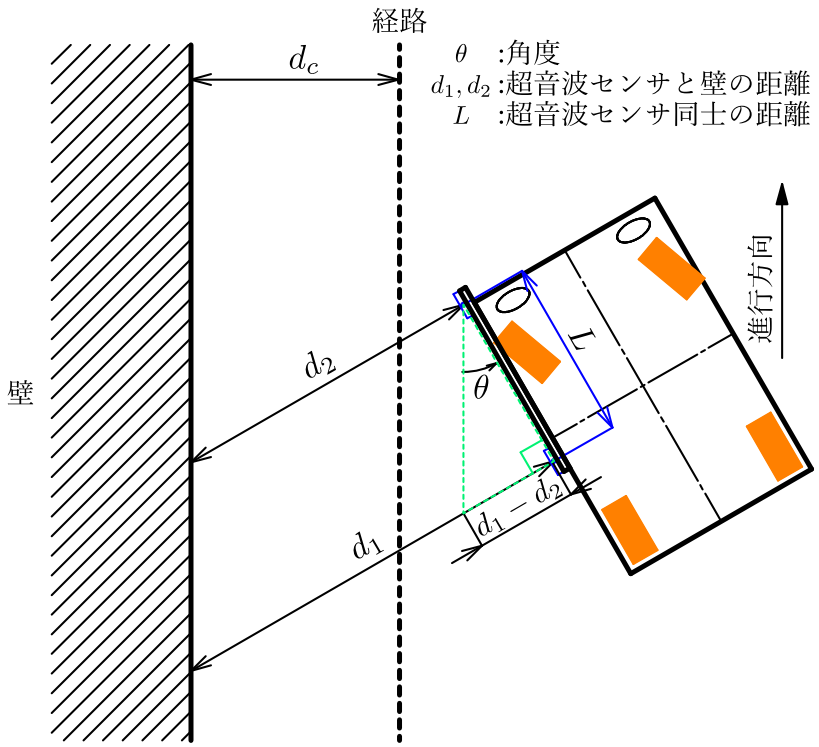


図 3.3　超音波センサによる角度の算出

◆ ずれ量と角度を組み合わせる

ずれ量または角度のどちらかのみで制御するわけではないので，これらを**組み合わせる**必要がある．

具体的には，まず，**ずれ量**と**車体の角度**を**フィードバック**し，それぞれに**ゲイン定数**を掛けて足し合わせた値をハンドルの**切れ角**とする．

制御の概要をブロック線図で表現すると，図 3.4 のようになる．

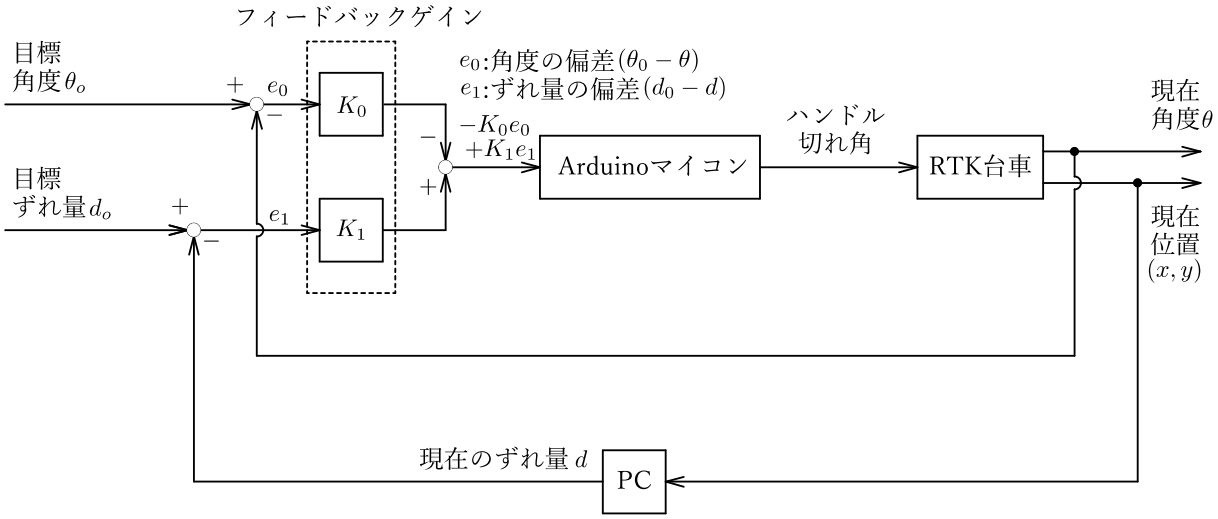


図 3.4　RTK 台車の制御方法の概要

これで，**ずれ量**と**角度**の両方を制御に活用できる．

3.1.4 実験結果とその考察

超音波センサを用いて，ずれ量と角度の計算を行い，壁から一定の距離を保って直線走行をすることが出来た．しかし，実験をしていると以下のような問題点が観察された．

① **走行開始時**に大きく挙動が乱れる（蛇行が大きくなる）．
② 車体角度が大きくなる場合も，挙動が大きく乱れる．

①に関しては，超音波センサの作動開始時に出力される，**外れ値**が原因であると考えた．そこで，走行開始前に安定するまで値を 10 個程度出力しておき，安定した状態の値を使って走行を開始するように，プログラムを改良した．

3.2　電子コンパスによる直線走行実験

前節では，超音波センサを用いて車体の角度を求めている．RTK 測位から得られる位置情報では，車体の角度を計算することができないので，**電子コンパス**を用いて車体の**角度**を求める．

3.2.1 電子コンパスのキャリブレーション

RTK 台車では走行開始時にキャリブレーションを行うように設定している．キャリブレーションの時には，ステッピングモータを使って電子コンパスを回転させる動作を行う．この動作をする理由は，センサの特性が周りの温度や磁気によって変わる[3]ためである．キャリブレーションをすることで，センサの値と実際の角度の**ずれを減らす**ことができる．（図 3.5）

ちなみに図 3.5 の横軸と縦軸の B_x, B_y はそれぞれ，センサが出力する地磁気の値である．

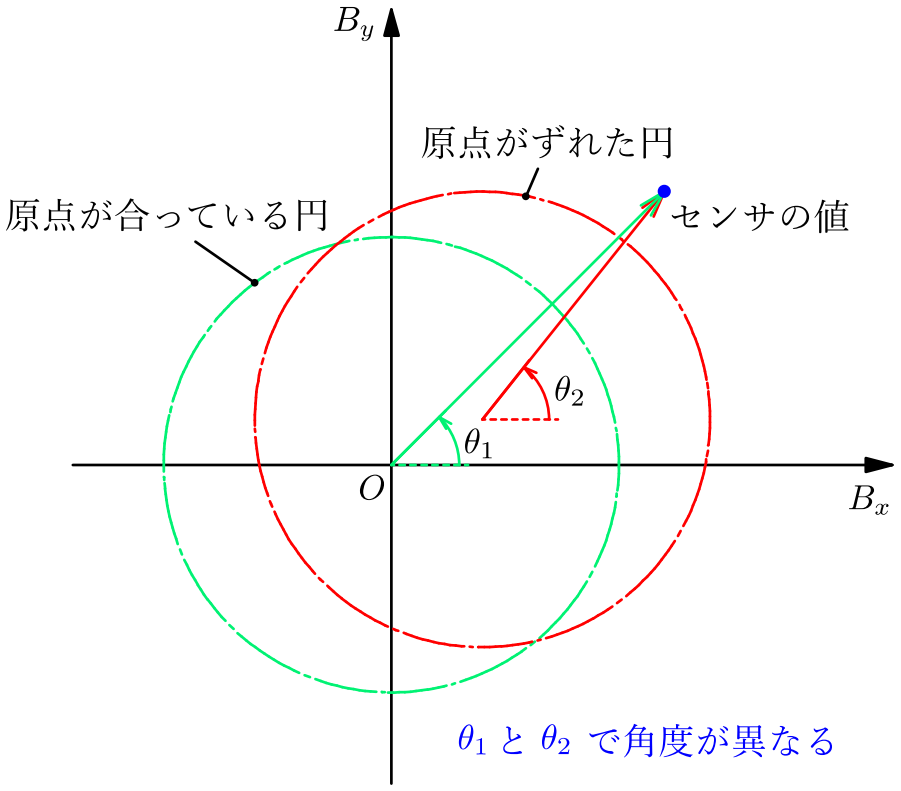


図 3.5　原点のずれによる測定角度の誤差（文献[3]を参考に作成）

図 3.6 に**キャリブレーション**のイメージ図を示す．まず，ステッピングモータで電子コンパス回転させることで，円形に点がプロットされる．しかし，プロットされた点によって作られた円は，原点が少しずれているので，この**ずれを修正**する必要がある．

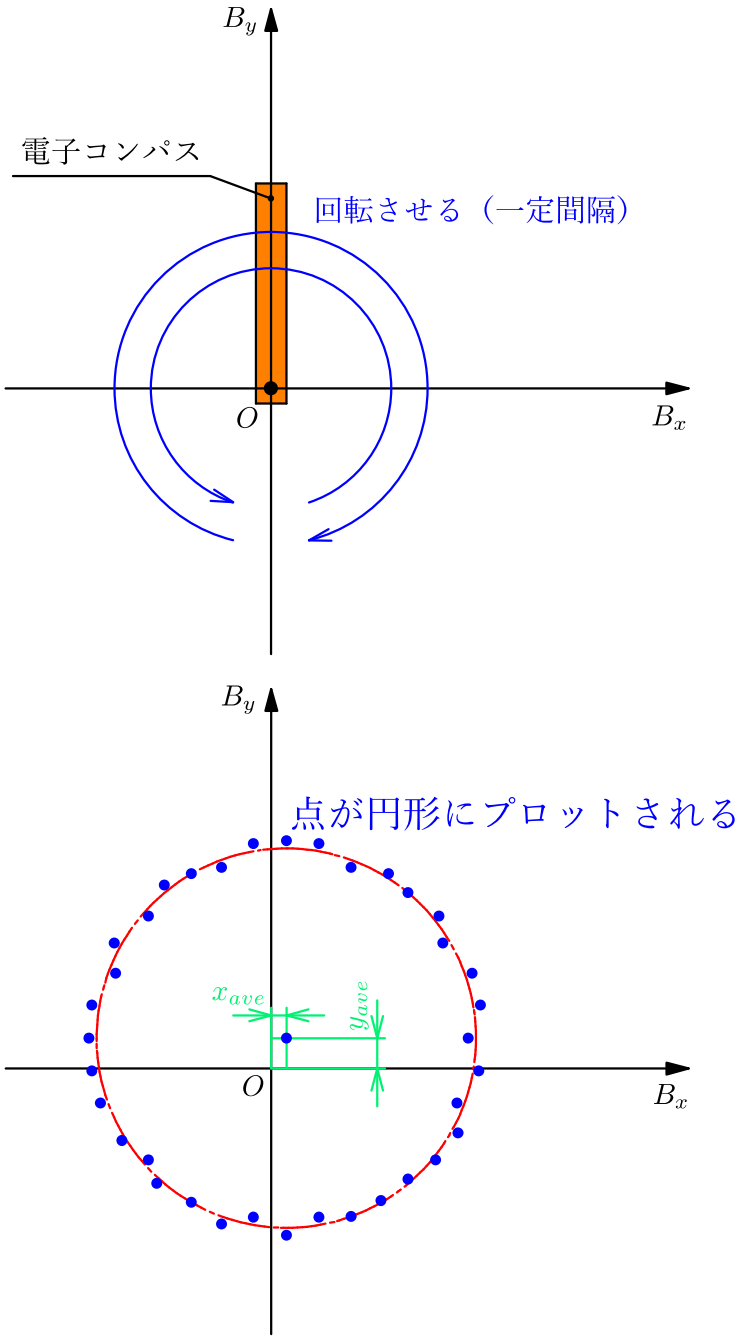


図 3.6　キャリブレーションの原理（文献[4]を参考に作成）

ずれを修正したときの x, y 方向のセンサの値 x_{real}, y_{real} を計算する方法を考えてみる．ここで，センサから出力されたセンサの値 x_{mag} , y_{mag} ，プロットされた点の平均値（原点からのずれ）を x_{ave}, y_{ave} と定義すると，

$$x_{real} = x_{mag} - x_{ave} \tag{3.4}$$

$$y_{real} = y_{mag} - y_{ave} \tag{3.5}$$

と求められる．

3.2.2 方位の計算方法

RTK 台車の方位の**基準**は，**キャリブレーション**の段階で決定する．車体の角度は，**キャリブレーション時の磁北角度[4]を基準**にして，この角度から**どのくらいずれたのか**を計算している．

図 3.7 に方位の計算方法のイメージを示す．図中の $xMag, yMag$ はぞれぞれ，センサが出力する x, y 方向の地磁気の値である．

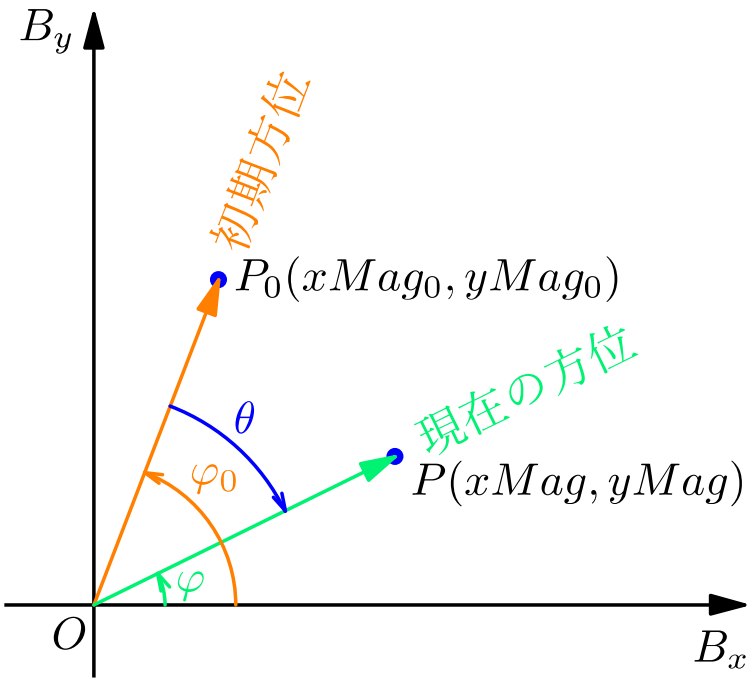


図 3.7　車体角度の計算方法

図 3.7 を参考にして車体の角度 θ を計算する．初期方位を φ_0, 現在の方位を φ とすると，

$$\theta = \varphi - \varphi_0 \tag{3.6}$$

ここで，センサの出力値と逆三角関数を用いて，φ_0, φ を具体的に求めると，

$$\varphi_0 = \tan^{-1}\frac{yMag_0}{xMag_0} \tag{3.7}$$

$$\varphi = \tan^{-1}\frac{yMag}{xMag} \tag{3.8}$$

(3.6)式に(3.7)~(3.8)式を代入することで，**電子コンパスの値**から**車体角度** θ が計算できる．

[4] 方位磁石の指す向きを磁北と呼ぶ．電子コンパスで取得できるのは，この磁北を基準とした角度である．

$$\theta = \varphi - \varphi_0 = tan^{-1}\frac{yMag}{xMag} - tan^{-1}\frac{yMag_0}{xMag_0} \tag{3.9}$$

3.2.3 実験結果とその考察

電子コンパスのみを用いて，直線走行を行ったときの走行軌道を図 3.8 に示す．経路の始点と終点を破線で結んでいる．この結果を見ると，電子コンパスのみでも直線走行が出来ているが，**経路から離れてしまう**ことが確認できる．

このため，正しく経路上を走行するためには，経路からの**ずれ量**を計算する必要がある．ずれ量の計算方法は 3.3.2 項に示す．

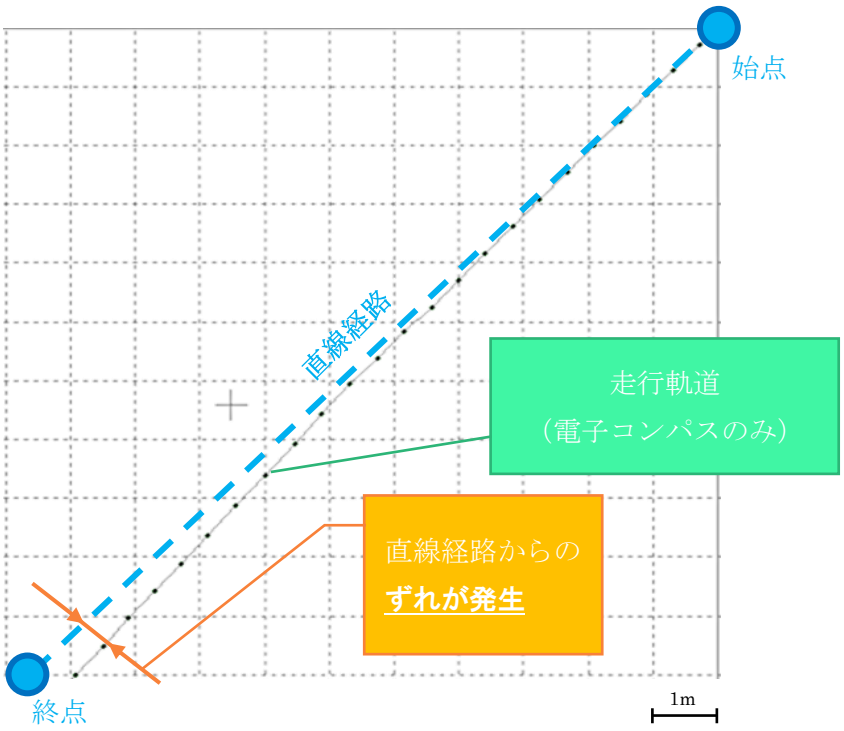


図 3.8　ずれ量計算の必要性

直線経路から離れて走行する結果になったが，**ほぼ一定の方位**に保って走行することが出来たので，走行方位は車体の姿勢検出（車体角度の算出）に利用できることを確認できた．

3.3　RTK と電子コンパスによる直線走行実験 ①

3.3.1 直線経路を数式で表現するには

ここで考えている経路は，始点と終点がそれぞれ 1 つずつの**直線経路**である．今求めたい**ずれ量**というのは，この**直線経路から車体が離れた（垂直）距離**のこと（図 3.9）を指す．これを計算するためには，図 3.9 のように**車体の位置情報**と**経路の方程式**が必要である．

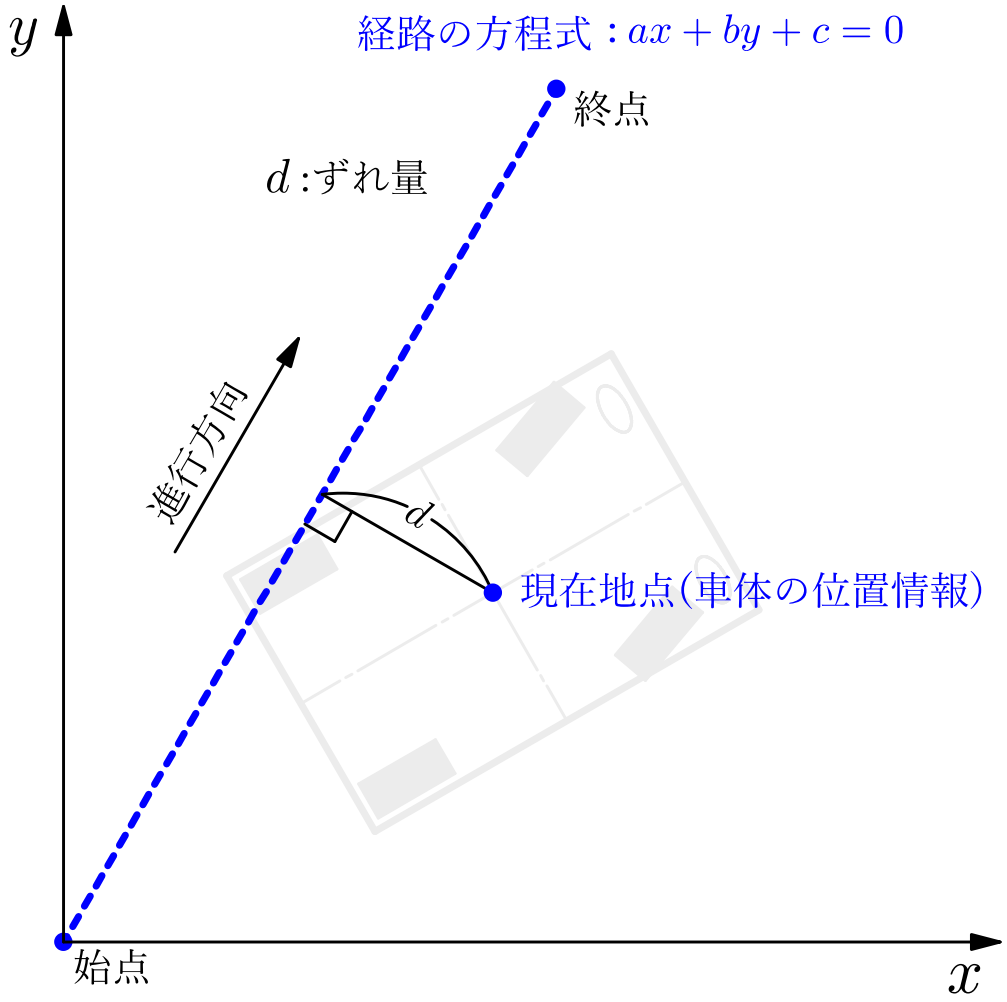


図 3.9　ずれ量の計算に必要な情報

図 3.9 では，**2 次元平面**を考えているが，実際には地球上の位置を**球面上**[5]で考える必要がある．

そこで，地球上の（球面上の）点でも図 3.9 のような 2 次元平面（xy 平面）で考えられるように，**緯度**と**経度**を用いて**近似**を行う．

[5] 本研究では地球を球面と考えているが，厳密には楕円体を用いる。（文献[5]より）

ここでは，事前の研究（RTK 測位による自律走行原理：2020 年度）から用いている，曲面を平面近似する理論について紹介する．

まず, RTK 測位によって得られる**車体の位置情報**は, **緯度** *lat*（latitude）と**経度** *lng*（longitude）を用いて表現されており，イメージは図 3.10 のようになる．

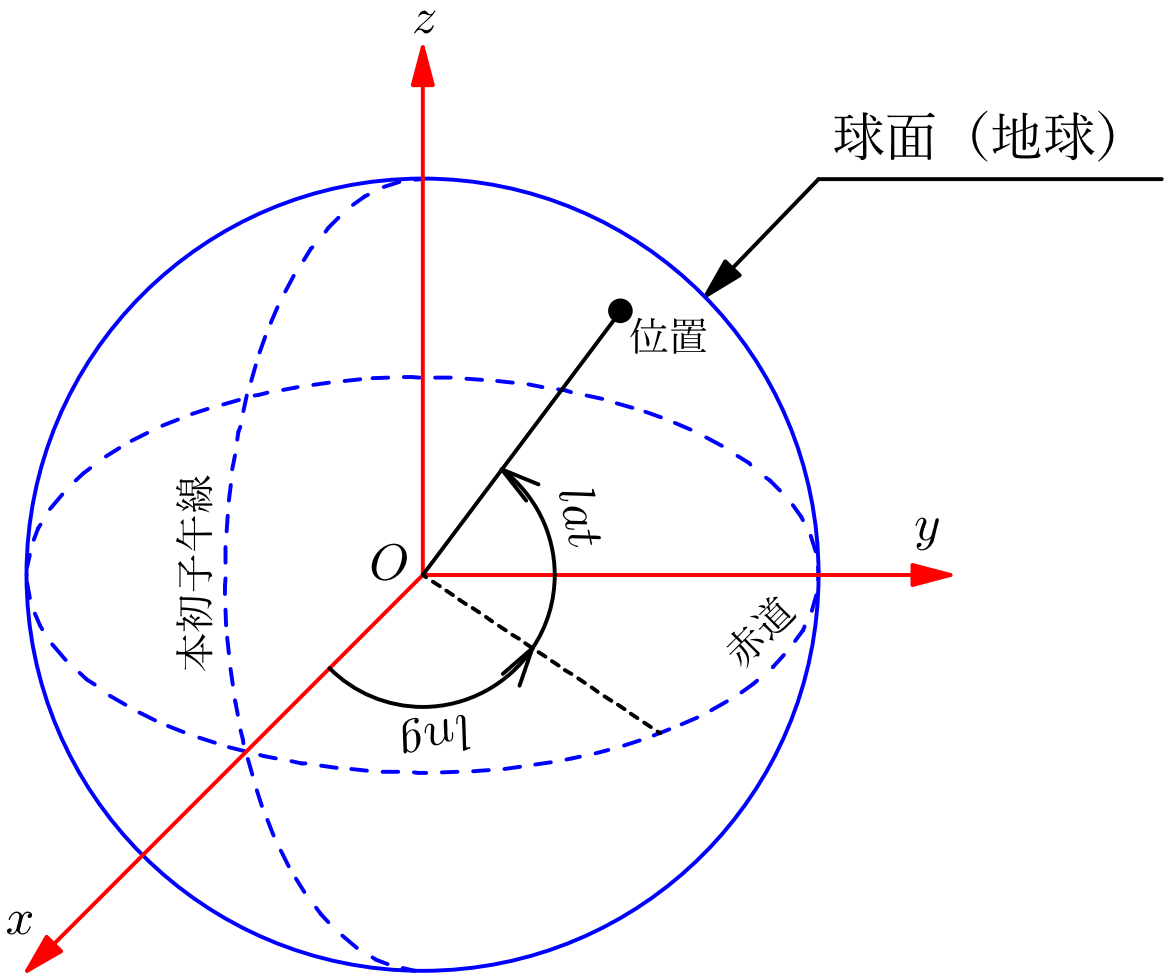


図 3.10　緯度と経度のイメージ図

次に球面上に，RTK 台車が走行するときの経路を考えてみる．考えた（xy 平面上の）**直線経路**は，球面上に描くと図 3.11 のように（球面上の）**曲線**になる．そのため，**曲面を平面近似**する必要がある．曲面を平面近似することができれば，**経路の方程式**が得られる．

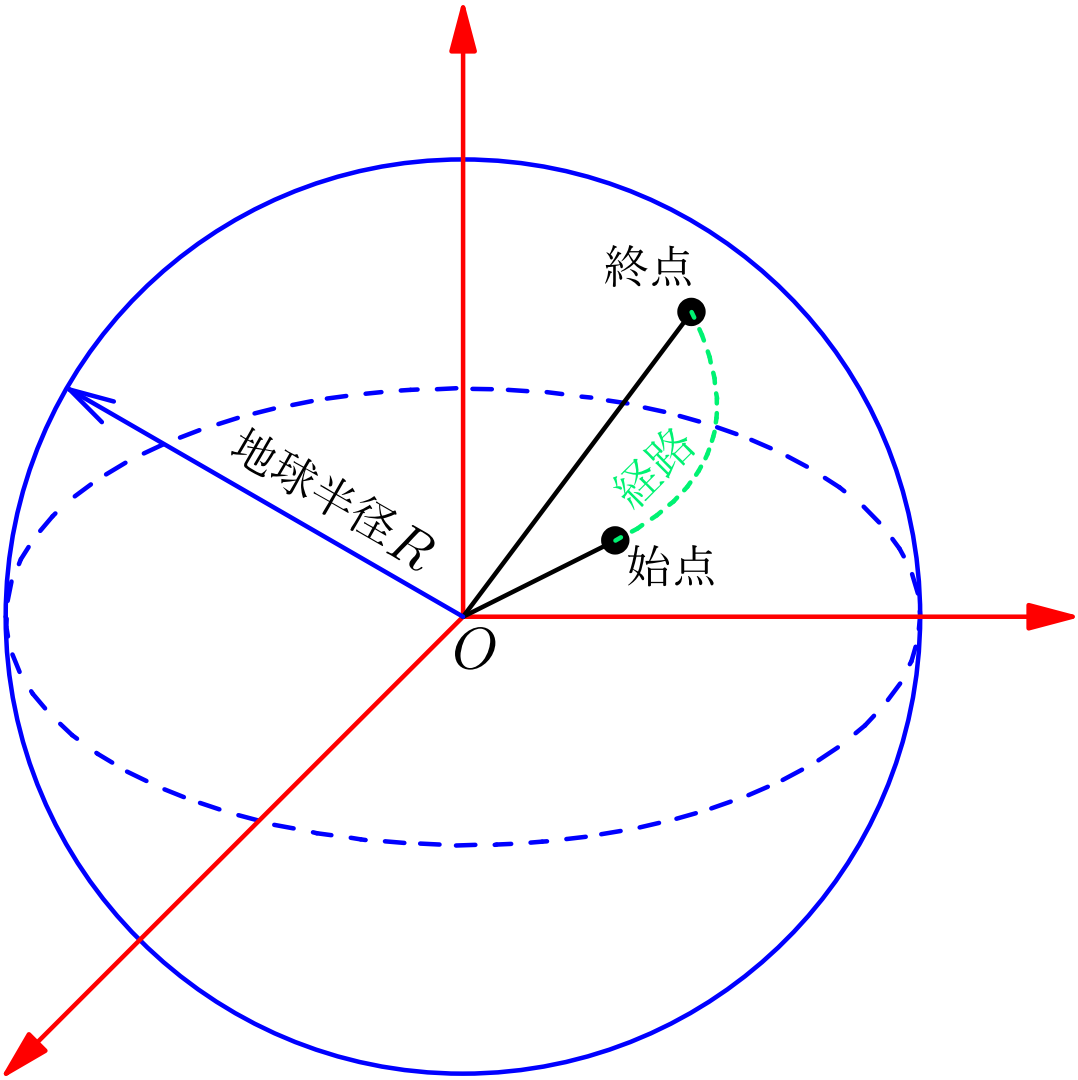


図 3.11　曲面上に描かれる走行経路

曲面を平面近似するときのイメージは図 3.12 のようになる．この近似方法では，RTK 台車の**走行距離（経路の長さ）**が**地球の半径**に比べて**非常に小さい**（微小）であることを用いる．このとき，2 点（始点と終点の位置座標）の**経度の差**，**緯度の差**から**弧長**を計算する状況を考えてみる．

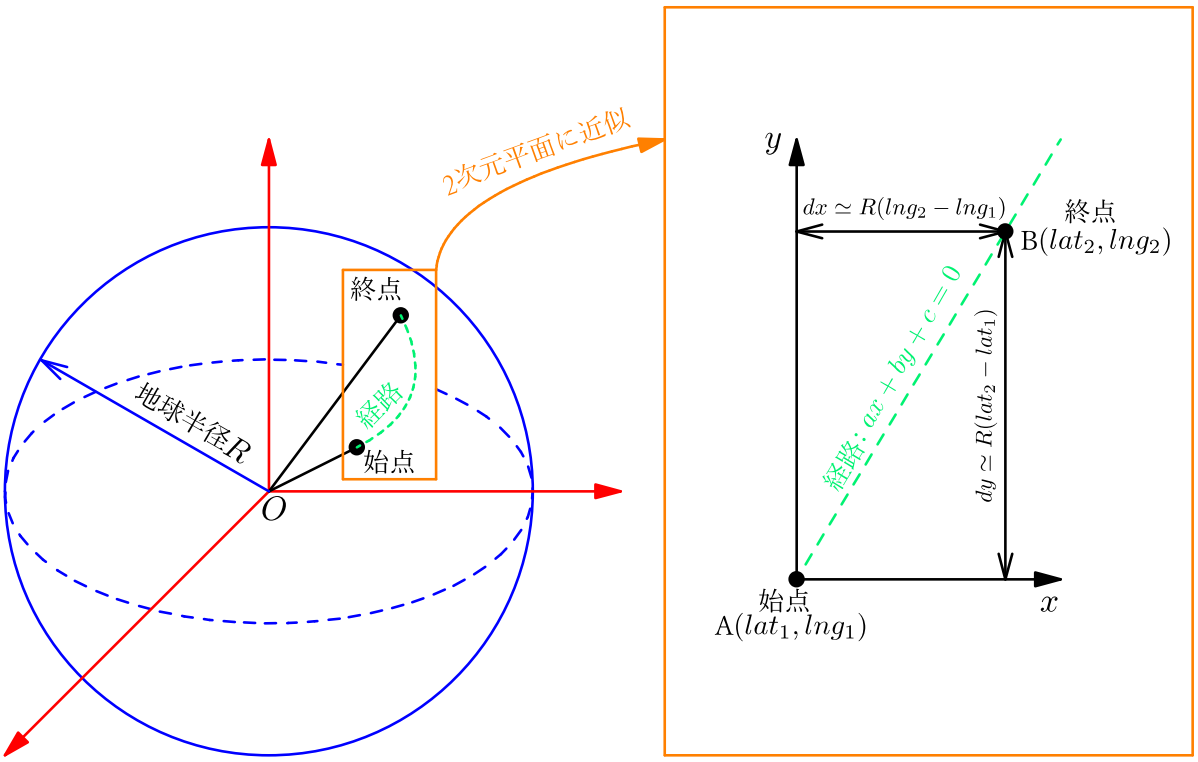


図 3.12　曲面を平面に近似するときのイメージ

始点 A の座標を (lat_1, lng_1)，終点 B の座標を (lat_2, lng_2) として弧長 l_1, l_2 を計算する．l_1, l_2 はそれぞれ**経度の差**，**緯度の差**から計算される弧長である．

半径 r，中心角 θ の弧長 l は，

$$l = r\theta \tag{3.10}$$

で求められるから，

$$l_1 = R(lng_2 - lng_1) \tag{3.11}$$

$$l_2 = R(lat_2 - lat_1) \tag{3.12}$$

RTK 台車の**走行距離（経路の長さ）**が**地球の半径**に比べて**非常に小さい**（微小）であるから，平面での x 方向変位を dx，y 方向変位を dy としたとき，

$$dx \cong l_1 = R(lng_2 - lng_1) \tag{3.13}$$

$$dy \cong l_2 = R(lat_2 - lat_1) \tag{3.14}$$

と考えることができる．図 3.12 の右側に，経路を表す次のような直線の方程式

$$ax + by + c = 0 \tag{3.15}$$

が示されている．ここで，始点を原点としたとき(3.15)式の切片が 0 になるから，$c = 0$ が成り立つ．したがって(3.15)式は次のように書き直せる．

$$ax + by = 0 \tag{3.16}$$

(3.16)式を y について解くと，

$$y = -\frac{a}{b}x \tag{3.17}$$

次に，(3.17)式を x について微分し，直線の傾きを計算すると，

$$\frac{dy}{dx} = -\frac{a}{b} \tag{3.18}$$

(3.18)と(3.13)~(3.14)式を比較すると，直線経路の係数 a, b は次のように計算できる．

$$a = R(lat_2 - lat_1) \tag{3.19}$$

$$b = -R(lng_2 - lng_1) \tag{3.20}$$

※(3.19)式と(3.20)式において，R の手前に付くマイナス符号は a と b の場合で逆になってもよい．

これで，**曲面上の台車の移動**を**平面として近似**し，**経路の方程式**を得ることができた．

3.3.2 得られた情報からずれ量を計算

3.3.1 項で，**車体の位置情報**と**経路の方程式**が得られたので，図 2.1 の**ずれ量** d を計算することができる．ここでは，このずれ量を**符号付き**で求めるための理論について紹介する，

まず，図 3.13 に示すように，車体の位置を $P(x_1, y_1)$，経路の方程式を $s_1 : ax + by + c = 0$ とおいて考える．いま考えている**ずれ量** d というのは，点 P から経路 s_1 に下した**垂線の長さ**のことである．この垂線と経路 s_1 との交点を $Q(x_2, y_2)$ とする．

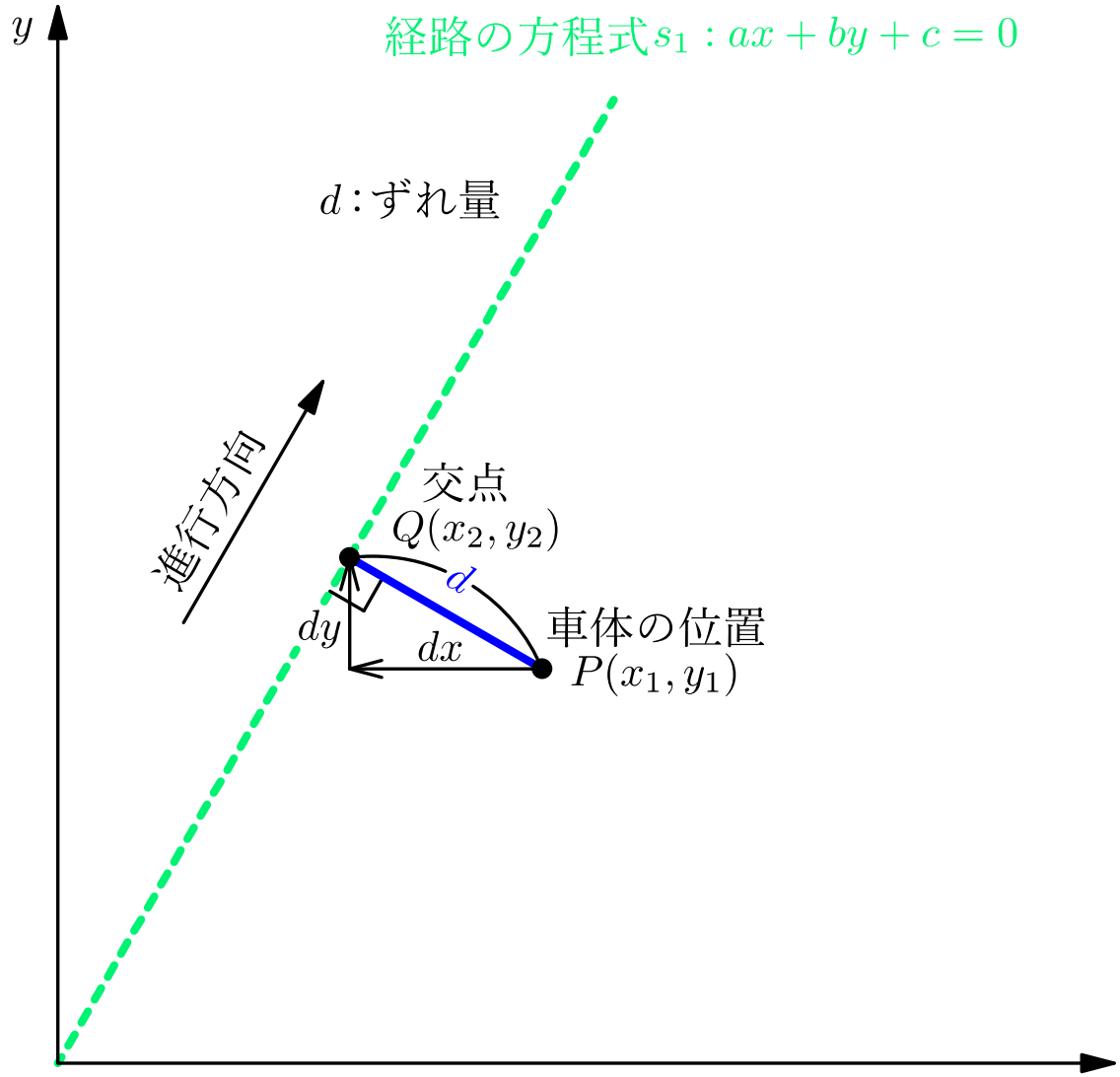


図 3.13　経路からのずれ量 d とは

直線の傾きは，(3.18)式より，$dx = x_2 - x_1$，$dy = y_2 - y_1$ であることを考慮（図 3.13 参照）すると，

$$(\text{直線 } s_1 \text{ の傾き}) = \frac{y_2 - y_1}{x_2 - x_1} = -\frac{a}{b} \tag{3.21}$$

ある直線 とその垂線の傾きには，

$$(\text{ある直線の傾き}) \times (\text{その垂線の傾き}) = -1 \tag{3.22}$$

の関係が成り立つから，今求めたい垂線の傾きは，

$$(\text{垂線の傾き}) = -\frac{1}{(\text{ある直線の傾き})} \tag{3.23}$$

(3.21)式と(3.23)式より，点 P と Q を通る垂線の傾きは，

$$\left(\text{垂線の傾き}\right) = -\frac{1}{\left(\text{直線 } s_1 \text{ の傾き}\right)} = -\frac{1}{-\frac{a}{b}} = \frac{b}{a} \tag{3.24}$$

(3.24)式より，垂線の方程式は

$$y - y_1 = \frac{b}{a}(x - x_1) \tag{3.25}$$

と表現することができる．

次に，(3.25)式に点 $Q(x_2, y_2)$ を代入して，x_2, y_2 のそれぞれについて解くと，

$$y_2 - y_1 = \frac{b}{a}(x_2 - x_1)$$

$$\frac{b}{a}x_2 - \frac{b}{a}x_1 = y_2 - y_1$$

$$x_2 - x_1 = \frac{b}{a}(y_2 - y_1)$$

$$x_2 = x_1 + \frac{a}{b}(y_2 - y_1) \tag{3.26}$$

$$y_2 = y_1 + \frac{b}{a}(x_2 - x_1) \tag{3.27}$$

(3.26)，(3.27)式を，経路の方程式 $s_1 : ax + by + c = 0$ に代入して整理すると，

$$ax_2 + by_2 + c = 0$$

$$a\left\{x_1 + \frac{a}{b}(y_2 - y_1)\right\} + b\left\{y_1 + \frac{b}{a}(x_2 - x_1)\right\} + c = 0$$

$$a\left\{x_1 + \frac{a}{b}\left(y_1 + \frac{b}{a}(x_2 - x_1) - y_1\right)\right\} + b\left\{y_1 + \frac{b}{a}(x_2 - x_1)\right\} + c = 0$$

$$a\{x_1 + (x_2 - x_1)\} + b\left\{y_1 + \frac{b}{a}(x_2 - x_1)\right\} + c = 0$$

$$ax_1 + a(x_2 - x_1) + by_1 + \frac{b^2}{a}(x_2 - x_1) + c = 0$$

$$(x_2 - x_1)\left(a + \frac{b^2}{a}\right) = -ax_1 - by_1 - c$$

$$x_2 - x_1 = \frac{-ax_1 - by_1 - c}{\left(a + \frac{b^2}{a}\right)}$$

$$x_2 - x_1 = \frac{-ax_1 - by_1 - c}{a^2 + b^2}a \tag{3.28}$$

経路からのずれ量 d は，図 3.14 のように**水平方向にずれた距離** dx と**鉛直方向にずれた距離** dy を使って計算できる．そのために， dx, dy を(3.24)式と図 3.14 を参考に求めると，

$$dx = x_2 - x_1 \tag{3.29}$$

$$dy = dx\frac{b}{a} = (x_2 - x_1)\frac{b}{a} \tag{3.30}$$

そして，三平方の定理を用いてずれ量 d を計算すると，

$$d = \sqrt{dx^2 + dy^2} = \sqrt{(x_2 - x_1)^2 + (x_2 - x_1)^2\left(\frac{b}{a}\right)^2} = (x_2 - x_1)\sqrt{1 + \left(\frac{b}{a}\right)^2} \tag{3.31}$$

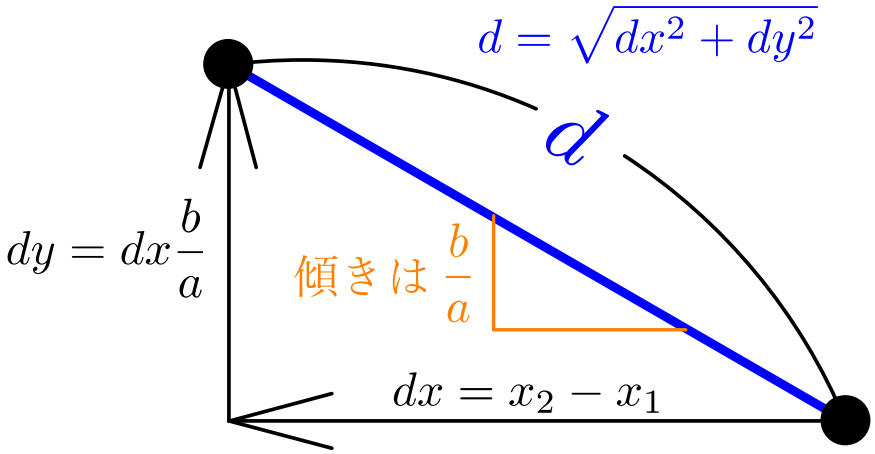


図 3.14　三平方の定理を用いてずれ量を計算

(3.31)式に(3.28)式を代入すると，

$$d = \frac{-ax_1 - by_1 - c}{a^2 + b^2}a\sqrt{1 + \left(\frac{b}{a}\right)^2} = \frac{-ax_1 - by_1 - c}{a^2 + b^2}a\sqrt{\frac{a^2 + b^2}{a^2}} = \frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}} \tag{3.32}$$

経路の始点は原点を通り，経路の方程式 $s_1 : ax + by + c = 0$ の係数において，$c = 0$ となるから，

$$d = \frac{|ax_1 + by_1|}{\sqrt{a^2 + b^2}} \tag{3.33}$$

経路から**左右のどちらにずれたか**知りたいので，絶対値を外して**符号付き**にする．

$$d = \frac{ax_1 + by_1}{\sqrt{a^2 + b^2}} \tag{3.34}$$

(3.34)式を用いれば，車体が経路から**どちら側（符号）**に，**どのくらいの距離ずれているか（大きさ）**計算できる．制御用プログラム中では，この計算式を用いてずれ量を計算している．

3.3.3 超音波センサ予備実験の応用

ここまでで，RTK による位置情報からずれ量を，電子コンパスから車体角度を求めることが出来た．超音波センサによって計算したずれ量と角度をこれらに**置き換え**れば，フィードバック制御を用いて自律走行が可能になるだろう．**ずれ量**は **PC で計算**するため，その値を**マイコンに送信**する動作が必要になる．

そこで，マイコンにずれ量を送信する方法を考えた．ずれ量は［cm］単位で計算を行い，**整数値**としてマイコンに送信することにした．

シリアル通信で PC からマイコンに送信できる変数は byte 型であり，その大きさは 1 バイト[6]である．したがって，0~255 までの整数値を送信することが可能である．

ただし，ずれ量は 3.3.2 項で示したように，符号付きで算出するので，0~255 までの整数値では都合が悪い．そこで，下図 3.15 のような手順でこの問題を解決することにした．

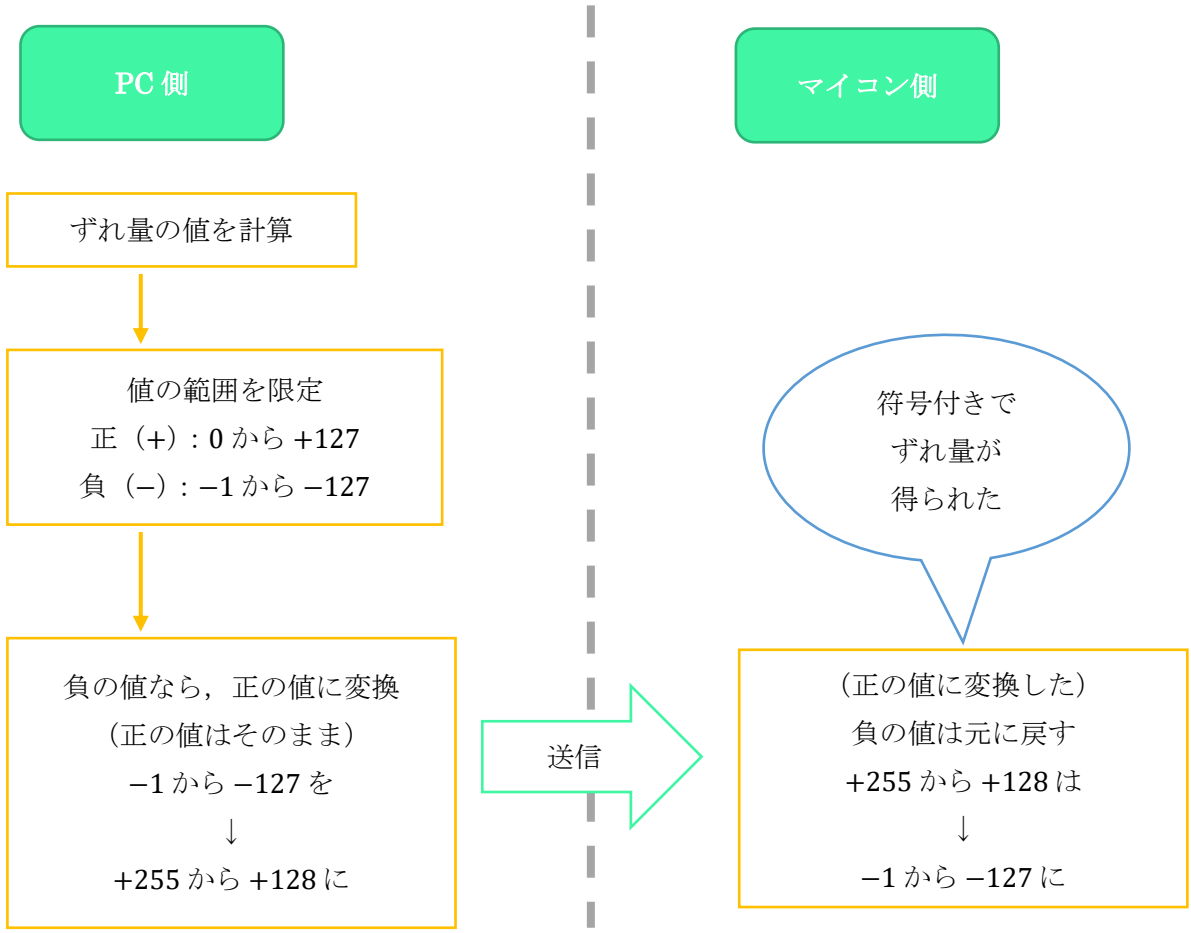


図 3.15　ずれ量を符号付きで送るには

符号なしの整数と**符号付き**整数の対応関係は，表 3.1 のようになる．マイコンで正の値を負の値に戻すときは，この関係を用いている．送られてきた正の値を char **型（符号付き）**に変換した後，int **型（整数）**に変換すれば，**符号付きの整数**が得られる．

表 3.1　符号付き，符号なし 1 バイト 10 進数の対応関係（参考：文献[7]）

符号なし 10 進数	符号付き 10 進数
0	0
1	1
2	2
⋮	⋮
125	125
126	126
127	127
128	-128
129	-127
130	-126
⋮	⋮
253	-3
254	-2
255	-1

3.3.4 実験結果とその考察

この実験で走行するのは，図 3.16 のような約 15 [m] の直線経路である．開始地点と目標地点の座標は，緯度と経度で与える．

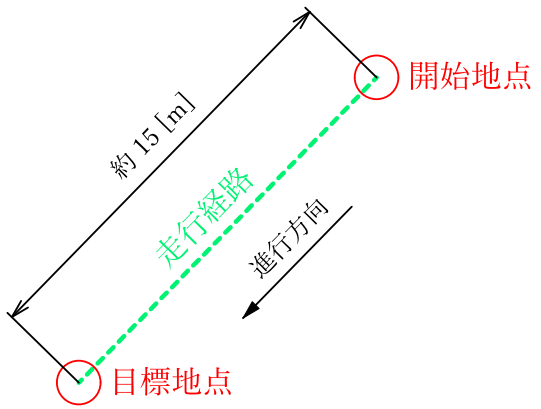


図 3.16　直線走行経路の概要

この経路を RTK（ずれ量）と電子コンパス（角度）を用いて，直線走行した結果が図 3.17 である．

実験結果（図 3.17）を見ると，細かい蛇行が見られるが**大きな蛇行がなく**，概ね直線に沿って走行することが出来た．

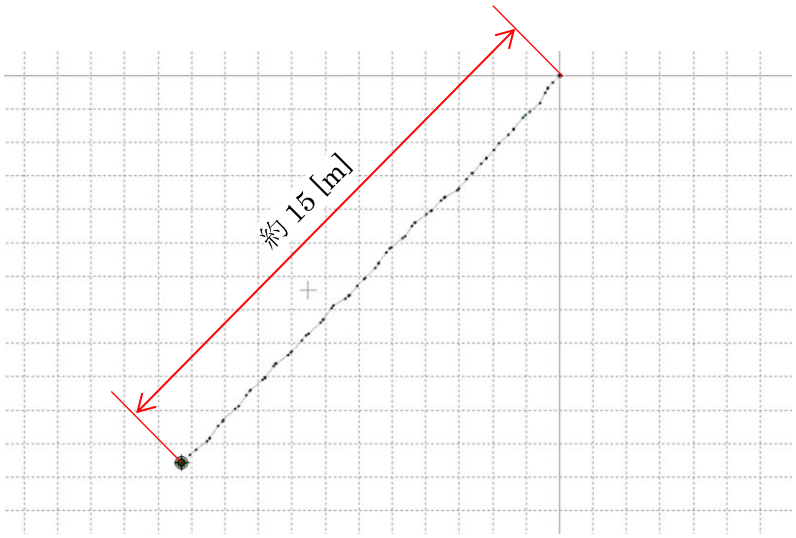
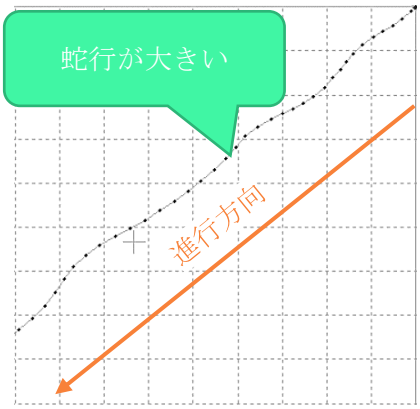
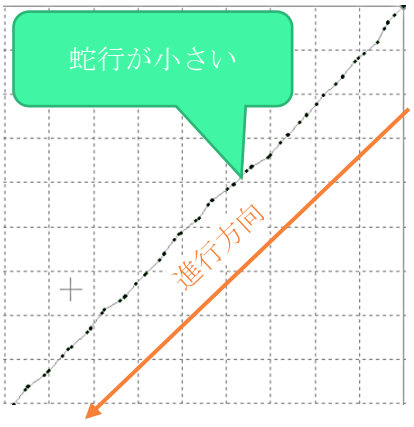


図 3.17　RTK と電子コンパスによる直線走行経路

次に，図 3.17 の経路を走行したときの軌道を，先行研究の場合と比較してみる．図 3.18 の(a)は先行研究の手法（RTK から得たずれ量のみ），(b)は本研究の手法で得られた走行軌道である．



(a) ずれ量のみによる走行［先行研究］



(b) ずれ量と角度による走行［本研究］

図 3.18　走行軌道の比較

＊実験結果の比較＊

図 3.18 の(a)を見てみると大きな蛇行が見られるが，(b)では**蛇行が小さく**抑えられていることが確認できた．

＊実験結果の考察＊

実験結果に差が表れた原因として，**切れ角の算出方法**の違い（2 章参照）が挙げられる．具体的には，車体が**傾いたとき**の切れ角を，先行研究に比べて**大きく出力**していることが影響しているだろう．このことで，車体の姿勢に応じて切れ角を適切に変化させることが出来るので，直線経路から大きく離れずに走行したと考えられる．

3.4 RTK と電子コンパスによる直線走行実験 ②

3.4.1 実験の目的

直線経路を走行することが出来たので，次は経路の終点で**左折**をすることを考えた．左折をするにあたって，以下のような点を考慮する必要がある．

- 90 度を**一気に曲がることが出来ない**（方向転換にステアリングの機構を用いているため）
- 走行できるのは，**直線経路**である．（曲線には対応していない）

このような点を踏まえて，図 3.19 のような経路を設計し，走行することを考えた．この経路は，1 本目の直線の終点に，45 度反時計周りに回転した直線を結合した形となっている．

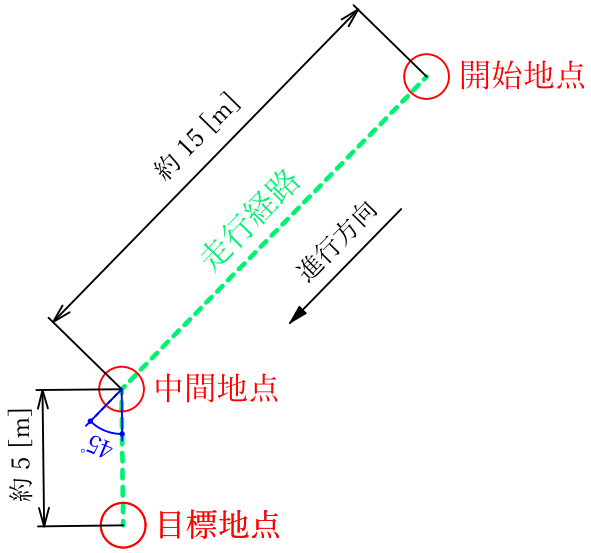


図 3.19　左折走行のための経路

3.4.2 実験目的を達成するために

まず，図 3.19 のような経路を走行するためには，2 本の経路情報が必要である．そこで，経路情報を管理しやすくするために，先行研究に比べて表 3.2 のような改良を行った．

表 3.2　経路管理方法の改良点

改良した項目	改良前	改良後
始点・終点の座標の保存場所	プログラム内に記述	外部のテキストファイル
地点の名前	特になし	以下のような形式でテキストファイルに格納 外部ファイルの中身) 地点名 1，緯度 1，経度 1 地点名 2，緯度 2，経度 2 … …

実験時に使用した，具体的なスタート地点・中間点・ゴール地点の位置情報は，図 3.20 のようになっている．

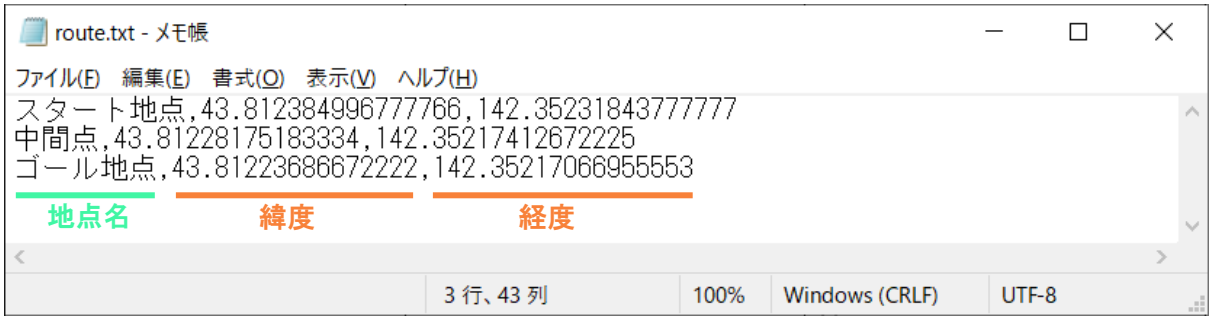


図 3.20　実験に使用した経路情報ファイル

ここまでで，複数の経路情報を管理することが出来た．次に，車体角度の算出方法に関して修正を行った．

車体角度は，初期の磁北方位を基準として算出している．そのため，経路の向きを変更したときに，**方位を算出する基準**も変える必要がある．

そこで，磁気コンパスから出力される値から 45 度差し引くことを考えた．そのようにすることで，経路変更後も直線からの**角度を適切に計算**することが出来る．（図 3.21）

仮に 45 度値を差し引かない場合，図 3.22 のようになってしまうため，経路変更後に角度を正しく算出することが出来ない．

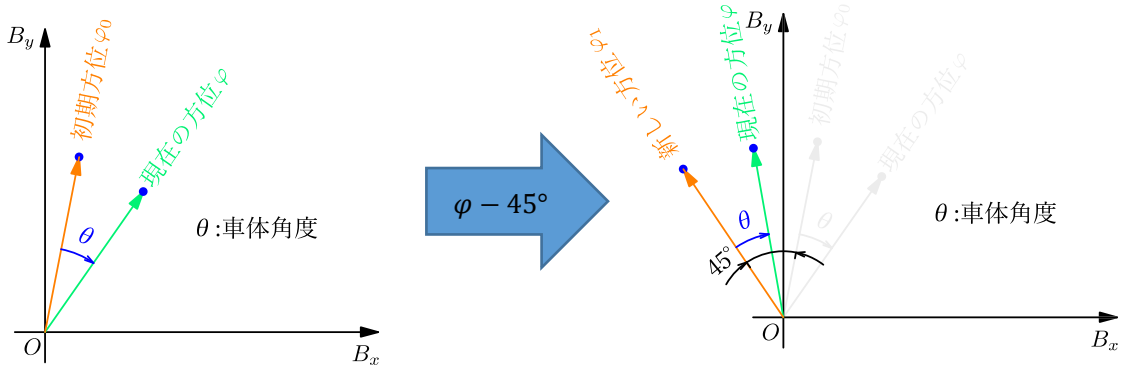


図 3.21　経路変更後における車体角度の計算

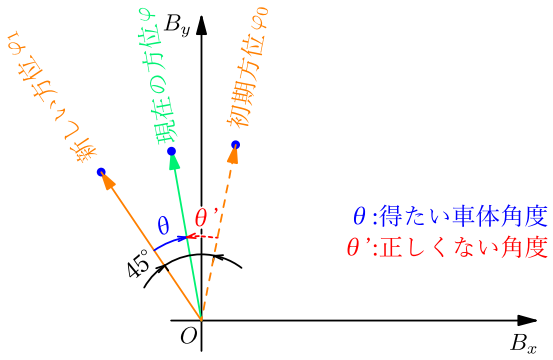


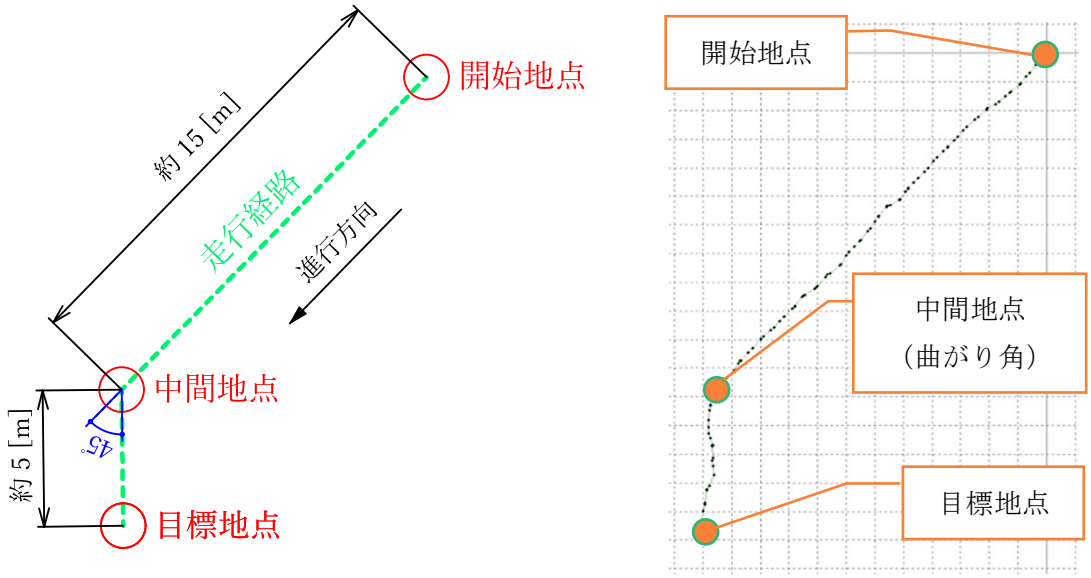
図 3.22　角度を正しく計算できない場合

3.4.3 実験結果とその考察

図 3.23 に設計した経路と，実際に RTK 台車が走行した軌道を示す．

図 3.23(b)を見てみると，初めの直線部分（**開始地点⇒中間地点**）は**蛇行が少なく**走行できていることが分かる．次の直線部分（**中間地点⇒目標地点**）に関しては，少し**蛇行が見られる**結果になった．

このような結果になった理由として，中間地点（曲がり角）に入った後に，切れ角を決定していることが考えられる．次の経路の向きを考慮して，曲がり角へ**進入する姿勢**を決める必要があるだろう．



(a) 設計した経路　　(b) 実験結果

図 3.23　設計した経路と実験結果の比較

3.5　RTK と電子コンパスによる直線走行実験 ③

3.5.1 実験の目的

3.4 節では，経路の終点で左折する実験を行ったが，曲がることが出来る角度は 45 **度限定**である．しかし，これでは柔軟な経路変更に対応できない．

そこで，この実験では曲がる角度を**柔軟に変更できるよう**に工夫を行った．

3.5.2 経路の角度差を計算

経路変更後，もとの経路に対してどのくらい**（何度）変化したか**計算する必要がある．ここで図 3.24 のように経路変更前，変更後の経路をそれぞれ，s_1, s_2 とおくと，s_1, s_2 の**角度差**は以下のような手順で計算できる．この計算方法は，文献[8]を参考にした．

まず，経路 s_1, s_2 の角度差を α_1, α_2 と定義すると，求めたい角度差 θ は次のようになる．

$$\theta = \alpha_1 - \alpha_2 \tag{3.35}$$

角度差 θ は，**正接の加法定理**より

$$\tan\theta = \tan(\alpha_1 - \alpha_2) = \frac{\tan\alpha_2 - \tan\alpha_1}{1 + \tan\alpha_2 \tan\alpha_1} \tag{3.36}$$

となる．

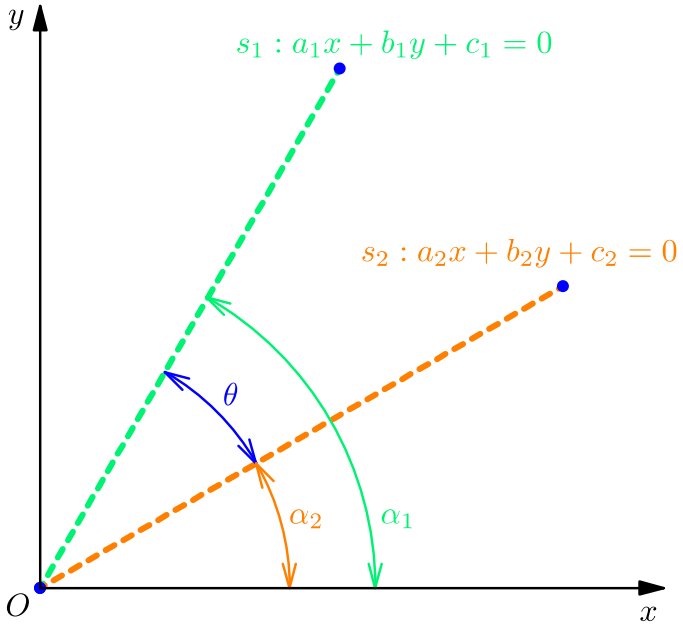


図 3.24　角度差を計算したい 2 つの直線

グラフの傾きが**正接**の値と等しくなるので，経路 s_1, s_2 をそれぞれ y について解くと，

$$y = -\frac{a_1}{b_1}x - \frac{c_1}{b_1} \tag{3.37}$$

$$y = -\frac{a_2}{b_2}x - \frac{c_2}{b_2} \tag{3.38}$$

(3.37)，(3.38)式より，$\tan\alpha_1, \tan\alpha_2$ は次のようになる．

$$\tan\alpha_1 = -\frac{a_1}{b_1} \tag{3.39}$$

$$\tan\alpha_2 = -\frac{a_2}{b_2} \tag{3.40}$$

(3.36)式より，角度差 θ は，

$$\theta = \tan^{-1}\frac{\tan\alpha_2 - \tan\alpha_1}{1 + \tan\alpha_2 \tan\alpha_1} \tag{3.41}$$

(3.41)式に(3.39)，(3.40)式を代入すれば，具体的に角度差を計算することが出来る．この角度は直線の方程式の係数（パラメータ）によって変化する．そのため，経路変更後の**角度を柔軟に変化**させることが出来る．

3.5.3 実験結果とその考察

図 3.25 に左折実験の結果を示す．実験車は，中間地点から経路変更を行い，目標地点に向かって走行する．基準直線と経路変更後の角度差は，目標地点①，②，③でそれぞれ 22°, 45°, 62° である．これらの**角度差**は事前に設定せず，直線の方程式の**係数から計算**している．(3.5.2 項参照)

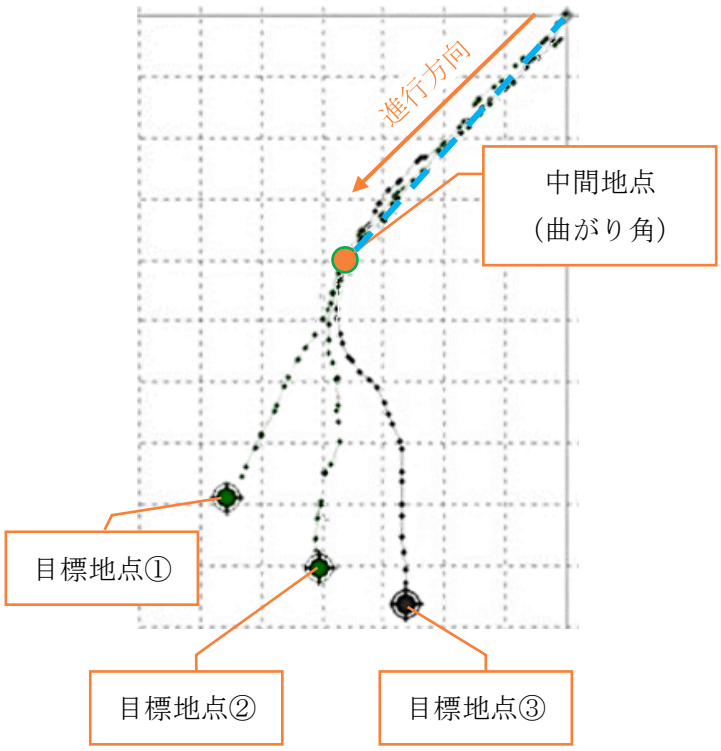


図 3.25　経路情報からの角度計算による左折

実験結果より，**角度差が小さい**場合は，経路変更後も①に向かって**直線的に走行**することが出来た．しかし，目標地点②，③の場合は経路変更後に**大きな蛇行**が見られた．

角度差が大きいほど**蛇行が大きく**なってしまったので，中間地点への進入方向を考慮した制御方法の考案に加え，ステアリングの切れ角を大きくするための**機構的工夫**も必要であると考えられる．

第4章　回路の改良

4.1　目的

昨年度に製作された基板では機能が多く複雑化していたため，**簡易化**することを目的として回路の改良を行った．また，後退走行用のリレーの追加や走行時に起こる**不具合の解決**を目的として行った．

4.2　基板の製作

昨年度，製作されたものを元に基板の改良を行った．製作した基板の回路図が図 4.1，図 4.2 である．これらの回路図と実際の部品の対応関係は，付録 B.2 に示す．

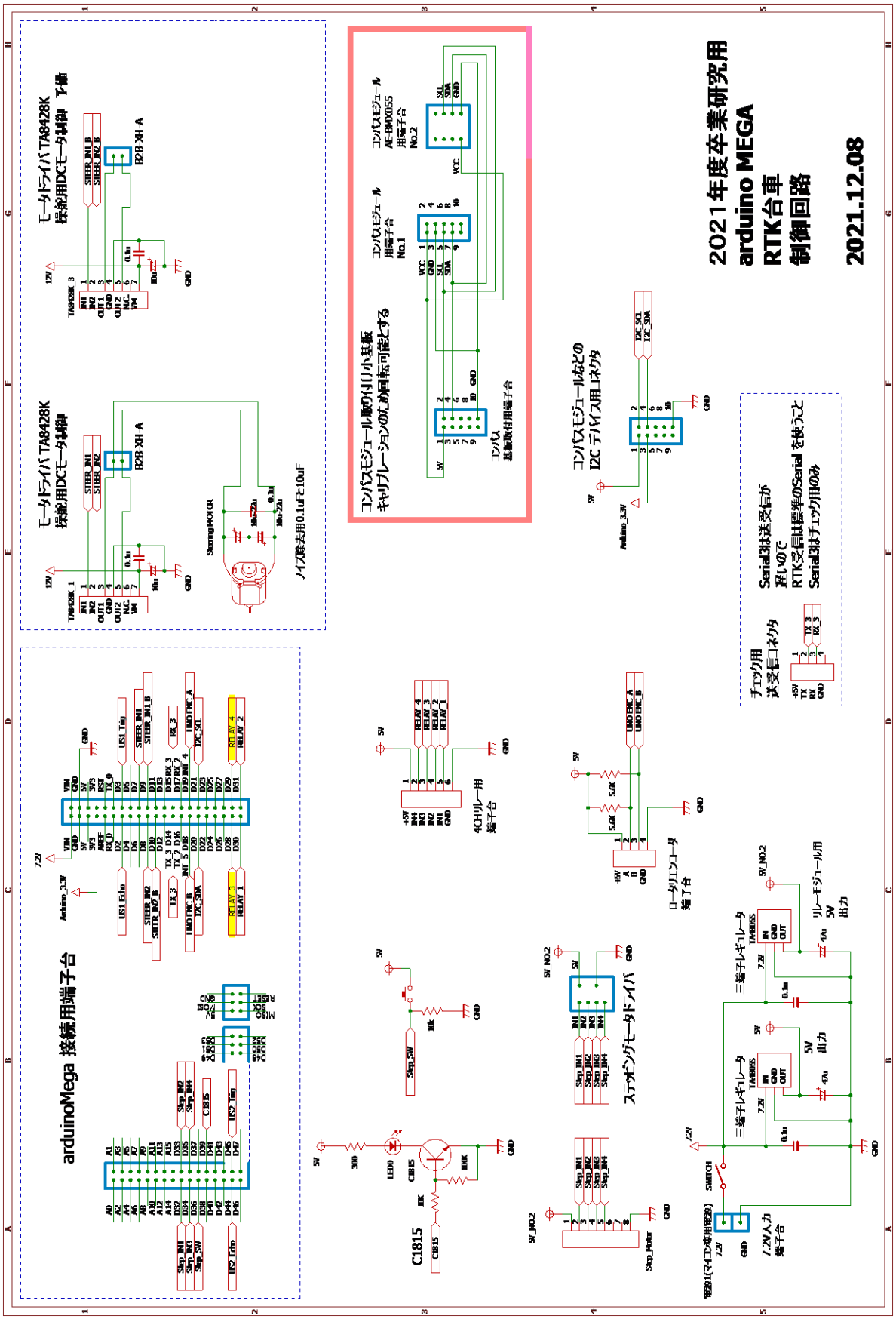

2021年度卒業研究用
arduino MEGA
RTK台車
制御回路
2021.12.08
arduinoMega 接続用端子台
モータドライバ TA8428K
操舵用DCモータ制御 予備
モータドライバ TA8428K
操舵用DCモータ制御
コンパスモジュール基板
キャリブレーションの為の回転可能とする
コンパスモジュール
用端子台
No.1
コンパスモジュール
AE-HMC6352
用端子台
No.2
コンパス
基板取付用端子台
コンパスモジュールなどの
I2C デバイス用コネクタ
Serial3は送受信が
難しいので
RTK受信は標準のSerial を使うこと
Serial3はチェック用のみ
チェック用
送受信コネクタ
4CH リレー用
端子台
ロータリエンコーダ
端子台
ステッピングモータドライバ
三端子レギュレータ
三端子レギュレータ
リレーモジュール用
5V
出力
5V
出力
7.2V入力
端子台
C1815

図 4.1 回路図 1

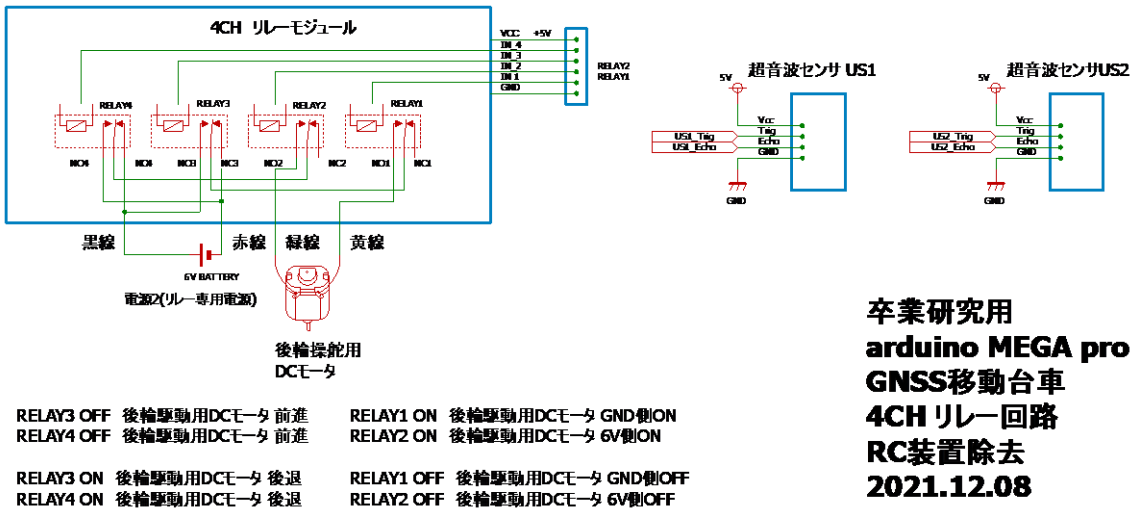


図 4.2 回路図 2

4.3 回路改良の過程

4.3.1 後退走行用のリレーを追加

昨年度は 8CH リレーモジュールであったが，必要な動作は**前進と後退のみ**であるため，4CH リレーモジュールに変更した基板を作成した．しかし，走行実験を行うと数回に一度の頻度で**走行が停止**してしまう問題が発生した．

4.3.2 走行用モータにノイズ対策用のコンデンサを追加

初めに走行用モータに直接的な原因があると考え，**ノイズ対策**を行うことにした．図 4.3 のようなノイズ対策用の**コンデンサと抵抗**を追加したが，4.3.1 項で起こった問題が発生した．走行実験を行う中で分かったこととして，走行が停止し，ハンドルの制御も完全に停止していることがわかった．走行用モータやハンドルの制御はマイコンで行っているので，**マイコンの電源が切れてしまう**ことが原因と考えられる．

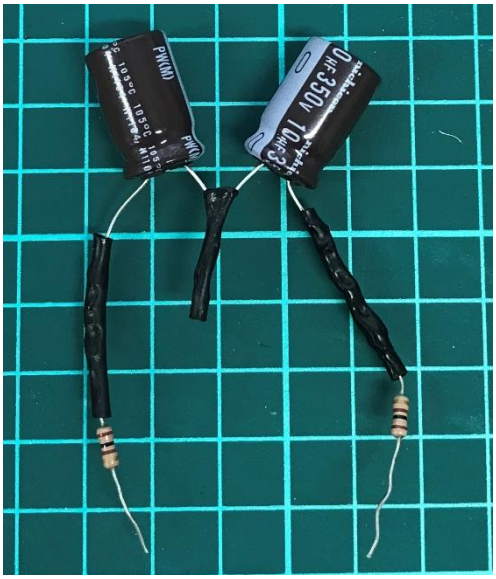


図 4.3 ノイズ対策用のコンデンサと抵抗

4.3.3 マイコン用の電源回路作成

4.3.2 項の不具合を踏まえて，**マイコン専用電源**として 12V バッテリーを割り当てる回路を作成した．走行実験を行った結果，マイコンの電源が ON の状態でも走行が停止していた．このことから**リレー**の動作が直接的な原因であると考えられる．これまではリレーが ON/OFF を繰り返すプログラムで走行実験を行っていたが，試しにリレーが ON の状態で連続走行をするプログラムで走行実験を行った．結果としては，走行が停止することはなかった．このことから**リレーの** ON/OFF が電源に**負荷**を与え，走行が停止していたと考えられる．

4.3.4 リレー用の電源回路の作成

4.3.3 項の結果を踏まえて，マイコン専用電源のほかに，**リレー専用電源**として 6V バッテリーを割り当てる回路を作成した．そのため，バッテリーは 2 個（マイコン用とリレー用）搭載している．また，マイコン専用電源とリレー専用電源の**グランドを分け**，リレーで発生するノイズの影響をマイコンが受けないようにした．走行実験を行った結果，バッテリーがフル充電の状態では走行停止することがなくなった．

4.4 問題に対する解決策のまとめ

最終的に 6V バッテリーを 2 つ搭載し，それぞれマイコン専用電源とリレー専用電源とした．また，走行を行う上でバッテリーの充電状態が走行やハンドルの制御にかなりの影響を与えるため，バッテリーは**常にフル充電**にする必要がある．これにより，マイコンの電圧が足りなくなることやリレーの影響で走行が停止することはなくなった．また，バッテリーが長持ちするようになり，走行可能時間が長くなった．

第5章　結言

本研究では先行研究の手法に改良を加えることで，安定した直線経路上の自律走行を試みた．そこで，直線走行に車体角度の検出をするためにセンサ追加，制御方法の改良を行った．実験を通して車体角度を検出することの有効性について確認できた．

現状の課題点は，連続走行ができないことである．現在は，位置情報の取得とハンドルの切れ角決定のタイミングを合わせるために，車体を一時停止させている．この処理によって走行と停止を繰り返すため，実際の自動運転車のよう一定速度で走行できていない．

また，研究目的であるマイコンカー教材の開発に関しては，不具合の除去やマニュアル作成をさらに進める必要があると考えている．

最後に実験中の様子を図 5.1 に示す．

図 5.1　走行中の RTK 台車

参考文献

[1] 旭テクノロジー「驚異のセンチメートル測位！「RTK」の特徴と GPS との違い，DJI ドローンの使用方法など」，(https://atcl-dsj.com/work/2685/ 閲覧日：2021 年 10 月 28 日)

[2] 文部科学省「小学校プログラミング教育の手引」，(https://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1403162.htm 閲覧日：2021 年 11 月 16 日)

[3] AICHI MI「磁気センサの校正一般」、(https://aichi-mi-test.jimdo.com/home/%E9%9B%BB%E5%AD%90%E3%82%B3%E3%83%B3%E3%83%91%E3%82%B9/%E7%A3%81%E6%B0%97%E3%82%BB%E3%83%B3%E3%82%B5%E3%83%BC%E3%81%AE%E8%BC%83%E6%AD%A3%E4%B8%80%E8%88%AC/ 閲覧日：2021 年 9 月 14 日)

[4] AICHI MI「較正ソフトウェアの原理」，(https://aichi-mi-test.jimdo.com/home/%E9%9B%BB%E5%AD%90%E3%82%B3%E3%83%B3%E3%83%91%E3%82%B9/%E3%82%B3%E3%83%B3%E3%83%91%E3%82%B9%E3%81%AE%E8%BC%83%E6%AD%A3%E3%82%BD%E3%83%95%E3%83%88%E3%81%AE%E5%8E%9F%E7%90%86/ 閲覧日：2021 年 10 月 10 日)

[5] 国土地理院 「日本の測地系―1.地球上の位置を表す」、（https://www.gsi.go.jp/sokuchikijun/datum-main.html#p1 閲覧日：2021 年 9 月 6 日）

[6] Qiita @hidakitai 「Arduino でバイナリ送受信のシリアル通信をするときのパケットの構造」，(https://qiita.com/hideakitai/items/347985528656be03b620 閲覧日：2021 年 11 月 30 日)

[7] Infonet「符号つき 2 進数 (signed binary)」，(http://www.infonet.co.jp/ueyama/ip/binary/signedbin.html 閲覧日：2021 年 11 月 30 日)

[8] おいしい数学「2 直線のなす角を求める方法(加法定理利用)」，(https://hiraocafe.com/note/2lineangletri.html 閲覧日：2021 年 12 月 8 日)

[9] Arduino 日本語リファレンス「Arduino 言語」、(http://www.musashinodenpa.com/arduino/ref/ 閲覧日：2021 年 9 月 23 日)

[10] CQ 出版社： トランジスタ技術 2019 年 10 月号，2019,vol56,no10，pp.47-68.

[11] Electrical Information「【3 端子レギュレータとは？】『使い方』や『型番(7805 等)』などを分かりやすく解説！」，(https://detail-infomation.com/3-terminal-regulator/ 閲覧日：2021 年 12 月 20 日)

謝辞

本研究を進めるにあたって，いつも RTK の基地局を管理してくださった戸村先生，夏休み中も研究活動に協力してくださった大柏先生，制御方法に関するアドバイスをしてくださった佐竹先生に心から感謝申し上げます．また，プログラムの保管方法やシリアル通信に関して同クラスの武井君にもアドバイスをいただきました．

活動に協力してくださった方々に心から感謝申し上げます．

池神　那京
長谷部　航

付録A　ソフトウェアのセットアップ

A.1　u-center と移動局の設定

u-center ではパソコンと受信機を有線で接続し受信機の設定を行うことで，衛星数や現在地座標，信号強度などを確認できる．設定を保存する際は必ず「Send」を押す．

A.1.1 受信衛星の選択

初めに図 A.1 の青色枠内で ZEP-F9P を接続した USB シリアルデバイスのポート番号を選択し，接続する．ポート番号はパソコンの「コントロールパネルのデバイスとプリンター」または「デバイスマネージャー」にて確認することができる．

次に ZED-F9P が受信可能な衛星を選択する．タブの「View – Messages View – UBX – CFG– GNSS」を順に選択し，図 A.1 に示すように衛星と周波数を選択する．

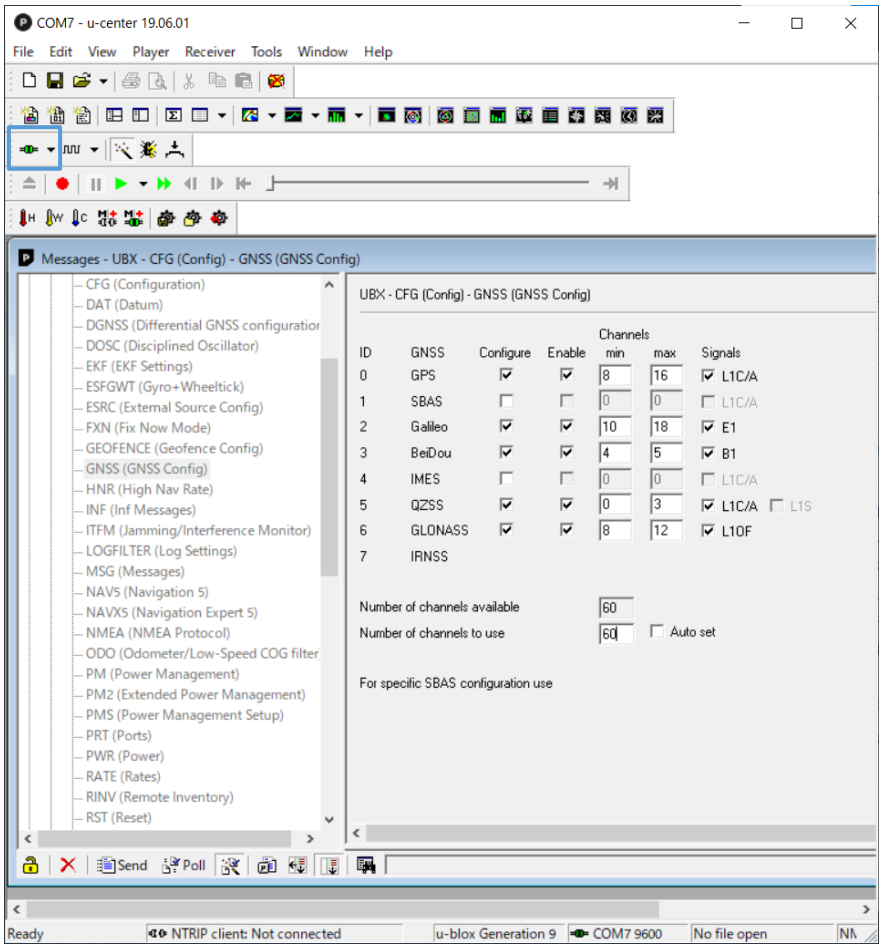


図 A.1 受信衛星の選択画面

図 A.1 で選択している衛星と周波数は ZED-FP9 で受信可能な衛星と周波数帯である．

A.1.2 2 周波受信の設定

タブの「View – Generation 9 Advanced Configuration」を順に選択し，図 A.2 に示すように 2 周波を選択する．

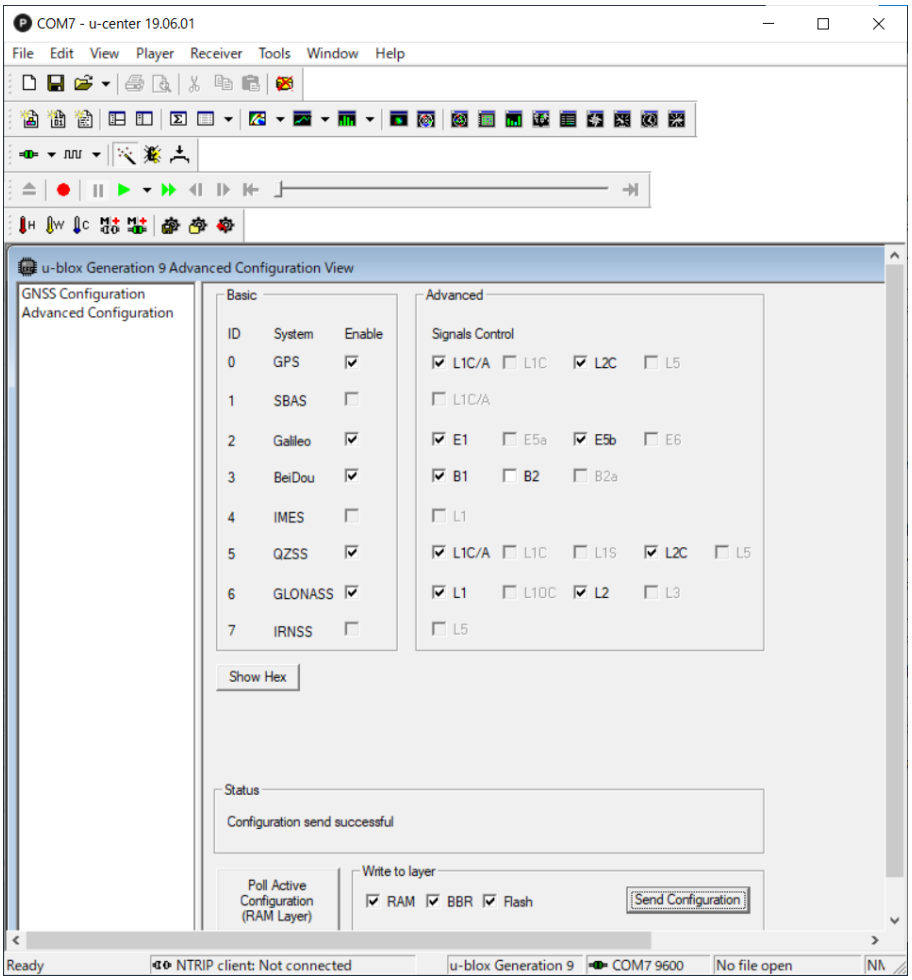


図 A.2 2 周波選択画面

図 A.2 画面下にある「Write to layer」欄の全てを選択し，「Send Configuration」を押す．

A.1.3 出力センテンスの選択

測位データは様々なセンテンスを含んでいる．それらから必要なセンテンスのみを出力することで，データの欠陥を防ぐ．タブの「View – Messages View – NMEA」を順に選択すると図 A.3 に示すような画面が表示される．

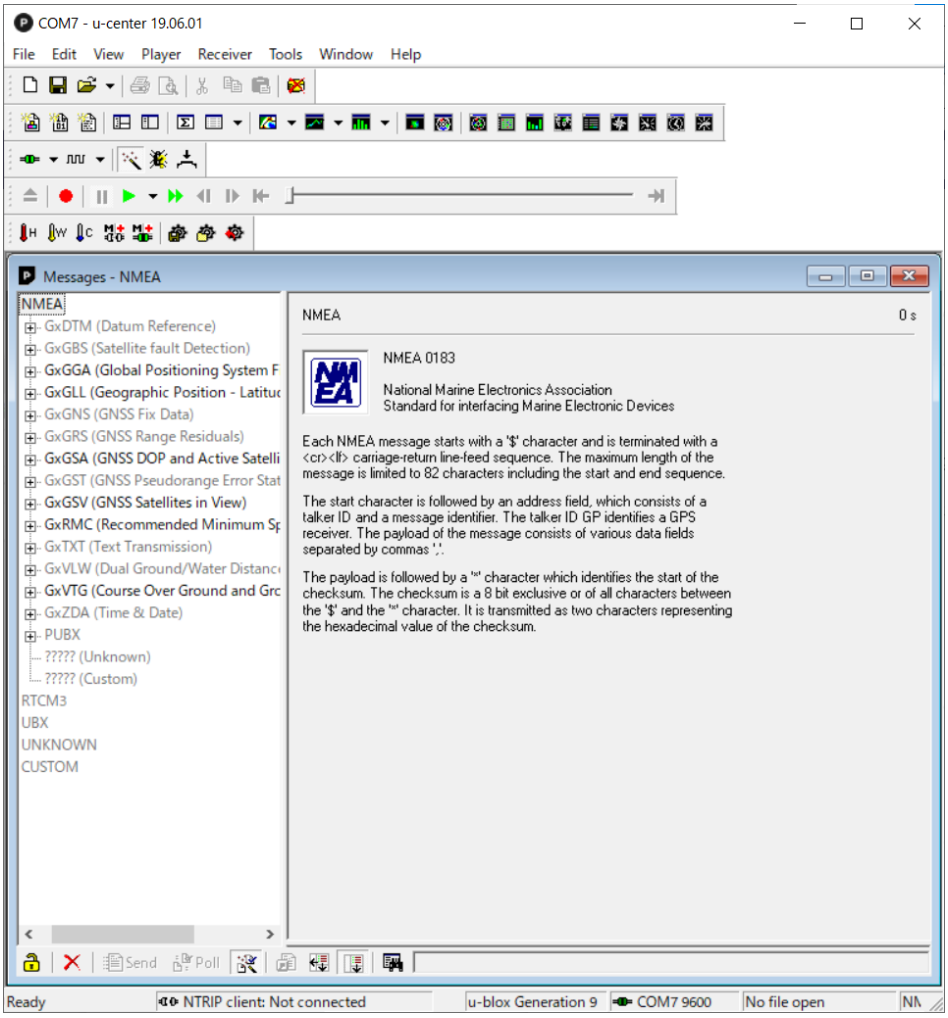


図 A.3 出力センテンスの設定画面

図 A.3 画面左の各センテンスを右クリックし，「Enable Message」または「Disable Message」を選択する．

A.1.4 測位品質の設定

高度な測位を行うには，位相と波数を計測することは必要不可欠である．目的に合わせて，Fix 解を得るか Float 解を得るかを選択する．タブの「View – Messages View – UBX – CFG – DGNSS」を順に選択し，Fix 解が必要な場合は図 A.4 に示すように「3=RTK fixed:Ambiguities are fixed whenever possible.」を選択する．

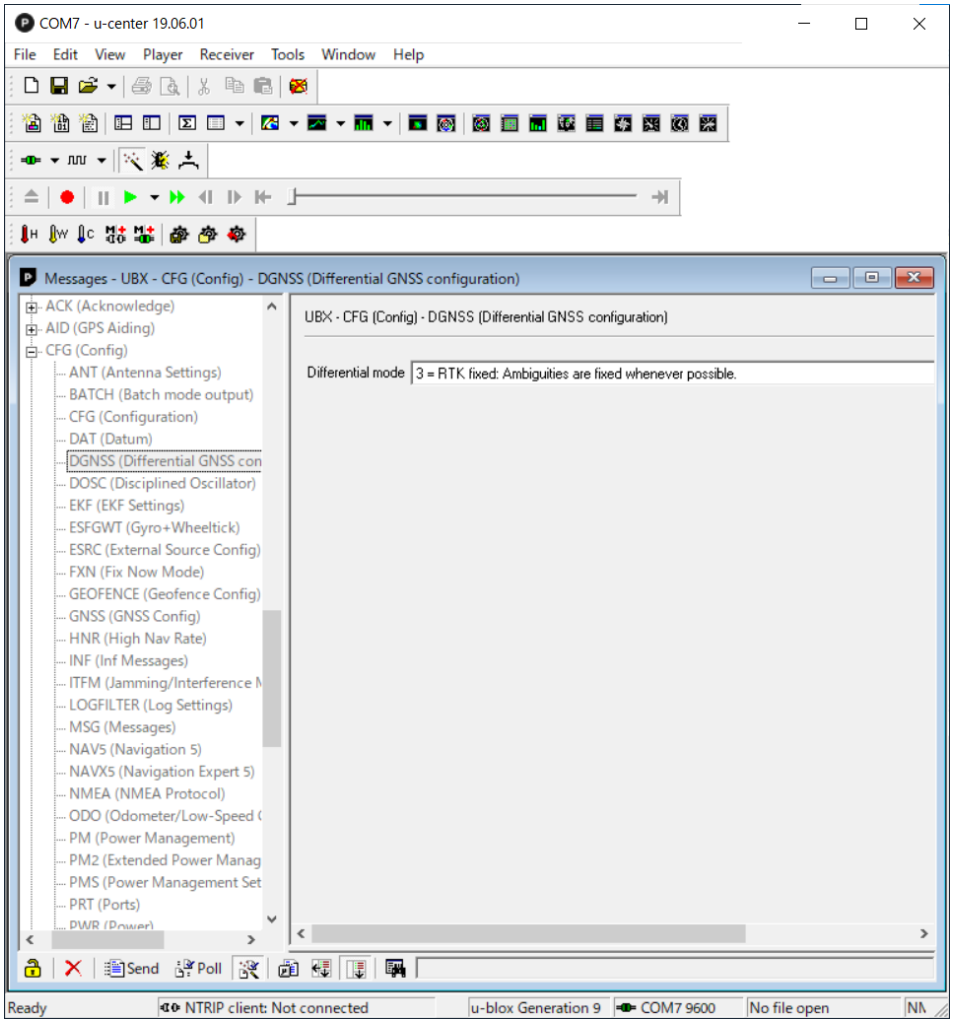


図 A.4 測位解の選択画面

図 A.4 の設定では，なるべく Fix 解を維持する．

A.1.5 受信する衛星の角度設定

マルチパスによる誤差を考慮し，移動局から受信する衛星までの角度設定を行う．角度が大きい(高度が高い)とマルチパスの影響は軽減できるが，衛星数が少なくなってしまう．タブの「View – Messages View – UBX – CFG – NAV5」を順位選択し，図 A.5 に示すように設定する．

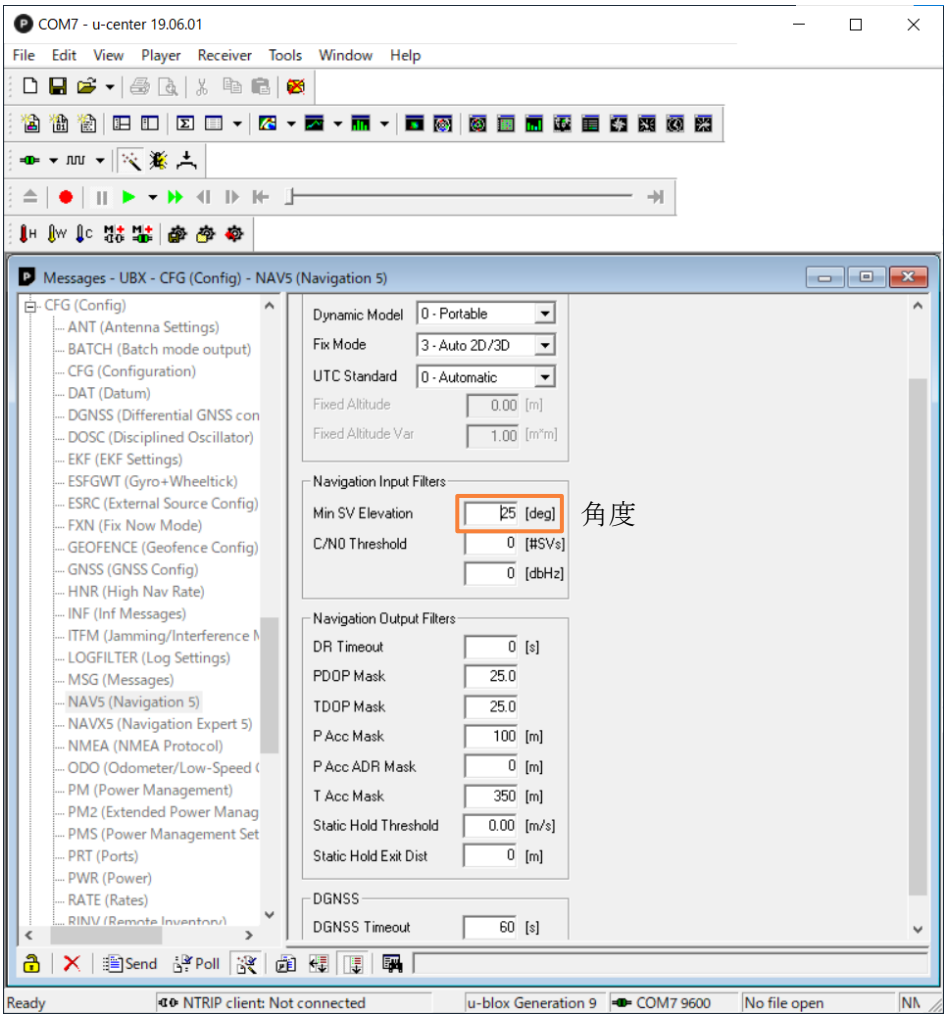


図 A.5 受信衛星の角度設定画面

移動局は「Dynamic Model」欄の「0 - Portable」を選択する．「Fix Mode」欄の「3- Auto 2D/3D」は平面精度と立体精度が表示される．

A.1.6 衛星別測位データの出力頻度設定

測位データの出力頻度を設定する．「View – Messages View – UBX – CFG – RATE」を順に選択し，に示すように「Measurement Period」欄に出力間隔を入力する．入力に伴って「Measurement Frequency」と「Navigation Frequency」は変化する．

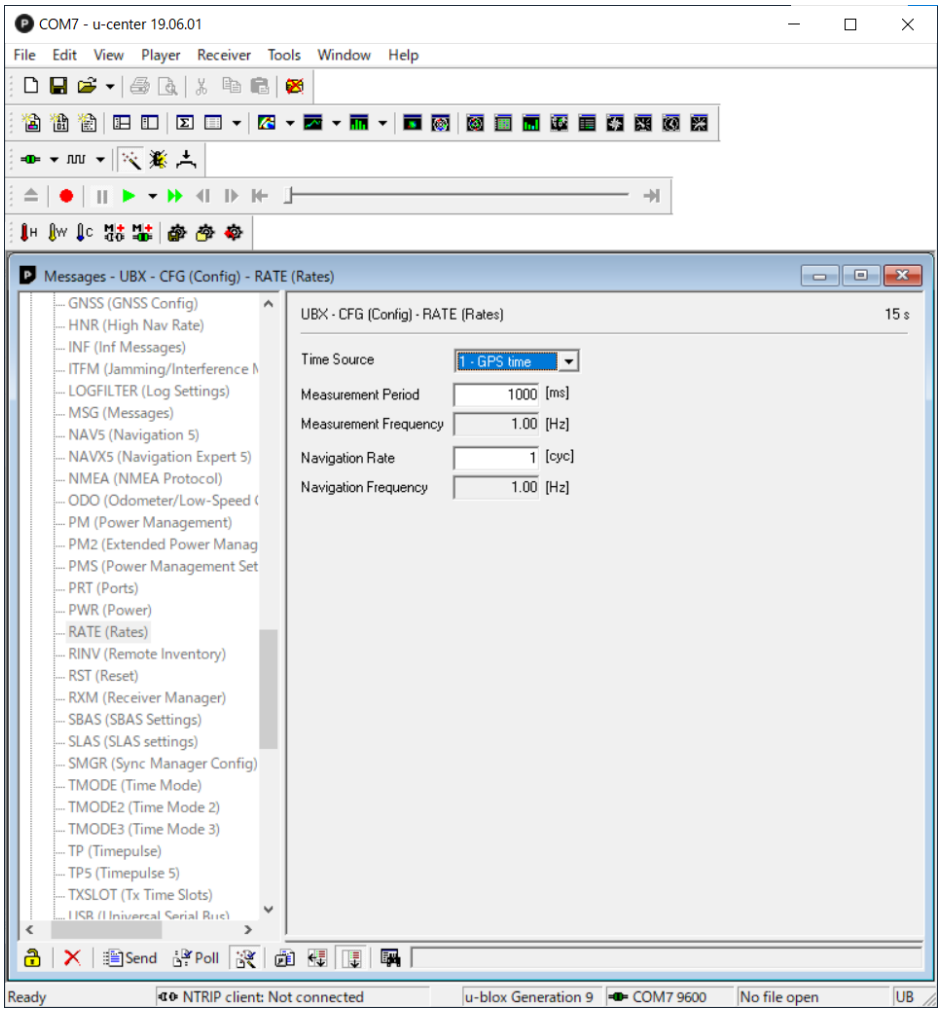


図 A.6 測位データの出力頻度の設定画面

A.1.7 基準局の設定

測位で使う基準局を指定する．タブの「Receiver – NTRIP Client」を順に選択すると図 A.7 に示すような画面が表示される．

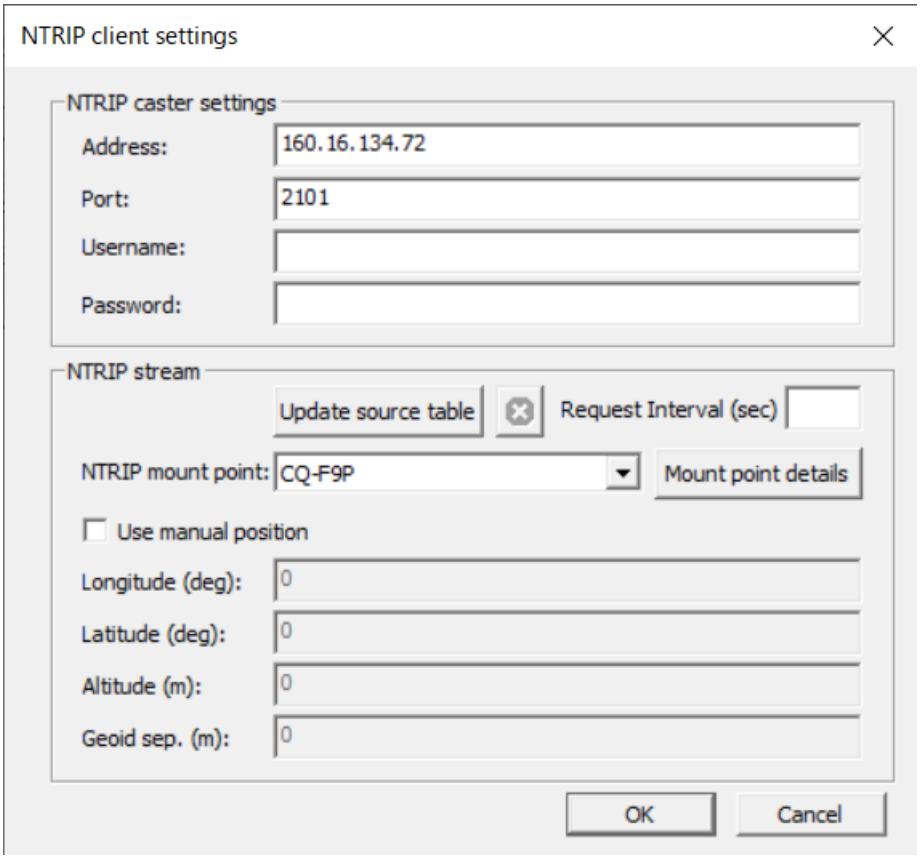


図 A.7 基準局情報入力画面

「NTRIP caster settings」欄に基準局のアドレス，ポート番号，名前，パスワードを入力する．
「NTRIP stream」欄には「NTRIP mount point」にマウントポイントを入力する．

A.1.8 各状況確認の例

画面上部にあるアイコンをそれぞれ押すことで図 A.8 のように各状況の確認をすることができる．

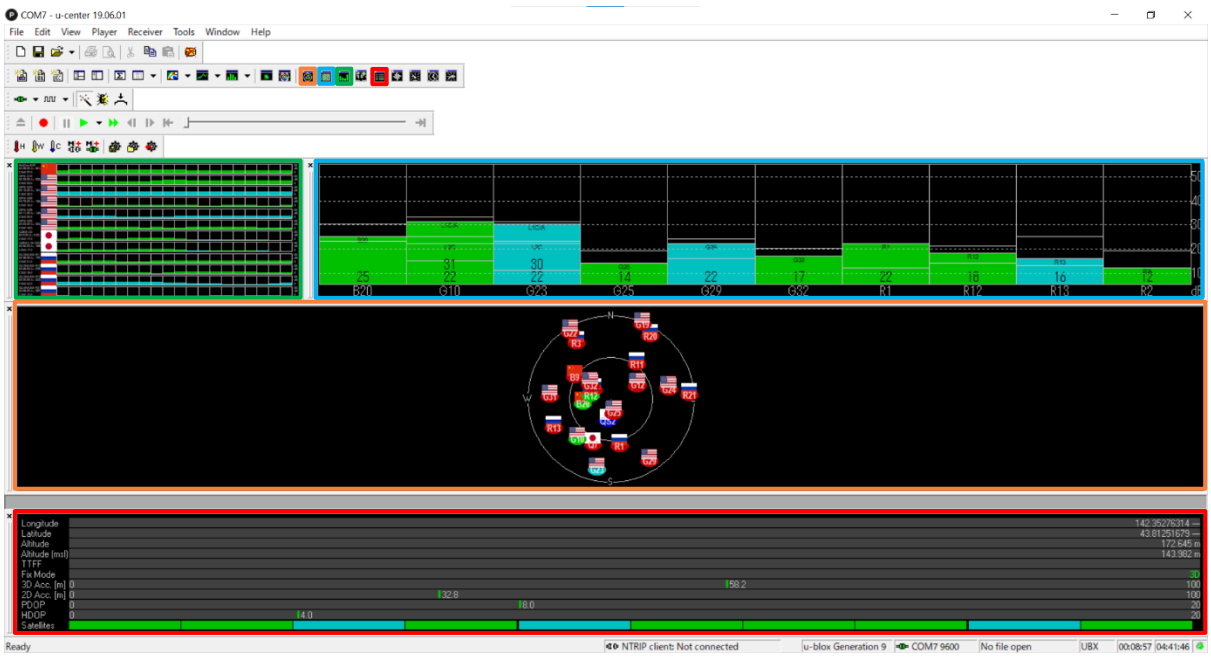


図 A.8 各機能

橙色枠内では各衛星の位置が確認できる．青色枠内では信号強度が確認できる．緑色枠内では各信号の時間経過が確認できる．赤色枠内では経度，緯度，高度，測位モードなどが確認できる．

A.2 strsvr のセットアップ

strsvr は基準局から測位データを受信し，複数のストリームに測位データを送信することができる．RTKLIB フォルダの「strsvr.exe」を開くと図 A.9 に示すような画面が表示される．

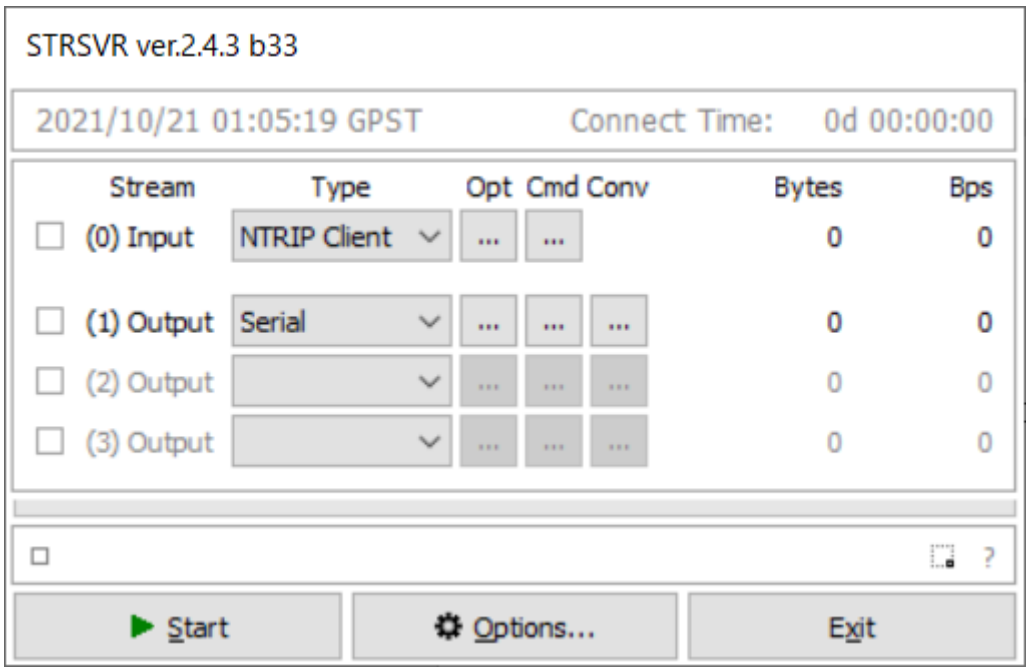


図 A.9 strsvr 初期画面

図 A.9 の「Input」に基準局，「Output」に移動局の設定を行う．
まずは「Input」の「Type」欄に「NTRIP Client」を選択し，「Opt」を押すと，図 A.10 に示す画面が表示される．
各項目に基準局の情報を入力し，「OK」を押す．

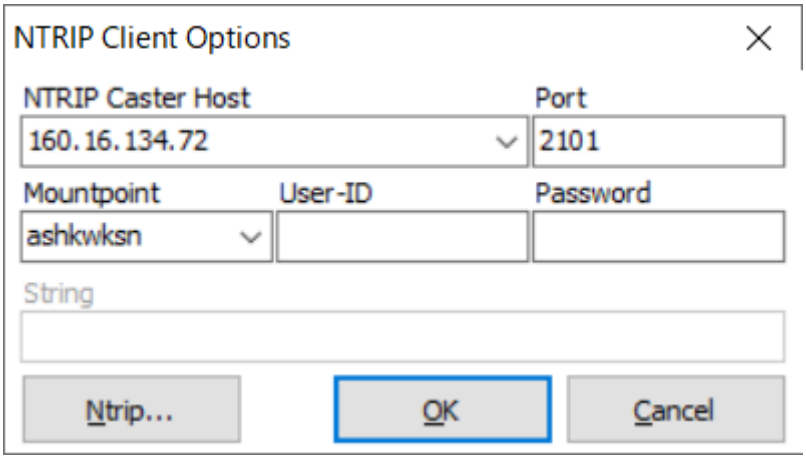


図 A.10 基準局の設定画面

次に，図 A.9 の「Output」の「Type」欄に「Serial」を選択し，「Opt」を押すと図 A.11 に示す画面が表示される．

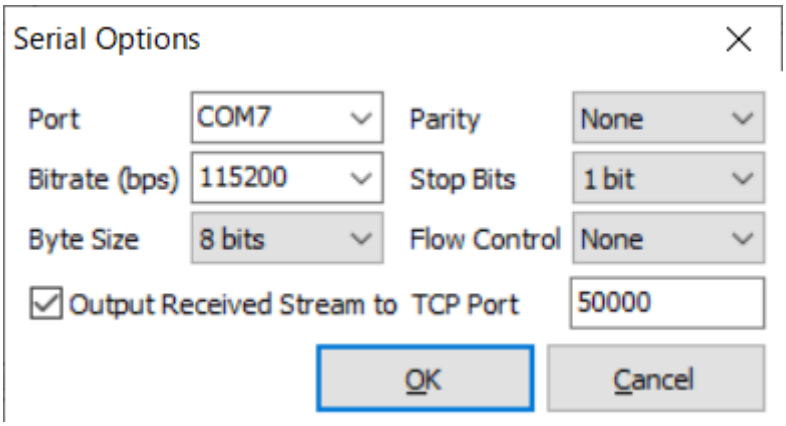


図 A.11 移動局の設定画面

図 A.11 の「Port」欄には，ZEP-F9P を接続したポート番号を選択する．ポート番号はパソコンの「コントロールパネルのデバイスとプリンター」または「デバイスマネージャー」にて確認することができる．「Bitrate(bps)」欄には，「115200」を選択する．「9600」では欠落が起こる可能性がある．rtkplot を用いて座標をプロットするため，「Output Received Stream to TCP Port」をチェックし，TCP ポートを指定する．ここで指定した TCP ポートに測位データを流す．他のポート番号と被らないような番号を決め，「OK」を押す．

図 A.9 の「Start」を押すと測位データの受信が始まり，「Input」と「Output」の左側の四角が緑色に変化する．緑色にならず，黄色や赤色の場合は基準局または移動局の設定が間違っている可能性がある．

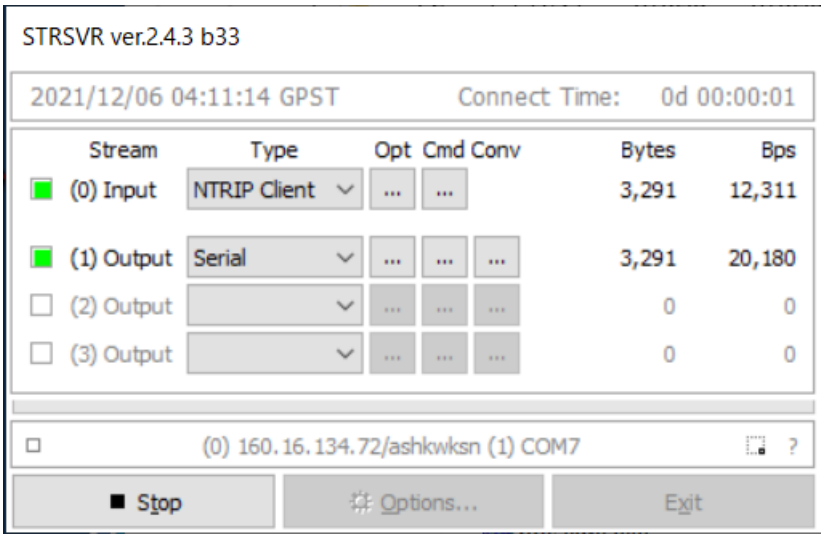


図 A.12 測位データ受信中の画面

A.3 rtkplot の使い方

rtkplot は strsvr と併用することで，簡単に現在座標をプロットすることができるソフトウェアである．RTKLIB フォルダの「rtkplot.exe」を開くと，図 A.13 に示すような画面が表示される．

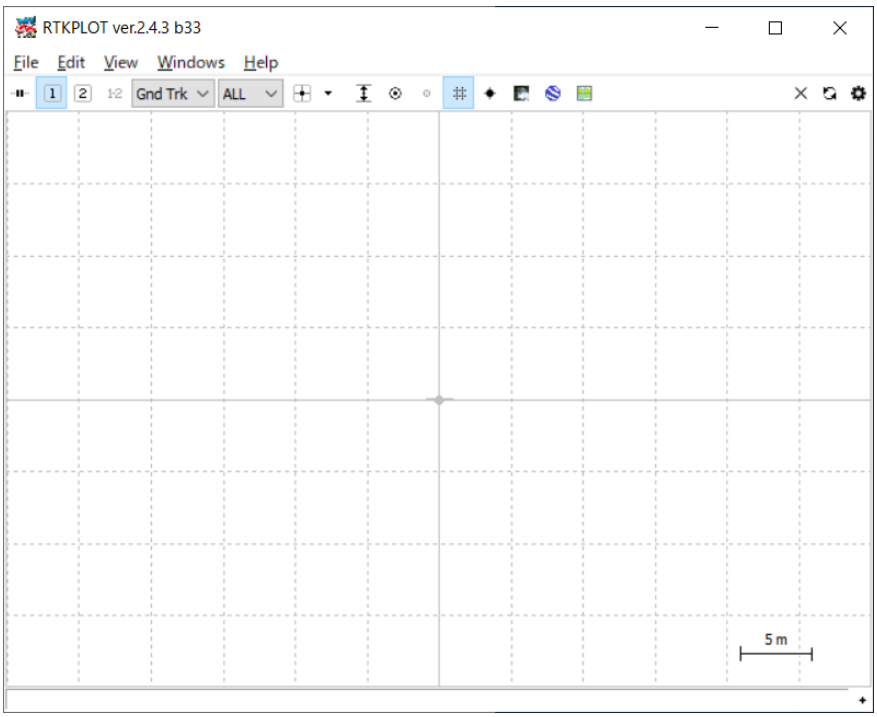


図 A.13 rtkplot 初期画面

まず，タブの「File – Connection Setting」を順に選択すると，図 A.14 に示すような画面が表示される．

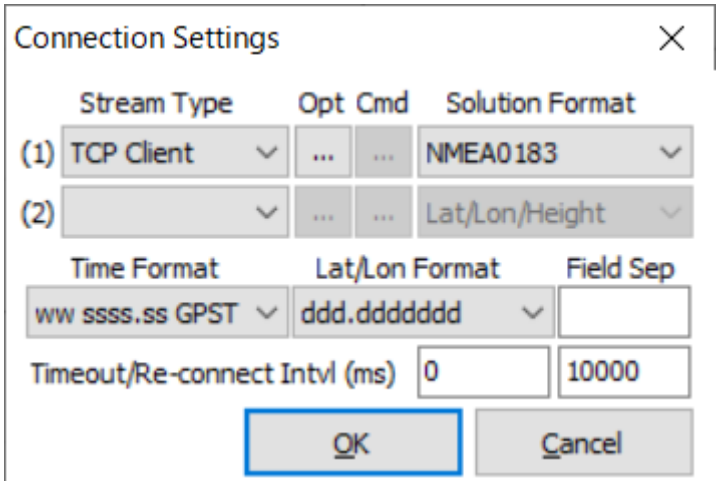


図 A.14 接続設定画面

図 A.14 の「Stream Type」欄に「TCP Client」を選択し，「Opt」を押すと図 A.15 に示すような画面が表示される．

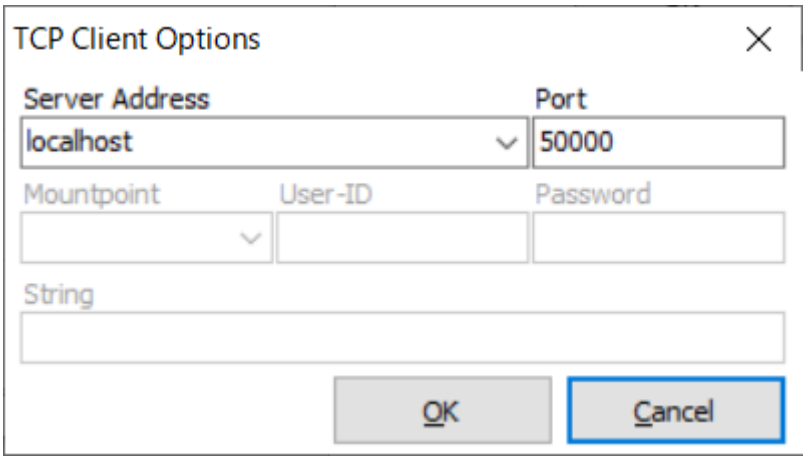


図 A.15 測位データの受信元の TCP ポート設定画面

図 A.15 の「Server Address」欄に「localhost」と入力し，「Port」欄に A.2 項 strsvr セットアップ時に決めた TCP ポートと同じ番号を入力し,「OK」を押す. strsvr を開き, 図 A.9 の「Start」を押し，図 A.13 タブの「File – Connect」を順に選択すると図 A.16 に示すように現在座標が表示される．緑色の丸印で表示されているのが，現在座標である．

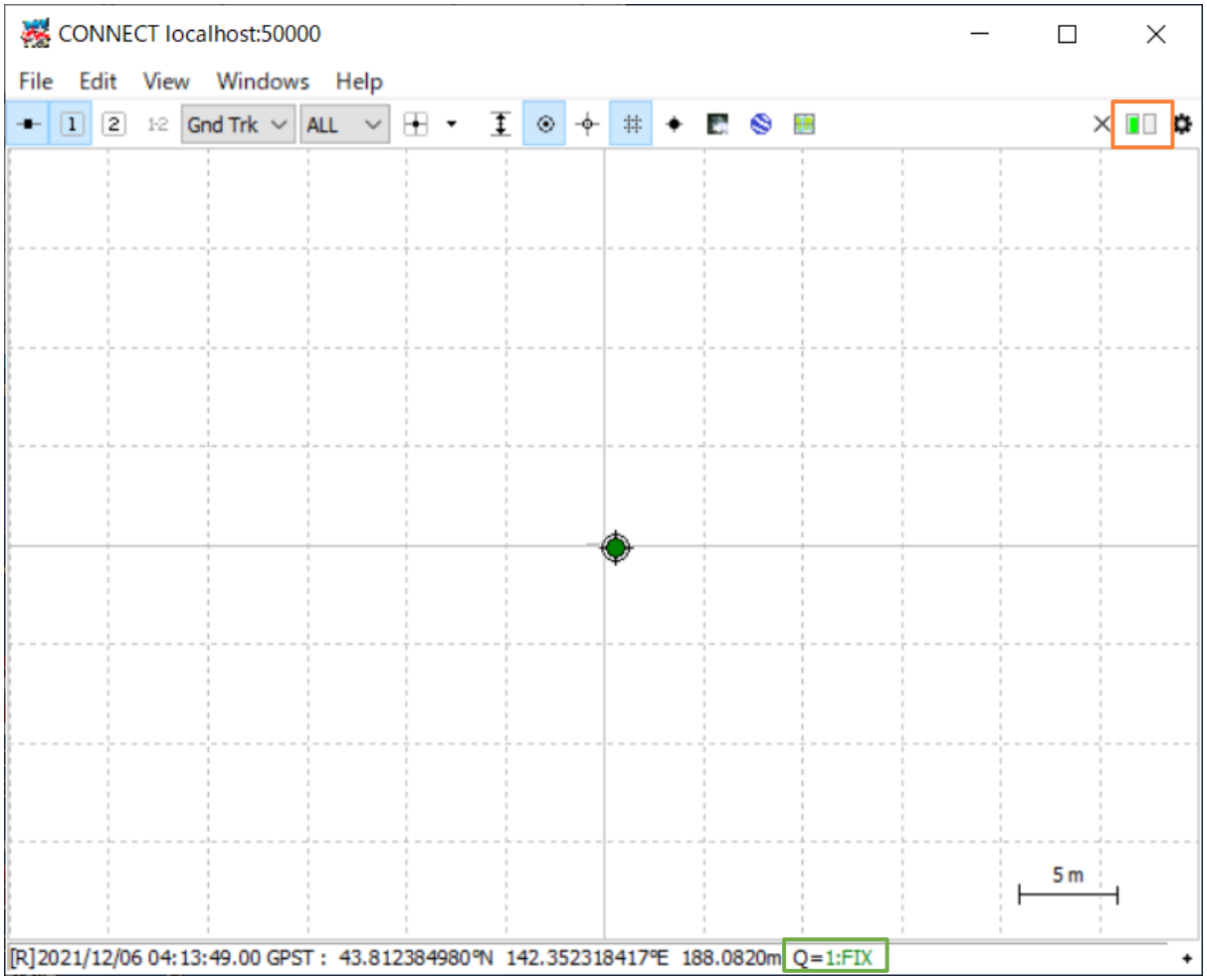


図 A.16 座標プロットの様子

測位データが指定の TCP ポートから得ることができると図 A.16 右上の橙色枠内が緑色で表示される．下の緑色枠内には測位品質フラグが表示される．単独測位(Single）は 5(赤)，DGPS 測位は 4(青)，Fix 解に測位は 1(緑)，が表示される．

付録B　制御プログラムと回路

本章では，RTK 台車の動作や制作に関わる内容を掲載する．B.1 節では PC やマイコンの動作に必要なプログラムのソースコード，B.2 節では回路図と実際の機器の対応関係を示す．

B.1　制御プログラムのソースコード

RTK 台車の走行に使用しているソースコードをここに掲載する。ソースコードは PC 側と Arduino マイコン側で 2 種類あり、PC 側は Python、Arduino マイコン側は Arduino 言語（C/C++に近い[9]）でプログラムの作成を行っている。

◆ PC 側（プログラム名：daisha_PC.py）

```
  ''' daisha_PC_v2.0
#########################################################################

・RTK モジュールから、位置情報を取得する。
・取得した位置情報(スタートとゴールの座標)から、直線経路の方程式（ax+by+c=0）を算出する。
・直線経路からのずれ量を計算し、マイコンに送信する。
・左折用の経路に，始点と終点をつなぎ変える

#########################################################################
############# '''

import socket        # ソケット通信用モジュール
import math      # 数学関係モジュール
import serial        # シリアル通信用モジュール
import time      # 時間関数用モジュール
import re        #北緯と東経を分離するために使用

# ###### --左折用(カンマ区切り版)-- 出発点と目的地(スタート→中間点)
########################################

#座標（スタート，中間点 1，中間点 2...，ゴール）のテキストファイルを開く
fp = open('route.txt', 'r', encoding='UTF-8')

#ファイルを文字列として読み込む
route_txt=fp.read()
```

```
fp.close()

#それぞれの行のデータをリストの一要素にする.
route_data_set=route_txt.splitlines()
print(route_data_set)
pattern="(.*),(.*),(.*)"

Point_NUM=len(route_data_set)
print("経路上の点の個数:{0}個".format(Point_NUM))
print("")

#ファイルから取得した座標を表示する.
for i in range(Point_NUM):
    Point_info=re.search(pattern, route_data_set[i])
    Label=Point_info.group(1); LAT=Point_info.group(2);
LNG=Point_info.group(3)
    print("{0}: 北緯{1}, 東経{2}".format(Label,LAT,LNG))

print("")

def check_fix():    # fixしているか確認
    print("¥n 現在 fixしているか確認しています...")
    while True:
        codelist_check=get_codelist()   # $GNGGA センテンスを取得
        if int(codelist_check[6])==4:
            print("現在 fixしています。¥n")
            # time.sleep(1) # 1秒処理停止
            break

def min_deg(phi_min):# 分から度に変換
    # 緯度、経度共に dddmm.mmmm 表記なので100で割って分をすべて小数点以下にする
    f,i=math.modf(phi_min/100.0)  # i 整数部(ddd度), f 小数部(.mmmmmm)
    phi_deg=i+f*100.0/60.0 # ddd.dddd 度表記になる
    return phi_deg

def deg_rad(phi_deg):# 度からラジアンに変換
    phi_rad=phi_deg*math.pi/180.0
    return phi_rad
```

```
def get_course(LAT_s,LNG_s,LAT_f,LNG_f):# コースの数式係数を取得
    # LAT_s 緯度スタート, LNG_s 経度スタート
    # LAT_f 緯度終点, LNG_f 経度終点
    R=6378100.0 # 地球半径[m]
    dy=deg_rad(LAT_f - LAT_s)*R # y 方向変位(緯度)
    dx=deg_rad(LNG_f - LNG_s)*R # x 方向変位(経度)
    course=[ dy , -dx , 0.0 ]   # ax+by=0(原点を通るから) 右方向のずれが正
    return course

def get_codelist(): # $GNGGA センテンスを取得
    while True:
        j=0 # $の場所の定義
        data1 = str(s.recv(300)) # データ 300 バイト分取得
        while True:
            data2 = str(s.recv(300))
            data1 += data2  # データを追加する
            if len(data2) < 300: # 1パケット分を取得したらループ抜ける
               break

        j=data1.find("$GNGGA")  # 取得データから$GNGGA の頭文字$の要素番号を取得
        code=data1[j:j+88]      # $GNGGA の頭文字要素数から 88 文字目までを読み込
む
        if(code.count("$")==1)and(code.count("'")==0):
            # 欠陥がなければちょうど 88 文字で $マークが 1 個
            # なぜか'が入るときがあり, エラーが出るから
            codelist=code.split(',')    # $GNGGA センテンスを','区切りでリスト化
            break
    return codelist # コードのリストを返す

def get_d(a,b,c,LAT,LNG,LAT_s,LNG_s): #経路からのずれの量を取得
    R=6378100.0  #地球半径[m]
    y=deg_rad(LAT - LAT_s)*R  #現在地と出発地の y 方向変位
    x=deg_rad(LNG - LNG_s)*R  #現在地と出発地の x 方向変位
    d = (a*x + b*y + 0.0) / math.sqrt(a*a + b*b) #経路からのずれ量
    return d #経路からのずれ量を返す

def get_limit_d(LAT,LNG,LAT_f,LNG_f): #経路からのずれの量を取得
```

```
    R=6378100.0  #地球半径[m]
    y=deg_rad(LAT - LAT_f)*R  #現在地と出発地の y 方向変位
    x=deg_rad(LNG - LNG_f)*R  #現在地と出発地の x 方向変位
    limit_d = math.sqrt(x*x + y*y) #目的地からの距離
    return limit_d #目的地からの距離

def get_edge(LAT1,LNG1,LAT2,LNG2):  # 2地点の距離を取得
    R=6373100.  #地球半径[m]
    dy=deg_rad(LAT2 - LAT1)*R
    dx=deg_rad(LNG2 - LNG1)*R
    edge=math.sqrt((dx)*(dx) + (dy)*(dy))
    return edge

limit_d=20          # ずれ量許容範囲[m]
limit_LL=0.0000004      # 緯度経度許容範囲[deg]

ser = serial.Serial('COM6',115200,timeout = 0.1)   # シリアル通信開始 ここには
Arduino のシリアルポート番号を設定する.

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:    # ソケット通信
開始
    s.connect(('localhost',50000))      # IPf vvvvca アドレス、ポート番号指定
    check_fix()                 # fix しているか確認

    #input("計測を開始するときは Enter を押してください。")
    print("計測開始")
    print("出発点から動きます")



    time.sleep(36)
    starttime=time.time()

    flag=0
    flag2=0
    #i=0
    for i in range(Point_NUM-1):

        print("")
```

```
        Point_info_s=re.search(pattern, route_data_set[i])
        Point_info_f=re.search(pattern, route_data_set[i+1])
        Label_s=Point_info_s.group(1); LAT_s=float(Point_info_s.group(2));
LNG_s=float(Point_info_s.group(3))
        Label_f=Point_info_f.group(1); LAT_f=float(Point_info_f.group(2));
LNG_f=float(Point_info_f.group(3))
        print("{0}から{1}に向かって走行中".format(Label_s,Label_f))
        print("{0}: 北緯{1}, 東経{2}".format(Label_s,LAT_s,LNG_s))
        print("{0}: 北緯{1}, 東経{2}".format(Label_f,LAT_f,LNG_f))

        course=get_course(LAT_s,LNG_s,LAT_f,LNG_f)  # コースの数式係数を取得
        a=course[0];b=course[1];c=course[2]     # ax+by+c=0
        limit_d=20

        j=0
        k=0
        while True:
            #print("内側ループ:",j,sep='')
            j=j+1
            codelist=get_codelist()
            Time=round((float(codelist[1])/10000+9),4)  # 日本時間
            NS=codelist[3]          #  北緯南緯
            LAT=min_deg(float(codelist[2])) # 緯度の単位を度に変換
            EW=codelist[5]          # 東経西経
            LNG=min_deg(float(codelist[4])) # 経度の単位を度に変換
            FF=int(codelist[6])     # fix か float か : fix は 4, float は 5 を判
断


            if FF==4:               # fix は 4, float は 5 を判断

                if limit_d <= 0.4:
                    flag=1
                    time.sleep(10)
                    print("ストップ")
                    break
```

```
                d=get_d(a,b,c,LAT,LNG,LAT_s,LNG_s)  # 経路からのずれの量[m]を取
得
                edge=get_edge(LAT_s,LNG_s,LAT,LNG)  # 出発地と現在地の距離を取得
                limit_d=get_limit_d(LAT,LNG,LAT_f,LNG_f) # 目的地からの距離[m]
を取得

                currenttime=time.time()
                #print("limit_d={0}".format(limit_d))

                if(currenttime-starttime > 3.5):    # 2 秒毎にシリアル通信 走行
用の周期は 3 秒
                    k=k+1

                    if  flag2==1:
                        print("d=0 による走行を終了しました. ")
                        flag2=0

                    starttime=currenttime

                    d_cm=d*100 #ずれ量を[m]から[cm]に変換

                    if d_cm>=127:
                        d_cm=127

                    if d_cm<=-127:
                        d_cm=-127

                    #目的地に到着したときは，ずれ量がないと仮定する
                    if flag==1:
                        d_cm=-128
                        print("d={0}に初期化して走行中(3 秒間)".format(d))
                        flag2=1
                        #time.sleep(3)
                        flag=0

                    d_int=int(d_cm)
                    if d_int<0:
                        d_int=256-abs(d_int)
```

```
            bina_d=bytes([d_int])

            print(k,LAT,LNG,limit_d,d,sep=",")

            ser.write(bina_d) #マイコンに値を書き込むと，車が動き始める

            time.sleep(0.1)
            c = ser.read() #マイコンから値を読み取ってくる



    else:
        print("float")  # Fix 解以外をまとめて Float 解とする
        continue
```

◆ Arduino 側（プログラム名：daisha_Arduino.ino）

```
/* daisha_Arduino_v2.0 ****************************

  ・PC と磁気コンパスから、ぞれぞれずれ量と角度を取得する。
  ・取得したずれ量と角度を使用して、スタート地点から中間点へ直線走行を行い，中間点からゴール地点は 45°曲がった直線経路を設定する.

**************************************************/

// モータードライバ TA8428K は前輪操舵のために使う
// バッテリーは6V 使用する
// 後輪はモータードライバはリレー制御で前進のみ
// 前輪を左右に往復させロータリーエンコーダで回転角を検出する
// Rotary Encoder 読み込み:割り込み使用
// 初めにハンドルを左限界に回して enc_countA = 35 とする
// 反時計回り++    時計回り--
// 常にハンドル真ん中 enc_countA=17 で停止させる

// ******************************************************/
#include<math.h>
#include <Wire.h> //I2C ライブラリ
#include <Stepper.h>
#define MOTOR_1  (32)  // orange
#define MOTOR_2  (33)  // yellow
#define MOTOR_3  (34)  // green
#define MOTOR_4  (35)  // blue

#define Addr_Mag 0x13

#define SW  (36)
#define MOTOR_STEPS (2048) // 出力軸 1 回転ステップ数:2048(2 相磁励)
// カタログ値です 1 ステップ 0.17578 度、10 ステップ 1.7578 度

// Relay pin //
#define RELAY1 (30)
#define RELAY2 (31)
```

```
// センサーの値を保存するグローバル関数
int   xMag;
int   yMag;
int   zMag;


const int pinA = 19;//ロータリーエンコーダ A 相 割り込み番号 4
const int pinB = 18;//ロータリーエンコーダ B 相
volatile long enc_countA = 0;
const int STEER_IN1 = 7;   // 前輪操舵用
const int STEER_IN2 = 8;   // 前輪操舵用
const int STEER_IN1_B = 9;    // 前輪操舵用 予備
const int STEER_IN2_B = 10;   // 前輪操舵用 予備
const int duty0 = 0;

void compass_Rawdata();
void compass_Rawdata_Real();
double getDirection();
void stopMotor() ;

int flag = 0; //曲がり角の検出を保存するグローバル変数
int t;

// ライブラリが想定している配線が異なるので 2 番、3 番を入れ替える
Stepper myStepper(MOTOR_STEPS, MOTOR_1, MOTOR_3, MOTOR_2, MOTOR_4);

//pinA の割り込み処理
void enc_RisingPinA()
{
  if (( digitalRead(pinA) == 0 && digitalRead(pinB) == 1 ) ||
( digitalRead(pinA) == 1 && digitalRead(pinB) == 0))
    --enc_countA; // ハンドルを上から見て時計回りで--
  else if (( digitalRead(pinA) == 0 && digitalRead(pinB) == 0 ) ||
( digitalRead(pinA) == 1 && digitalRead(pinB) == 1))
    ++enc_countA; // ハンドルを上から見て反時計回りで++
}

int duty_s = 255; // 前輪操舵用モータ duty 比 (0~255)
```

```
unsigned long tm0;
int ii = 0;

const double One_step_angle = 360.0 / MOTOR_STEPS;
const int User_steps = 57;// 回転してほしいステップ数 57 で 10 度です
double v_angle = 0;
double cal_x = 0;
double cal_y = 0;
double cal_x_real = 0;
double cal_y_real = 0;
double ave_cal_x = 0;
double ave_cal_y = 0;
double cal_x_north = 0;
double cal_y_north = 0;
double rd = 0;
double rd_north = 0;
//double seihen = 0.16872;
//磁北の状態 旭川市春光台では西偏 9 度 40 分 = 9.667 度 = 0.16872rad
// しかし電子コンパスは磁北を指さないので西偏値を引く必要は無い
int count = 0;
int rot_dir = 0;
int hensa = 0;
//ステッピングモータ回転方向 rot_dir=0 は上から見て反時計回り
//ステッピングモータ回転方向 rot_dir=1 は上から見て時計回り
//ステッピングモータ回転方向 rot_dir=4 は停止


int dest = 17;//ロータリーエンコーダとギヤ取り付け部がソフトなのでずれるが7にする 8 から
17.5 に変更
double theta;
int delta_l;



void setup() {
  Serial.begin(115200);// arduino IDE モニタ用
  while (!Serial);
  Serial3.begin(115200);// arduino TeraTerm モニタ用
```

```
  Serial.println("プログラム開始");
  Serial3.println("プログラム開始");
  xMag=yMag=zMag=1; //コンパスの値を初期化

  // RELAY Setting //
  pinMode(RELAY1, OUTPUT);
  pinMode(RELAY2, OUTPUT);
  digitalWrite(RELAY1, 1); // 0 -> RELAY on , 1 -> RELAY off
  digitalWrite(RELAY2, 1);

  // Rotary Encoder Setting //
  pinMode(pinA, INPUT);
  pinMode(pinB, INPUT);
  attachInterrupt(4, enc_RisingPinA, CHANGE); //両エッジで割り込み発生

  // 前輪操舵 PWM 用
  // Timer2(8bit timer) -> pwm 10, 9
  TCCR2B = (TCCR2B & 0b11111000) | 0x07; //30.64 [Hz]
  // Timer4(16bit timer) -> pwm 8, 7, 6
  TCCR4B = (TCCR4B & 0b11111000) | 0x05; //30.64 [Hz]

  myStepper.setSpeed(5);//5rpm ステッピングモータの回転速度
  // この回転速度で回り続けるのではなく
  // stepper.step(steps)で設定した steps 数をこの回転速度で回る
  // steps=10 ならば10ステップをこの回転速度で回る

  //Wire(Arduino-I2C)の初期化
  Wire.begin();

  //BMX055 初期化
  BMX055_Init();
  delay(300);

  // 前輪操舵ピン設定
  pinMode(STEER_IN1, OUTPUT);
  pinMode(STEER_IN2, OUTPUT);
  analogWrite(STEER_IN1, 255);
  analogWrite(STEER_IN2, 255);
```

```
enc_countA = 35; //14 から 35 に変更

while (1) {
  Serial.println(enc_countA);

  if ( enc_countA < dest) {
    // ハンドルを反時計回りに回転
    analogWrite(STEER_IN1, duty_s);
    analogWrite(STEER_IN2, duty0);
    ii = 0;
  }
  else if ( enc_countA == dest) {
    analogWrite(STEER_IN1, 255);
    analogWrite(STEER_IN2, 255);
    if (ii == 0) {
      tm0 = millis();
      ii = 1;
    }
    else {
      if ((millis() - tm0) > 3000) {
        enc_countA = 0;
        dest = 0;
        break;
      }
    }
  }
  else if ( enc_countA > dest) {
    // ハンドルを時計回りに回転
    analogWrite(STEER_IN1, duty0);
    analogWrite(STEER_IN2, duty_s);
    ii = 0;
  }
}

Serial.println(enc_countA);
ii = 0;

//ステッピングモーターを往復回転だけさせる
```

```
  while (rot_dir != 5) {
    if (rot_dir == 0) { // rot_dir=0 は上から見て反時計回り
      myStepper.step(User_steps);  // User_steps だけ回す
      v_angle = v_angle + 10.0;
      // 電子コンパスのデータ取得
      count++;
      compass_Rawdata();
      if (v_angle >= 360.0) rot_dir = 1;
    }
    else if (rot_dir == 1) { // rot_dir=1 は上から見て時計回り
      myStepper.step(-1 * User_steps); // -1*User_steps だけ回す
      v_angle = v_angle - 10.0;
      // 電子コンパスのデータ取得
      count++;
      compass_Rawdata();
      if (v_angle <= 0.0) rot_dir = 4;
    }
    else if (rot_dir == 2) {
      myStepper.step(User_steps);  // User_steps だけ回す
      v_angle = v_angle + 10.0;
      // 電子コンパスのデータ取得
      count++;
      compass_Rawdata();
      if (v_angle >= 360.0) rot_dir = 3;
    }
    else if (rot_dir == 3) {
      myStepper.step(-1 * User_steps); // -1*User_steps だけ回す
      v_angle = v_angle - 10.0;
      // 電子コンパスのデータ取得
      count++;
      compass_Rawdata();
      if (v_angle <= 0.0) rot_dir = 4;
    }
    else if (rot_dir == 4) {
      stopMotor();
      delay(500);
      rot_dir = 5;
      Serial.println("平均値");
```

```
      ave_cal_x = cal_x / count; //楕円中心のx座標
      Serial.println(ave_cal_x);
      ave_cal_y = cal_y / count; //楕円中心のy座標
      Serial.println(ave_cal_y);

      compass_Rawdata_Real();
      cal_x_north = cal_x_real - ave_cal_x;
      cal_y_north = cal_y_real - ave_cal_y;
      rd_north = getDirection(cal_x_north, cal_y_north);// y 軸が角度の基準としている

      // rd_north は台車を置いた位置での磁北の方向
    }
    delay(100);
  }

  delay(400);
}

double U;   //制御量 ロータリエンコーダのパルス数
double Sum_y = 0.0; //積分要素


//フィードバック制御プログラム
int Feed_Back(double delta_rad, double delta_m) {

  // delta_rad [rad]です。 delta_m [m]です。

  float k[3];     //フィードバックゲイン

  double angle_num = 6; //この値を 12→35→6 に変更 ここの数字が大きくなると，蛇行が大きくなる。小さくし過ぎても操舵が出来なくなった。
  double DR = 12; //DR:Dual Rate ，舵角のこと。中心から片側に操舵したときに出力される、(ロータリーエンコーダの)パルスのカウント数。
  double delta_rad_2; //補正後の角度

  ////////////////////////////////////////////////////
  // 比例・積分制御                                 //
  // 進行方向の角度とずれ量の2つの比例制御 //
```

```
  ////////////////////////////////////////////////////

  k[0] = 2.0;// 進行方向の角度のゲイン（初めは 2.0）
  k[1] = 50.0; // ずれ量のゲイン（初めは 25.0）
  k[2] = 0.00;  // ずれ量の積分のゲイン

  if ((int)(delta_m * 100) == -128) {

    flag = 1;
    k[1] = 0; //左折時の 1 回のみ距離のゲインをゼロにする

  }

  delta_rad_2=delta_rad;

    if (flag==1){
      delta_rad_2=delta_rad-45*(PI/180); //左折をするとき、初期方位から 45°(π/4)ずれるため
      Serial3.print("flag = "); Serial3.println(flag);
      }


  double hen_rad = delta_rad_2 / (PI / angle_num) * DR;


  U =  (-k[0] * hen_rad  + k[1] * delta_m + k[2] * Sum_y);  //制御量の計算
  //Serial3.println(U);
  Sum_y = delta_m + Sum_y;

  Serial3.print("d = "); Serial3.println(delta_m);
  //Serial3.print("θ1 = "); Serial3.println(delta_rad * 180 / PI);
  Serial3.print("θ = "); Serial3.println(delta_rad_2 * 180 / PI);
  Serial3.print("U = "); Serial3.println(U);
  Serial3.print("¥n");


  if (U >= DR) U = DR;
  else if (U <= -DR) U = -DR;
```

```
  while (1) {
    if ( enc_countA < (int)U) {

      //Serial3.println("CCW"); // 反時計回り

      analogWrite(STEER_IN1, duty_s);
      analogWrite(STEER_IN2, duty0);
      Serial.print("1:enc_count,U, "); Serial.print(enc_countA);
Serial.print(','); Serial.println(U);
      //ii = 0;
    }
    else if ( enc_countA == (int)U) {

      //Serial3.println("Stop!");

      analogWrite(STEER_IN1, 255);// ブレーキ
      analogWrite(STEER_IN2, 255);// ブレーキ
      Serial.print("2:enc_count,U, "); Serial.print(enc_countA);
Serial.print(','); Serial.println(U);
      break;
    }
    else if ( enc_countA > (int)U) {

      //Serial3.println("CW"); // 時計回り

      analogWrite(STEER_IN1, duty0);
      analogWrite(STEER_IN2, duty_s);
      Serial.print("3:enc_count,U, "); Serial.print(enc_countA);
Serial.print(','); Serial.println(U);
      //ii = 0;
    }
  }
  return (0);
}

void loop() {

  get_theta_and_d(); //車体角度:theta とずれ量:d を取得する
```

```
  ////////////////////////
  // 前輪操舵制御をする //
  ////////////////////////
  int stop_f = Feed_Back(theta, (double)delta_l / 100); //delta_l/100 → 単位を[cm]から[m]にするため.



  /*距離を受信したら走行開始*/
  digitalWrite(RELAY1, 0); // 0 -> RELAY on , 1 -> RELAY off
  digitalWrite(RELAY2, 0);

  delay(1000);

  /*一定時間走行したら停止*/
  digitalWrite(RELAY1, 1); // 0 -> RELAY on , 1 -> RELAY off
  digitalWrite(RELAY2, 1);

  if (stop_f == 1) exit(0);
  delay(300);

}

void get_theta_and_d(void) {
  ////////////////////////
  // コンパスデータ受信です //
  ////////////////////////
  compass_Rawdata_Real();
  cal_x_real = cal_x_real - ave_cal_x;
  cal_y_real = cal_y_real - ave_cal_y;
  rd = getDirection(cal_x_real, cal_y_real);

  // atan:－π/2 から π/2

  theta = rd_north - rd; //角度の値を磁気コンパスのものに置き換え
  Serial3.print("theta_real = "); Serial3.println(theta);

  //PC からずれ量を取得
```

```
  while (1) {
    if (Serial.available() > 0) {
      byte cc = (byte)Serial.read();
      delta_l = (char)cc; //経路からのずれ量[cm]
      break;
    }
  }
}
/*磁気コンパス関連の関数*/

// コンパスの生データを取得する
void compass_Rawdata() {
  //BMX055 磁気の読み取り
  BMX055_Mag();
  Serial3.print("xMag,yMag,zMag,rot_dir,count :  ");
  Serial3.print(xMag); cal_x += (double)xMag;
  Serial3.print(",");
  Serial3.print(yMag); cal_y += (double)yMag;
  Serial3.print(",");
  Serial3.print(zMag);
  Serial3.print(",");
  Serial3.print(rot_dir);
  Serial3.print(",");
  Serial3.print(count);
  Serial3.print(",");
  Serial3.println();
  if (xMag==0&&yMag==0&&zMag==0){
    Serial3.print("compass value error!");
    xMag=yMag=zMag=1;
    delay(100);
    exit(0);
    }
}

void compass_Rawdata_Real() {
  //BMX055 磁気の読み取り
  BMX055_Mag();
```

```
  cal_x_real = (double)xMag;
  cal_y_real = (double)yMag;
  if (xMag==0&&yMag==0){
    Serial3.print("compass value error!");
    xMag=yMag=zMag=1;
    delay(100);
    exit(0);
    }
}

//** 角度を求める 出力は Radian **//
double getDirection(double x, double y) {
  double dir;
  if (y == 0) dir = 0;
  else if (x == 0 && y > 0) dir = PI / 2.0;
  else if (x == 0 && y < 0) dir = -PI / 2.0;
  else {
    dir = atan(y / x); // 戻り値 -pi/2 から +pi/2
    if (x < 0 && y >= 0) dir = dir + PI;
    else if (x < 0 && y <= 0) dir = dir - PI;
  }
  // 結果として -PI < dir < PI
  return dir;
}


//================================================================================
===========//
void BMX055_Init()
{
  //BMX055 初期化
  //-------------------------------------------------------------//
  Wire.beginTransmission(Addr_Mag);
  Wire.write(0x4B);  // Select Mag register
  Wire.write(0x83);  // Soft reset
  Wire.endTransmission();
  delay(100);
  //-------------------------------------------------------------//
  Wire.beginTransmission(Addr_Mag);
```

```
  Wire.write(0x4C);  // Select Mag register
  Wire.write(0x00);  // Normal Mode, ODR = 10 Hz
  Wire.endTransmission();
  //-------------------------------------------------------------//
  Wire.beginTransmission(Addr_Mag);
  Wire.write(0x4E);  // Select Mag register
  Wire.write(0x84);  // X, Y, Z-Axis enabled
  Wire.endTransmission();
  //-------------------------------------------------------------//
  Wire.beginTransmission(Addr_Mag);
  Wire.write(0x51);  // Select Mag register
  Wire.write(0x04);  // No. of Repetitions for X-Y Axis = 9
  Wire.endTransmission();
  //-------------------------------------------------------------//
  Wire.beginTransmission(Addr_Mag);
  Wire.write(0x52);  // Select Mag register
  Wire.write(0x0F);  // No. of Repetitions for Z-Axis = 15
  Wire.endTransmission();
}

//============================================================================
===========//
void BMX055_Mag()
{
  //BMX055 磁気の読み取り
  unsigned int data[6];

  for (int i = 0; i < 6; i++)
  {
    // Start I2C Transmission
    Wire.beginTransmission(Addr_Mag);
    // Select data register
    Wire.write((66 + i));
    // Stop I2C Transmission
    Wire.endTransmission();

    // Request 1 byte of data
    Wire.requestFrom(Addr_Mag, 1);
```

```
    // Read 6 bytes of data
    // xMag lsb, xMag msb, yMag lsb, yMag msb, zMag lsb, zMag msb
    if (Wire.available() == 1)
    {
      data[i] = Wire.read();
    }
  }

  // Convert the data
  xMag = ((data[1] * 256) + (data[0] & 0xF8)) / 8;
  if (xMag > 4095)
  {
    xMag -= 8192;
  }
  yMag = ((data[3] * 256) + (data[2] & 0xF8)) / 8;
  if (yMag > 4095)
  {
    yMag -= 8192;
  }
  zMag = ((data[5] * 256) + (data[4] & 0xFE)) / 2;
  if (zMag > 16383)
  {
    zMag -= 32768;
  }
}


// モーターへの電流を止める
void stopMotor() {
  digitalWrite(MOTOR_1, LOW);
  digitalWrite(MOTOR_2, LOW);
  digitalWrite(MOTOR_3, LOW);
  digitalWrite(MOTOR_4, LOW);
}
```

B.2　回路図と実際の写真

B.2.1 回路図

全体的な回路図は 4 章で示したが, 本節ではその回路図に番号をつけたもの (図 B.1, 図 B.2) を示す.

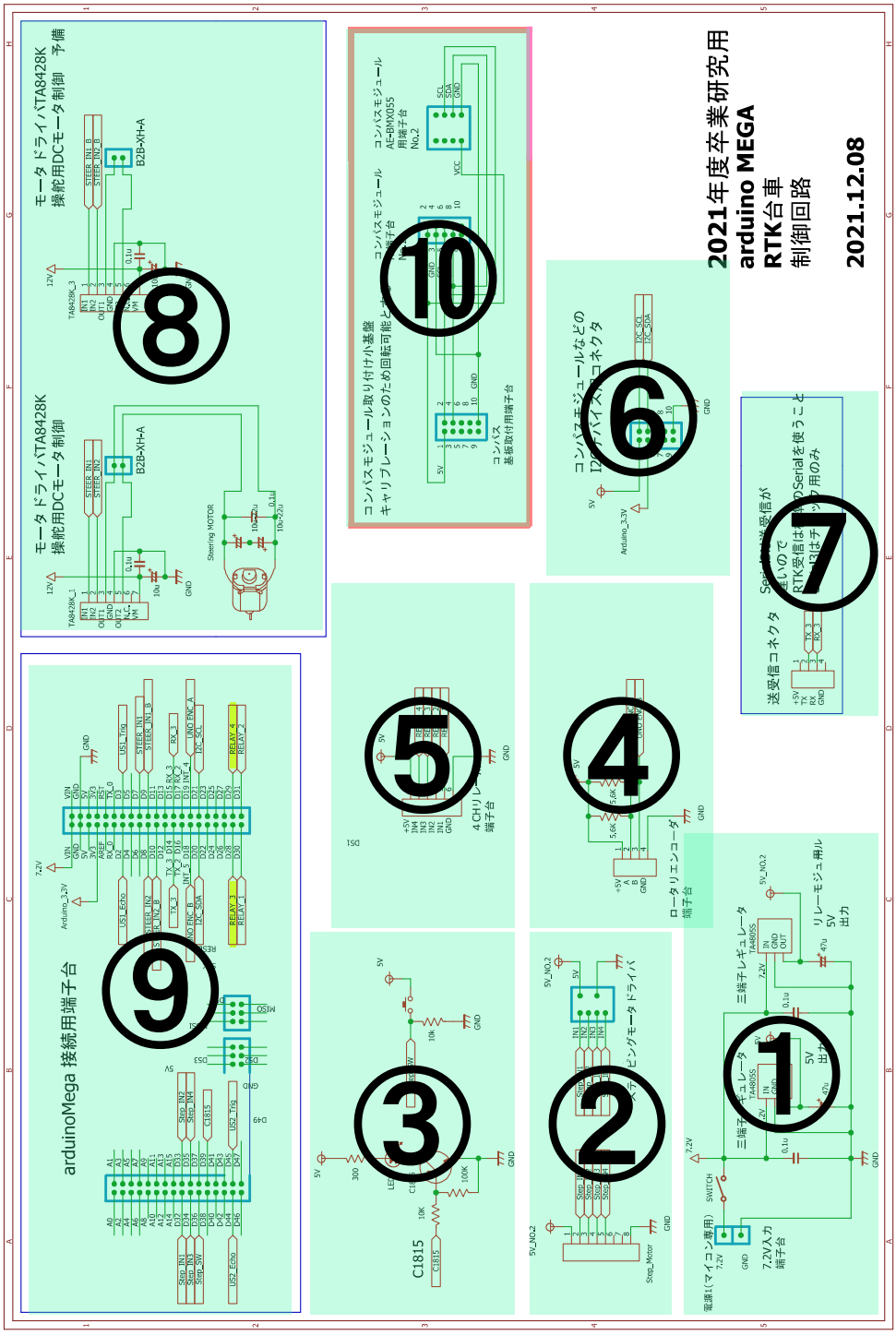


図 B.1　制御基板の回路図　基板 A【①～⑨】と基板 B【⑩】

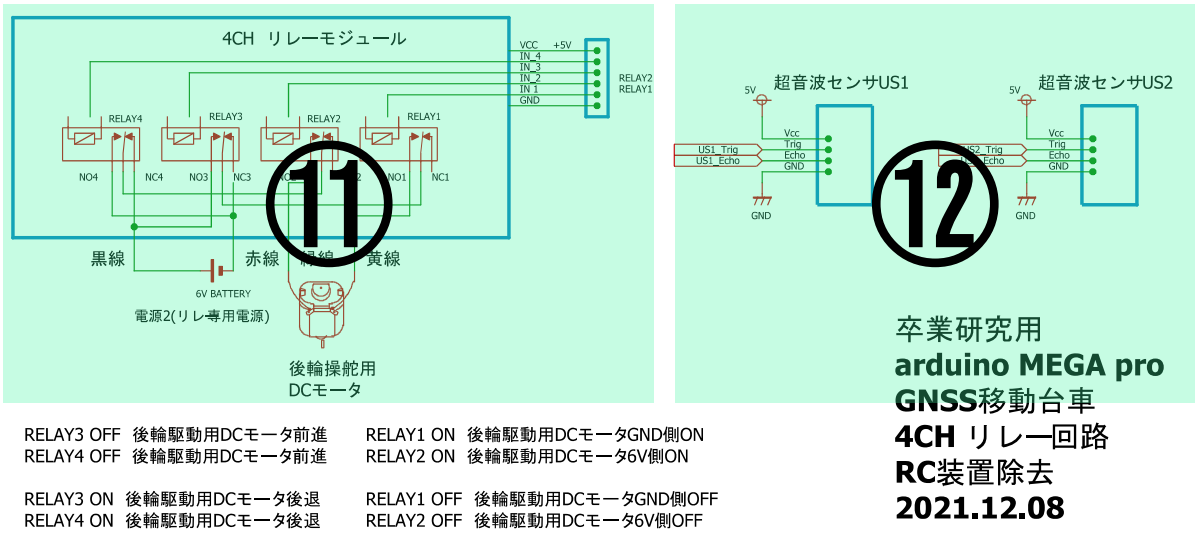


図 B.2 制御基板の回路図 基板 C【⑪】と基板 D【⑫】

B.2.2 実際の回路

B.2.1 項で回路図を示したが，実際の回路は図 B.3，図 B.4 のようになる．

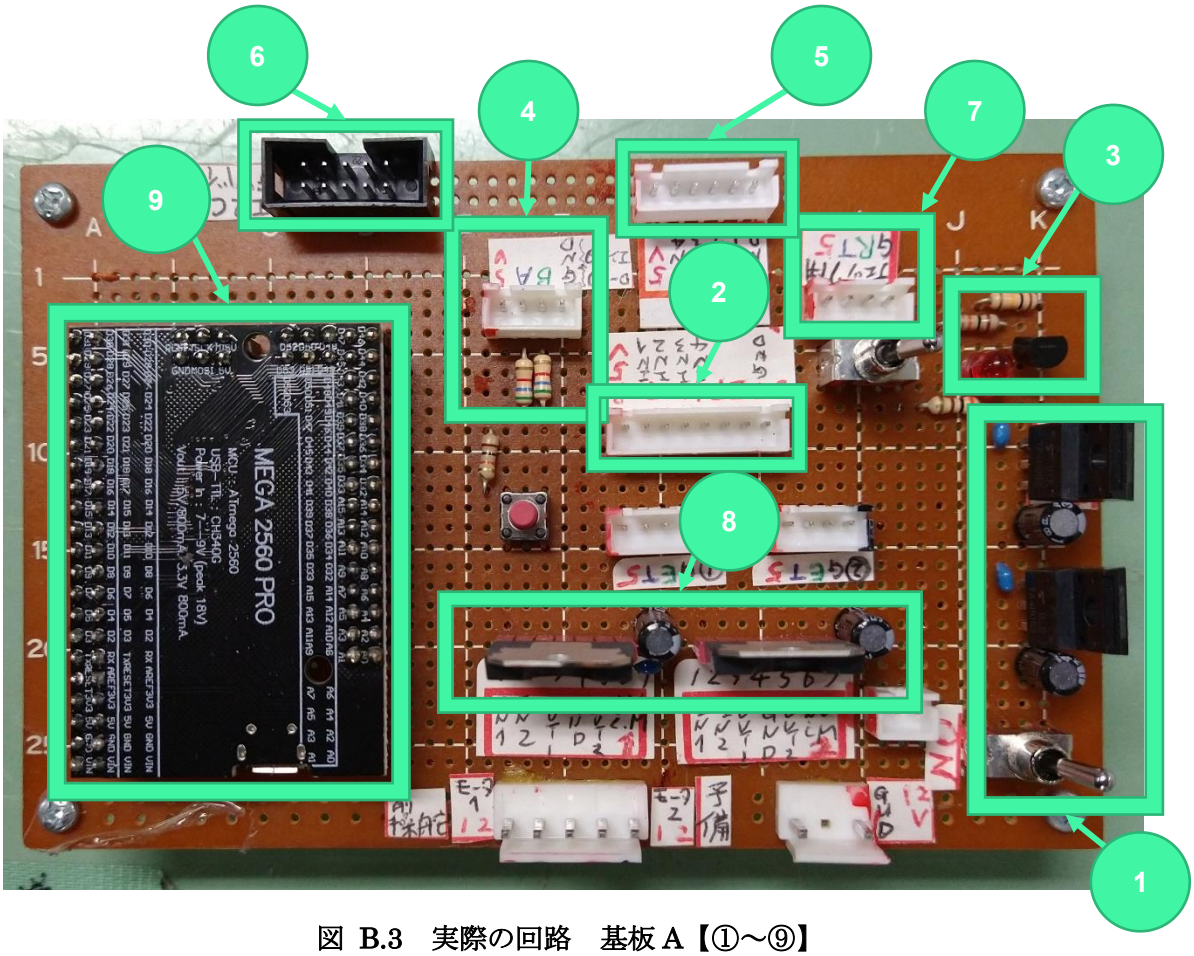


図 B.3 実際の回路 基板 A【①～⑨】

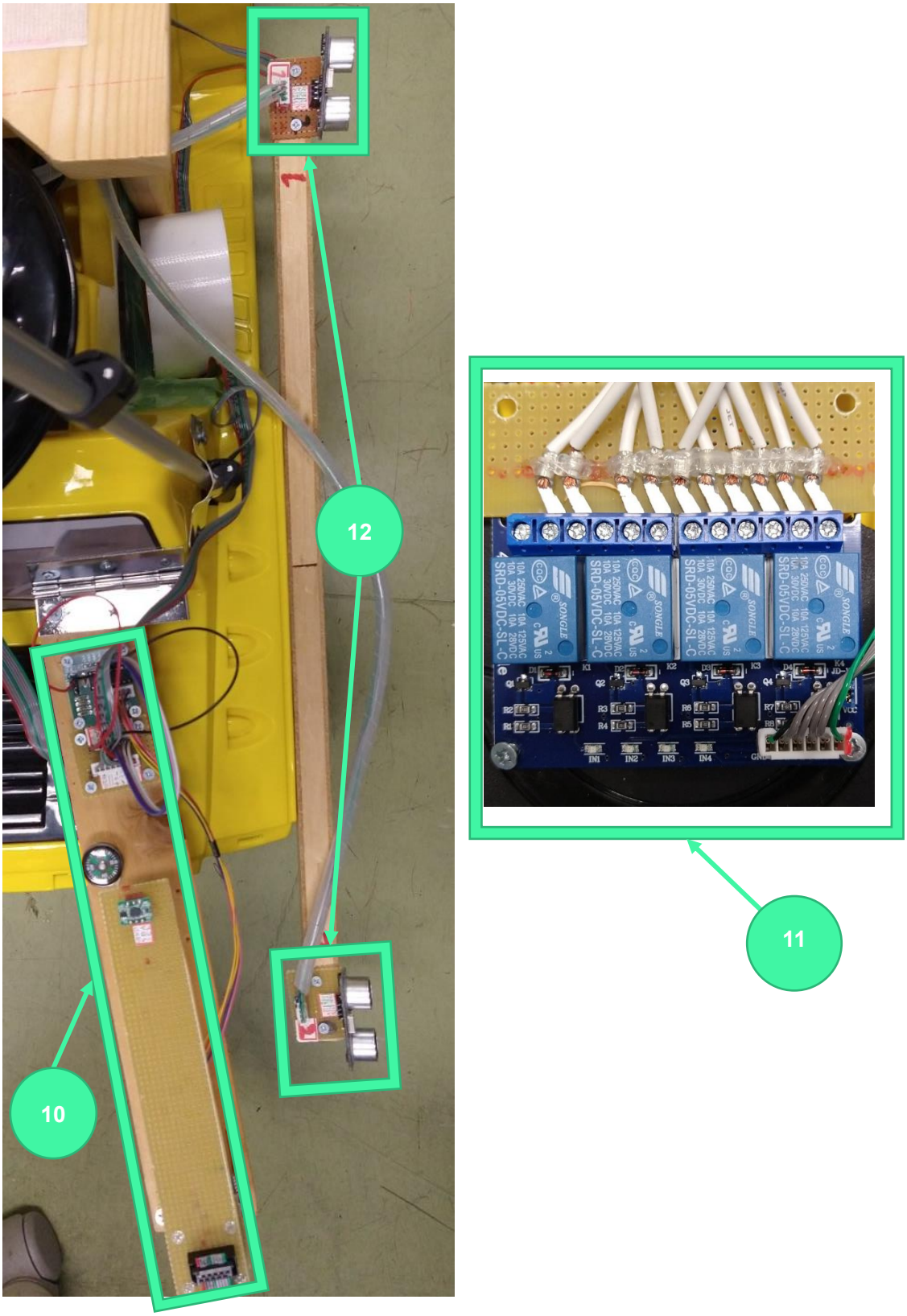


図 B.4　実際の回路　基板 B【⑩】, C【⑪】, D【⑫】

B.2.3 各部の説明

B.2.1 項で示した回路図のそれぞれの部分には重要な役割がある．ここでは，それぞれの回路が担っている機能について解説する．

＊ ①の部分 ＊

この部分の役割は大きく分けて 2 つある．

- **電圧を下げる**（6 V ⇒ 5 V）
- **電圧の平滑化・安定化**

電圧を下げるためには，三端子レギュレータ（型番：TA4805S）を使用している．**三端子レギュレータ**の入力（IN），出力側（OUT）に接続されている**コンデンサ**は，それぞれ電源の発振防止・平滑化[11]の役割を担っている。

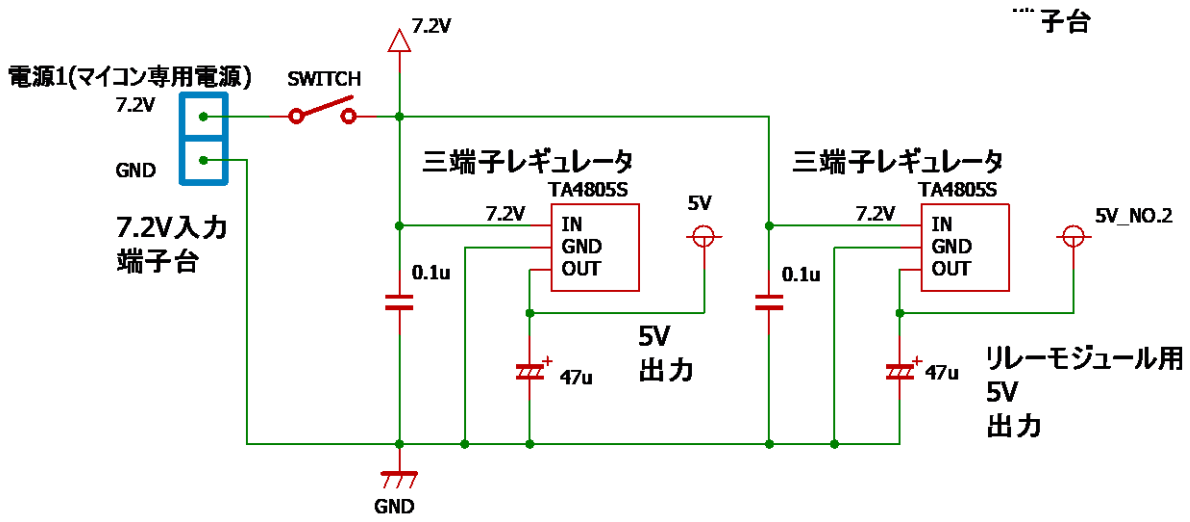


図 B.5 ①の部分 回路図

＊ ②の部分 ＊

ステッピングモータの駆動に使用する回路である．ステッピングモータは，キャリブレーション（3.2.1 項）時に**電子コンパスを回転**させるために使用する．

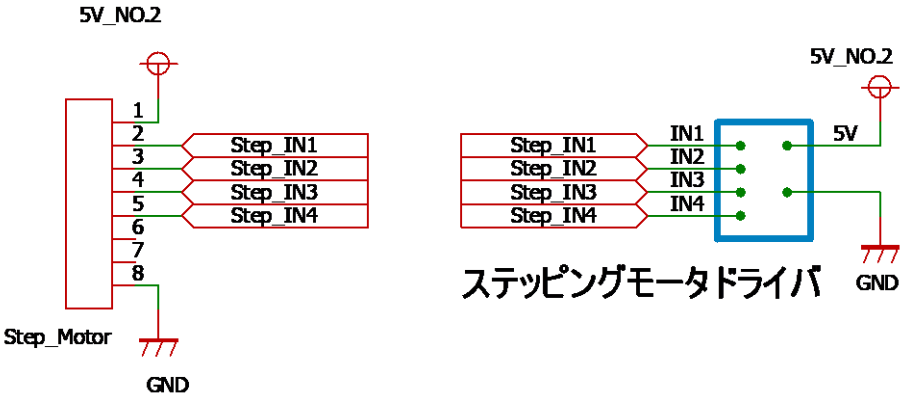


図 B.6 ②の部分 回路図

＊ ③の部分 ＊

LED **点灯**用の回路である．Arduino マイコンのデジタルピン（D39）に電圧をかけると，回路上の赤色 LED が点灯する．

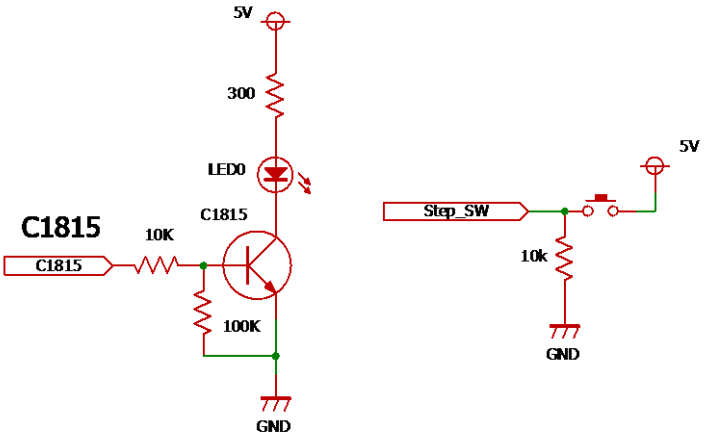


図 B.7 ③の部分 回路図

＊ ④の部分 ＊

ハンドルの切れ角検出用の回路である．**ロータリーエンコーダ**から出力される 2 種類（A 相，B 相と呼ばれる）のパルスの数を数えることで，ハンドルの切れ角を計算する．

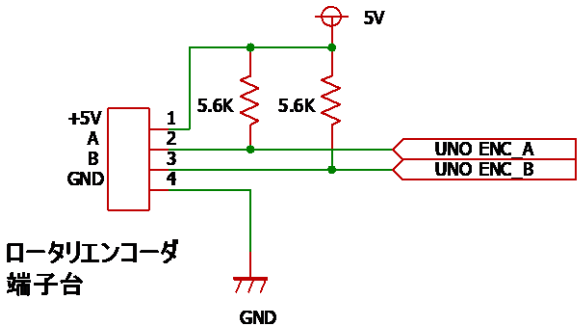


図 B.8 ④の部分 回路図

＊ ⑤の部分 ＊

後輪の駆動制御用回路である．リレーモジュールに 4 つの信号（1 か 0）を入力することで，モータ駆動用の**リレー**を制御する．

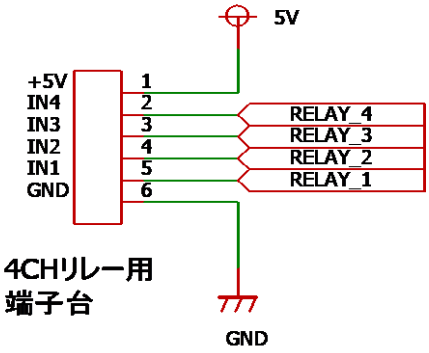


図 B.9 ⑤の部分 回路図

＊ ⑥の部分 ＊

電子コンパスの値をマイコンに取り込む回路である．Arduino マイコンに電子コンパスの地磁気の出力値が取り込まれる．

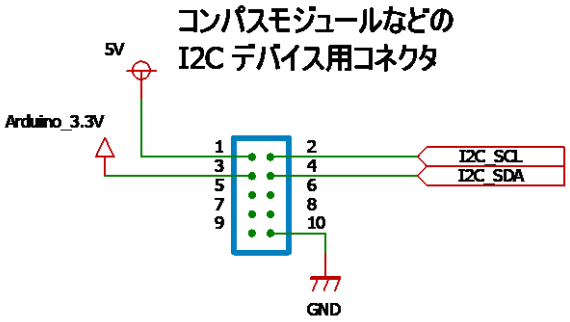


図 B.10 ⑥の部分 回路図

＊ ⑦の部分 ＊

Arduino **の値確認用**回路である．Arduino と PC 間でシリアル通信をすると，Arduino のシリアルモニターを使うことが出来ないという問題がある．この部分から値を出力して，PC に USB 接続することで**シリアルモニターを使用せずに値を観察**することが出来る．値の観察には，TeraTerm を使用した．

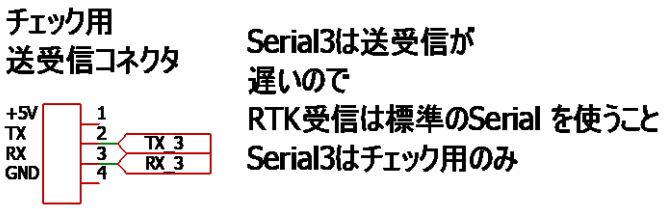


図 B.11 ⑦の部分 回路図

＊ ⑧の部分 ＊

前輪操舵用モータの駆動を行う回路である．モータの駆動には**モータドライバ IC**（型番：TA8428K）を使用している．モータドライバ IC を使用する理由は，マイコンの電源のみでは DC モータの駆動に不十分なためである．

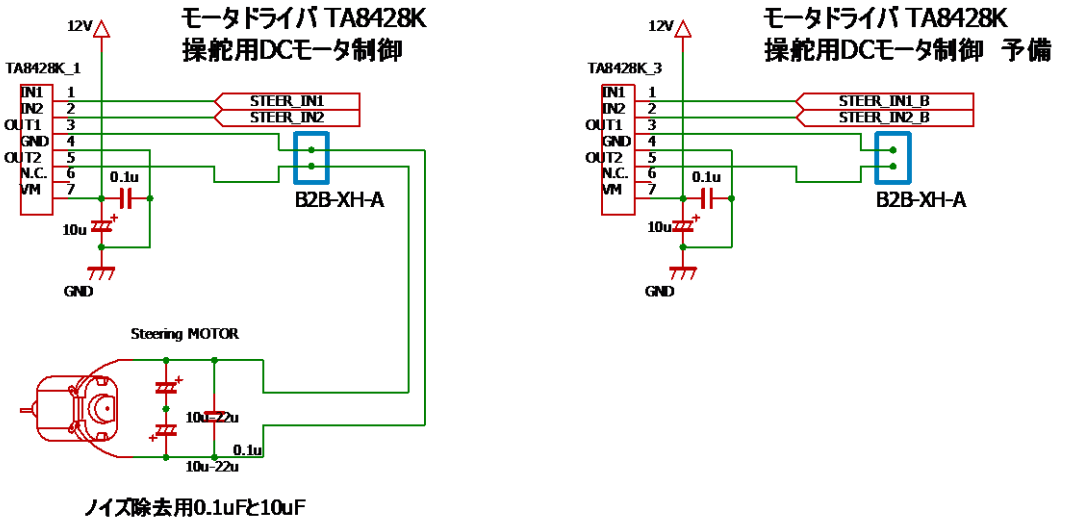


図 B.12 ⑧の部分 回路図

＊ ⑨の部分 ＊

マイコンで扱う信号の入出力用回路である．前輪操舵，後輪駆動，電子コンパスの値取得… といった様々な処理をこのマイコンで行っている．

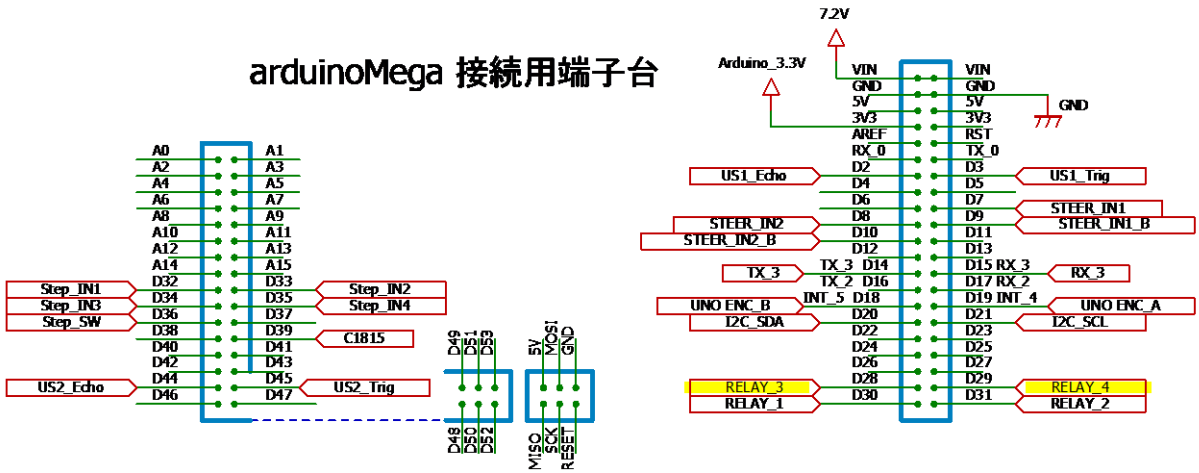


図 B.13 ⑨の部分 回路図

＊ ⑩の部分 ＊

電子コンパスを取り付けるための回路である．キャリブレーションの際には，ステッピングモータによって電子コンパスが回転する．

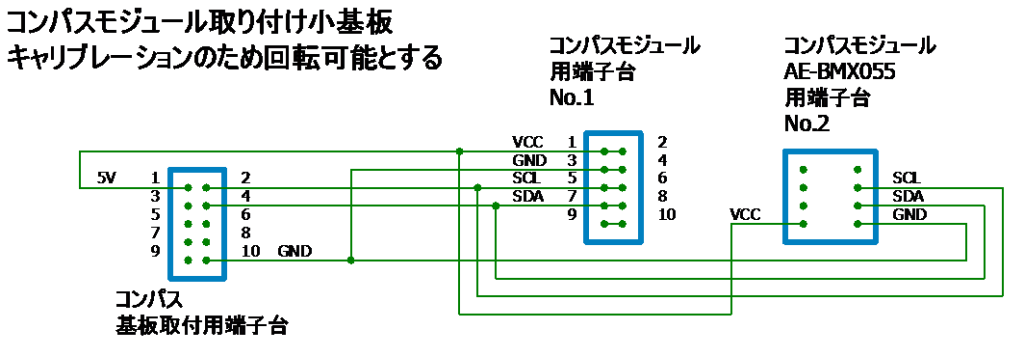
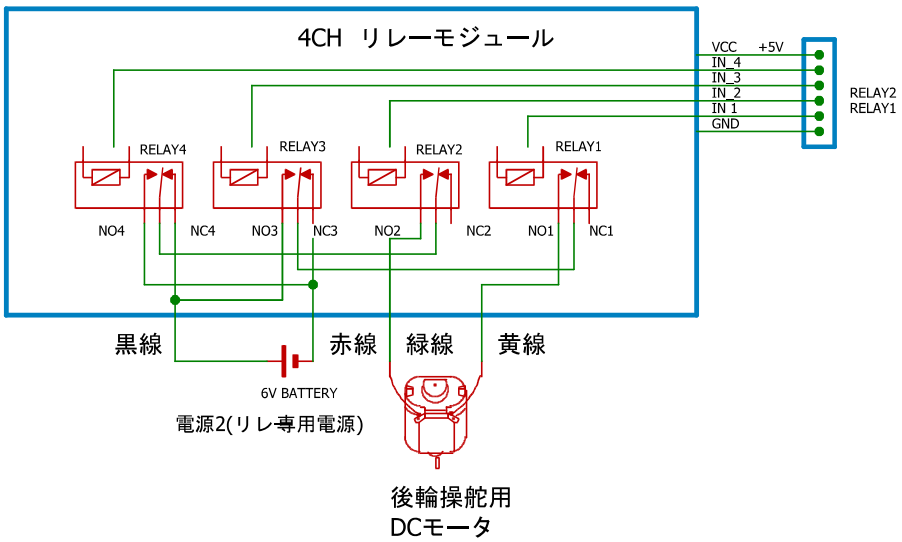


図 B.14 ⑩の部分 回路図

＊ ⑪の部分 ＊

後輪モータの駆動をするための回路である．モータの駆動には、4CH リレーモジュールを使用する． 図 B.2 にある通り，4 つのリレーをそれぞれ ON/OFF することで前進と後退ができる．



RELAY3 OFF 後輪駆動用DCモータ前進　　RELAY1 ON 後輪駆動用DCモータGND側ON
RELAY4 OFF 後輪駆動用DCモータ前進　　RELAY2 ON 後輪駆動用DCモータ6V側ON

RELAY3 ON 後輪駆動用DCモータ後退　　RELAY1 OFF 後輪駆動用DCモータGND側OFF
RELAY4 ON 後輪駆動用DCモータ後退　　RELAY2 OFF 後輪駆動用DCモータ6V側OFF

図 B.15 ⑪の部分 回路図

＊ ⑫の部分 ＊

3.1 節にある超音波センサによる直線走行実験を行う際に用いる**超音波センサ動作用の回路**である．

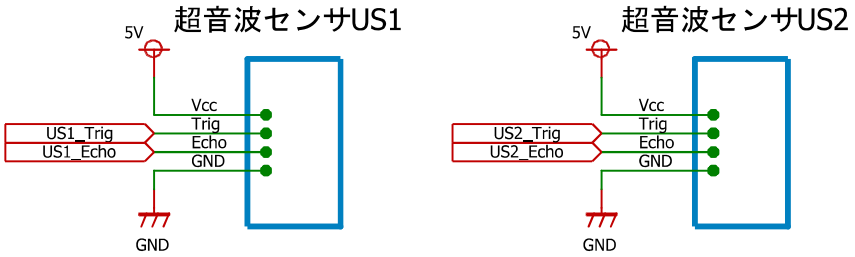


図 B.16 ⑫の部分 回路図

B.2.4 基盤の取り付け位置

次に制作した基板の取り付け位置について示す．図 B.17 を参考にして基板 A～D の取り付け作業を行う．

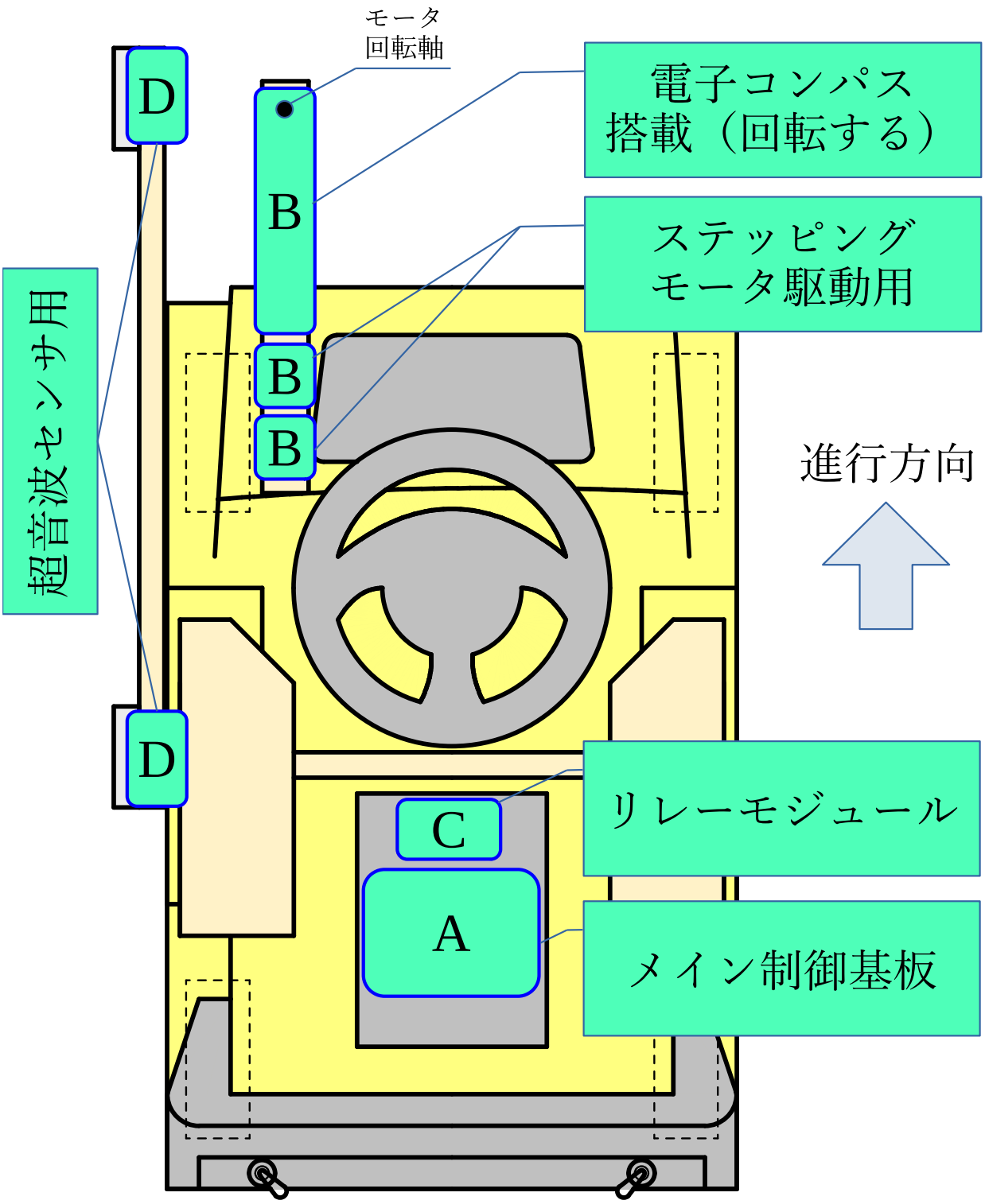


図 B.17　基板レイアウト

B.2.5 コネクタによる各基盤の接続

基板同士を接続するためのコネクタ種類とケーブル長さ，個数は表 B.1 のようになる．

表 B.1　製作するコネクタ付きケーブル

コネクタ種類	ケーブル長さ	個数
① XHP-4（オス）	120 cm	4 セット
② XHP-6（オス）	30 cm	1 セット
③ XHP-8（オス）	100 cm	1 セット
④ JST-NV コネクタ（オス：3 ピン）	80 cm	1 セット
⑤ JST-NV コネクタ（オス：5 ピン）	80 cm	1 セット
⑥ 10 ピンミルコネクタ（オス）	100 cm	1 セット
⑦ 差込形接続端子 FA 形	40 cm	2 セット

完成したコネクタは、図 B.18，図 B.19 を参考に接続する．

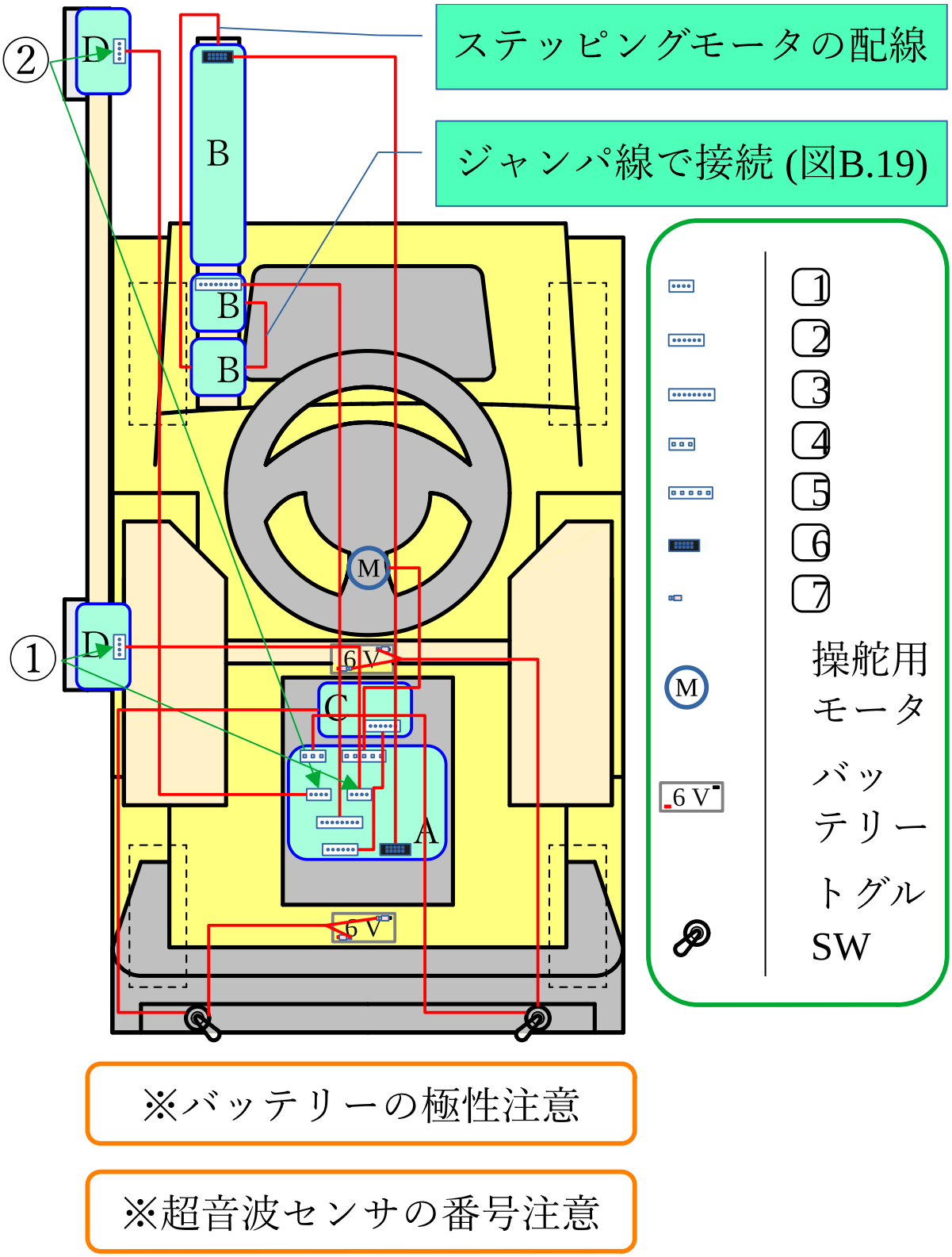


図 B.18　基板同士の接続

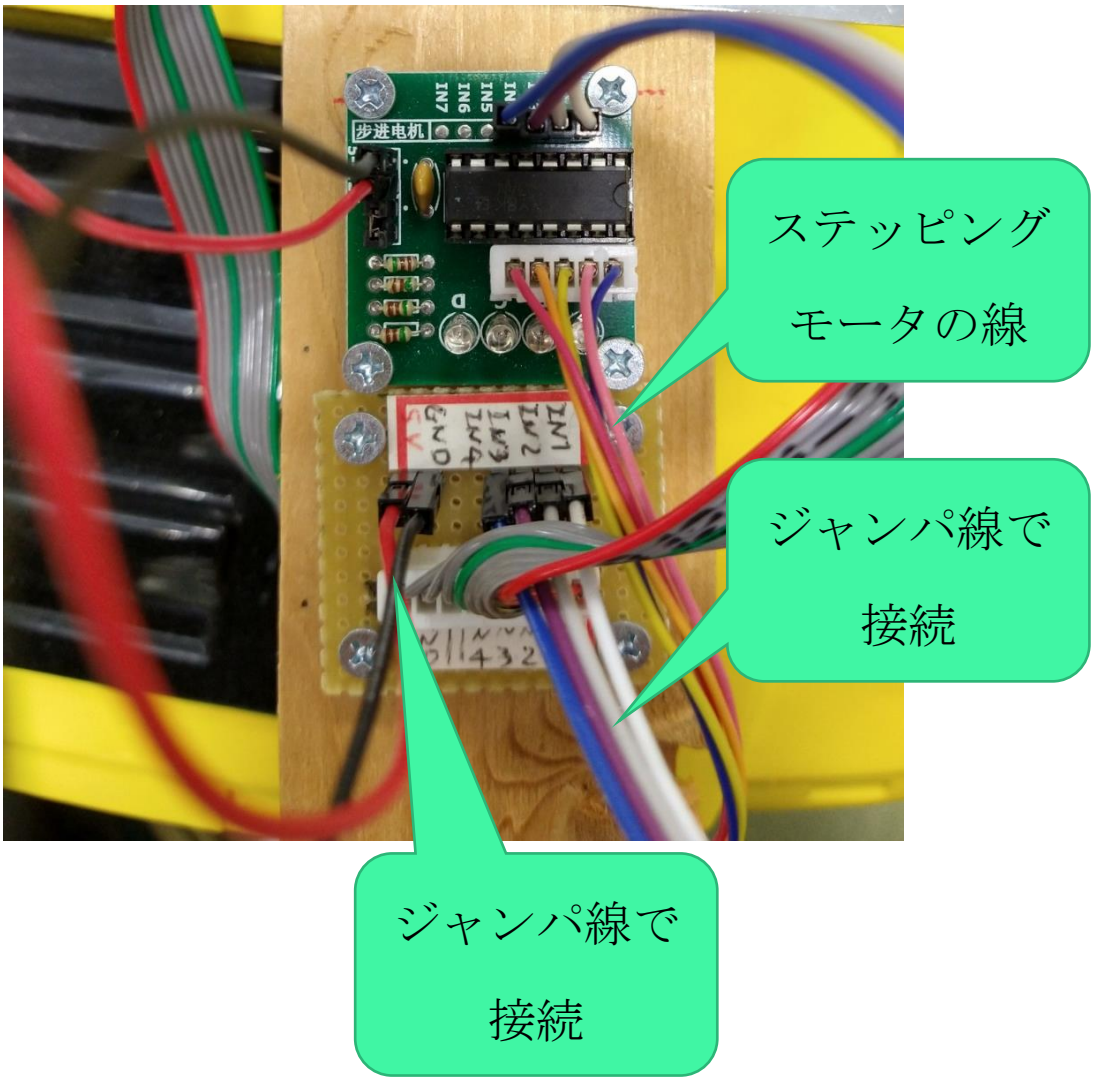


図 B.19　基板 B 同士の接続方法

B.2.6 各機器の接続

基板同士の接続が完了したら，最後に PC やマイコン，RTK 用アンテナなどの接続を行う。 図 B.20 にそれぞれの装置の接続方法を示す.

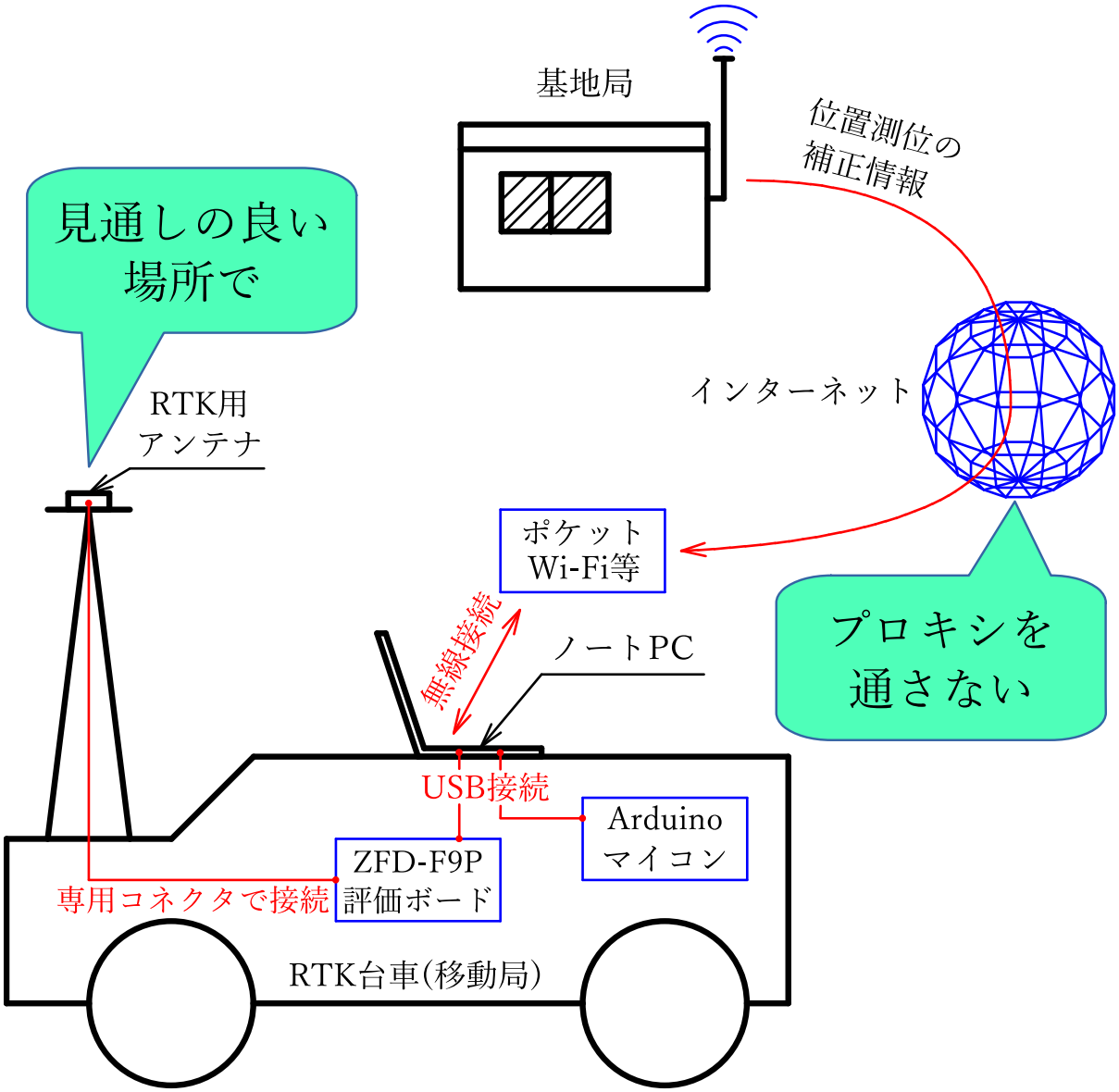


図 B.20　各機器の接続方法