修士論文

ステアバイワイヤにおける追従遅れの許容範囲の評価

–停止状態から発進における感覚評価–

令和２年度

三重大学 工学研究科

前野 友亮

令和 2 年度　修士論文

ステアバイワイヤにおける追従遅れの許容範囲の評価

-停止状態から発進における感覚評価-

所属　　三重大学 工学研究科

研究室　知能ロボティクス研究室

令和 元年度入学　419M145

氏名　　**前野 友亮**

目 次

図 目 次

第1章

目的

本研究の目的は車のハンドルと転舵アクチュエータを機構的に結合しないことで新しい操舵方法を提案することである．ハンドルと転舵アクチュエータ間を機構的に結合をせず電気信号に置き換えることで独立した動きが可能となり，ハンドル自体を操縦桿など他の入力装置に置き換えることも可能となる．しかし，入力装置を置き換えてしまうと操舵感覚が大きく変わり，操舵者に抵抗感を与えると考えたため，本研究では機構的結合がある場合と比較し違和感のない範囲で提案する．

第2章

研究背景

現在，ほとんどの自動車にはパワーステアリングが採用されている．パワーステアリングとはハンドルの操舵に対してアシスト力を加えることで操縦者の操舵力の負担を低減するものである．電動パワーステアリング (EPS) を選定する際，必要アシスト力と搭載スペースについての検討が重要である．しかし，現在の車両電源電圧の 12V では搭載スペースに収まらない場合がある [1]．このことからモーター及び減速機の小型化の必要性があるといえる．

2.1 ステアバイワイヤ

次に車載電装部品の最大消費電力について測定した結果を示す (Fig.2.1,[2])．最大消費電力について，電動パワーステアリングのすえ切り時が最大であり，直進走行時と大きく差が開いている．瞬間当たりの電力消費量が多くなると電源への負担が大きくなるため電源装置が大きくなり，大型車になるにつれて顕著である．そこで瞬間的な電源への負荷を防ぐため，あらかじめ補助電源としてキャパシタに充電をしておく [2] という手法があり，大型車へパワーステアリングを搭載する際の電力不足への対策になっているといえる．しかし，補助電源を搭載する分装置が大きくなるため，より消費電力を抑えつつ小型化を図る必要がある．

そこで航空機技術を参考にしようと考えた．航空機では計算機による制御を前提として性能，運動性および経済性を向上させるアクティブ制御技術 (Active Control Technology) が

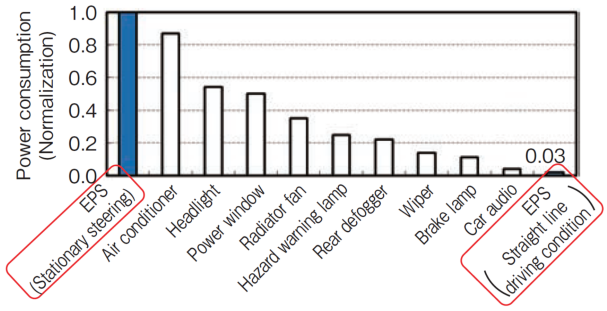

Fig. 2.1　Comparison of maximum electric power consumption

積極的に採用されている [3]．さらに，航空機の操縦桿と各アクチュエータ間の機構的結合をとりはらい電気信号によって制御するフライバイワイヤ技術が登場し，F-16 戦闘機で初めて採用され高い信頼性が実証された [4]．フライバイワイヤにおいて外乱補償や振動抑制などの制御は操縦桿に影響せず，信頼性，ヒューマンインタフェース，コストといった課題を克服したことで実用化されたと考えられる [5]．

フライバイワイヤの考え方を車に適用したものにステアバイワイヤ (Steer-By-Wire) があり，ハンドルと転舵アクチュエータを機構的に結合せず電気信号に置き換えるという技術である [6]．パワーステアリングは機構的に結合されている (Fig.2.2(a)) が，ステアバイワイヤでは機構的結合がない (Fig.2.2(b))．本研究ではハンドルと転舵アクチュエータを機構的に結合しないということによって，パワーステアリングでは不可能であった新しい操舵方法を提案する．しかし，フライバイワイヤの採用理由から，あまりに操舵感覚を変えてしまった場合に操舵者に抵抗感を与えてしまうと考えた．そこでパワーステアリングのようにシャフトで機構的に結合されている場合と比較して違和感がない範囲で検証する．

2.2　従来研究

ステアバイワイヤについての研究では，油圧パワーステアリングのような挙動をモータのみで再現しようとする研究 [6] や電動パワーステアリング用モータをステアバイワイヤに転用し油圧パワーステアリングと比較する研究 [7] がされている．また，近年では小径自転車

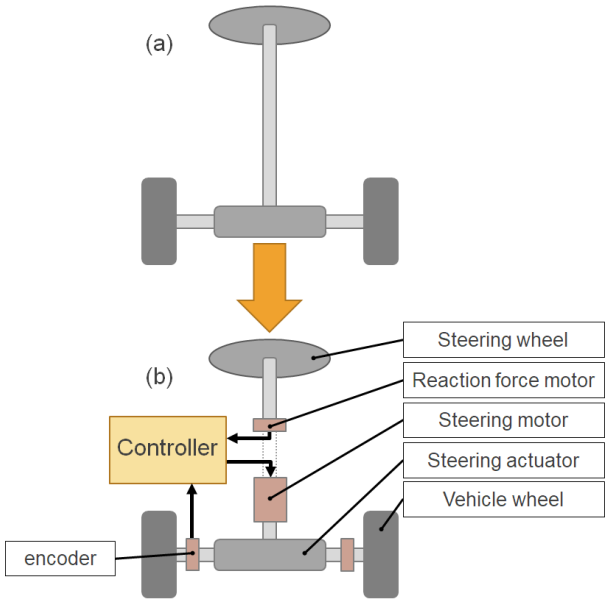

Fig. 2.2　Conversion from conventional steering system to Steer-By-Wire

の操縦性を向上させるためにステアバイワイヤを取り入れる研究 [8] もされている．ハンドルと転舵アクチュエータを機構的に結合したままでは得られず，結合しないことによって可能になる操舵方法について，その中でも操舵感について着目した研究を紹介する．

2.2.1　ハンドルへの反力調節による操舵時の負担低減

超高齢社会に対する輸送機械の要求から超小型モビリティが注目されている．小回りが利く点から大都市の近距離移動，山間部における生活交通の確保として，外出の制約により孤立した高齢者の自立支援に適した車両と考えられている．しかし，小型化のためにパワーステアリングが実装されていない車両がほとんどであり，筋力が低下した高齢者にとっては負担となる．そこで操舵時の腕部の表面筋電位を測定することで操縦者の負担を評価し，ハンドルへの反力の調節 (Fig.2.3) をし腕部への負担を低減するという手法を提案するものである．一定のハンドル反力のもと，各ハンドル角度において肩にかかるモーメントが計測されている [9].

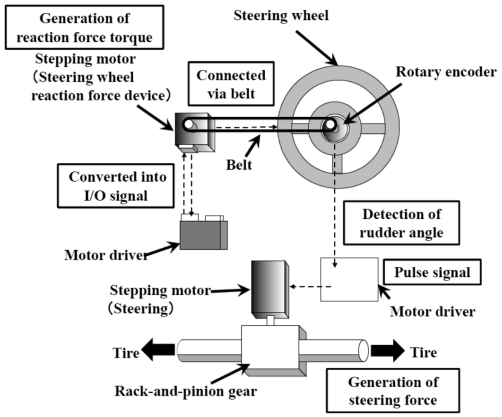

Fig. 2.3 Control System

転舵力と独立してハンドルへの反力を自在に変化できるという，パワーステアリングにはなくステアバイワイヤにはある特性を生かした手法であり，従来と同じくハンドルという入力装置を用いる点からも操縦者への抵抗感は小さいと考えられる．本研究も転舵力とハンドルへの反力が独立しているという点を生かしたものである．従来研究ではハンドルと転舵の位置が一致しているが，本研究では時間によって位置を遅らせるという点が異なる．

2.2.2 手動車いすによる自動車の操縦

車いす利用者の自立や社会参加への意欲の高まりから，車いす利用者のための福祉自動車市場は年々増加傾向にある．高齢者人口の増加に伴い車いすを利用する高齢者も増加すると予想される．そこで，外出しやすくするために車いすのまま自動車に搭乗し，操縦できるものを提案している．車いす固定時に車輪が浮いた状態になり，車輪を回転させることで自動車の操縦が可能 (Fig.2.4) となる [10].

入力装置と転舵アクチュエータの機構的接続が不要であるという，パワーステアリングにはなくステアバイワイヤにはある特性を生かした手法である．車いす利用者にとって慣れている車輪を入力装置として用いる点から，抵抗感の小さい操縦方法であると考えられる．本研究も機構的接続をしないが，従来研究では車いすに慣れている人を対象としているのに対

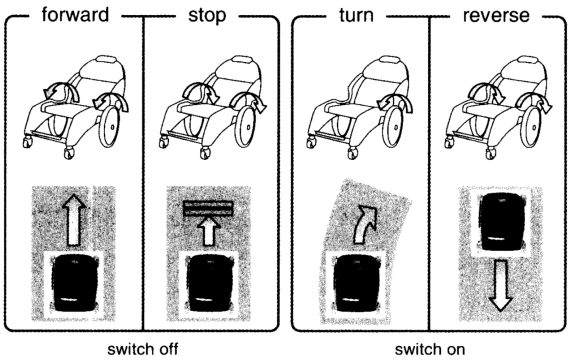

Fig. 2.4 Steering by wheelchair

し，本研究ではハンドル操作に慣れている人を対象としているという点が異なる．

第3章

提案手法

単位移動距離当たりの転舵可能な角度を変えていき，被験者にハンドルからの位置入力に対して遅れを感じたかを評価してもらう．集めたデータから，転舵時の距離遅れ追従を違和感のない範囲で動作するよう提案する．

3.1 設定条件

車両が停止した状態から開始し，ハンドルから切り角を指示する．その後アクセルが踏まれてから転舵アクチュエータが動作しハンドルの位置に追従したのち保持する．

3.2 提案説明

本研究ではハンドル操舵による転舵アクチュエータの動作を遅らせる．これにより，すえ切りが防止され，過度なタイヤのすり減り防止，瞬間的な電源への負荷の低減，モーターの小型化を達成できる．

まず，あらかじめハンドルの操舵から転舵により追いつくまでの基準となる移動距離と角度の関係を決める必要がある．本研究では単位距離当たりに動くことのできる転舵角度を決める．ハンドルの操舵角度と実際の転舵角度を読み取り，指定距離をかけて目標となる転舵

角度になるよう転舵用サーボモータを制御する．例えば，車両が前進せず移動距離が 0 である場合，単位距離当たりの転舵角度やハンドルの操舵角度によらず，転舵角度は 0 となりすえ切りが防止される (Fig.3.1).

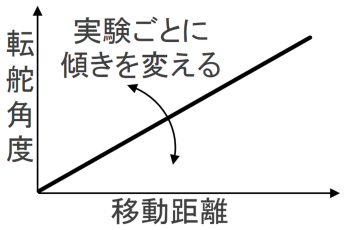

Fig. 3.1　Steerable angle

第4章

実験装置

本研究で使用する車両はトヨタ車体製の超小型 EV のコムス (Fig.4.1,[11]) である．RaspberryPi の上に FPGA を接続した LOGI-PI をメインの制御装置として使用する．搭載されている FPGA は Xilinx 社の Spartan-6 である．また，外部発振器として 50MHz の MEMS 発振器が搭載されている．入出力ポートとして 8 ポートがひとまとめになった PMOD 端子が 4 個用意されている．プログラムの実行にはまず，Xilinx 社のソフトウェア ISE 14.7 を使用しハードウェア記述言語である Verilog で記述しビルド・コンパイルすることで bit ファイルを作成する．次に bit ファイルを RaspberryPi に送り，さらに RaspberryPi から FPGA 側へ書き込むことでプログラムが実行される．さらに安川電機製のサーボパックとデータの授受をし，転舵アクチュエータに接続した同じく安川電機製のサーボモータを駆動する (Fig.4.2)．FPGA への指令はさらに別の RaspberryPi から送られ，上位装置となる RaspberryPi にはステアリングコントローラが接続される．

4.1 サーボドライバの接続

使用するサーボドライバは安川電機製のサーボパック SGDV-2R9EP1A であり，サーボモータは同じく安川電機製の SGMMV-A3E2A21 である．電源にはサーボパック，サーボモータともに DC24V を使用しサーボパック CN3 に接続する (Fig.4.3)．サーボモータは 3 相

Fig. 4.1 COMS

で回転し入力信号はサーボパックの CN4 に接続する．エンコーダの出力信号はサーボモータからサーボパック CN2 に接続する．また，電源切断時にも回転位置を失わないためにエンコーダ信号線にバッテリを追加する．上位装置とサーボパック間の制御入出力信号は CN1 に接続する．CN1 は 26 本の信号線からなるため配線が複雑になる．配線同士の意図しない接触を避けるため市販の端子台に加え別途端子台を作成した (Fig.4.4)．制御方式や信号線の割り当てなど内部パラメータの設定および実行時変数のリアルタイムモニタとして CN7 をパソコンに接続し，安川電機製のソフトウェア SigmaWin+から操作，監視する．

4.2 絶対値データ要求信号

電源切断時に外力によりモーターが回転した場合を考慮し，電源を再投入するごとにある基準位置からの回転数の絶対値データを受け取る必要がある．電源投入時のモータ軸回転数の絶対値データをサーボパックから出力するには，サーボパックへ SEN 信号を送る必要がある．SEN 信号とシーケンス信号用制御電源入力間に 24V をかけることでサーボパック内のフォトカプラがターンオンする．まず，フォトカプラターンオン時の SEN 信号の挙動を設定する必要があるので，SigmaWin+を操作し内部パラメータを特定の信号線のフォトカプラのターンオン時に SEN 信号が ON になるよう設定する．これで SEN 信号線を GND 接

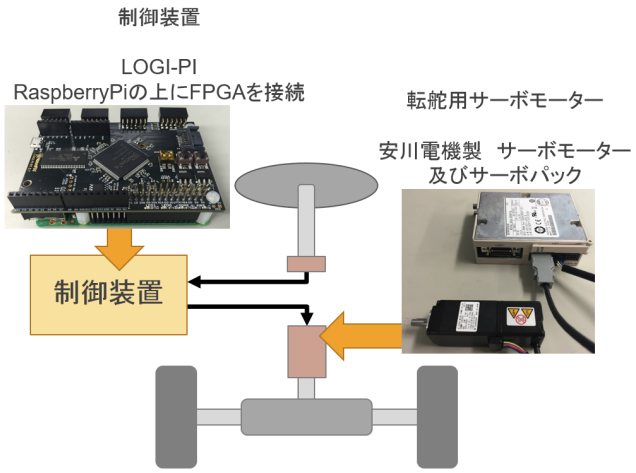

Fig. 4.2　Devices

続することでサーボパックへ SEN 信号が送られるようになる．

上位の制御装置として使用する FPGA は LVTTL で入出力を行うため 3.3V または GND 電圧が出力可能である．SEN 信号線には 24V または GND 電圧をかける必要があるため，FET を経由することで SEN 信号線の電位を操作する (Fig.4.5)．FPGA3.3V 出力時には 1k Ω の抵抗と 10k Ω の抵抗で分圧するため，FET のゲートーソース間には 3V かかる．今，FET に 2SK2961 を使用したときドレイン電流は 2A まで流すことができる．また，ドレインーソース間オン抵抗も低いため SEN 信号線がほぼ GND 電圧になりサーボパック内のフォトカプラはターンオンする．一方，FPGA が GND 電圧出力時には FET のゲートーソース間電圧は 0V となるため，ゲート閾値電圧を下回り，ドレインーソース間は絶縁され，SEN 信号線は 24V となりサーボパック内のフォトカプラはターンオフする．

FPGA からの信号入力回路を実装し (Fig.4.6)FPGA の出力電圧を操作する．FPGA へは回路の書き込みを行うため，書き込み途中であってもすでに書き込み終わった回路は動作を始める．そのため，書き込みが終わる十分な時間が経過したのちに FPGA の SEN 信号ポートの出力電圧の操作をするよう記述する．具体的には，初期値を 0 としておき 50MHz のオシレータで 49,999,999 回カウントアップすることで 1sec をカウントし，出力に 1 を代入する．

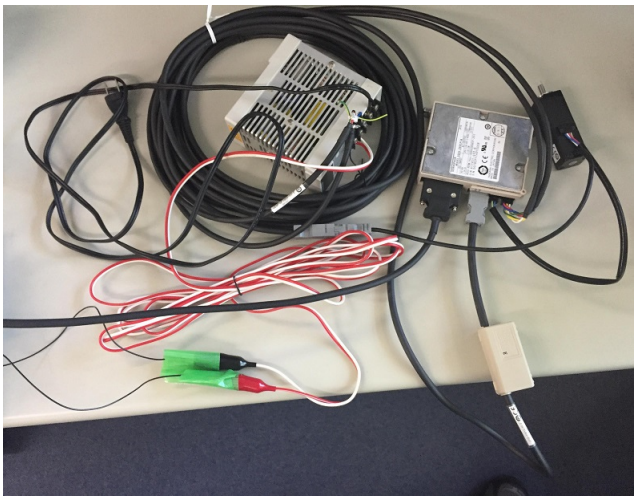

Fig. 4.3　Connection of Servopack

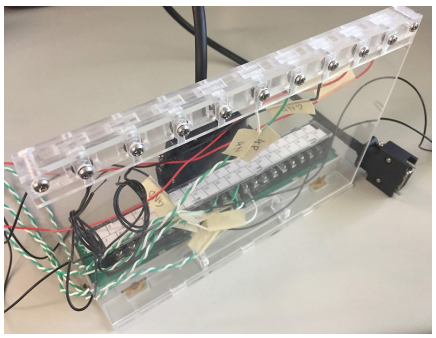

Fig. 4.4　Terminalblock

SigmaWin+から実行時変数を監視した結果，SEN 信号の値の追従が確認できた (Fig.4.7).

4.3　絶対値エンコーダ

絶対値エンコーダからモータ軸回転数の絶対値データを受け取るよう接続する (Fig.4.8). データは PAO 信号線から差動で送られ，オシロスコープで測ったところ 0V と 5V の差動であった．LVTTL の入力は 4.1V が上限であるため，抵抗分圧により 0V と 2.5V にし FPGA へ入力した．また，差動入力において一方がマスタ，もう一方がスレーブであるため，(マス

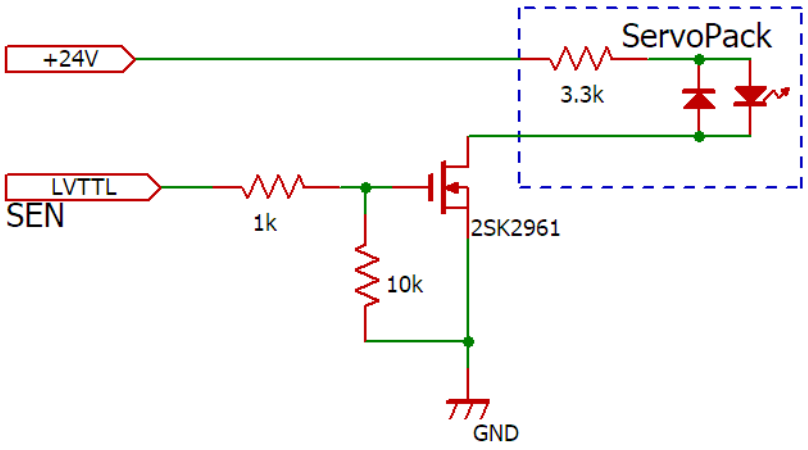

Fig. 4.5　Circuit diagram of SEN signal

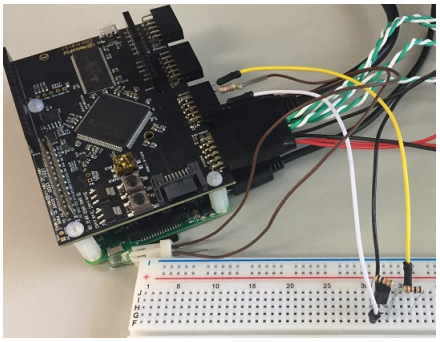

Fig. 4.6　Mounting SEN signal

タ，スレーブ) としたとき，入力 (0,1) で出力 0，入力 (1,0) で出力 1，入力 (0,0) または (1,1) で出力変化なしとなるように差動バッファを通す．これで内部的に一つの受信信号として扱うことができる．絶対値データ要求信号を送ったのち PAO 信号は 50ms の不定領域を経て Hi になり，さらに 60ms〜90ms 後に回転量シリアルデータを送信し始める．受信された内容を確認するために切削機 MDX-20(Fig.4.9) を用いて生基板を削り 8bit を LED で表示する基板を作成した (Fig.4.10)．

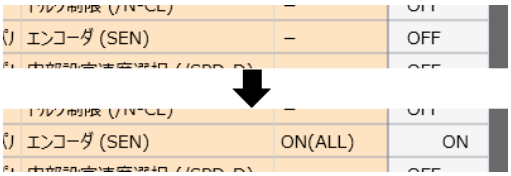

Fig. 4.7　Monitoring SEN signal

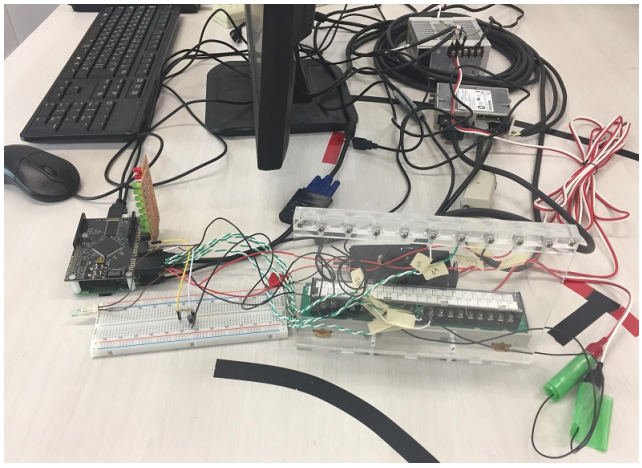

Fig. 4.8　Encode Devices

4.3.1　ボーレイト

回転量シリアルデータはボーレイト 9600bps で伝送される．そのため 1/9600 秒おきに PAO の状態を記録する必要がある．LOGI-PI には 50MHz の MEMS 発振器が搭載されているため，50,000,000/9600=5208.333…より，発振器の信号を 5208 回受けるごとに PAO の状態を記録するよう指示を送るようなクロックを生成する．回転量シリアルデータの最初の立ち下げをトリガにおよそ 9600Hz のクロックを生成するが，サーボパックから伝送される bit 列の切り替えの瞬間に読んでしまうと切替の直前に読んだか直後に読んだかによって信号が曖昧になるため，最初の立ち下げから半周期待機つまり発振器の信号を 5208/2=2604 回受けたのちにクロックの生成を開始する．

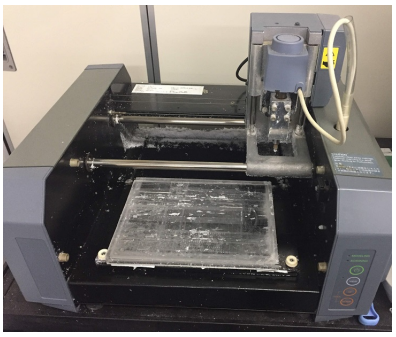

Fig. 4.9　MDX-20

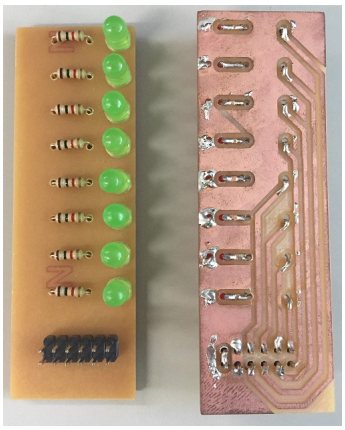

Fig. 4.10　LEDs

4.3.2　調歩同期

回転量シリアルデータは調歩同期で伝送され，同期信号を送るための信号線がない．調歩同期ではあらかじめボーレイトを取り決めているため，スタートビットとストップビットによって送受信のタイミングを合わせる．データフォーマットは，まず Hi であったところにスタートビットで Low が伝送され，次に ASCII 7-bit のキャラクタおよび偶数パリティ，最後にストップビットで Hi が伝送され Hi が続き，次のスタートビットが始まる．このためストップビットを受信したのち次のスタートビットつまり信号の立ち下げに備える必要がある．よって，ボーレイト用のクロック信号の挙動としては PAO 不定領域を無視するため SEN 信

号送信後に 100ms 待機したのち PAO の立ち下げ検出を有効にする．スタートビットによる立ち下げを検出してから半周期待機したのち一定数のクロックを送り，スタートビットを検出するごとに繰り返す.

4.3.3 信号の受け取り

回転量シリアルデータは 8 キャラクタあり，“P”，“+” または “-”，回転量 5 桁 “0”〜“9”，“CR” の順で伝送される．1 キャラクタあたりスタートビット，ASCII 7-bit，パリティ，ストップビットの計 10-bit ある．10-bit を 8 キャラクタ分伝送するため 80 個のボーレイト用クロックが必要となる．また，クロックが立ち上げられるたびに PAO をレジスタに記録するよう記述する.

まず，PAO 不定領域を無視するため SEN 信号送信後に 100ms 待機したのち PAO の立ち下げ検出を有効にする．回転量シリアルデータの最初のスタートビットから連続した 80 個のクロックを生成し，特定回数目のクロック立ち上げ時の PAO からの入力を LED 出力用のレジスタに代入することで LED 表示し，複数回電源を投入することで記録した (Fig.4.11, 右下が 1 クロック目であり左上が 80 クロック目である)．ASCII コードが送信されたのちに Hi が続き再びスタートビットの立ち下げが起きていることが確認された.

さらに調歩同期とするため，スタートビットを含め 10 クロック生成ごとに次のスタートビットまで待機をし，10 個のクロックを 8 回生成し，PAO からの入力を記録した (Fig.4.12, 右下が 1 クロック目であり左上が 80 クロック目である)．データを順番に取り出しコードを翻訳していくと P-03497(CR) となった (Table 4.2)．反対方向に 3487 回転の位置にあるという意味であり，確かに回転量のデータフォーマット通りに読み取れていることがわかる.

4.3.4 ASCII7 ビットの表示

1 度の SEN 信号ですべてのデータを受け取っていることを確認するため，受け取った回転量シリアルデータ 8 キャラクタ分のスタートビット，ASCII 7-bit，パリティ，ストップ

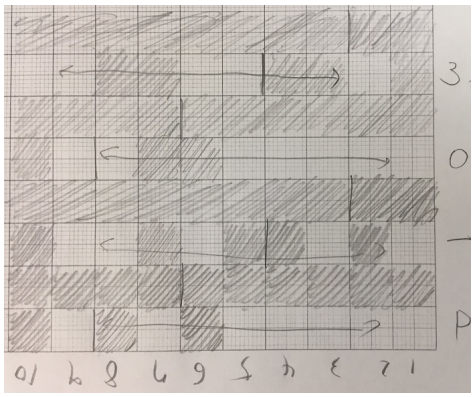

Fig. 4.11 Continuous PAO signal

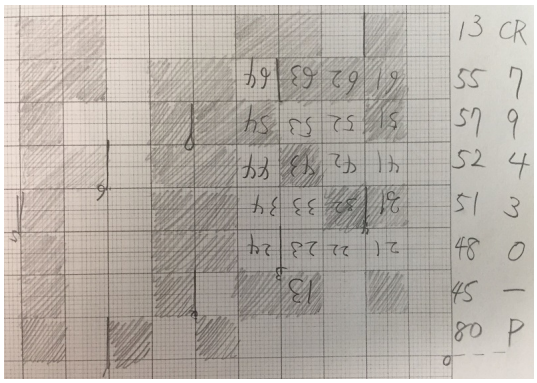

Fig. 4.12 Asynchronous PAO signal

ビットを 10*8 のレジスタに格納し，データ部である ASCII 7-bit とパリティつまり [9:0] のうち [8:1] を LED によって順送り表示した (Fig.4.13).

4.3.5 ASCII7 ビットから 2 の補数表現への変換

FPGA で受け取った ASCII コード形式の回転量を RaspberryPi へ送るとき，RaspberryPi 側で扱いやすい bit 列にして送るのがよいと考える．ここで，回転量シリアルデータについて回転量の範囲は-32768〜+32767 であるため，2 の補数として表現することで 16bit に抑えることができるうえ，ASCII コードに比べ計算する上で扱いやすい．よって FPGA 側で

Table 4.1 Decode of ascii 7-bit

stop	parity	data	start	code
1	1	0001101	0	CR
1	1	0110111	0	7
1	0	0111001	0	9
1	1	0110100	0	4
1	0	0110011	0	3
1	0	0110000	0	0
1	0	0101101	0	-
1	0	1010000	0	P

ASCII コードから符号付 16bit へ変換する．

回転量シリアルデータは 8 キャラクタあり，“P”，“+”または “-”，回転量 5 桁 “0”〜“9”，“CR”の順で伝送され，符号付 16bit へ変換するために必要であるのは符号部と回転量 5 桁である．クロック検出ごとに 0 から 79 までカウントアップしていき特定のカウントでデコードに必要な処理をするように記述する (Table 4.2)．まず，1 キャラクタ目 (0〜9) に “P(1010000)”が伝送されるが特に意味を持たないため無視する．2 キャラクタ目 (10〜19) には “+(0101011)” または “-(0101101)”が伝送され，両者を比較するとデータ部の bit1 および bit2 が 01 または 10 という違いがある．そこで bit2(13 クロック目) を符号記憶用のレジスタに代入し，0 であれば正の数，1 であれば負の数と判断できるようにする．3 キャラクタ目 (20〜29) には十進の 10000 の位の数が伝送され，bit0 から bit3 にかけて 2 進数で表現されている．よって bit0(21 クロック目) には 2^0 を乗じさらに桁数から 10000 を乗じ，16bit 回転量へ加算する．bit1(22 クロック目) には 2^1 を乗じさらに桁数から 10000 を乗じ，16bit 回転量へ加算する．bit2(23 クロック目) には 2^2 を乗じさらに桁数から 10000 を乗じ，16bit 回転量へ加算する．bit3(24 クロック目) には 2^3 を乗じさらに桁数から 10000 を乗じ，16bit 回転量へ加算する．以降 4 キャラクタ目から 7 キャラクタ目も同様に処理し 8 キャラクタ目は復帰コードなので

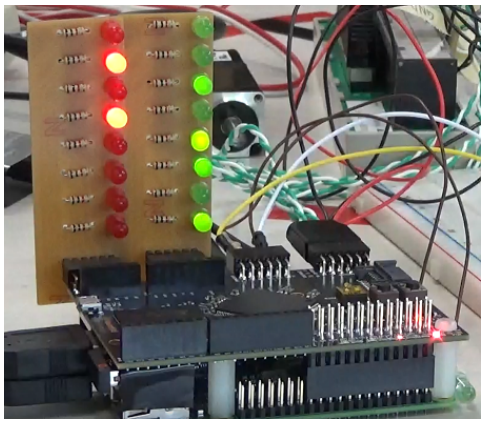

Fig. 4.13　7-bit ASCII and parity bit

無視する．以上から求めた 16bit 回転量に符号をつけるため，符号記憶用のレジスタが 1 であった場合に 16bit 回転量を bit 反転させ 1 を加算する．以上から 2 の補数表現による 16bit 回転量が得られる．LED によって 16bit 回転量を表示することで 2 の補数表現による-03497 が確認された (Fig.4.14).

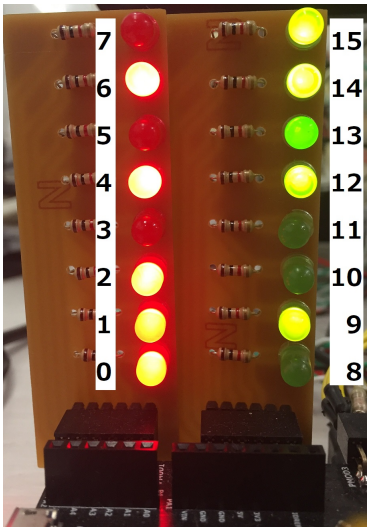

Fig. 4.14　two's complement

Table 4.2 Encode to two's complement

クロック	ビット	処理内容と実際の処理
		符号，回転量レジスタを確保
13	2-bit2	符号を記憶 ($sign$<=pao)
21	3-bit0	$data$<=$data$+$pao * 10000 * 2^0$
22	3-bit1	$data$<=$data$+$pao * 10000 * 2^1$
23	3-bit2	$data$<=$data$+$pao * 10000 * 2^2$
24	3-bit3	$data$<=$data$+$pao * 10000 * 2^3$
31	4-bit0	$data$<=$data$+$pao * 1000 * 2^0$
…	…	…
61	7-bit3	$data$<=$data$+$pao * 1 * 2^3$
		$sign$==1 の時 $data$<=~$data$+1

4.4 SPI 通信

RaspberryPi と FPGA 間のデータ授受は，同期式シリアル通信として一般的に用いられる SPI 通信で行い，RaspberryPi をマスタ，FPGA をスレーブとする．RaspberryPi 側のドライバには spi-bcm2835 及び spidev を用い，一度の通信で 8bit のデータを授受する．FPGA 側は，基板に搭載されている 50MHz のクロックと RaspberryPi から入力される同期用のクロックからビットシフト用のクロックを生成し 8bit のデータを順番に送る．

RaspberryPi から FPGA に 2 の補数表現に変換した 16bit の回転量を要求する場合，RaspberryPi から回転量の上位 8bit を要求する 8bit 信号，下位 8bit を要求する 8bit 信号，受信のためのダミーの 8bit 信号を順番に送る．このとき RaspberryPi へはダミーデータ 8bit，回転量上位 8bit，下位 8bit の順番で伝送される．

例えば RaspberryPi から “00000001”，“00000010”，“00000000” の順に FPGA に送る場合，RaspberryPi は “00000000”，“11110010”，“01010111” の順に受け取る．さらに上位 8bit

と下位 8bit を結合し 16bit にしたのち 2 の補数から 10 進数へ変換することで-3497 となり負方向の回転量が得られる (Fig.4.15).

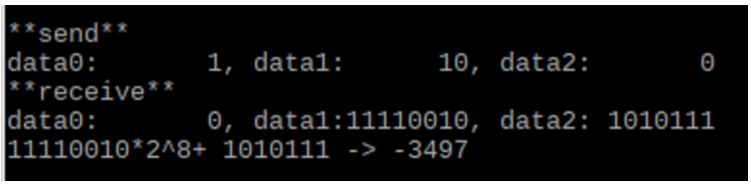

Fig. 4.15 receive 16-bit amount of rotattion

4.5 サーボモータのエンコーダ値

サーボパックは絶対値データを送ったのち，インクリメンタルパルスの伝送を始める．絶対値データが電源投入前の回転量であるのに対し，インクリメンタルパルスは 1 回転をさらに分割した細かい回転量を表現するパルス列であり，モータの動きに対応して 4 逓倍されたインクリメンタルパルスがサーボパックから出力される．

絶対値データを FPGA で受信したのち，インクリメンタルパルスとして PAO の差動および PBO の差動を監視することでモータの位置を読み取る．まず，パルス数をインクリメンタルパルス用のレジスタに記録していく．同時にあらかじめ受信した回転量の絶対値データをパルス数に変換し，インクリメンタルパルス用のレジスタと加算し，絶対値パルスのレジスタに格納する．以上から，現在のモータの絶対位置がパルス数によって表現され，RaspberryPi から FPGA へ SPI 通信にて絶対位置のパルス数が要求されるごとに授受がなされる．

4.6 ハンドル

本研究に使用するハンドルは指定した反力を出力し，位置を返す機能が必要である．そこでステアリングコントローラの G29 Racing Wheel(Fig.4.16) を使い，RaspberryPi に USB

接続する．

Fig. 4.16 G29 Racing Wheel

4.6.1 デバイスファイル名

RaspberryPi に G29 の USB を接続すると自動で番号が割り当てられる．どの番号が割り当てられたかを調べ，その番号に対応するファイルとデータを授受することで G29 と通信する．/proc/bus/input/devices に複数のデバイス情報が記述されている．RaspberryPi に USB コネクタを接続することで G29 Racing Wheel という項目が追加されるので，文字列を検索し接続した G29 Racing Wheel の Handlers の項目からデバイスの認識番号を抜き出すことができる．デバイスファイル名が/dev/input/event*(番号) と分かる (Fig.4.17) のでファイルに書き込むことで反力の出力と，読み込むことで位置の取得が可能となる．

4.6.2 G29 への書き込みと読み込み

G29 はキーボードと同じくイベントデバイスなので，入力がされるごとに出力がされる．例えばボタンを押した場合は，ボタンの変数の値が 1 と出力され，しばらくしてからボタ

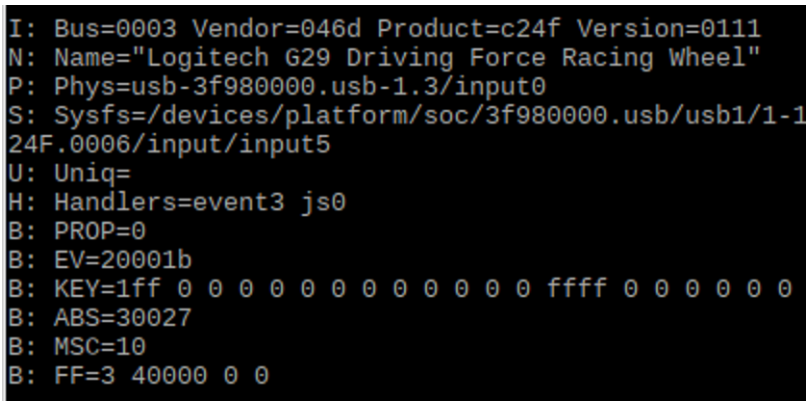

Fig. 4.17 detect device file

ンの変数の値が 0 と出力される．ハンドルの位置の場合は USB 接続時に位置の変数の上限 (65535) と下限 (0)(Fig.4.18(a)) が得られ，ハンドルが動くごとに位置が得られる (Fig.4.19)．ハンドルの右端つまり位置の上限値 65535 を実数型の 1.0，ハンドルの左端つまり位置の下限値 0 を実数型の-1.0 に変換する．また，反力は-1.0〜1.0 の値を書き込むことで設定できる．

次に，反力の指令値を格納する force という名のファイルと位置の現在値を格納する position という名のファイルを用意する．ただひたすら反力の指令値を force ファイルから読み込み G29 へ反力を設定し，G29 から位置を受け取り位置の現在値を position ファイルに出力する．以上の内容のプログラムを C 言語で記述する．

4.6.3 G29 通信用の子プロセス呼び出し

メインとなる python のプログラムから，G29 への書き込みと読み込みを行う C 言語のプログラムを子プロセスとして呼び出す (Fig.4.18(b))．メインの python のプログラムから force ファイルに反力を書き込むと C 言語で記述された子プロセスから G29 へ指令が送られハンドルの反力は更新される．また，ハンドルを動かすと G29 から子プロセスへハンドルの位置が送られ position ファイルには位置が書き込まれ保持されるので，メインのプログラムから position ファイルを読みむことでハンドルの位置が得られる．

4.6.4 ハンドルの初期位置の同期

電源投入後にハンドルの初期位置をタイヤの方向に合わせる必要がある．具体的には転舵用サーボモータの初期位置がタイヤが正面を向く位置にある場合，ハンドルも正面を向く位置に合わせる．電源が投入され RaspberryPi がハンドルを認識した後でハンドル反力のフィードバックによって目標位置へハンドルを移動させる．

メインの python プログラムが初期位置を FPGA から受け取る必要があるので，FPGA が RaspberryPi との通信準備を終えているかを確認する (Fig.4.18(c))．FPGA 側が準備を終えていれば C 言語の子プロセスを呼び出し (Fig.4.18(b))，さらに FPGA へサーボモータのエンコーダの値を SPI 通信にて要求する．8bit3 回分のデータが送られてくるので (Fig.4.18(d))，結合し符号付 24bit の回転パルス量へ変換し (Fig.4.18(e)) さらに-1.0〜1.0 に変換し (Fig.4.18(f)) ハンドルの初期位置の目標値となる．次に，現在のハンドルの位置を position ファイルから読み取る．ハンドルの位置の現在値と目標値から force ファイルに-1.0〜1.0 の値を書き込みフィードバックループによって現在値を目標値へと収束させる．以上から電源投入後のハンドルの初期位置とサーボモータの同期がなされる．

4.7 サーボモータの駆動

サーボモータは 2 相からなる位相差 90 °の 4 逓倍パルスをサーボパックへ入力することで駆動する．サーボパックへのパルス列は FPGA から入力し，サーボモータの絶対値パルス数の現在値と目標値からフィードバックループによってパルスを生成する．現在値はエンコーダから取得され，目標値は上位装置である RaspberryPi から FPGA へ SPI 通信にて入力される．

ハンドル位置をパルス数に変換し FPGA へ伝送し FPGA からフィードバックループによってサーボモータを駆動した．ハンドル位置およびサーボモータの回転位置をパルス数と回転数の 2 表現で示す (Fig.4.18(g))．

4.8 各装置の電源投入

FPGA に bit ファイルを書き込む RaspberryPi，サーボモータおよびサーボパック，上位装置となる RaspberryPi に電源投入する必要があるため，制御基板を作製した (Fig.4.20).

4.8.1 OS の起動と終了

RaspberryPi は USB+5V を加えるとスイッチを介さず電源が入るため，FET 経由で USB+5V を操作する (Fig.4.21)．シャットダウンをせずに USB+5V を切断すると RaspberryPi が破損するため，スイッチが OFF の時は切断ではなくシャットダウン信号にする必要がある．スイッチが ON の時には起動信号を出力するが，初回起動時に限り USB+5V を加え以降保持する必要があるため，自己保持回路を追加する (Fig.4.22)．また，誤操作を防ぐためトグルスイッチに加えプッシュスイッチも押されなければ起動しないようにする．また，OS の起動時にメインの python プログラムも起動するよう boot を編集する．

4.8.2 サーボ系の電源投入

サーボ系にはサーボモータ，サーボパックともに+24V を加える必要がある．COMS のバッテリーから DC-DC コンバータを介して+24V を取り出すが，コンバータはリモートコントロール-入力側 GND 端子間をショートさせることで動作する．トグルスイッチを切り替えることでリレーを駆動しコンバータの端子間をショートさせサーボ系の電源を操作する (Fig.4.21).

4.8.3 FPGA への書き込みと消去

起動時は RaspberryPi から FPGA に bit ファイルを書き込み，終了時は空の bit ファイルを書き込むことで消去する．FPGA へ書き込む RaspberryPi はメインの python プログラムを動かす RaspberryPi の自己保持回路と同等のものにプッシュスイッチを常時押し下げにし

ておく．FPGA に書き込み時は I2C を使用するが，RaspberryPi の起動信号と共通のピンと使うため信号が衝突する．起動後は起動信号を無視し，I2C の機能を有効にするため，外部に回路を追加する (Fig.4.23).

4.9 車体への取り付け

COMS のステアリングシャフトを切断しサーボモータとギアボックスに置き換える (Fig.4.24). さらに運転席にステアリングコントローラの G29 Racing Wheel および電源スイッチを配置する (Fig.5.2)．トランクにはサーボパック，FPGA，FPGA 書き込み用の RaspberryPi，上位装置となる RaspberryPi，モニタ，プログラム送受信用の無線 LAN および車載バッテリーから電源を取るためのコンバータを取り付ける (Fig.5.5)．また，後輪のドライブシャフトに，走行距離測定用のエンコーダとして OMRON のインクリメンタル形ロータリーエンコーダを左右に取り付け FPGA に接続する (Fig.5.4).

4.10 タイヤの回転数

エンコーダ値とタイヤの回転数が対応しているかを確認するため，タイヤを回転させその時のエンコーダ値が理論値とどの程度ずれているかを調べる．開始位置と終了位置のずれによる測定誤差を小さくするため 5 回転させる．車体とタイヤにシールを貼っておき 5 回転させる．開始位置においてエンコーダ値は 1 であり，終了位置において-79989 であった．つまり変動量は 79990 パルスである．今回使用したエンコーダは 1 回転当たり 2000 パルス出力する．出力は 4 逓倍なのでエンコーダの 1 パルスは 4 倍の 4 パルスとして出力される．さらにエンコーダ及びドライブシャフト間に 2 倍のギアを入れ接続したのでパルスは 2 倍になる．以上を踏まえ，2000(1 回転分) × 4(4 逓倍) × 2(ギア) × 5(回転数) より，80000 パルス出力されるはずである．さらに，さしがねによる計測点における回転周長はおよそ 690cm であり，さしがねが 1mm 幅の目盛りであることを考慮すると，80000 パルス出力されると

ころを 79990 パルス出力されたのでタイヤの回転数に対応するエンコーダ値はおよそ正しいと言える．

4.11 サーボモータによる転舵

あらかじめタイヤ 1 周あたりにサーボモータが回転することのできる角度を設定する．次に，ドライブシャフトに取り付けたエンコーダから走行距離を定期的に取り続け，前回取得した走行距離と今回取得した走行距離の差分からサーボモータが回転することのできる角度を計算する．現在のサーボモータの回転位置とハンドルの位置と比較し，FPGA へサーボモータの回転位置指令を送る．以上を高速で繰り返すことでサーボモータによる転舵がなされる．

4.12 各変数の記録

OS 起動時からコマンド実行時に至るまでの時間，ハンドルの角度，サーボモータが回転可能な角度，サーボモータへの指令値，サーボモータの現在の回転位置，ドライブシャフト左右それぞれのエンコーダ値および前回のループとの差分を csv ファイルに保存することで記録を取っていく (Fig.4.28)．結果，1 秒間におよそ 575 回ループされていた．約 20 秒間 11737 ループを計測した結果，ループ毎の時間差は 0.083931198,0.059780052,0.043532239,0.019519219,0.014791354 の順に大きく，逐次メモリを確保する際に瞬間的な負荷がかかるためと考えられ，あらかじめメモリを確保することで解決すると考えられる．

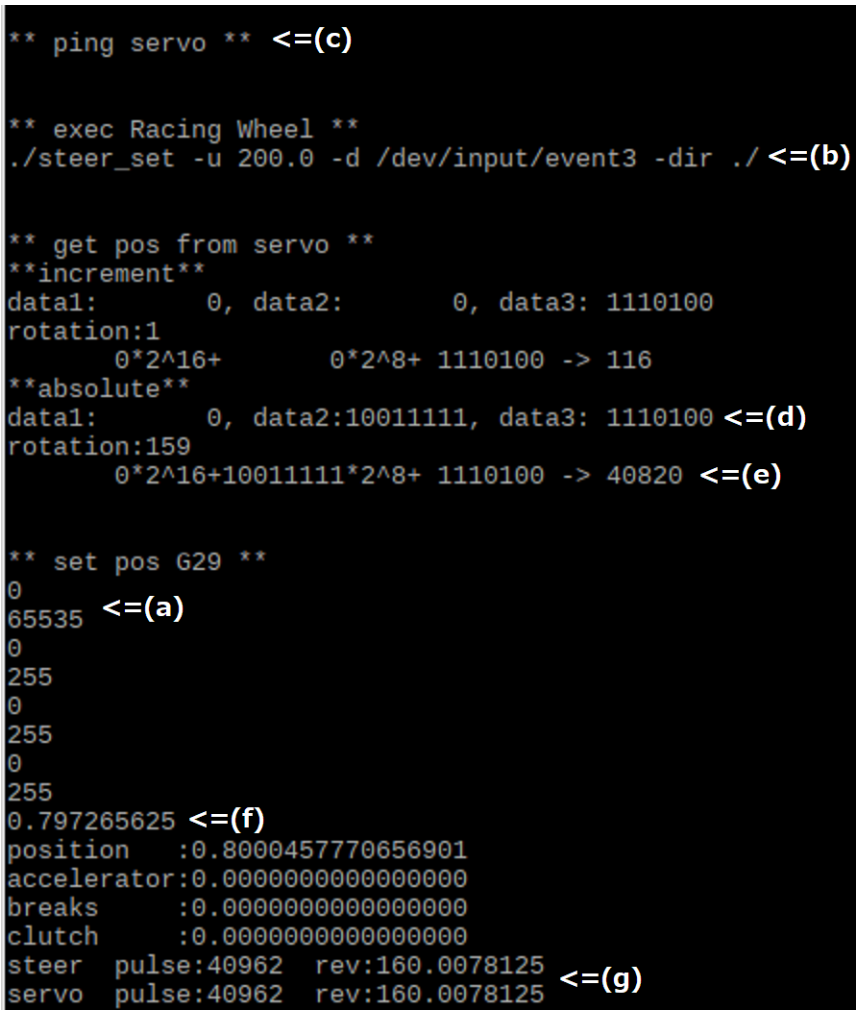

Fig. 4.18 event log

```
type 4 (EV_MSC), code 4 (MSC_SCAN), value 90005
type 1 (EV_KEY), code 292 (BTN_TOP2), value 1
------------- SYN_REPORT ------------
type 4 (EV_MSC), code 4 (MSC_SCAN), value 90005
type 1 (EV_KEY), code 292 (BTN_TOP2), value 0
------------- SYN_REPORT ------------
type 3 (EV_ABS), code 0 (ABS_X), value 33156
------------- SYN_REPORT ------------
type 3 (EV_ABS), code 0 (ABS_X), value 33157
------------- SYN_REPORT ------------
type 3 (EV_ABS), code 0 (ABS_X), value 33158
------------- SYN_REPORT ------------
```

Fig. 4.19 input from event device

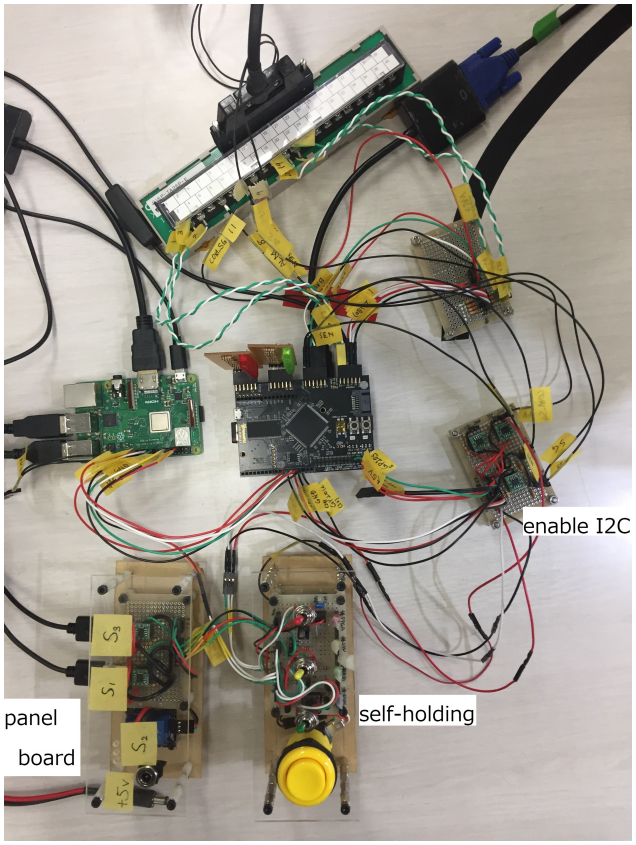

Fig. 4.20 control board

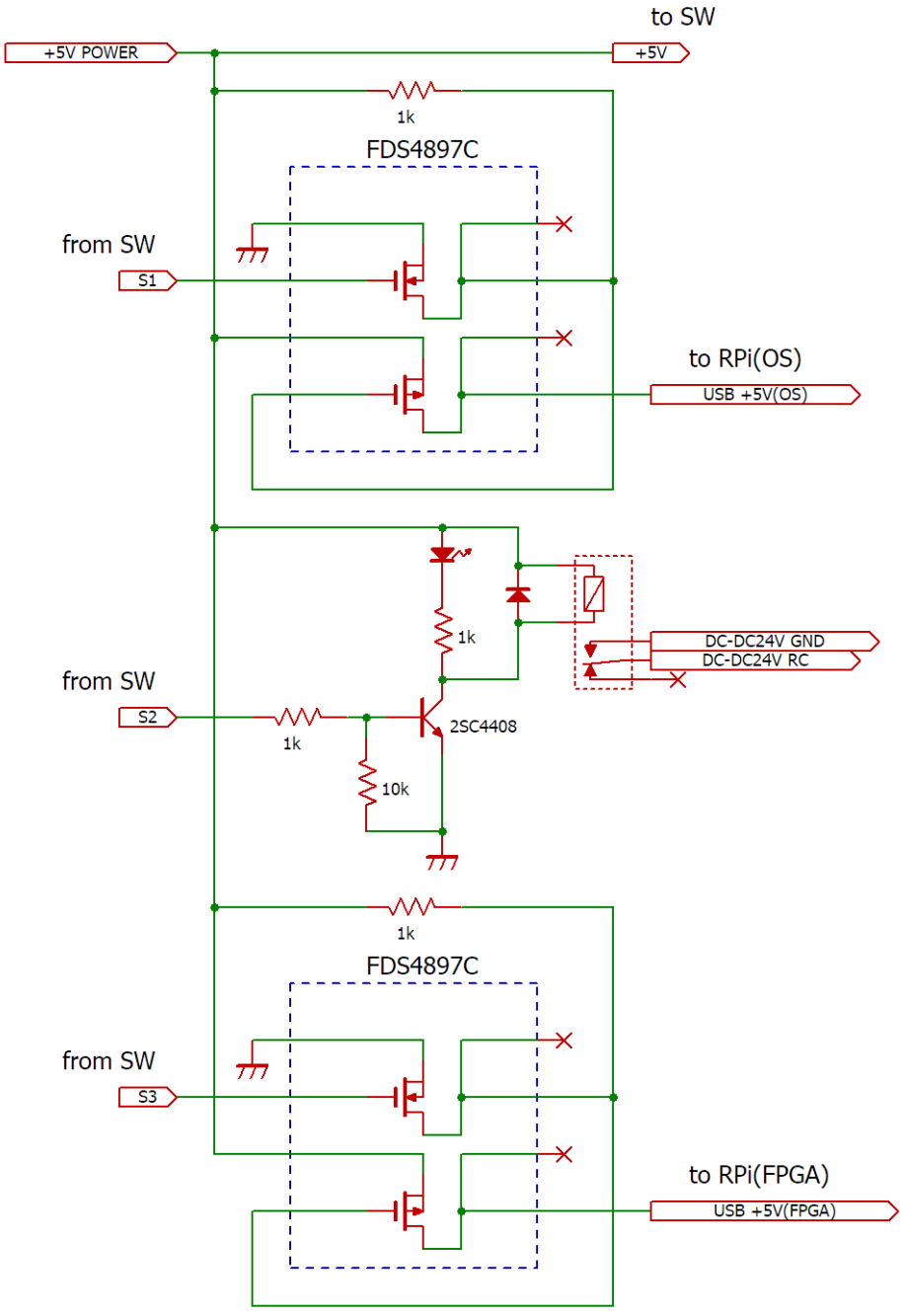

Fig. 4.21 panel board

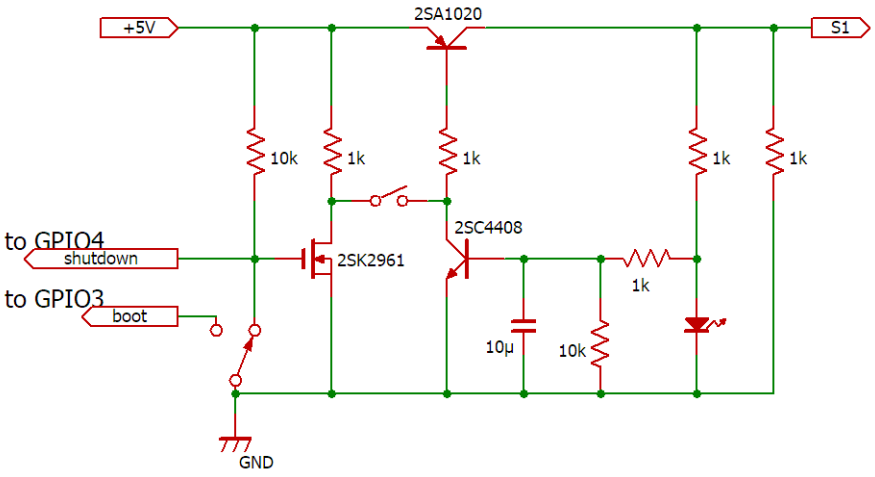

Fig. 4.22　self-holding circuit

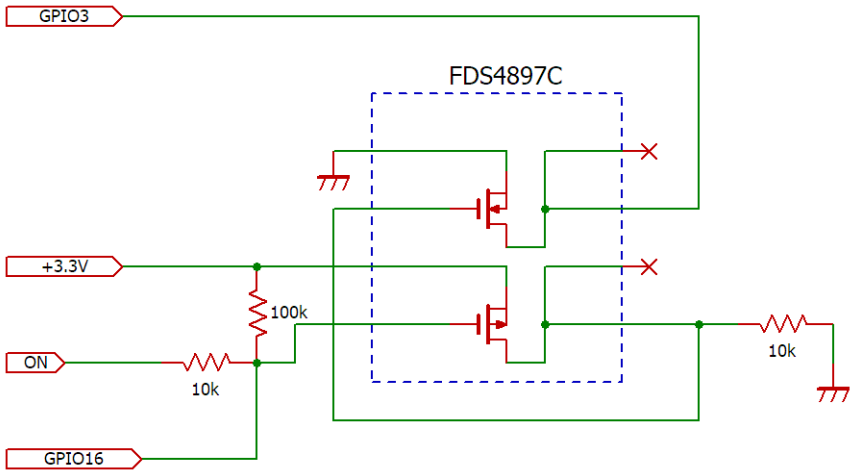

Fig. 4.23　circuit to enable I2C

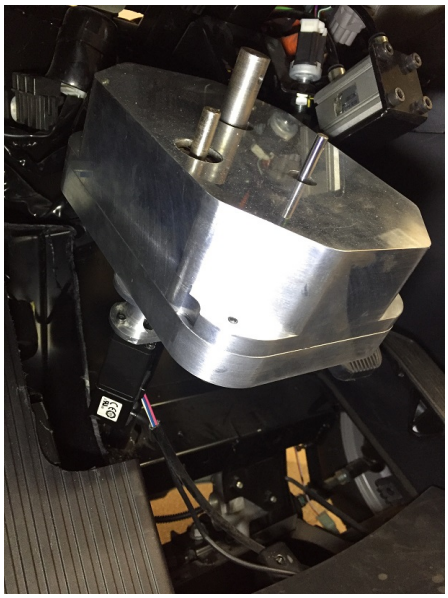

Fig. 4.24　Steering Gearbox

Fig. 4.25　Driver's seat

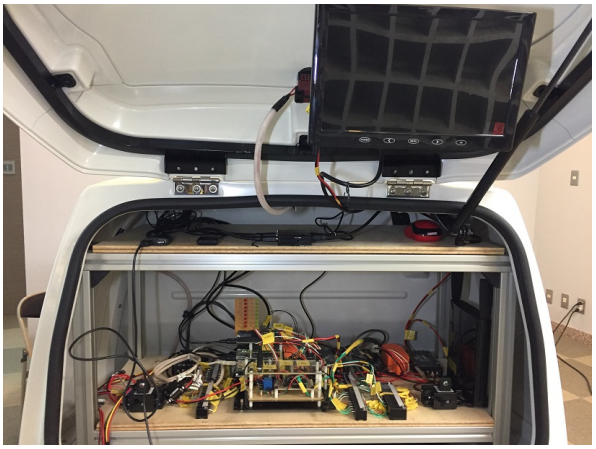

Fig. 4.26 trunk

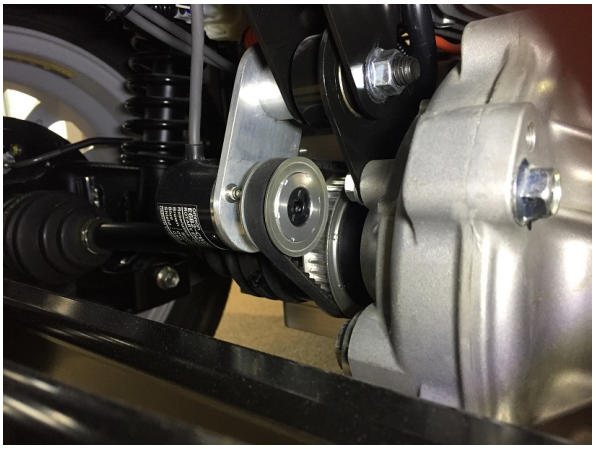

Fig. 4.27 encoder on Driveshaft

2019_01_30_16h07m58s								
absolute_max	51200	(rev of servo)*(2**8) if rev=1: -1~+1						
permitted	512000	(pulse of servo)/(rev of vehicle wheel)						
time	sw	permitted	command	servo	encoder_l	encoder_l_diff	encoder_r	encoder_r_diff
1030.976614	-69	48	-48	0	1064	-1	-1142	2
1030.978673	-69	112	-69	0	1060	-4	-1139	3
1030.980873	-69	96	-69	0	1057	-3	-1136	3
1030.98228	-69	64	-64	0	1055	-2	-1134	2
1030.983529	-69	64	-64	0	1053	-2	-1132	2
1030.985108	-69	96	-69	0	1050	-3	-1129	3
1030.987266	-69	96	-69	0	1047	-3	-1126	3

Fig. 4.28 csv file

第5章

実験装置のさらなる改良

これまでの改造によりとりあえず車両を動かすことはできたが，違和感無く動かすには処理速度も故障も多いので，さらなる改良を進めていく．被験者が一定の経路で小型電気自動車を運転する実験をする. 実験用電気自動車として，カスタマイズされたトヨタの COMS を使用している (Fig.5.1)．ステアリングハンドルをゲーム用のステアリングハンドルである Logitech の G29 レーシングハンドルに変更する (Fig.5.2)．Logi G29 には，エンコーダとフォースフィードバックシステムがある．ステアリングシャフトを切断し，サーボモータに交換する (Fig.5.3)．サーボモータの信号は FPGA に送信される．左右両方の後輪にエンコーダを取り付け，車両の軌跡を測定する (Fig.5.4)．後輪エンコーダパルスは FPGA で受信される．ほとんどの回路とコンピュータはトランク内にある (Fig.5.5)．bit ファイルの書き込み専用のカードサイズコンピュータ RaspberryPi を用意している．RaspberryPi はコンパイルされた Verilog ファイルを FPGA に書き込む．ハンドルの角度，車輪のエンコーダ値，サーボモータによる舵角はマスターとなる RaspberryPi に送信され，舵角の目標値は FPGA を介して操舵部のサーボモータに送信される (Fig.5.6).

Fig. 5.1　Experimental electric vehicle

5.1　サーボドライバの故障

車体取り付け後にサーボモータを駆動したところ，サーボドライバが故障した．車体取り付け後に故障したことから，取り付け時の誤配線による故障と過負荷による故障が考えられるが，配線の誤りがないことを確認したため過負荷による故障と考える．ドライバ内部にはモータドライバ基板と制御基板があり，今回は制御基板が故障した．両基板は共通の+24V電源に接続されていた．コンデンサの容量不足により過負荷と判断され，さらにモータ切断時のサージにより制御基板が破壊されたと考えられるため，モータドライバ基板及び制御基板それぞれにコンバータを接続しモータドライバ基板側の電源にはコンデンサを追加した．しかし，コンバータを別々にしコンデンサを取り付けてからも，故障はしないものの過負荷あるいは振動によるエラーが発生した．FPGA に書き込む Verilog による指令制御を書きなおすも解決しないため，車体に取り付けたギアボックスを確かめたところモータとピニオンギアの接続部のゆるみが発覚した．接続部品を取り換えたのち各種ゲインパラメータを調整することでエラーが解決された．

Fig. 5.2　Steering wheel G29 on the vehicle

5.2　コンバータ入力部

車載の走行用の 72V バッテリーからコンバータに接続し各装置の電源を取るが，安全のため，コンバータの入力側にリレーを追加し，電源 OFF 時に 72V が流れないようにする．3D プリンタとソレノイドで作られたリレーを使用していたが夏場は動作不良を起こすため，車載用のリレーを使用する (Fig.5.7).

サーボモータの電源を入れると RaspberryPi の電源が落ちるという問題が発生した．DCDC コンバータの+5V 出力にコンデンサを追加することで多少は改善されたがやはり電源が落ちる場合がある．RaspberryPi 用に無停電装置を用意しようと考えたが根本的な解決にはならないためオシロスコープで電源周りを見ていく．コンバータの+5V 出力にコンデンサがある場合とない場合とで RaspberryPi の電源電圧に確かに変化が見られたが，同様のコンデンサを十数個を並列につなぐ必要があるほどの電圧降下を起こしていた．RaspberryPi の電源投入用の FET や DC ジャックでの電圧降下が原因とも考えられるためコンバータの+5V 出力を見たところ，RapberryPi までにわずかに電圧降下はしているがそれ以上に V の字を描くように 2.4V まで落ちている．以上よりコンバータを含めたバッテリー側に問題があると考えらる．取り付けたリレーが原因の可能性もあるので導線に置き換えたが解決しないので，サーボモータのコンバータに電荷が吸われ電圧降下を起こし，RaspberryPi 側のコンバータ

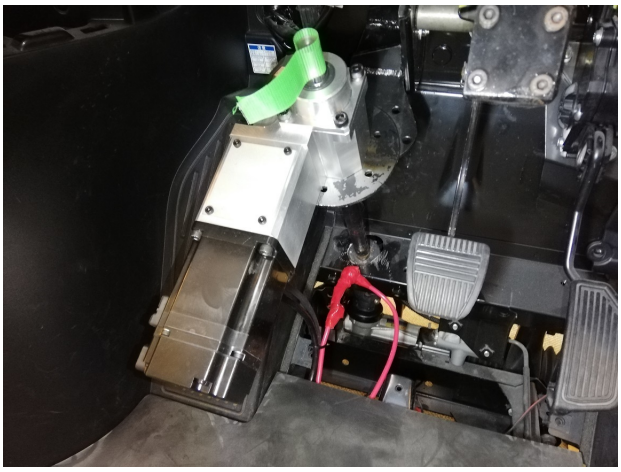

Fig. 5.3　replace to the servomotor for Steering

が動かなくなっていると考える．コンバータの入力をオシロスコープで見たところ，52V まで低下しその後発振していた．今回使用したコンバータは 64V〜144V 入力であり，異常を検知したために停止しその後コンバータ内部にフィルターとして入れられているチョークコイルにより発振したと考えられる．64V を割らせなければいいため，サーボモータ側の電荷の引き抜きに耐えられるよう，RaspberryPi 側のコンバータの入力にコンデンサを追加した．以上より，RaspberryPi 起動中にサーボモータの電源を投入しても RaspberryPi の電源が落ちないようになる．また，コンデンサを追加したことによりショートの時間が長くなり 72V に入れていた 5A のヒューズがじわじわと切れたため，10A のヒューズに置き換えた．

5.3　FPGA の top モジュール

FPGA 内のモジュールの中で最上位となる top モジュールはそのまま外部への IO ピンと接続される (Table 5.1)．サーボパックは原則 24V で信号を扱うが FPGA は 3.3V が基本であるため，一部のピンには FPGA で扱うことのできる電圧にするための電圧のレベル変換回路を外付けする．また，内部に複数のモジュールを含む (Fig.5.8)．

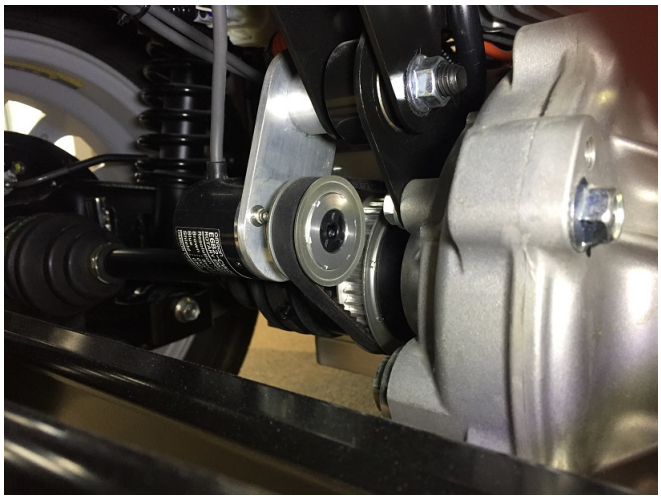

Fig. 5.4 encoders on Driveshaft

Table 5.1: connection of FPGA

PIN	I/O	function
FPGA board		
OSC_FPGA	In	50MHz clock oscillator
PMOD2(7:0)	Out	state monitor LEDs
SERVO MOTOR		
ALM	In	alarm
SEN	Out	ask absolute encoder
S_RDY	In	servo ready
S_ON	Out	supply motor power
PAO_P	In	encoder A-phase+
PAO_N	In	encoder A-phase−
PBO_P	In	encoder B-phase+
PBO_N	In	encoder B-phase−

Table 5.1: connection of FPGA

PAI_P	Out	motor A-phase+
PAI_N	Out	motor A-phase−
PBI_P	Out	motor B-phase+
PBI_N	Out	motor B-phase−
CLR_P	Out	motor clear target+
CLR_N	Out	motor clear target−
COIN	In	position coincidence
P_OT	In	positive limit sw
N_OT	In	negative limit sw
REAR WHEEL ENCODER		
PHASE_LA	In	left encoder A-phase
PHASE_LB	In	left encoder B-phase
PHASE_RA	In	right encoder A-phase
PHASE_RB	In	right encoder B-phase
RaspberryPi		
ARD_MOSI	In	spi MOSI
ARD_MISO	Out	spi MISO
ARD_SCK	In	spi SCK
ARD_SS	In	spi SS

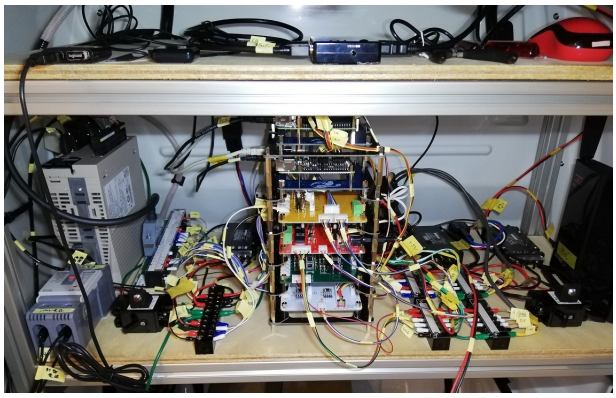

Fig. 5.5　FPGA and RaspberryPi in trunk

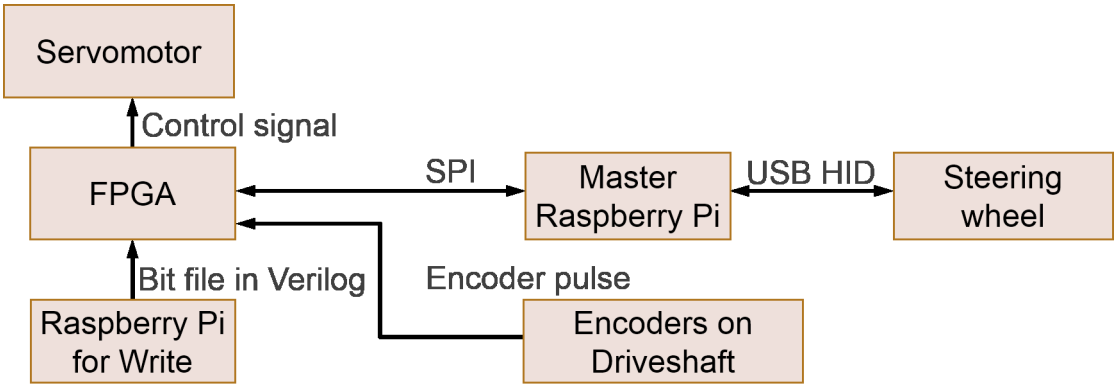

Fig. 5.6　System

5.3.1　ibuf_ain および ibuf_bin モジュール

サーボパックのエンコーダ信号線は差動による出力がされる．エンコーダは A 相と B 相の二つの信号があり，A 相について，差動の正側である PAO_P と負側である PAO_N を差動バッファを通したのちエンコーダ信号を受け取る servo_encoder モジュールにつなげる (Fig.5.9)．また，B 相も A 相と同様に差動バッファに通し servo_encoder モジュールにつなげる．

Fig. 5.7　relay

5.3.2　servo_encoder モジュール

servo_encoder モジュールは，サーボパックからの ALM,ibuf_ain を通した pao,ibuf_bin を通した pbo, 上位装置との spi 通信による SEN 出力指令である sen_sw, 初期化指令の reset を入力に持ち，サーボパックへの絶対値エンコーダ要求信号である SEN, 起動時の回転量を保存するレジスタの rotation(15:0), 回転量情報の受信を完了したことを知らせる ascii_finish, 現在の絶対位置を保存するレジスタの absolute(25:0)，起動時からの相対位置を保存するレジスタの increment(25:0) を出力にもつ (Fig.5.10)．これらの入出力は内部の複数のモジュールから構成される．

5.3.3　almcheck モジュール

almcheck モジュールはサーボパックからの警告を検出するモジュールである．サーボパックからの警告を示す ALM が検出されず，かつ，上位装置からの SEN 信号出力指令を受けた時，SEN 信号を出力する (Fig.5.11)．また，SEN 信号は ON あるいは OFF の切り替え後に一定時間その状態を持続させる必要があるため，タイマを用意する．

5.3.4 pao_busy モジュール

almcheck モジュールにより送られた SEN 信号をサーボパックが受け取ったのちエンコーダ信号線の A 相である pao に絶対位置情報が伝送されるが，SEN 信号受信直後は pao に不定領域が存在する．そのため，タイマを入れることで pao の状態が確定するまでの一定の時間は pao を絶対位置情報の翻訳部から切り離す (Fig.5.12).

5.3.5 pao_baudrate モジュール

絶対位置情報の翻訳について，一定の間隔ごとにデータが送られる調歩同期式のため，pao_busy モジュールによって不定領域を排除した pao を監視することで翻訳部と信号との同期用のクロックを生成する (Fig.5.13).

5.3.6 ascii モジュール

pao に送られる絶対位置情報について，pao_baurate によって生成された同期用クロックの立ち上げ時に pao の中身を取り出し，取り出した ASCII コードを順次翻訳していき，回転量保存用レジスタの rotation(15:0) に記録する．また，最後の同期用クロックの立ち上げが確認されたのち，ascii_finish を出力し，絶対位置情報の ascii コード部の取得完了を外部のモジュールに知らせる (Fig.5.14).

5.3.7 servo_increment モジュール

pao,pbo の 2 相からなる 4 逓倍のインクリメントパルスを記録し，起動時からの相対位置のレジスタである increment(25:0) に格納する (Fig.5.15)．また，SEN 信号出力後には pao に絶対位置情報のパルス列が出力されるが，pbo が一定に保たれているため，絶対位置情報のパルス列が相対位置として加算され続けることはない.

5.3.8 add_absolute モジュール

回転量保存用レジスタの rotation(15:0) をビットシフトすることでパルス数に変換したものと起動時からの相対位置である increment(25:0) を足し合わせることで，現在における絶対位置が求められ，求めた絶対位置を absolute(25:0) に格納する (Fig.5.16).

5.4 servo_initial モジュール

サーボパックが絶対位置を出力し終えたのち，サーボモータへの通電が可能であればサーボパックから S_RDY 信号が出力される．ここで，S_RDY 入力信号の立ち上げを検出した瞬間のサーボモータのエンコーダの絶対値を absolute_initial レジスタに格納する (Fig.5.17). この値を起動時のサーボモータの絶対位置として以降のサーボモータ駆動の基準点にする．

5.5 s_on_in モジュール

s_on_in モジュールはサーボモータへの通電指令の許可を出すモジュールである．絶対位置情報の ascii コードの受け取り成功を示す ascii_finish が出力され，サーボモータへの通電可能を示す S_RDY 信号が出力され，かつ，上位装置からの S_ON 信号出力指令を受けた時，S_ON の出力を指示する信号を出力する (Fig.5.18).

5.6 servo_on モジュール

s_on_in から S_ON 信号の出力許可が出たのち，S_ON 信号を出力する．接続時はほかのモジュールの処理を追いつかせるために遅れが出ても問題ないが，切断時は瞬時に切り離したいため，オンディレイを入れる (Fig.5.19).

5.7 encoder_l および encoder_r モジュール

後輪のエンコーダは左右で合計 2 個ある．左側のエンコーダについて PHASE_LA,PHASE_LB の 2 相からなる 4 逓倍のインクリメントパルスを記録し，起動時からの相対位置のレジスタである data(31:0) に格納し，spi モジュールに接続する (Fig.**??**)．また，右側のエンコーダも左側と同様に相対位置を格納し，spi モジュールに接続する．

5.8 servo モジュール

servo モジュールは，正転および逆転のオーバートラベルを検出する P_OT,N_OT，起動時の絶対位置を示す absolute_initial(25:0), 上位装置との spi 通信により設定される目標位置指令 absolute_target(25:0), およびパルス出力速度 speed(15:0), 目標値が更新されたことを示す absolute_seted, パルス出力速度が更新されたことを示す speed_seted, サーボモータ通電指令を示す S_ON, 初期化指令の reset を入力に持ち，サーボパックへの A 相指令である pai,B 相指令の pbi，サーボパック内部にためられた位置偏差をクリアする指令である clr を出力にもつ (Fig.5.21)．また，pai,pbi,clr はぞれぞれ PAI_P,PBI_P,CLR_P に接続され，PAI_N,PBI_N,CLR_N は top モジュール内部で GND 接続される．

5.8.1 servo_clr モジュール

P_OT あるいは N_OT についてチャタリングノイズを除去したのち，どちらかが反応したとき，サーボパック内部にためられた位置偏差をクリアする指令である clr の出力を許可する (Fig.5.22).

5.8.2 mux_clr モジュール

servo_clr モジュールによる clr 信号はサーボモータが通電しているときのみ有効であるため，オーバートラベルを検出する clr 出力時は S_ON の状態を clr 出力に反映し，オーバー

トラベル未検出の clr 出力停止時は GND を clr 出力にする (Fig.5.23).

5.8.3 s_on_busy モジュール

サーボモータの通電直後に位置指令を出力するとサーボパックの素子を痛めるので，タイマを入れることで通電しきってから位置指令を出力するモジュールを駆動するようにする (Fig.5.24).

5.8.4 servo_drive モジュール

s_on_busy モジュールによりサーボモータが完全に通電したのち，P_OT および N_OT によってオーバートラベルが検出されない状態で，上位装置による絶対値指令を受け付けたことを absolute_seted により検出する．servo_initial モジュールから得た起動時の絶対位置である absolute_initial(25:0) と指令値である absolute_target(25:0) の差分から出力すべきパルス数を求め，speed(15:0) をもとに pai および pbi による 4 逓倍パルス出力の速度を決定し，指令パルスを出力する (Fig.5.25).

5.8.5 spi モジュール

RaspberryPi と FPGA は SPI 通信によって接続され，すべての信号はこのモジュール (Fig.5.26) を経由する．SPI 通信によって伝送される内容とタイミングからコマンドとデータを切り分けていく (Table 5.2)．8bit からなるコマンドの最上位 bit が 0 の場合は FPGA 内のレジスタへの変更はされず RaspberryPi が FPGA のデータを読み込む命令であり，最上位 bit が 1 の場合は FPGA 内のレジスタを変更する命令とする．SERVO ABSOLUTE INITIAL については，サーボパックの初回起動時に自動で SERVO ABSOLUTE の初期値がコピーされる．RaspberryPi へ返すデータが必要ない場合はサーボが通電可能であるか，通電中であるかといったステータスを返すようにする．RESET，SEN，S_ON を順番にステータスを確認しながら立ち上げていくことで正常に起動できるようにしている．BACKWARDS

はモータと接続するギアに応じて反転方向に対応できるようにするために用意している．SPI 通信の 8bit を割り当てる時にサーボモータの値に関して反転させることで現在値，指令値のすべてが反転し，回転方向が反転したかのように見せかけている．実際に駆動する場合は SERVO TARGET を使用する．現在値と指令値を同時に授受することで少ない通信回数で済むようにしている．servo_speed ではサーボモータの回転速度（パルス打ち出しの最高速度）を指定できるようにしている．

Table 5.2: SPI command

command	function
SERVO ROTATION	
8'h01	return: rotation[15:8]
8'h02	return: rotation[7:0]
SERVO INCREMENT	
8'h11	return: increment[31:24]
8'h12	return: increment[23:16]
8'h13	return: increment[15:8]
8'h14	return: increment[7:0]
ENCODER LEFT	
8'h21	return: increment_l[31:24]
8'h22	return: increment_l[23:16]
8'h23	return: increment_l[15:8]
8'h24	return: increment_l[7:0]
ENCODER RIGHT	
8'h31	return: increment_r[31:24]
8'h32	return: increment_r[23:16]

Table 5.2: SPI command

8'h33	return: increment_r[15:8]
8'h34	return: increment_r[7:0]
SERVO ABSOLUTE	
8'h41	return: absolute[31:24]
8'h42	return: absolute[23:16]
8'h43	return: absolute[15:8]
8'h44	return: absolute[7:0]
SERVO ABSOLUTE INITIAL	
8'h51	return: absolute[31:24]
8'h52	return: absolute[23:16]
8'h53	return: absolute[15:8]
8'h54	return: absolute[7:0]
SERVO ABSOLUTE TARGET BUF	
8'h61	return: absolute_target_buf[31:24]
8'h62	return: absolute_target_buf[23:16]
8'h63	return: absolute_target_buf[15:8]
8'h64	return: absolute_target_buf[7:0]
RECORD TO BUF	
8'h70	return: stat
RESET	
8'h80	assign0: reset=0 (reset)
	return0: stat
8'h81	assign0: reset=1 (releaser)

Table 5.2: SPI command

	return0: stat
SEN	
8'h90	assign0: sen_sw=0 (Lo to SEN)
	return0: stat
8'h91	assign0: sen_sw=1 (Hi to SEN)
	return0: stat
S_ON	
8'hA0	assign0: s_on_sw=0 (Lo to S_ON)
	return0: stat
8'hA1	assign0: s_on_sw=1 (Hi to S_ON)
	return0: stat
BACKWARDS	
8'hE0	assign0: backwards=0 (Forward rotation)
	return0: stat
8'hE1	assign0: backwards=1 (Backward rotation)
	return0: stat
SERVO TARGET	
8'hF0	return0: absolute[31:24]
	assign1: servo_target[31:24]
	return1: absolute[23:16]
	assign2: servo_target[23:16]
	return2: absolute[15:8]
	assign3: servo_target[15:8]
	return3: absolute[7:0]

Table 5.2: SPI command

8'hF1	assign4: servo_target[7:0] return4: stat return0: stat assign1: servo_speed[15:8] return1: stat assign2: servo_speed[7:0] return2: stat

5.9 モータの置き換え

これまでは特別に小型のモータを使用していたが，回転速度が遅いため，より大きいモータに置き換える (Fig.5.27).

5.10 C による制御ループ

Python による制御ループではループそのものが遅いという問題と，一定の周期で動いていないという問題がある．そこで，同じような環境を用意し，C に変えることで解決を試みる．

まず，FPGA とは SPI 通信をするため，テスト環境として，SPI 通信ができるような DA コンバータと RaspberryPi を接続する (Fig.5.28)．そしてデータを送るループを作る．ここでは，SPI 通信で DA コンバータに 16bit のデータを送り，sleep によって時間をはかり，電圧を線形に上昇させていく．さらに，通信の合図である SS ピンについて，データをまとめて送ることでピンの操作回数を減らし，ピンの操作分速くする．また，データを double 型

から long 型にすることで送るデータを作る処理を速くする．以上によりループそのものは速くなるので，次は周期について手を加える．システムの時間を格納する構造体を作り，前回のループの時間を記録し，一定周期経過したかを do-while で判断させ続ける．これにより，ループ間の待機時間をはかることができる．さらに，周期を任意に変化させることで，今回は DAC を使用したため，スピーカーをつなぐことでシンセサイザーの機能を持つことになる．少なくとも 880Hz の三角波を作ることができ，任意のメロディーを鳴らすことに成功した．この仕組みを車両の制御に適用することで一定周期のループによって動かすことが可能だと考える．

5.11 基板の作成

モータの変更に伴い，授受する信号の数が増えたため，荷台のスペースに収まるように基板を作成する．(Fig.5.29,Fig.5.30,Fig.5.31,Fig.5.32,Fig.5.33,Fig.5.34)．

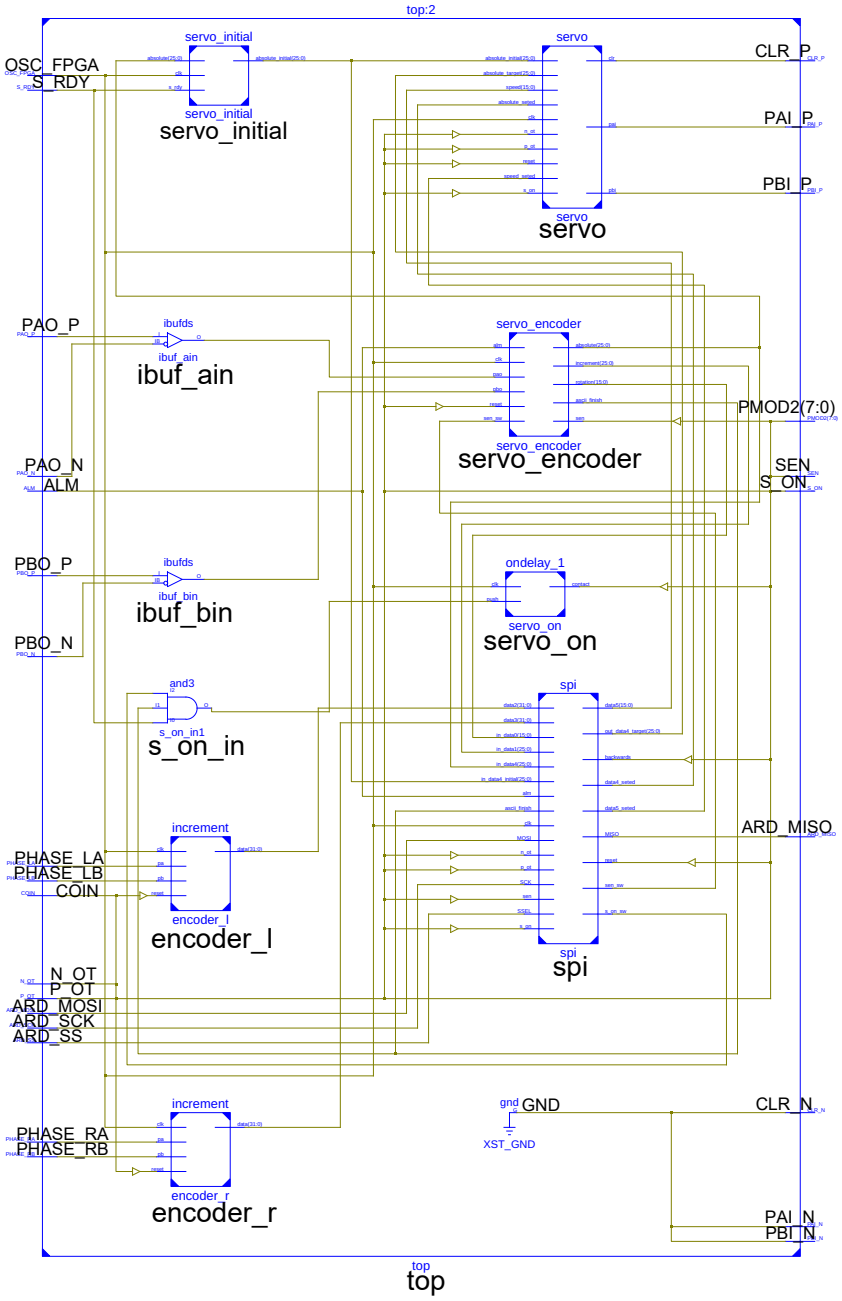

Fig. 5.8　Details of top module

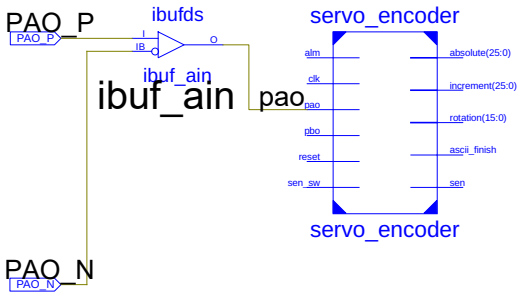

Fig. 5.9　Details of ibuf_ain module

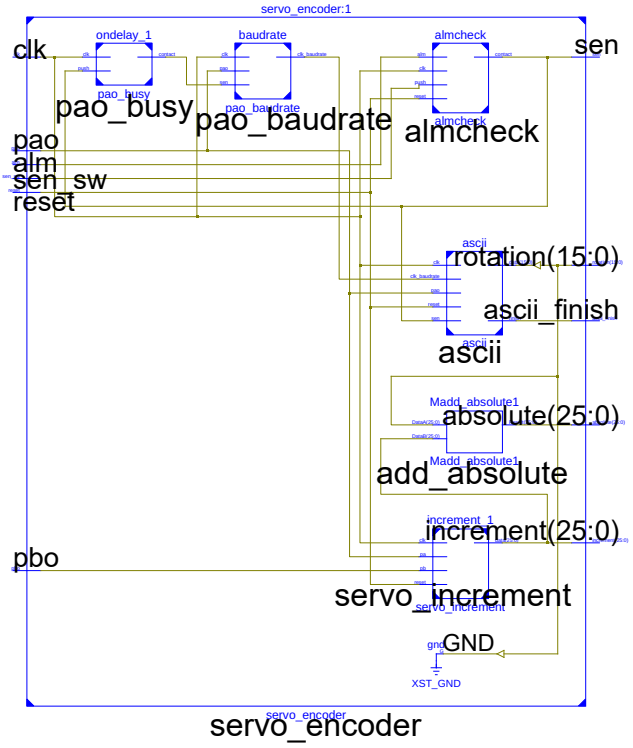

Fig. 5.10　Details of servo_encoder module

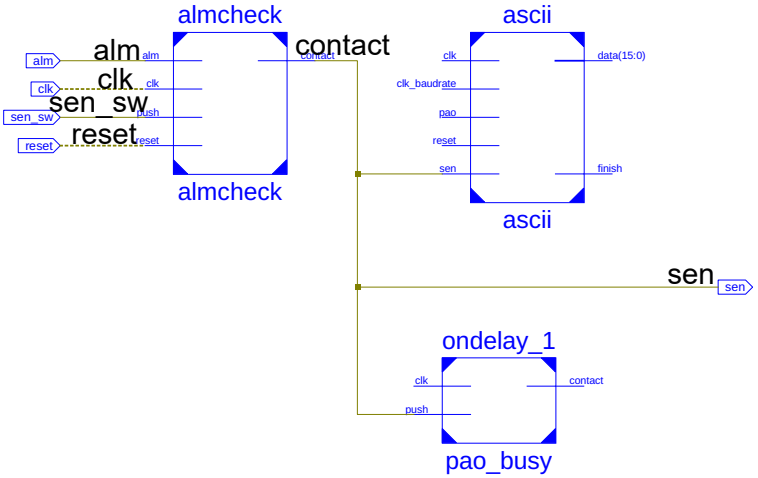

Fig. 5.11　Details of almcheck module

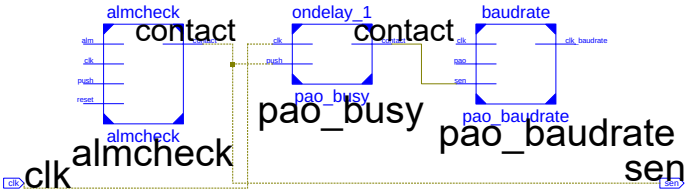

Fig. 5.12　Details of pao_busy module

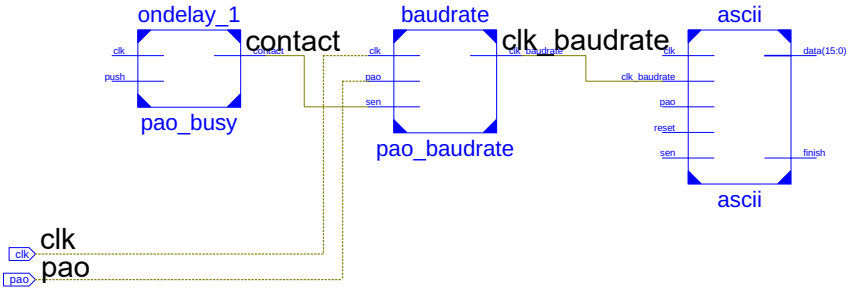

Fig. 5.13　Details of pao_baudrate module

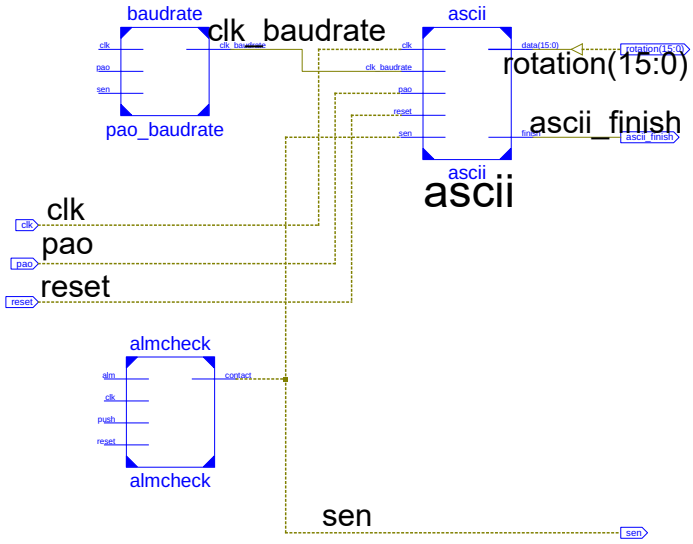

Fig. 5.14　Details of ascii module

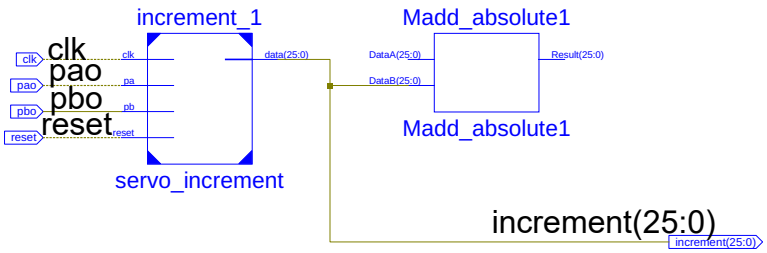

Fig. 5.15　Details of servo_increment module

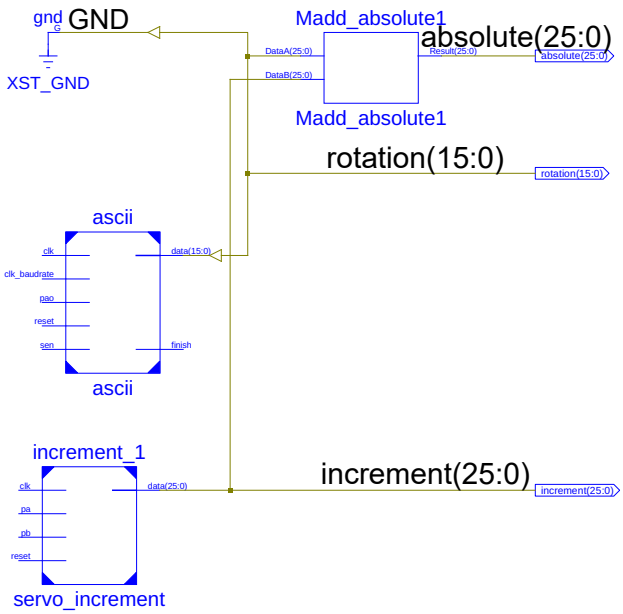

Fig. 5.16　Details of add_absolute module

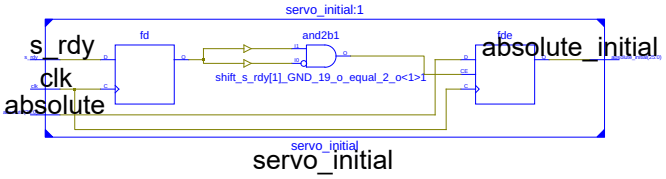

Fig. 5.17　Details of servo_initial module

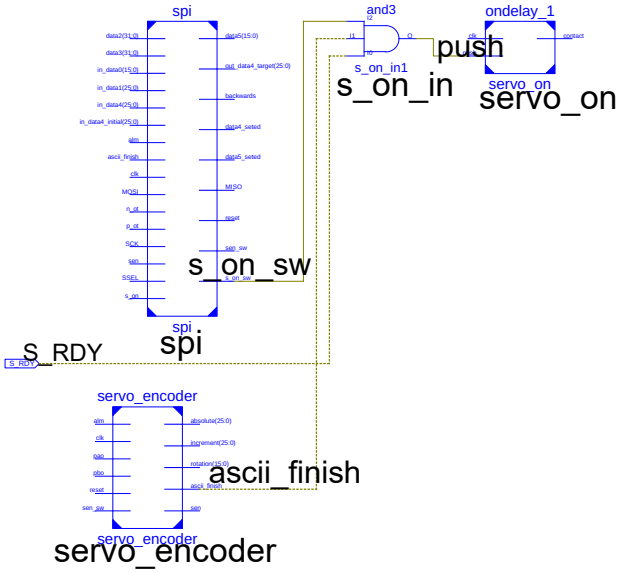

Fig. 5.18　Details of s_on_in module

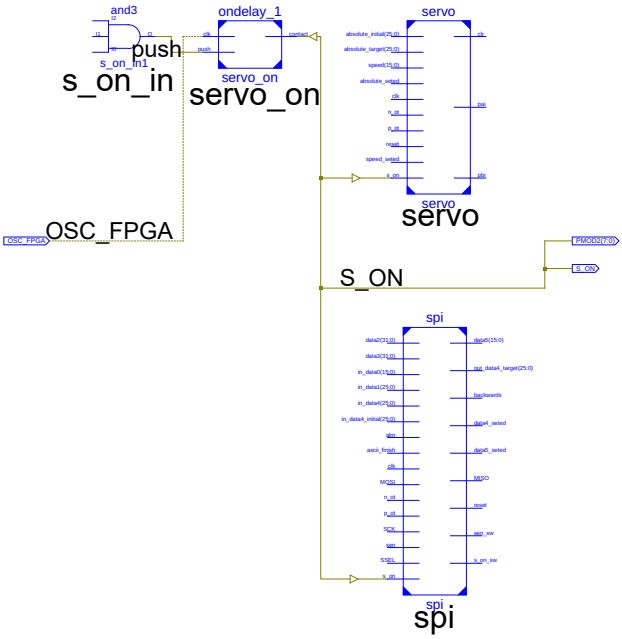

Fig. 5.19　Details of servo_on module

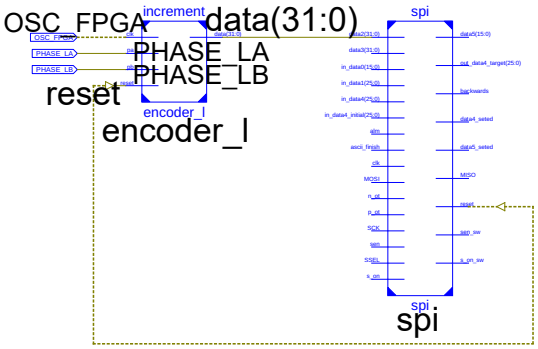

Fig. 5.20　Details of encoder_l module

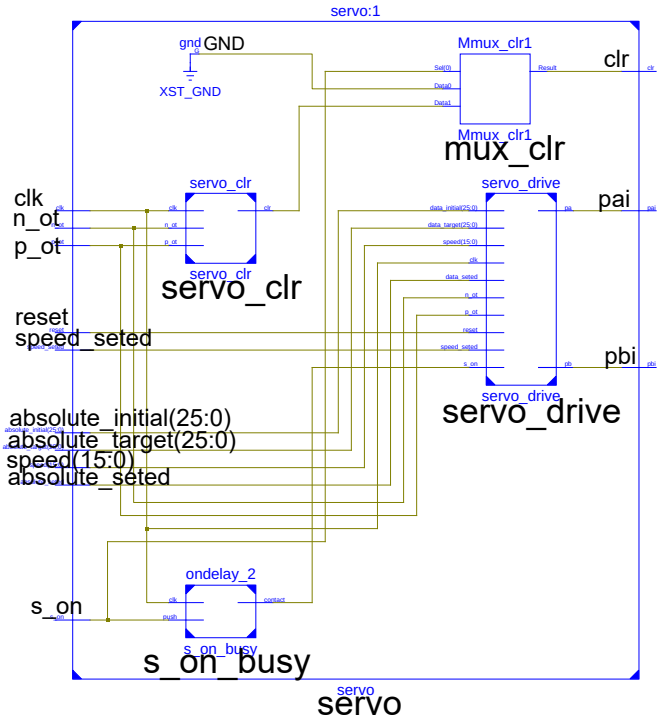

Fig. 5.21　Details of servo module

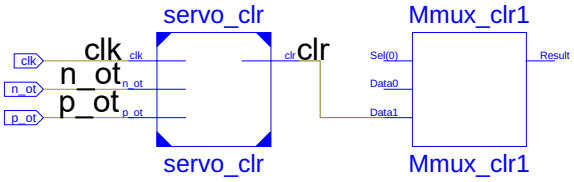

Fig. 5.22　Details of servo_clr module

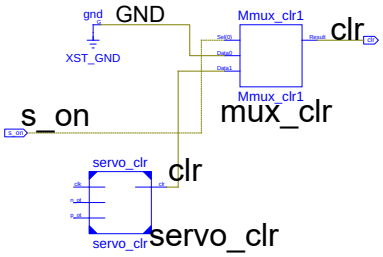

Fig. 5.23　Details of mux_clr module

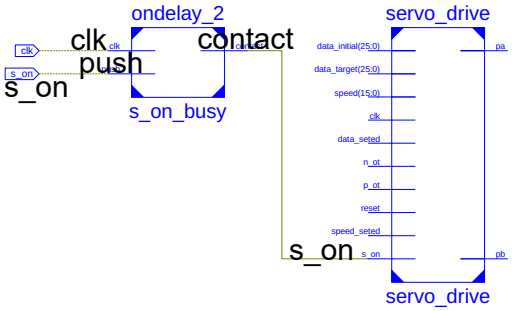

Fig. 5.24　Details of s_on_busy module

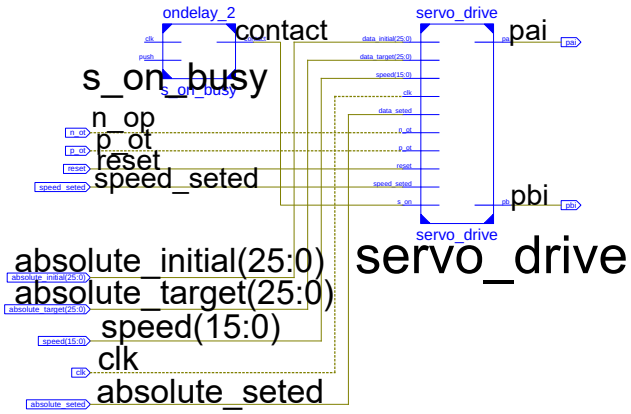

Fig. 5.25　Details of servo_drive module

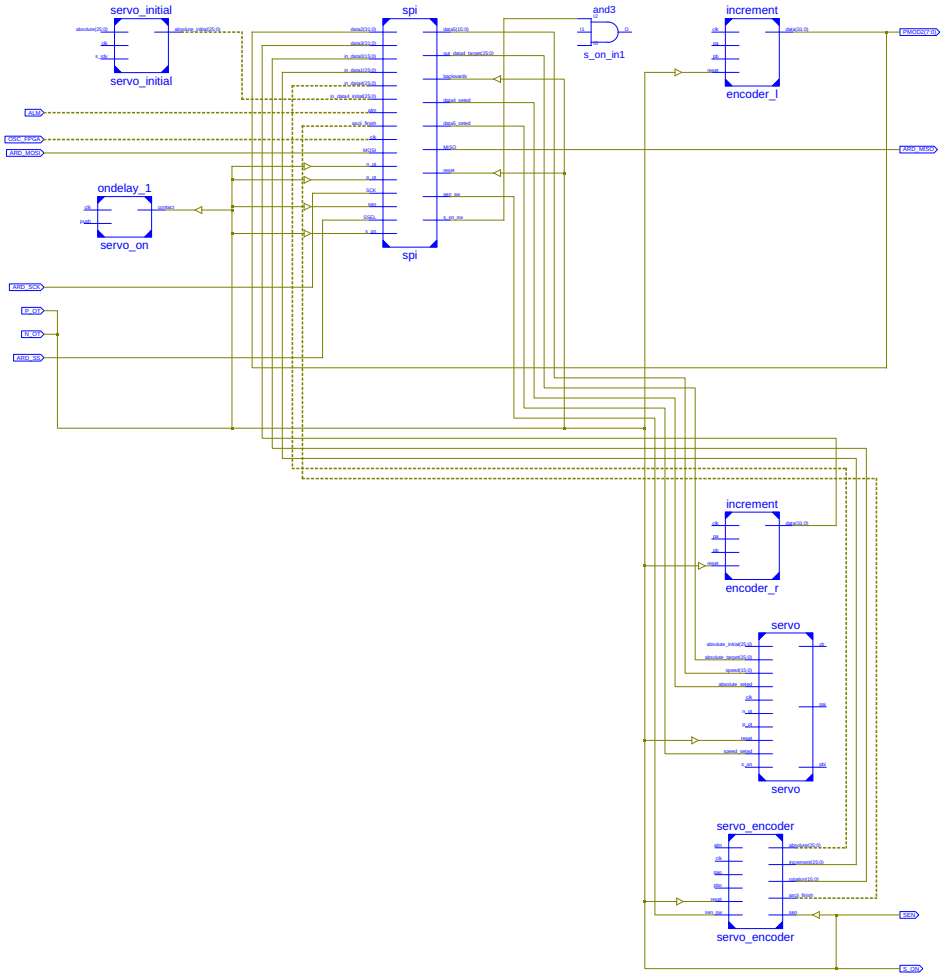

Fig. 5.26　Details of spi module

Fig. 5.27　replace with large motor

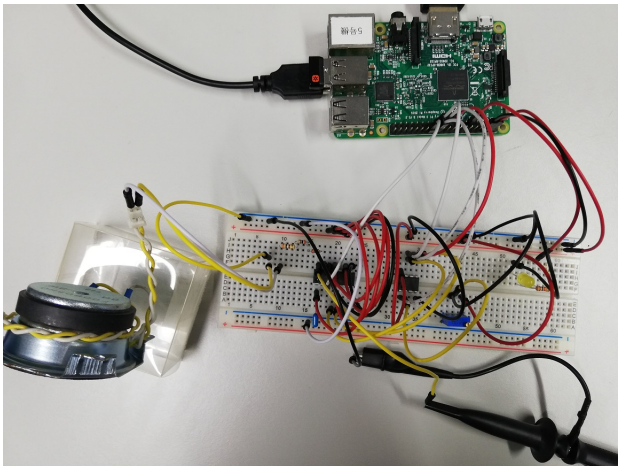

Fig. 5.28　synthesizer(C loop test)

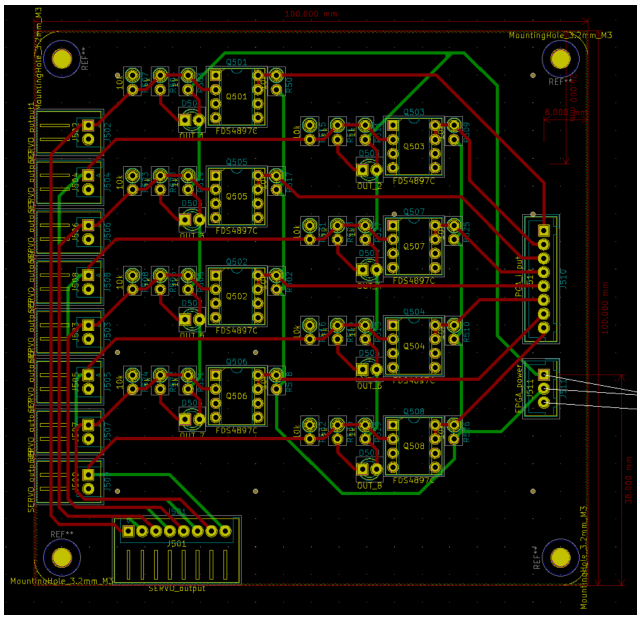

Fig. 5.29　pcb of servo output

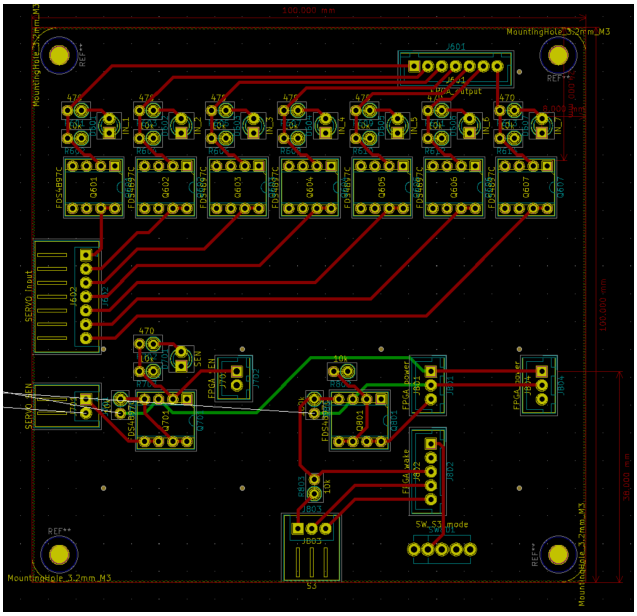

Fig. 5.30　pcb of servo input

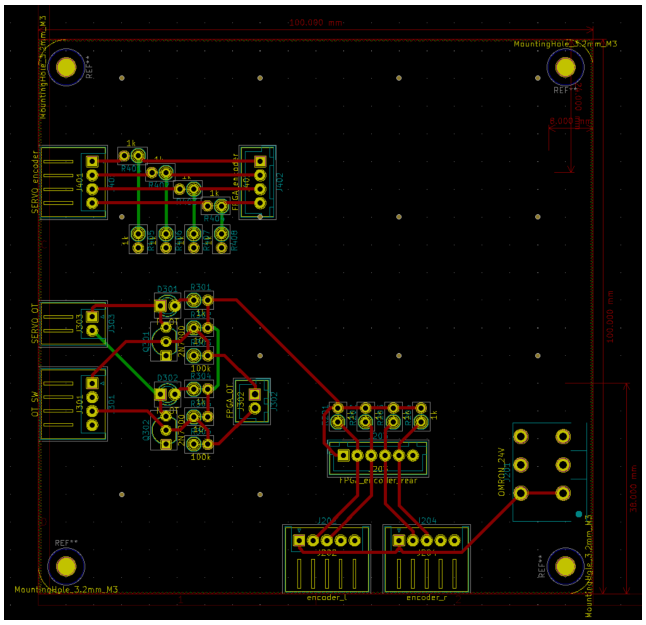

Fig. 5.31　pcb of encoder, overtravel

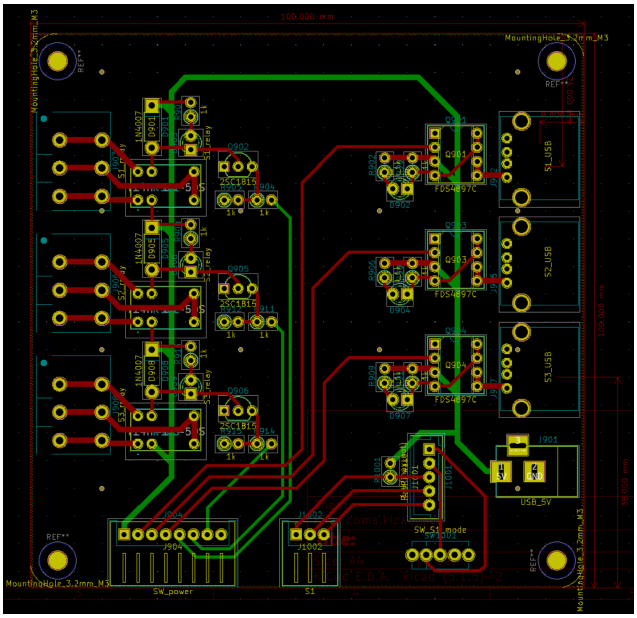

Fig. 5.32　pcb of power sw

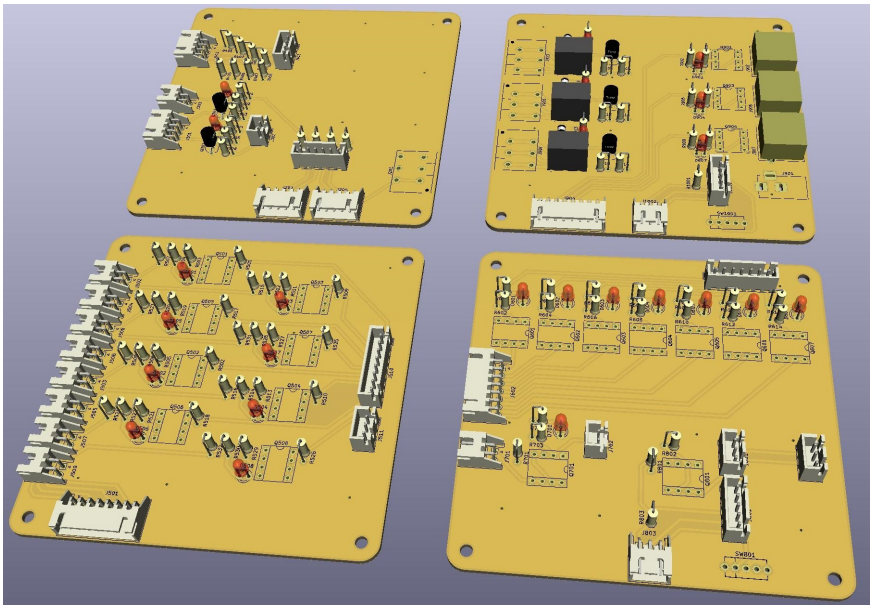

Fig. 5.33　pcb view from top

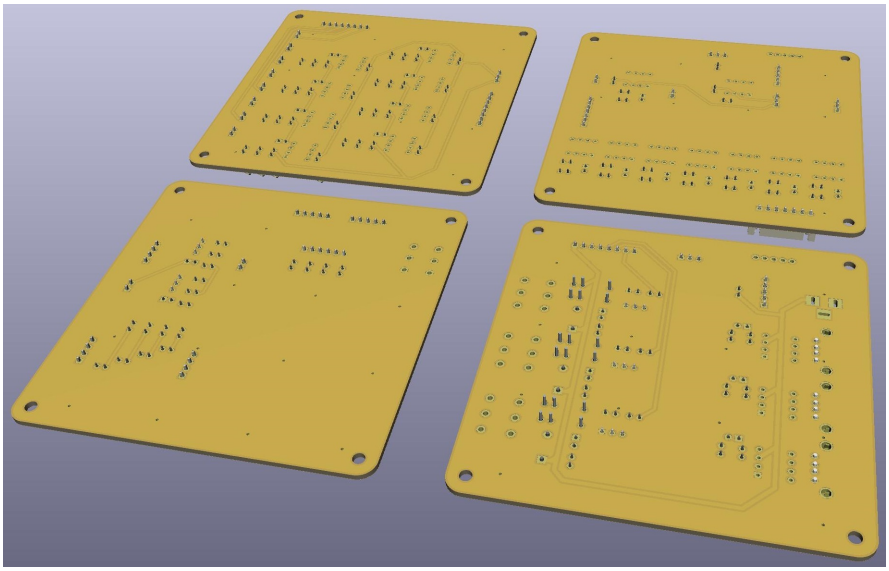

Fig. 5.34　pcb view from bottom

第6章

実験

まず，ハンドル-サーボモータ間が機械的に結合されているかのように運転する．サーボモータを初期化し，絶対値エンコーダの初期値を取得する．入力デバイスをイベントデバイスとして検出する．ハンドルとサーボモータを同期させる．次にハンドルを回して操縦し，ログを csv ファイルとして保存する (Fig.6.1).

6.1 提案手法による操舵

提案手法は，車両の移動距離に従い操舵可能角を決定するものである．この手法において操舵可能角を小さくすると，消費電力とタイヤの摩耗が減少する．しかし，違和感が増す．本研究の目標は提案手法においてこの不快感が感じられるようになる閾値を見つけ，単位距離あたりの操舵可能角を見つけることである．

後輪エンコーダパルスに従い操縦する．サーボモータのパルスは $\pm 17 \times 2^{16} (\pm 1, 114, 112)$ である．操舵可能角は後輪エンコーダパルス 1 個ごとの値である．タイヤ 1 回転ごとにそれぞれの後輪エンコーダから 16,000 パルスが出力される．シフトレバーを動かすことで車両が振動する．後輪エンコーダパルス 1 個ごとにサーボモータに 1024 個のパルスを送る設定 (Fig.6.2) と後輪エンコーダパルス 1 個ごとにサーボモータに 128 個のパルスを送る設定 (Fig.6.3) で操舵する．

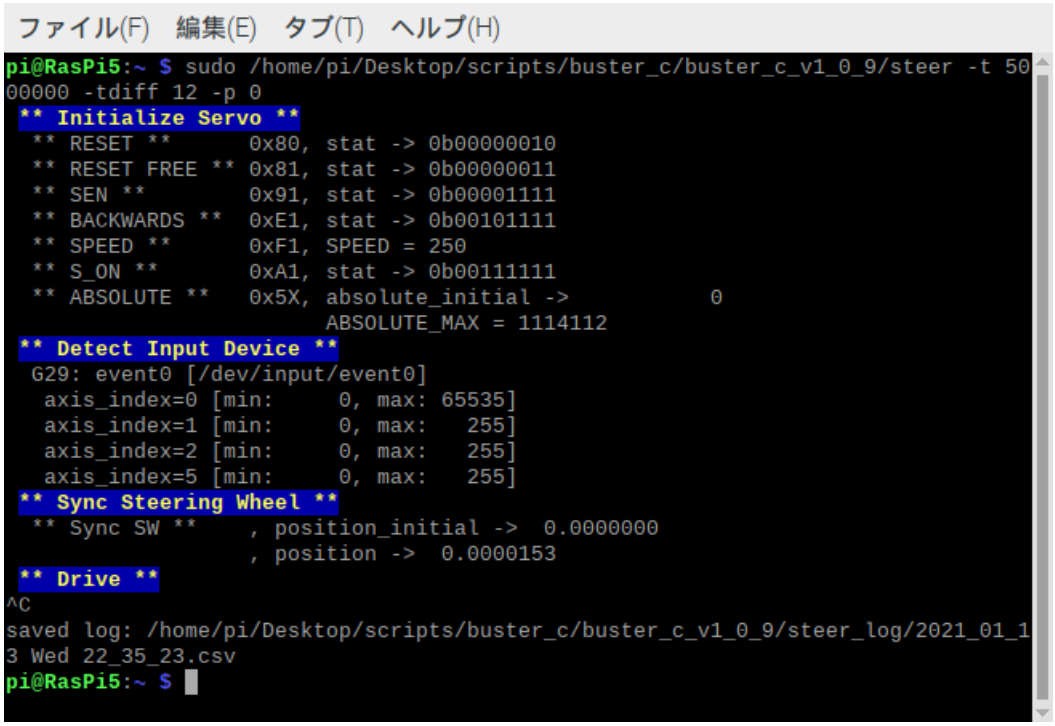

Fig. 6.1 terminal of RaspberryPi

6.2 被験者から得られた結果

カーブを曲がる動作で，二人の被験者によって実験した．ブレーキを踏んだままハンドルを切る．この時，舵は動かない．次にブレーキを解除すると，車両が前進するにつれて舵がハンドルに対応する位置に移動する．被験者の狙い通りのコースで車両が走るかどうかを，1 から 5 の指標により違和感を感じたかどうか回答してもらう (Fig.6.4)．

少ない被験者による実験では，移動距離に対する転舵可能角が減るにつれて，不快感が増すという傾向が表れた．ハンドルの位置に対してより転舵位置が追従しているにもかかわらず，不快に感じると回答した場合があるが，それよりも転舵可能角が小さい場合でも不快に感じないと回答した場合もあり，被験者の気のせいだといえる場合もある．今回は少人数ではあるが，すえ切り状態での検証において，被験者が違和感を感じる遅れは予想以上に大きく，すえ切り時に提案手法を用いることは摩耗防止に十分役に立つことが分かった．より多

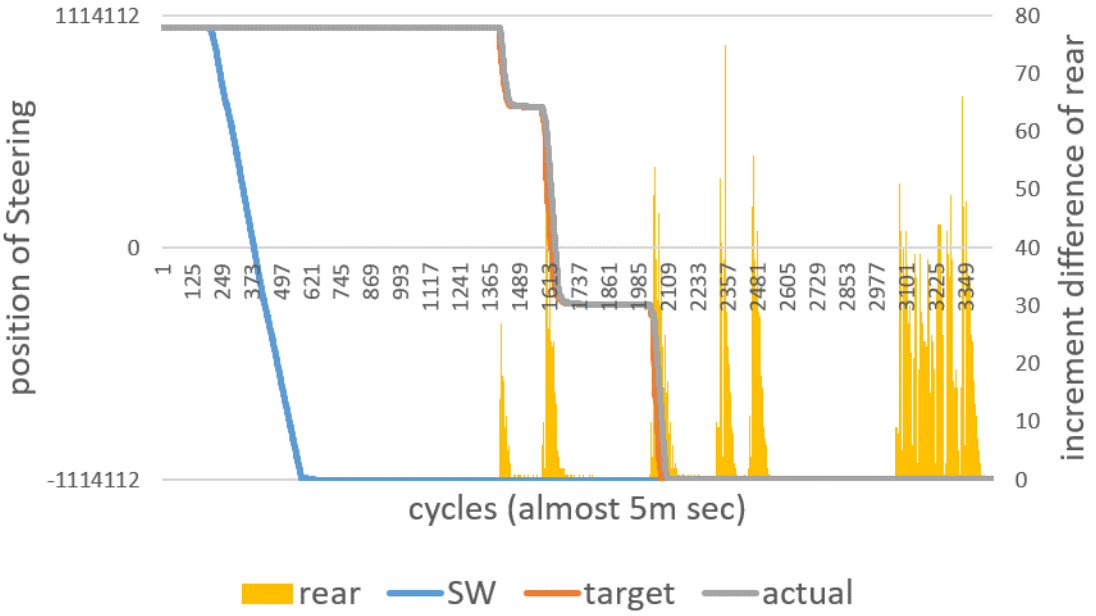

Fig. 6.2　fits 1024 steering pulses per 1 rear pulse

くの被験者に試してもらうことで，誰もが違和感を感じない範囲ですえ切りを防ぐことのできる境界を探すことができると考える．

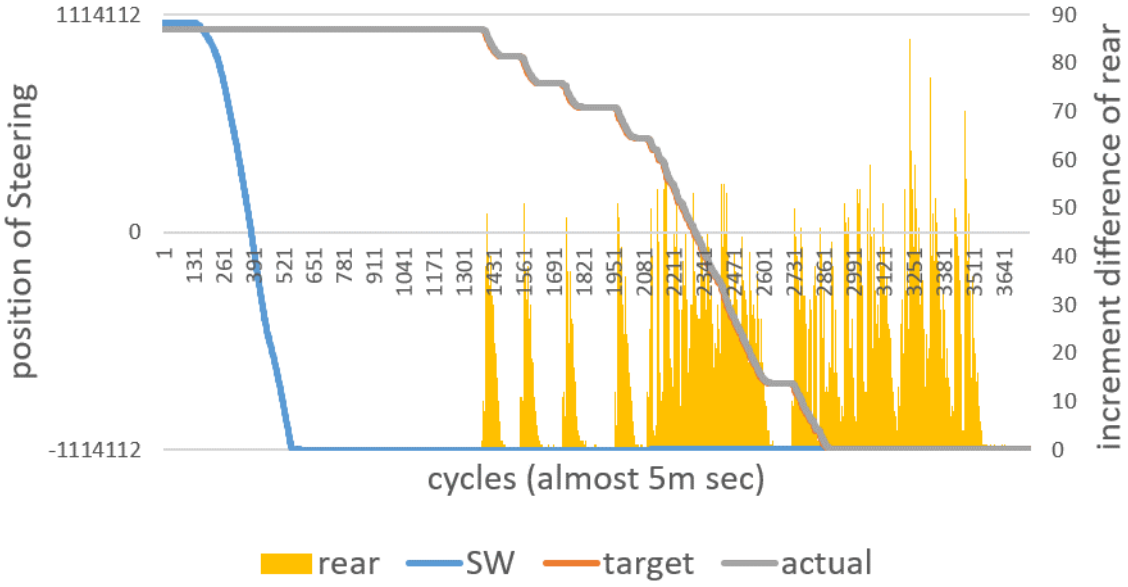

Fig. 6.3 fits 128 steering pulses per 1 rear pulse

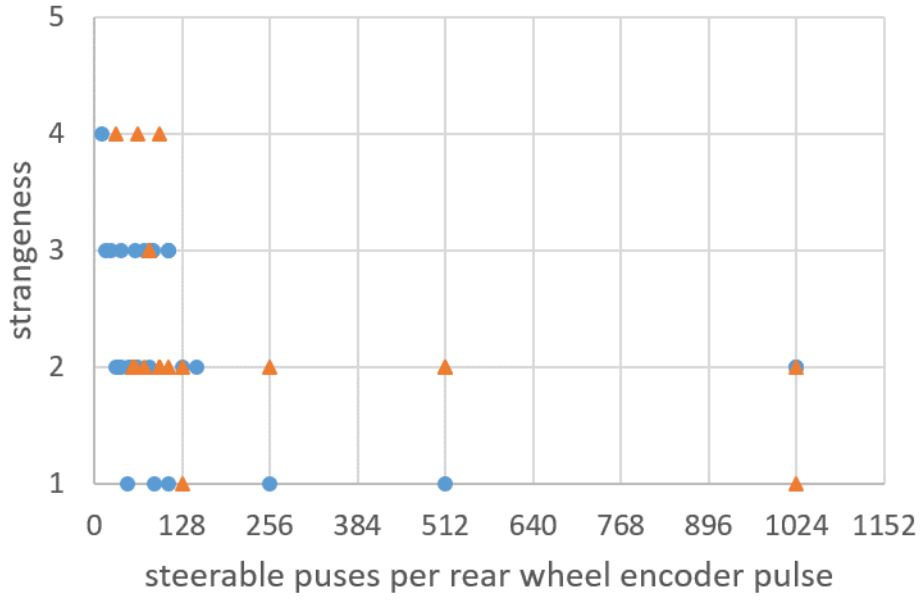

Fig. 6.4 running on a curve

第7章

まとめ

ハンドルと転舵アクチュエータを機構的に結合しないことで新しい操舵方法を提案する．現在採用されているパワーステアリングはスペースを多く必要としエネルギー消費も大きい．車両停止時の転舵を防ぐことで，モータ及び減速機の小型化とエネルギー消費を抑えることが可能だと考えられる．ドライブシャフトにエンコーダを取り付けることで移動距離を取得し，ハンドルの位置に対する転舵アクチュエータの動作に移動距離に応じた変化を加えることで機械的結合がある場合と比較して違和感のない範囲で新しい操舵方法を提案する．

ステアバイワイヤ車の場合の舵角余裕を計測する．一般に，ステアバイワイヤシステムはできるだけ早く車両の舵角がステアリングホイール角に応じた角度になるように設計されている．しかし，車速によっては，ステアリングホイール角と実際の舵角に多少の差があっても操作感覚に対して大きな影響がない．そこで，ステアリングホイール角に対して舵角を意図的に変え，その差に対する人間の許容度を調べる．その許容の範囲で，ステアリングホイール角に対する実際の舵角の差を利用し，エネルギー効率や車両の振動，タイヤの摩耗を減らすような手法が考えられる．そのための車両のシステムを作っていく．まず，操舵角はゲーム機に接続するようなハンドルコントローラ (G29) から取得し，車両の軌跡は左右の後輪にエンコーダを取り付けて FPGA にパルスを送り測定する．操舵角と軌跡，実際の転舵角を上位装置となる Raspberry-Pi に送り，FPGA を経由して転舵部のサーボモータに目標

値を送る．実際の車両（改造 COMS）においての車両の動きに対応する人間の許容誤差を測定し，その許容誤差を考慮して転舵方法を提案する．単位時間当たりの移動距離に応じた分だけ転舵可能な角度を設定する．求めた転舵可能角度を許容することですえ切りを防止し，人間の許容度の範囲内で，タイヤの摩耗を減らしエネルギー消費を減らす方法を提案する．

少ない被験者による実験では，移動距離に対する転舵可能角が減るにつれて，不快感が増すという傾向が表れた．ハンドルの位置に対してより転舵位置が追従しているにもかかわらず，不快に感じると回答した場合があるが，それよりも転舵可能角が小さい場合でも不快に感じないと回答した場合もあり，被験者の気のせいだといえる場合もある．今回は少人数ではあるが，すえ切り状態での検証において，被験者が違和感を感じる遅れは予想以上に大きく，すえ切り時に提案手法を用いることは摩耗防止に十分役に立つことが分かった．より多くの被験者に試してもらうことで，誰もが違和感を感じない範囲ですえ切りを防ぐことのできる境界を探すことができると考える．

参考文献

[1] 井尻　和一郎，筒井　高志，電動パワーステアリングの技術動向，KOYO Engineering Journal No.162，pp.28〜32，2002

[2] 佐藤　文彦，東　真康，杉山　豊樹，電動パワーステアリング用補助電源システムの開発，JTEKT ENGINEERING JOURNAL No.1013，pp.41〜46，2015

[3] 金井　喜美雄，航空機と自動車の共用制御技術，計測と制御　第 40 巻　第 12 号，pp.921〜926，2001

[4] 金井　喜美雄，航空機制御技術と自動車への適用，計測と制御　第 45 巻　第 3 号，pp.177〜184，2006

[5] 高松　孝修，中野　史郎，ステアバイワイヤの操舵制御，KOYO Engineering Journal No.158，pp.21〜26，2000

[6] Eid S. Mohamed，Saeed A. Albatlan，Modeling and Experimental Design Approach for Integration of Conventional Power Steering and a Steer-By-Wire System Based on Active Steering Angle Control，American Journal of Vehicle Design Vol.2 No.1，pp.32〜42，2014

[7] 葉山　良平，中野　史郎，熊本　博光，西原　修，ステアバイワイヤ車両の転舵応答性評価と運転者-自動車系の閉ループ解析，精密工学会誌　Vol.72 No.11，pp.1374〜1379，2006

[8] 中川　智皓，田中　冬也，新谷　篤彦，伊藤　智博，直進・旋回走行時のステアバイワイヤ小径自転車の挙動，日本機械学会論文集　Vol.85 No.876，p.18-00454，2019

[9] 梅本　貴史，劉　暁俊，成田　正敬，加藤　英晃，森山　裕幸，超小型電気自動車のステアバイワイヤシステム―三角筋の筋電位を用いた送り手上腕部の負担に関する基礎的検討―，日本 AEM 学会誌　Vol.25 No.2，pp.155〜161，2017

[10] 弓矢　将成，北村　武士，池町　優太，日尾　僚兵，氏家　良樹，古屋　繁，松岡由幸，2 人のためライフサポートビークル，SPECIAL ISSUE OF JSSD Vol.12 No.4，pp.80〜88，2005

[11] B・COM　—　超小型　EV「コムス」　：トヨタ車体，http://coms.toyotabody.jp/style/bcom.html，2017/05/18

謝辞

本研究を遂行するにあたり，装置設計のご助言に至るまでご指導ご鞭撻を賜りました知能ロボティクス研究室 松井博和 先生に対して，深く感謝いたします.

本研究の FPGA のプログラムを記述するにあたり，知能ロボティクス研究室 加藤典彦 准教授からご助言を頂き装置間の通信に成功しましたこと，謹んで感謝の意を表します.

本研究に際して，様々なご指導を頂きました知能ロボティクス研究室 矢野賢一 教授に対して深謝いたします.

また，本研究を進めるにあたり，知能ロボティクス研究室の諸氏の感謝いたします.