平成21年度

卒 業 研 究 報 告

研究題目

3次元ポインティングデバイスの製作

指導教員

由井 四海

著者

大場 敬充

下村 祐斗

平成22年 3月 9日提出

独立行政法人国立高等専門学校機構

富山高等専門学校　電子制御工学科

目次

1．　序論

1．1　はじめに

今日、ロボットティーチングをするためには、CAD/CAM（Computer Aided Design/Computer Aided Manufacturing）を用いてロボットに動作を教示することや専用のコントローラから動作を教示することが必要である。そのため、初心者がCAD/CAMや専用のコントローラを用いてロボットに動作を教示させるためには、その使い方から学ばなければならない。初心者がロボットに動作を教示させるには、ロボットの 3 次元ポインティングデバイスを使用して、小型模型を動かしロボットにどのように動けばいいか教える方が簡単である。

1．2　目的

従来の 3 次元ポインティングデバイスでは、小型模型から手を離すと、それが動いてしまうことがあった。また、従来のPCまでのデータの伝送方式だと、高分解能化に対応するためにチャンネル数の多いデジタル I/O インターフェースに付け替えなければならなくなる。そして、配線が多くなりメンテナンスが困難になるなどの問題点があった。そこで、本研究では、従来の問題点を改良した 3 次元ポインティングデバイスを作製することを目的とした。

2．　マニピュレータ

マニピュレータとは、人間の腕のように、関節とリンク（腕）が組み合わされた構造のロボットである。マニピュレータは、現在利用されているロボットの中で最も一般的な形態である。その中でも工場内で組立作業、溶接作業などに用いられるものを産業用ロボットと呼ばれている。図 2.1 に６軸マニピュレータをモデル化したものを示す。

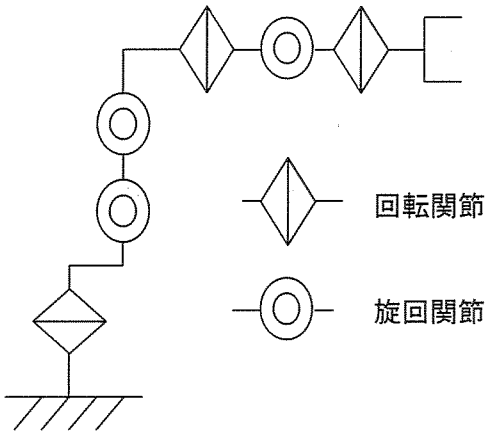


図 2.1　ロボットのモデル

3．　小型模型の作製

3．1　ロータリーエンコーダ

ロータリーエンコーダとは、回転の機械的変位量を電気信号に変換し、この信号を処理して位置・速度などを検出するセンサである。そのロータリーエンコーダには、アブソリュート形とインクリメンタル形がある。アブソリュート形とインクリメンタル形のそれぞれの特徴を以下に示す。

アブソリュート形ロータリーエンコーダ

- 回転軸の絶対的な位置を検出することが可能
- 回転軸の現在位置にあわせてグレイコードと呼ばれる 2 進データを出力

インクリメンタル形ロータリーエンコーダ

- 軸の回転変位量に応じてパルスを出力
- 回転軸の相対的な位置を測定
- 分解能が高い

これらの特徴から、本研究では OMRON 社製のインクリメンタル形ロータリーエンコーダ

を使用することにした。図 3.1 にロータリーエンコーダの外観を示す。また、表 3.1 に使用したインクリメンタル形のロータリーエンコーダの仕様を示す。

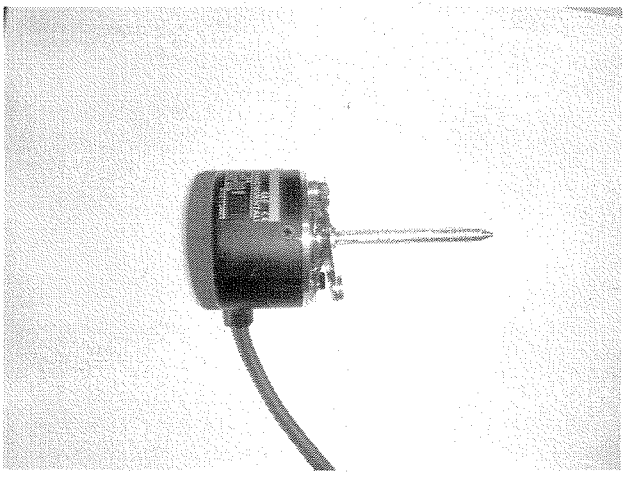


図 3.1　ロータリーエンコーダの外観

表 3.1　使用したインクリメンタル形ロータリーエンコーダの仕様

型式	E6H-CWZ3E
電源電圧	DC5～12V
消費電流	100mA 以下
出力相	A 相、B 相、Z 相
出力形式	電圧出力
出力容量	出力電流：30mA以下
	残留電圧：0.7V 以下
	負荷抵抗：1kΩ
最高応答周波数	100kHz
出力位相差	90±45°
出力立ち上がり・立ち下がり時間	1μs 以下
起動トルク	1.5mN・m 以下
慣性モーメント	2×10^{-6}kg·m² 以下
軸許容力	ラジアル方向　29.4N　/スラスト方向　4.9N
許容最高回転数	10,000r/min
分解能	3600

また、表 3.2 に従来の 3 次元ポインティングデバイスで使用したロータリーエンコーダの仕様を以下に示す。

表 3.2　従来使用していたロータリーエンコーダの仕様

型式	E6J-CWZ1E
電源電圧	DC5±5%(リップル含む)
消費電流	40mA
出力相	A 相、B 相、Z 相
出力形式	電圧出力
出力容量	出力電流：20mA以下
	残留電圧：0.5V 以下
	負荷抵抗：2.2kΩ
最高応答周波数	100kHz
出力位相差	90±45°
出力立ち上がり・立ち下がり時間	2μs以下
起動トルク	1.0mN・m 以下
慣性モーメント	0.045×10^{-7}kg·m² 以下
軸許容力	ラジアル方向　1.9N　/スラスト方向　1.9N
許容最高回転数	6000r/min
分解能	1000

上記の二つの表を比較すると、分解能の違いや出力容量の違いなどがある。そこで、ロータリーエンコーダの分解能が大きい方 (E6H-CWZ3E)を使用することにした。
本研究で使用したロータリーエンコーダは従来よりも分解能が大きいため、小型模型の先端位置と PC 上で示される座標の誤差が小さくなる。また、軸許容力が大きいため、1/3 スケールの小型模型でも十分に取り付けが可能となる。

３．２　姿勢を保持する装置の製作

従来の小型模型では、小型模型から手を離してしまうとその小型模型が動いてしまうことがあった。そのため本研究では、姿勢を保持するための装置を取り付け、小型模型から手を離してもその姿勢を保持するようにした。また、小型模型には、基本姿勢というものを決めた。

この姿勢を保持する装置は、ばねとばねの間の隙間に突起物をいれ、その突起物によって押す力によってばねの復元力が発生し、その復元力で姿勢を保持するという仕組みになっている。姿勢を保持する装置の原理を以下の図 3.2 に示す。本研究では、モーメントの計算からばね定数を決定した。これより、ばね定数が 2.0N/mm のばねを使用した。

設計内容としては、この突起物は、ふたの中心からの同心円からずれてしまうと回転しなくなってしまう。また、軸はアルミ材の中実丸棒を旋盤で製作したため太い所や細い所がある。そのため、突起物の付いた部品とばねの入っている部品の中心がずれないようにするため、突起物の付いた部品にはばねの入っている部品が入る大きさの円筒形のものをつけ、深さはカップの高さとした。この円筒形のものにより、中心のずれが無くなる。

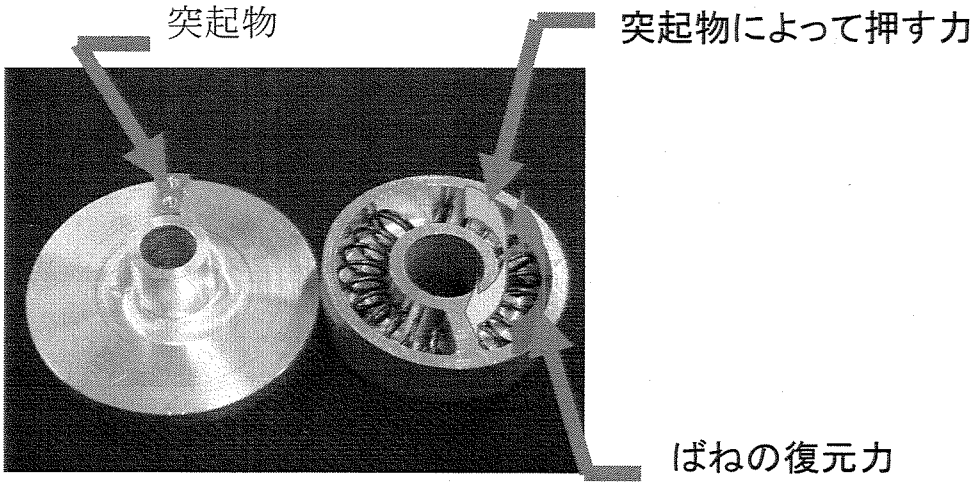


図 3.2　基本姿勢に戻る装置の原理

３．３　小型模型の作製

３．３．１　SolidWorks

SolidWorks とは、SolidWorks 社が販売している 3 DCAD ソフトである。このソフトは、3 次元設計ツールである CAD ソフトウェア、設計検証のためのシミュレーション・　ソフトウェア、製品データ管理ソフトウェア、および製品ドキュメント作成ソフトウェアという、3 次元製品開発環境を包括的に支援し、3 次元データを効率的に活用するための製品である。部品などをドラッグなどによって配置するだけで直感的に設計を行うことができるソフトである。

３．２．２　小型模型の作製

はじめに、SolidWorks を使用し、小型模型の完成予想図を作成した。その小型模型の完成予想図を図 3.3 に示す。完成予想図の黒い部分がロータリーエンコーダを取り付けるところであり、ロータリーエンコーダの付いた軸の反対側についているのが姿勢を保持する装置である。次に SolidWorks で作成した完成予想図のように小型模型を作製した。図 3.4 に製作した小型模型を示す。小型模型は、従来の小型模型はロータリーエンコーダが小さかったため、実際のロボットの 1/5 スケールであったが今回は、ロータリーエンコーダの大きさが大きくなったため、実際のロボットの 1/3 スケールで作製した。図 3.5 に小型模型の寸法を示す。また、本研究では、Misumi 製の部品を用いて小型模型の製作を行った。Misumi 製の部品は鉄製であったため重たくなった。そこで姿勢を保持する装置や、シャフト部分はアルミ材で製作した。Misumi 社の部品は表面加工がされており錆は出ない。しかし、シャフトなどの回転が行われる場所に鉄の部品を製作すると、錆により部品と部品が付着してしまう。この錆により、部品は回転しなくなってしまうと考えたからである。

また、完成予想図は、片側にロータリーエンコーダを取り付け、もう片方に姿勢を保持する装置を取り付けているが、組み立てたときに小型模型の荷重バランスが悪く

なってしまったため、ロータリーエンコーダの取り付け位置を変更し、荷重バランスが左右対称になるようにした。また、姿勢を保持する装置のばねが入っている部品は軸に固定し、突起物の付いている部品は小型模型に固定し、姿勢を保持する装置が回転しなくなるようにした。

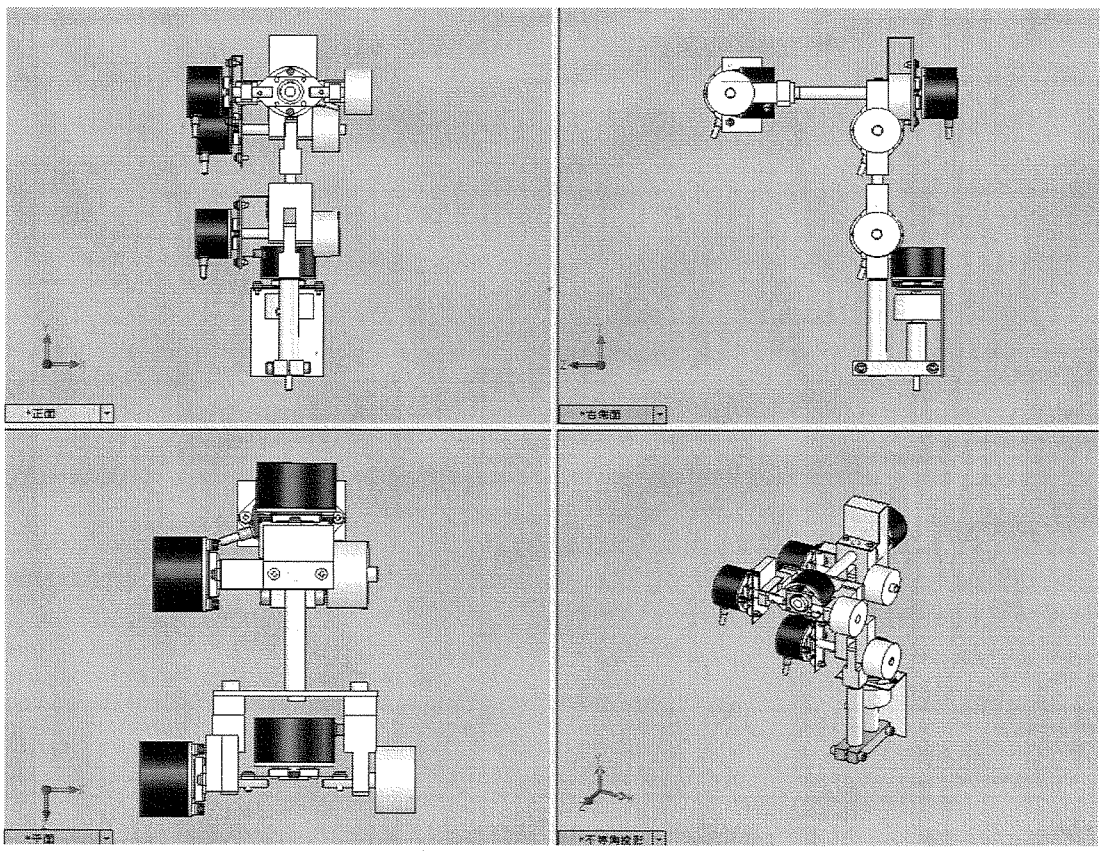


図 3.3　小型模型の完成予想図

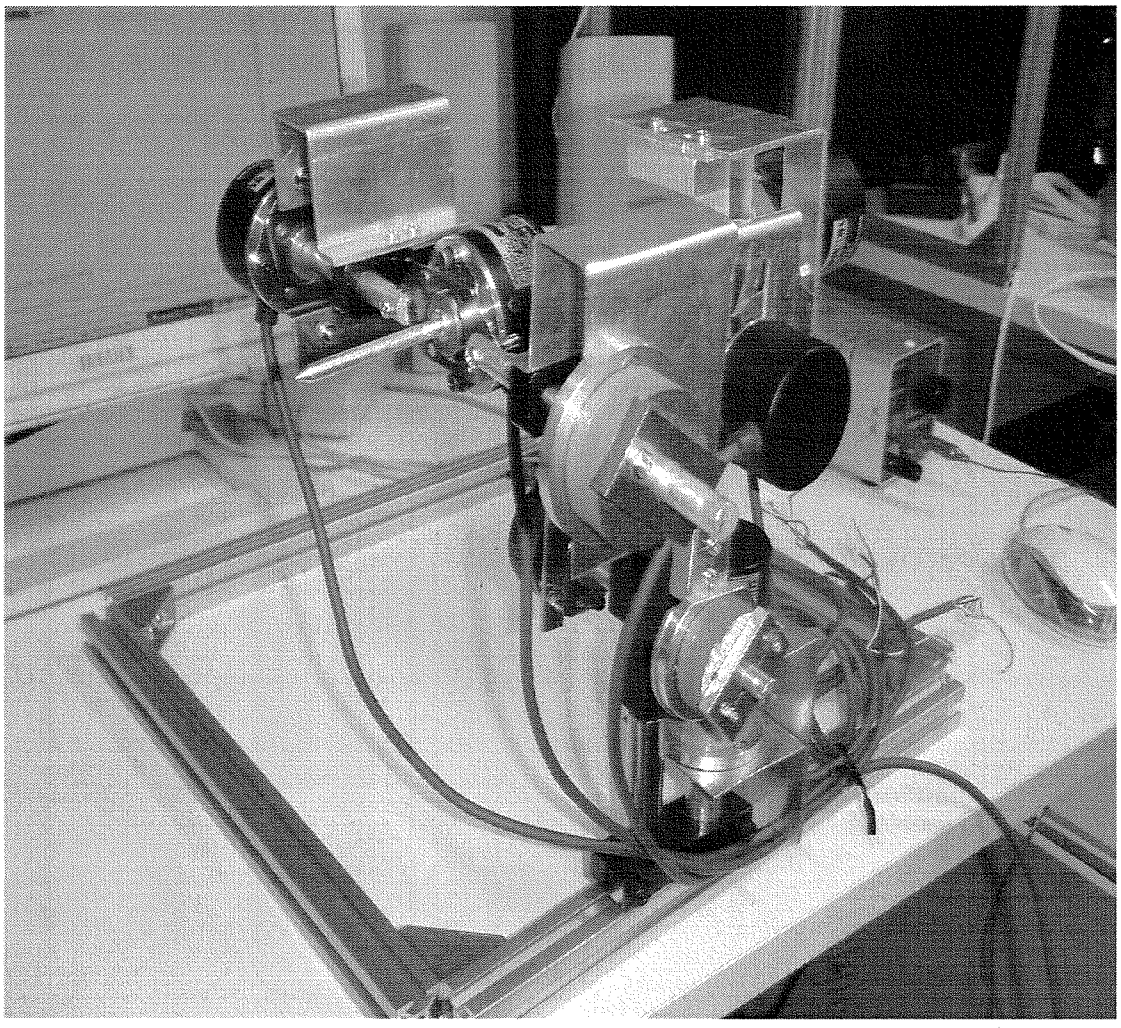


図 3.4　作製した小型模型

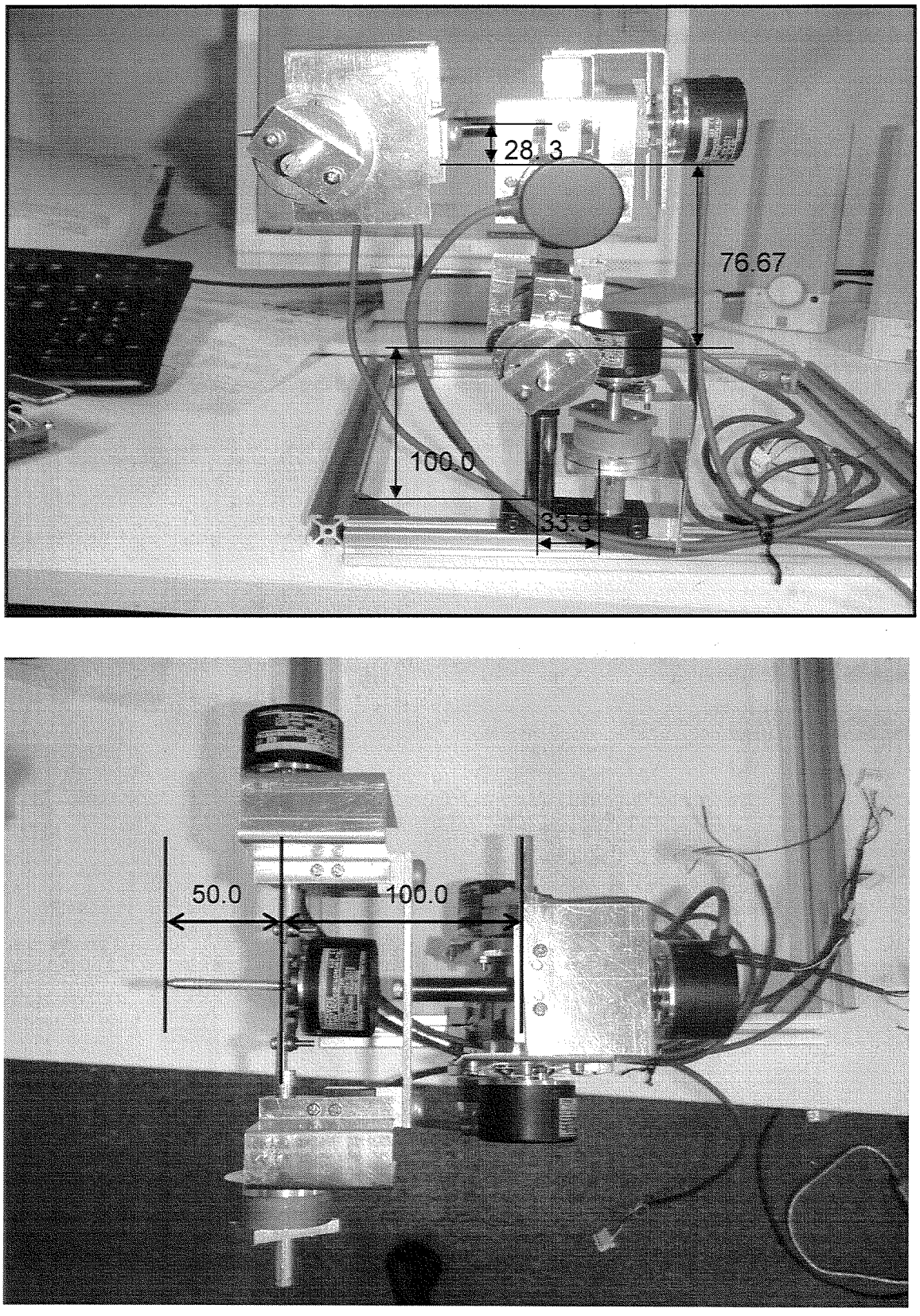


図 3.5　小型模型の寸法

4．　PSoC マイコン

従来と同様に PIC を用いてカウントや、通信を行おうとしたが SPI 通信に不向きであった。そこで、今回は SPI 通信を行えるように PSoC マイコンを用いることにした。本研究では Cypress 社製の PSoC を使用した。この PSoC マイコンの主な特性は以下に書いてあることである。

- ユーザーが手元でいつでも内部構成を変更可能なワンチップ・マイコン
- 構成変更は CPU からのレジスタ設定で行われるので、動作中にダイナミックに内部構成を変更することも可能である。
- A-D コンバータ、D-A コンバータ、フィルタ、タイマ、アンプ、カウンタといったものを選択して目的に応じたワンチップ・マイコンとして使うことも可能である。
- ブロック間を直接接続するための信号ラインも用意されている。
- PSoC Designer を使用して、積み木細工のようにチップを作り上げる感覚や、自動的に生成された API を呼ぶだけで手軽にできる。
- ブロック図にはカウンタやシリアル・ポートなどといったものがなく、その代わりにディジタル・ブロック・アレイ、アナログ・ブロック・アレイがありそれが PSoC の心臓部にあたるところである。
- ２種類の PSoC ブロックアレイによって、今まで外部回路で行わざるを得なかったようなアナログ処理も可能になった。

５．　インクリメンタル形ロータリーエンコーダの回転判別

図 5.1 にインクリメンタル形ロータリーエンコーダの出力波形（時計回り方向）を示す。図 5.1 のようにＡ相とＢ相の出力波形がある。Ａ相とＢ相は互いに半パルスだけずれた波形が出力される。エンコーダを軸側から見て時計回りに回転させたときの場合は A 相のパルスが先に出て、半パルスだけ遅れた B 相のパルスがあとから出力される。また、反時計回りに回転させた場合はＢ相のパルスが先に出て後からＡ相のパルスが出力される。この関係を利用してロータリーエンコーダの回転判別を行う。

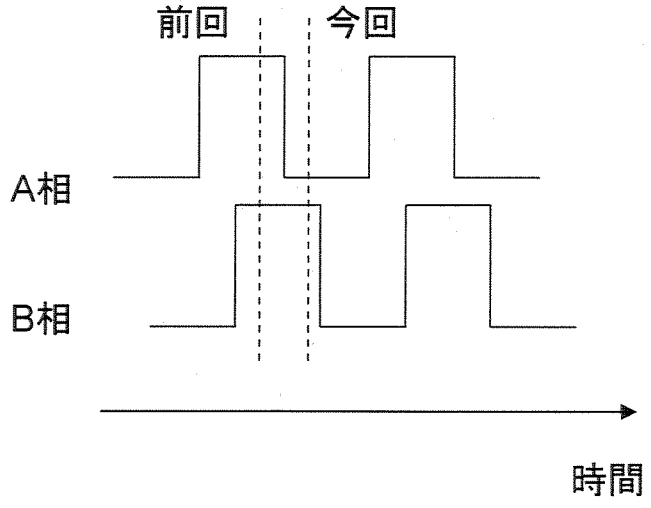


図 5.1　　出力波形（時計回り）

インクメンタル形ロータリーエンコーダの回転方向判別するためのフローチャートを図に示す。前回と今回のＡ相、Ｂ相の値で回転方向が判別することがわかる。

また、Ａ相、Ｂ相の立上り、立下りを数え、反時計回りのときに１増加し、時計回りのときに１減少するプログラムを作成した。また、Ｚ相が high になったときにカウント数がリセットされるようにプログラムを改良した。プログラムのフローチャートを図 5.2 に示す。

このロータリーエンコーダの分解能は 3600 である。そして、ロータリーエンコーダのＡ相とＢ相の立ち上がりと立下りを数えるプログラムであるため、0 から 14399 までカウント

するプログラムにした。プログラムのフローチャートを図 5.3 に示す。ロータリーエンコーダの回転判別、カウントを行うプログラムは付録のプログラムリスト 2 に示す。

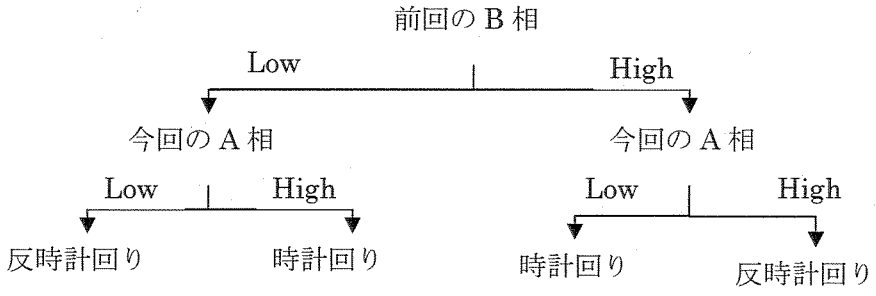


図 5.2　回転方向判別のフローチャート

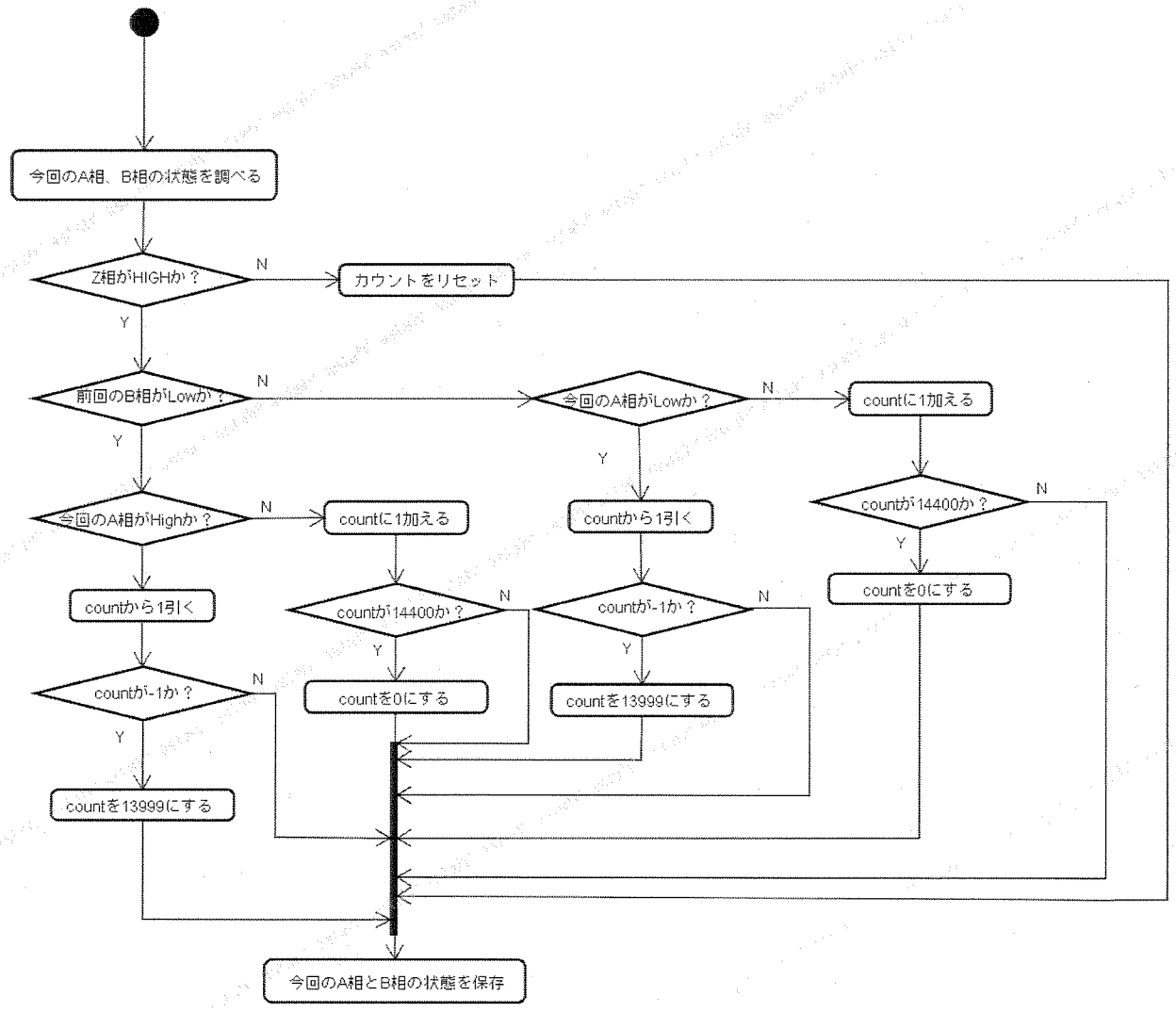


図 5.3　カウントのプログラムのフローチャート

6．　データ伝送方式

図 6.1 に従来のデータ伝送方式を示す。従来の伝送方式では、高分解能化に対応するためには、チャンネル数の多いデジタル I/O インターフェースに付け替える必要があった。また、図では示していないが、１つの PIC とデジタル I/O インターフェース間には伝送するビット分の線が必要になってくる。そのため、高分解能化を行うと、送信データのビット数が多くなりそのビット数分、配線も多くなるためメンテナンスしにくいという問題点があった。従来のロータリーエンコーダの分解能は 1000 であるため 0 から 3999 までカウントするので、ビット数は 12 だった。従来の伝送方式のまま本研究の分解能にすると、16 ビットになるためメンテナンスすることが更に困難になる。

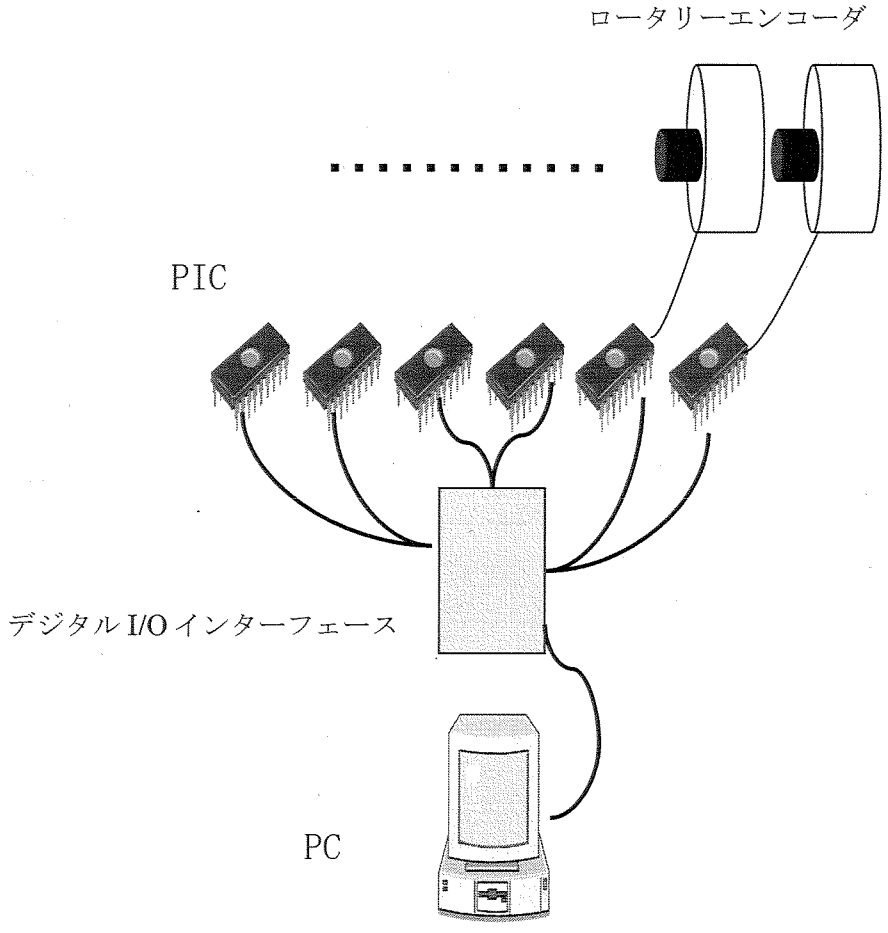


図 6.1　従来のシステムのデータ伝送方法

本研究の伝送方式はデジタル I/O インターフェースは使用せず、SPI 通信を行った。図 6.2 に本研究のデータ伝送方法を示す。SPI 通信だとビット数が増えても各マイコン間の配線は４本で済み、メンテナンスも従来に比べ容易になった。また、各 PSoC マイコンでロータリーエンコーダのカウントと PC までの伝送を行おうとすると、カウントミスを起こす可能性があるため１つの PSoC マイコンを介してから PC までデータを伝送することにした。今回のロータリーエンコーダのカウント数のデータは 16 ビット分あり SPI 通信では１度に 8 ビットまでしか送れないため、上位ビットと下位ビットに分けてデータを伝送するようにした。また、SPI 通信によってデータを受け取ったものをすぐに PC まで伝送するようにした。本研究のシステムのデータ伝送は各ロータリーエンコーダのカウント数を SPI 通信で行い、１つの PSoC マイコンを介し、RS-232C を用いて PC までデータを伝送することにした。

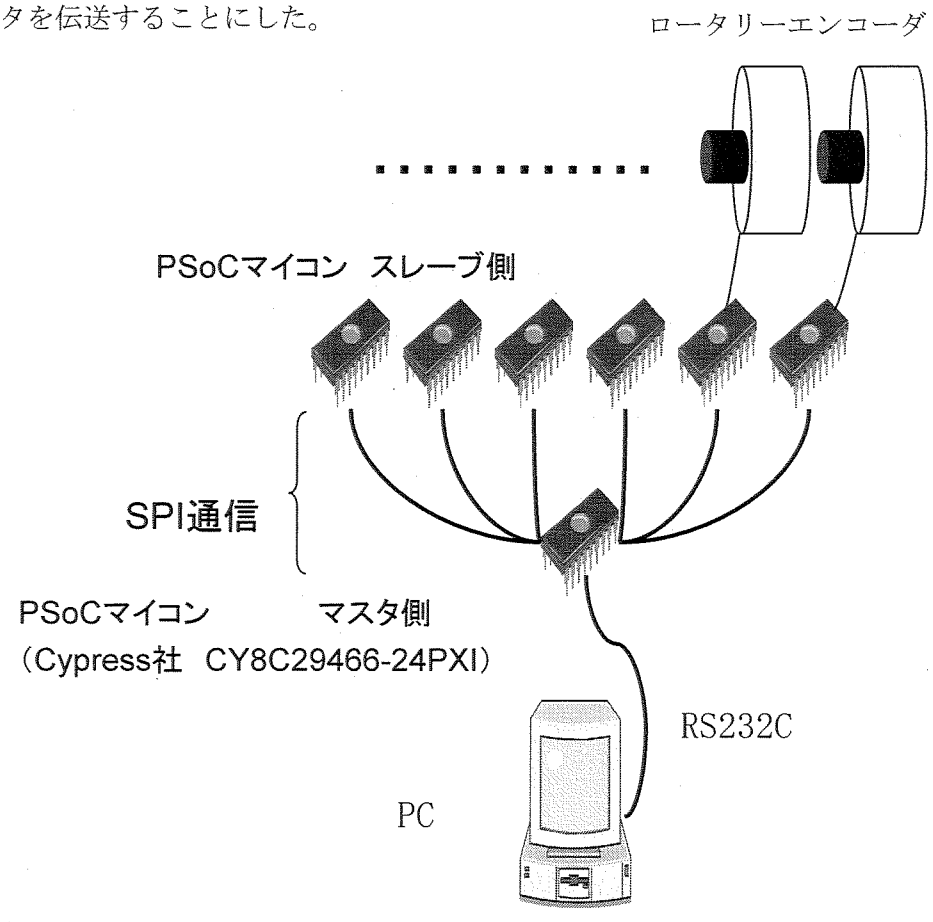


図 6.2　本研究のデータ伝送方法

６．１　SPI 通信（Serial Peripheral Interface）

６．１．１　SPI 通信とは

SPI 通信とは、同期式のシリアル通信の規格で、配線は４本で比較的速度を要する用途にしようされます。図 6.3 に SPI 通信の基本構成を示す。マスタ側とスレーブ側の間には SDI、SDO、SCK の３本とスレーブが複数接続する場合は SS を配線する。マスタ側が SCK を生成するが、このクロックは SCK 信号として共有されるので送信と受信はこのクロックによって同時に行われる。SS 信号は low で選択されるので複数のスレーブの中からマスタが選択して選択したスレーブと通信することができる。基本構成では、３つ以上スレーブを接続すると SPI 通信がうまく動作されなかった。本研究では、各関節に取り付けたロータリーエンコーダのカウント数を PSoC 間で SPI 通信を行い、１つの PSoC にデータをまとめた。そのまとめられたデータを送るときは下位ビットと上位ビットにわけて送るようにした。それぞれの下位ビットを送ってから上位ビットを送るようにした。

また、SPI 通信の配線方法をデイジーチェーン接続にした。デイジーチェーン接続は機器を数珠繋ぎに接続し、データ転送をバケツリレー式に転送する。また、図 6.4 にデイジーチェーン接続を用いた SPI 通信の配線図を示す。図 6.5 にスレーブ側の、図 6.6 にマスタ側のプログラムのフローチャートを示す。マスタ側のプログラムを付録のプログラムリスト１に示し、スレーブ側のプログラムをプログラムリスト２に示す。

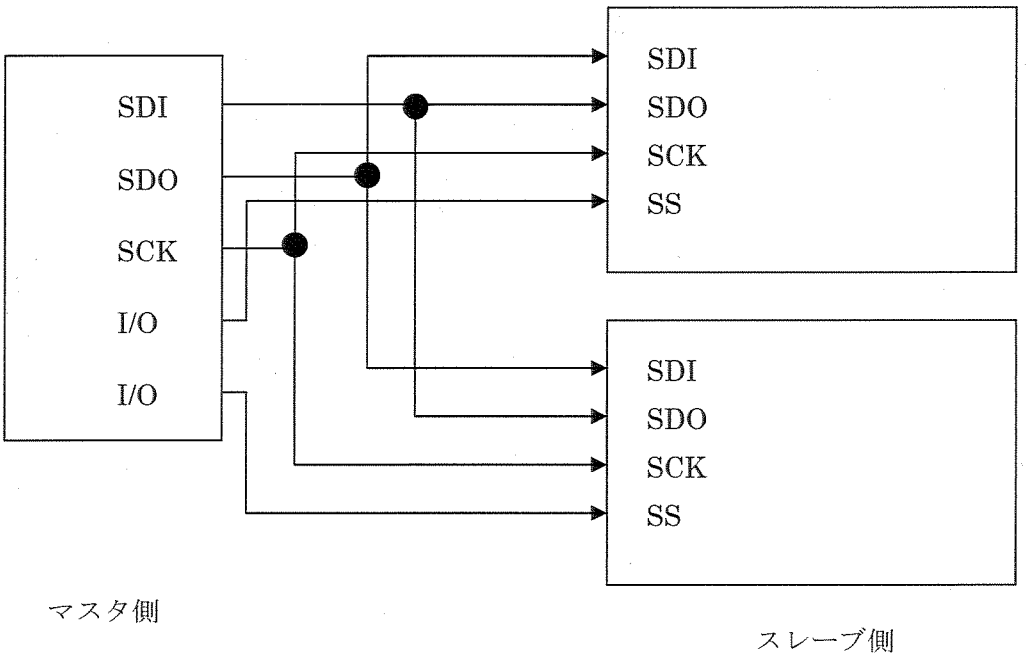


図 6.3　SPI 通信の基本構成

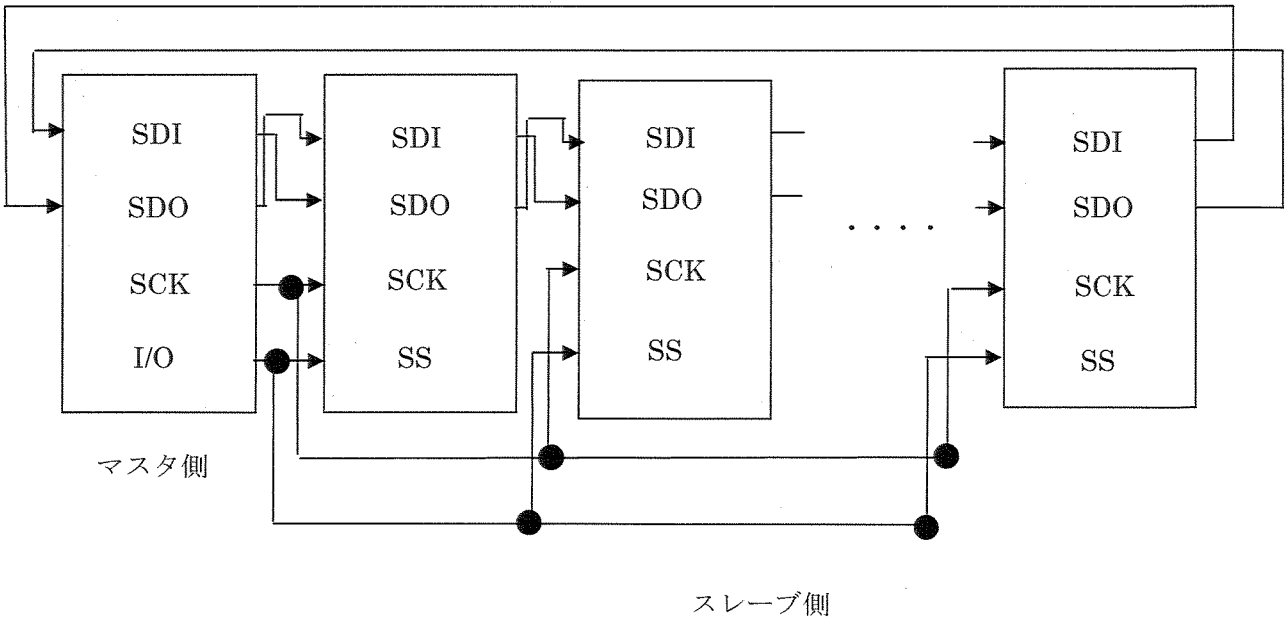


図 6.4　デイジーチェーン接続による SPI 通信の配線図

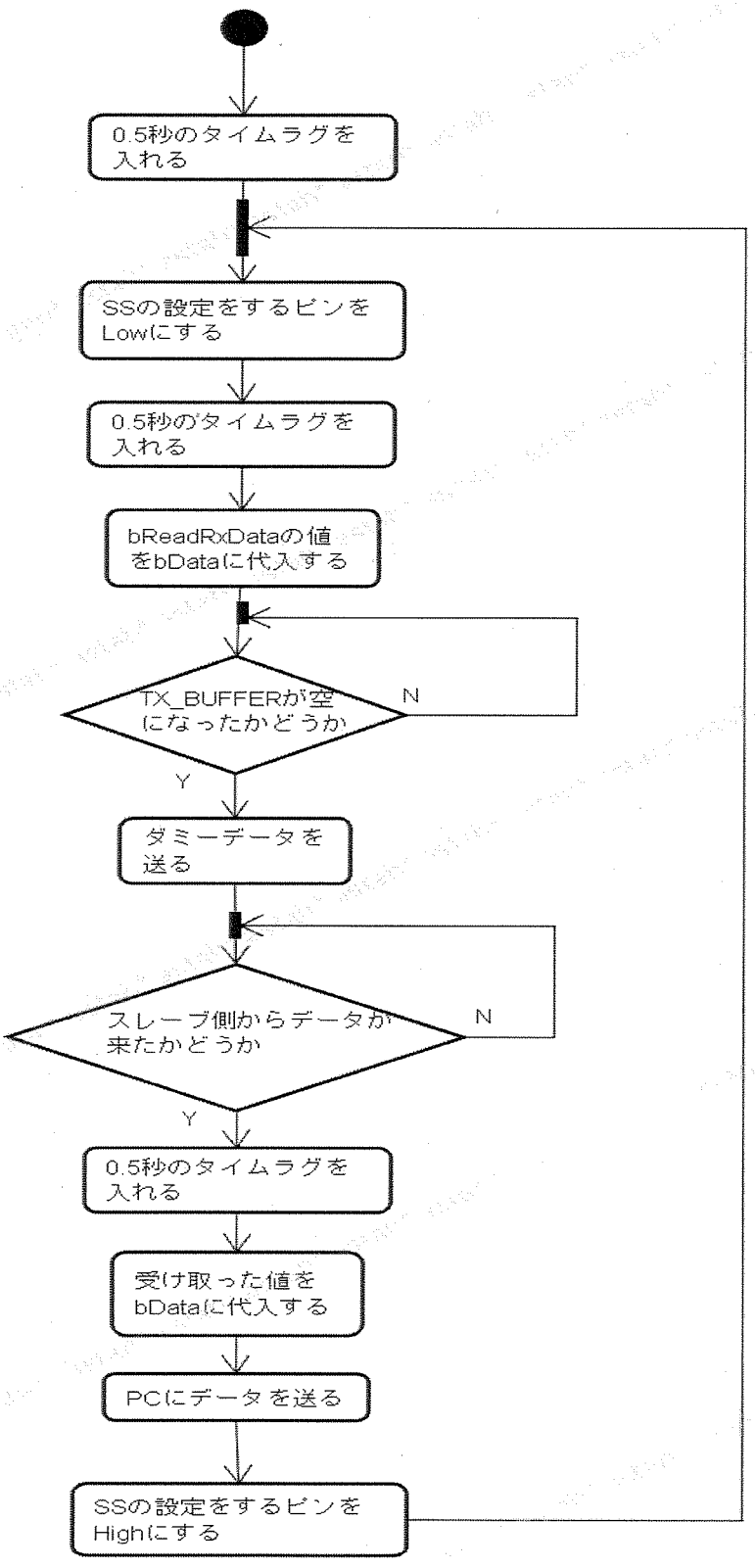


図 6.5　マスタ側のプログラムのフローチャート

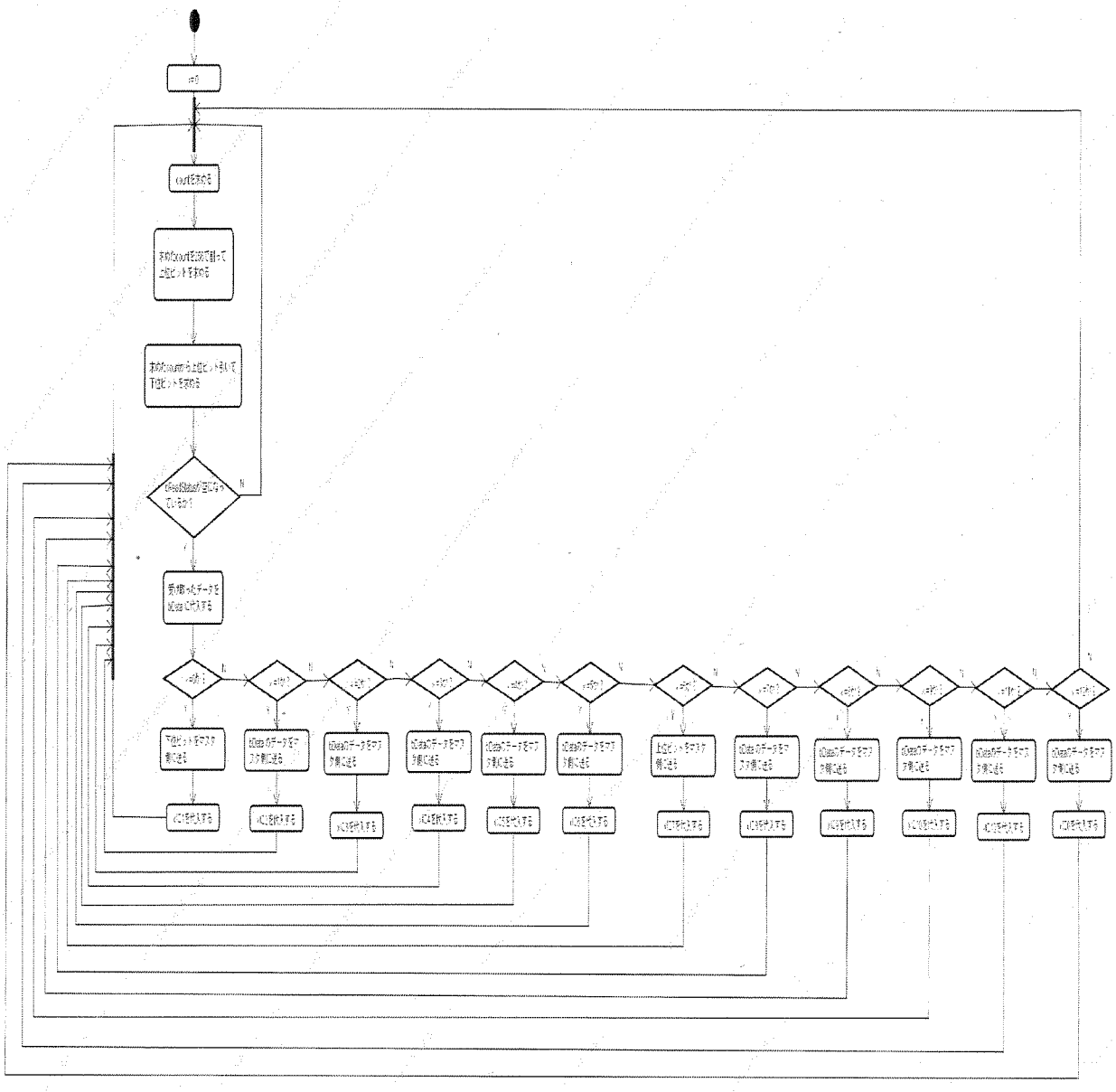


図 6.6　スレーブ側のプログラムのフローチャート

６．１．２　動作確認

SPI 通信の動作確認を行った。まず、マスタ、スレーブともに１つで SPI 通信を行った。マスタ側にはLCD 表示器をつけスレーブ側から送られてくるデータが確認できるようにした。マスタもスレーブも送るデータを自分で決め、SPI 通信を行い、マスタの LCD 表示器で自分が決めたデータと表示されているデータが一致するかを確認した。最初の方は正常に SPI 通信されてないことがあったので、ラトックシステム社製の REX-USB61 を使用して、スレーブだけの動作確認を行った。マスタとスレーブ１つでの動作確認したあと、スレーブを１つずつ増やしていき SPI 通信を行い、表示器で送られてくるデータを確認した。

自分で決めたデータでの SPI 通信の動作を確認したあとは、ロータリーエンコーダのカウント数をデータにして、SPI 通信を行い、表示器でデータを確認した。ロータリーエンコーダの場合は、正確なでデータの値がわからないので、ロータリーエンコーダを半分まわして、値も半分になるかどうかを表示器で確認した。ロータリーエンコーダの値は 16 ビットなので、上位ビットと下位ビットにわけデータを伝送した。また、複数スレーブがある場合は、それぞれのロータリーエンコーダの下位ビットを送ってから、上位ビットを送るようにし、表示器もその順番で表示させるようにした。これらにより、マスタ１つ、スレーブ６つで SPI 通信できていることを確認することができた。

6．1．3　REX-USB61

SPI 通信の動作確認を行う際に、スレーブ側の動作だけを確認するときに使用した。REX-USB61 はマスタ側の代わりになり、自分で送るデータを決め、送信することで PC 上にスレーブから受け取ったデータを表示する。これらによって、スレーブ側が正常に動作されているか確認することができる。表 6.1 に REX-USB61 の仕様を示す。また、図 6.7 に REX-USB61 の外観を示す。

表 6.1　REX-USB61 の仕様

インターフェース	USB2.0 Full Speed Device
接続コネクタ	USB mini B コネクタ
電源電圧	5V(USB バスパワーから取得)
消費電流	100mA
サポートインターフェース	SPI マスタ　最大周波数 12MHz
外部入出力レベル	3.3V/5V
外形寸法	57(W) × 75(D) × 18(H)mm
重量	約 60g

図 6.7　REX-USB61 の外観

６．２　RS—232C

６．２．１　RS-232C とは

本研究ではデータを集約した１つの PSoC から RS－232C を介して PC にデータを伝送した。RS－232C とは米国電子工業会によって標準化された、シリアル通信の１つでシリアル通信方式としては最も普及されている。規格としては、通信ケーブルの最大長は 15m で、最高通信速度は 115.2kbps である。RS-232C の一般的な通信速度は 9600bps である。本研究では通信速度を 19200bps にした。これは、マスタ側の PSoC マイコンのクロックの関係から通信速度を決定したためである。PSoC からパソコンが受け取ったデータを Tera Term で表示させるようにした。PC までデータを伝送するプログラムは付録のプログラムリスト 1 に示す。

シリアル通信を行う際には、RS-232C から USB コネクタに変換するケーブルを用いた。本研究で用いた RS-232C を USB コネクタに変換するケーブルを次の図６．８に示す。

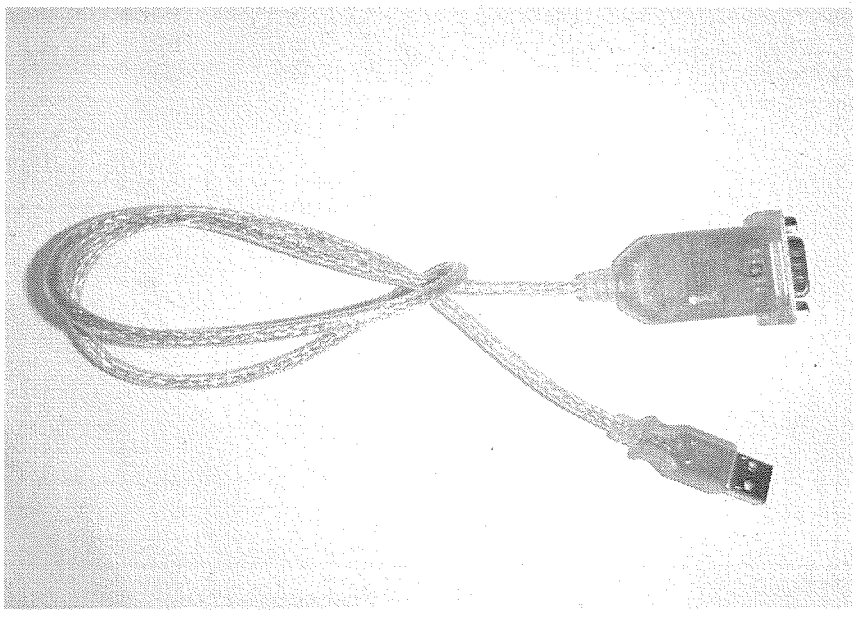


図 6.8 シリアル通信で用いた変換ケーブル

６．２．２　動作確認

PC までの伝送の動作確認を行った。PSoC 間で SPI 通信を行い、データを RS-232C を介して PC までデータを伝送した。PC では送られてきたデータを Tera Term で表示させるようにした。表示方法はどのロータリーエンコーダの値で下位ビットか上位ビットかわかるように表示させるようにした。ロータリーエンコーダの根元から順にそれぞれ a,b,c,d,e,f と置いた。例えば、a のロータリーエンコーダの送信データは下位ビットでは aL - □□と表示させ、上位ビットでは aH - □□と表示させるようにした。通信時のデータは a から f までの下位ビットを先に送り、次に a から f までの上位ビットを送信し、最後に Enter を送信するようにした。マスタ側の PSoC は SPI 通信によって受け取ったデータを直ちに PC まで伝送するようなプログラムにした。また、マスタ側の PSoC マイコンには、表示器を接続しておき、SPI 通信によって受け取ったデータを表示するようにしておき、表示器に表示されている値と PC 上に表示されている値が一致しているか確認できるようにした。これらにより、PC までの伝送の動作確認することができた。

7．　3次元座標の導出

PSoCから受け取ったデータは上位ビットと下位ビットに分かれて受け取るので、上位ビットと下位ビットを合わせて1つのデータにする必要がある。入力したデータは文字列で入力されるためカウント数の値のみを抽出し各ジョイントごとに回転カウントの文字列にする。その文字列のデータは16進数であるのでそれを10進数に直し、その各ロータリーエンコーダのカウント数を角度に直し、角度情報とリンクの長さから模型の先端座標を求めるプログラムを作成した。算出方法としては小型模型の先端から同時変換行列で座標系を変換させながら順にロボット機構の原点の方へ移動させ、最終的にはロボット機構の原点座標系での先端の座標を求めるようにした。図7.1に小型模型の座標の取り方について示す。同時変換行列について以下の式に示す。また図7.2にプログラムのフローチャートを示す。先端座標を求めるプログラムを付録のプログラムリスト3に示す。

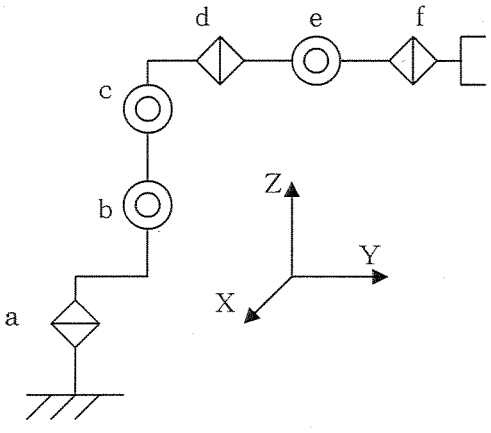


図7.1　小型模型の座標の取り方

各回転方向の回転行列は以下に示すものである。

X方向の回転による回転行列は、

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}$$

となる。

Y方向の回転による回転行列は、

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}$$

となる。

Z方向の回転による回転行列は、

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}$$

となる。

また、平行移動の行列は、

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & l_x \\ 0 & 1 & 0 & l_y \\ 0 & 0 & 1 & l_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}$$

となる。

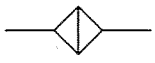
この回転行列と平行移動の行列の和が同時変換行列となる。

X 方向、Y 方向、Z 方向で回転する場合のそれぞれの同時変換行列を以下に示す。

X方向の回転は以下のように考える。

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & l_x \\ 0 & 1 & 0 & l_y \\ -\sin\theta & 0 & \cos\theta & l_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}$$

Y方向の回転は以下のように考える。



$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & l_x \\ \sin\theta & \cos\theta & 0 & l_y \\ 0 & 0 & 1 & l_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}$$

Z 方向の回転は以下のように考える。

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & l_x \\ 0 & \cos\theta & -\sin\theta & l_y \\ 0 & \sin\theta & \cos\theta & l_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}$$

これらの式を回転方向ごとに使うことで、(x_0, y_0, z_0)の座標系から X 方向にl_x、Y 方向にl_y、Z 方向にl_z平行移動させ、$\cos\theta$と$\sin\theta$の二つの三角関数によって回転させてできた新しい座標系(x_1, y_1, z_1)に変換することができる。そして、次の関節へ移動させる場合は先程算出した座標を(x_0, y_0, z_0)に代入し座標を計算する。

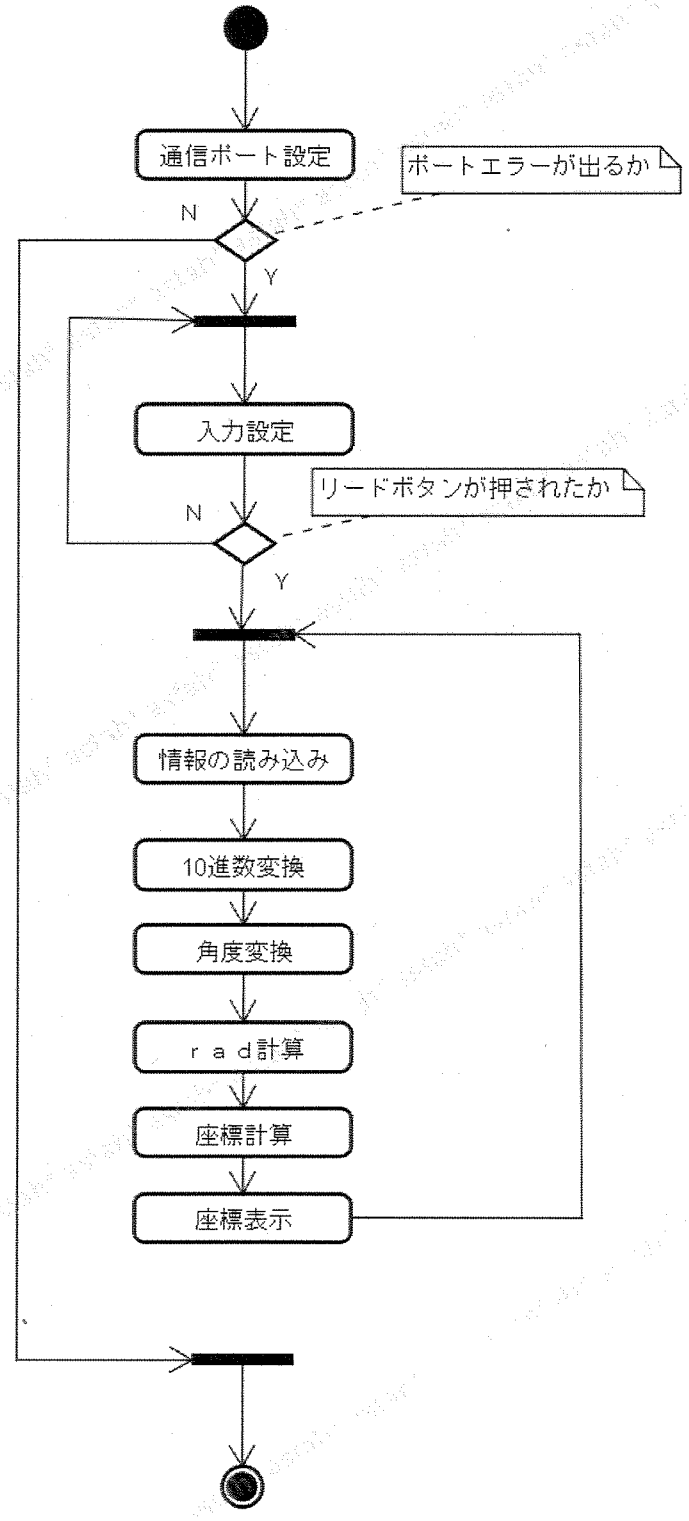


図 7.2　プログラムのフローチャート

８．　LabWindows/CVI

８．１　LabWindows/CVI とは

今回使用したプログラミングツールは LabWindows/CVI である。その LabWindows/CVI はテスト/制御アプリケーションを作成するためのプログラミングツールである。また、利用できる言語は ANSI C であり、その言語を用いて計測器制御、データ集録、解析、ユーザインタフェース開発などを行うソフトである。このソフトはオブジェクト指向プログラミングによる統合開発環境である。オブジェクト指向プログラミングとは、オブジェクトと呼ばれる機能の部品や関数でソフトウェアを構成させるものである。このオブジェクト指向プログラミングを用いた総合開発環境は Visual C++.NET などがある。今回は、シリアル通信をすることができる LabWindows/CVI を使用することにした。

本研究では、ユーザインタフェースに x , y , z の座標を表示するためのテキストボックスを設けた。このテキストボックスは float 型の変数を小数点以下第二位まで表示するような設定とした。また、変換ケーブルをつないでいるポート設定や通信速度の設定を行えるシステムにしてある。このときシリアル・ポートが繋がっていなければエラーを表示させ終了するようなシステムにしてある。そして、シリアル通信によって得られた情報を表示するようなテキストボックスも設けた。それは、受信データと PSoC の送信データに矛盾が生じないようにするためである。また、送信側がデータを送り続けるため受信も無限ループでを用いて受信を続けるようにした。そのため、この LabWindows/CVI はオブジェクト指向プログラミングなので受信部分のオブジェクトにある while 文から抜けることができなくなる。そのため、終了ボタンをクリックしても終了することができなくなった。改善点として、この while 文ではなく、時間制限を設けその時間を越えてしまうと終了するようにしなければならないと考えられる。

８．２　動作確認

はじめに、Tera Term を用いて座標計算に必要な各ロータリーエンコーダの角度情報として 0 を 60 文字と改行の 2 文字を送信する。そこで、小型模型の基本姿勢における先端の座標と一致するかを確認した。基本指定における先端座標と計算結果が一致しなかった。これは、リンクの長さの設定や正常にカウント数から角度に変換していなかったからであるといえる。これらの設定やプログラムの変更により小型模型の基本姿勢における先端の座標と一致させることができた。

次に、基盤とロータリーエンコーダを接続し、基盤とＰＣを RS-232C で接続した。プログラムを実行して、結果が正しく表示されるか確認した。また、基盤には LDC 表示器により各ロータリーエンコーダのカウントを表示させることができ、その表示する値と PC が受信したデータが一致するかを確認した。その座標計算の結果が示す位置と小型模型の先端の位置が一致するかを確認した。これらにより、ロータリーエンコーダのカウント数から座標計算とシリアル通信が正常に動作するかを確認することができた。

9．まとめ

小型模型を製作し、手を離しても姿勢を保持するようにばねを使った姿勢を保持する装置を小型模型に取り付けた。ロータリーエンコーダの回転判別をし、立上りと立下りをカウントするプログラムを作成した。また、各ロータリーエンコーダのカウント数を SPI 通信によって１つの PSoC マイコンに伝送した。SPI 通信を行った後に、RS-232C を介して PC までロータリーエンコーダのカウント数を伝送した。PC では、受け取ったデータのカウント数から各ロータリーエンコーダの回転角度を求め、その回転角度、リンクの長さを用いて小型模型の先端座標を計算するプログラムを作成した。

今後の課題として、座標計算の結果が示す位置と小型模型の先端位置が一致することを検証する。その検証方法としては、小型模型を四角の枠に設置し、その枠には一定間隔で穴を開けておく。小型模型の先端にボールのようなものをつけそのボールを枠に開いている穴にいれる。穴の位置は自分で決めているので、ボールを穴に入れたときに PC 上で表示されている値と実際に小型模型の先端座標の誤差を検証する。

１０．付録

プログラムリスト１

SPI 通信　マスタ用プログラム

```
#include <m8c.h>
#include "PSoCAPI.h"

void main()
{
 char Message1[] = "H";
 float i;
 BYTE  bData;
 LCD_1_Start();
 PRT2DR = 0b11111111;
 SPIM_1_Start(SPIM_1_SPIM_MODE_0 | SPIM_1_SPIM_MSB_FIRST);
 UART_1_Start(UART_1_PARITY_NONE);
 UART_1_CmdReset();
            for(i = 0; i < 600; i++)
            {

            }
            while(1)
            {
           PRT2DR = 0b11111110;

            for(i = 0; i < 600; i++)
            {

            }

            bData = SPIM_1_bReadRxData();
           while( ! (SPIM_1_bReadStatus() &
                       SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
           SPIM_1_SendTxData( 0x41 );
            while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
            for(i = 0; i < 300; i++)
            {
```

```
 }
 bData = SPIM_1_bReadRxData();
 LCD_1_Position(0,2);
 LCD_1_PrHexByte(bData);
 UART_1_PutChar(0x61);
 UART_1_PutChar(0x4C);
 UART_1_PutChar(0xB0);
 UART_1_PutSHexByte(bData);

 PRT2DR = 0b11111111;

 PRT2DR = 0b11111110;
 for(i = 0; i < 600; i++)
 {

 }

 bData = SPIM_1_bReadRxData();
while( ! (SPIM_1_bReadStatus() &
            SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
SPIM_1_SendTxData( 0x41 );
 while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
 for(i = 0; i < 300; i++)
 {

 }
 bData = SPIM_1_bReadRxData();
 LCD_1_Position(0,7);
 LCD_1_PrHexByte(bData);
 UART_1_PutChar(0x62);
 UART_1_PutChar(0x4C);
 UART_1_PutChar(0xB0);
 UART_1_PutSHexByte(bData);

 PRT2DR = 0b11111111;
```

```
PRT2DR = 0b11111110;

        for(i = 0; i < 600; i++)
        {

        }

        bData = SPIM_1_bReadRxData();
       while( ! (SPIM_1_bReadStatus() &
                   SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
       SPIM_1_SendTxData( 0x41 );
        while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
        for(i = 0; i < 300; i++)
        {

        }
        bData = SPIM_1_bReadRxData();
        LCD_1_Position(0,12);
        LCD_1_PrHexByte(bData);
        UART_1_PutChar(0x63);
        UART_1_PutChar(0x4C);
        UART_1_PutChar(0xB0);
        UART_1_PutSHexByte(bData);

        PRT2DR = 0b11111111;

        PRT2DR = 0b11111110;
        for(i = 0; i < 600; i++)
        {

        }

        bData = SPIM_1_bReadRxData();
       while( ! (SPIM_1_bReadStatus() &
                   SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
```

```
SPIM_1_SendTxData( 0x41 );
 while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
 for(i = 0; i < 300; i++)
 {

 }
 bData = SPIM_1_bReadRxData();
 LCD_1_Position(1,2);
 LCD_1_PrHexByte(bData);
 UART_1_PutChar(0x64);
 UART_1_PutChar(0x4C);
 UART_1_PutChar(0xB0);
 UART_1_PutSHexByte(bData);

 PRT2DR = 0b11111111;

 PRT2DR = 0b11111110;

 for(i = 0; i < 600; i++)
 {

 }

 bData = SPIM_1_bReadRxData();
while( ! (SPIM_1_bReadStatus() &
            SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
SPIM_1_SendTxData( 0x41 );
 while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
 for(i = 0; i < 300; i++)
 {

 }
 bData = SPIM_1_bReadRxData();
 LCD_1_Position(1,7);
 LCD_1_PrHexByte(bData);
 UART_1_PutChar(0x65);
```

```
UART_1_PutChar(0x4C);
UART_1_PutChar(0xB0);
UART_1_PutSHexByte(bData);

PRT2DR = 0b11111111;

PRT2DR = 0b11111110;
for(i = 0; i < 600; i++)
{

}

bData = SPIM_1_bReadRxData();
while( ! (SPIM_1_bReadStatus() &
            SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
SPIM_1_SendTxData( 0x41 );
while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
for(i = 0; i < 300; i++)
{

}
bData = SPIM_1_bReadRxData();
LCD_1_Position(1,12);
LCD_1_PrHexByte(bData);
UART_1_PutChar(0x66);
UART_1_PutChar(0x4C);
UART_1_PutChar(0xB0);
UART_1_PutSHexByte(bData);

PRT2DR = 0b11111111;

PRT2DR = 0b11111110;

for(i = 0; i < 600; i++)
{
```

```
}

bData = SPIM_1_bReadRxData();
while( ! (SPIM_1_bReadStatus() &
            SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
SPIM_1_SendTxData( 0x41 );
while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
for(i = 0; i < 300; i++)
{

}
bData = SPIM_1_bReadRxData();
LCD_1_Position(0,0);
LCD_1_PrHexByte(bData);
UART_1_PutChar(0x61);
UART_1_PutChar(0x48);
UART_1_PutChar(0xB0);
UART_1_PutSHexByte(bData);

PRT2DR = 0b11111111;

PRT2DR = 0b11111110;
for(i = 0; i < 600; i++)
{

}

bData = SPIM_1_bReadRxData();
while( ! (SPIM_1_bReadStatus() &
            SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
SPIM_1_SendTxData( 0x41 );
while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
for(i = 0; i < 300; i++)
{

}
```

```
bData = SPIM_1_bReadRxData();
LCD_1_Position(0,5);
LCD_1_PrHexByte(bData);
UART_1_PutChar(0x62);
UART_1_PutChar(0x48);
UART_1_PutChar(0xB0);
UART_1_PutSHexByte(bData);

PRT2DR = 0b11111111;

PRT2DR = 0b11111110;

for(i = 0; i < 600; i++)
{

}

bData = SPIM_1_bReadRxData();
while( ! (SPIM_1_bReadStatus() &
            SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
SPIM_1_SendTxData( 0x41 );
while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
for(i = 0; i < 300; i++)
{

}
bData = SPIM_1_bReadRxData();
LCD_1_Position(0,10);
LCD_1_PrHexByte(bData);
UART_1_PutChar(0x63);
UART_1_PutChar(0x48);
UART_1_PutChar(0xB0);
UART_1_PutSHexByte(bData);

PRT2DR = 0b11111111;
```

```
 PRT2DR = 0b11111110;
 for(i = 0; i < 600; i++)
 {

 }

 bData = SPIM_1_bReadRxData();
while( ! (SPIM_1_bReadStatus() &
            SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
SPIM_1_SendTxData( 0x41 );
 while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
 for(i = 0; i < 300; i++)
 {

 }
 bData = SPIM_1_bReadRxData();
 LCD_1_Position(1,0);
 LCD_1_PrHexByte(bData);
 UART_1_PutChar(0x64);
 UART_1_PutChar(0x48);
 UART_1_PutChar(0xB0);
 UART_1_PutSHexByte(bData);

 PRT2DR = 0b11111111;

 PRT2DR = 0b11111110;

 for(i = 0; i < 600; i++)
 {

 }

 bData = SPIM_1_bReadRxData();
while( ! (SPIM_1_bReadStatus() &
            SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
SPIM_1_SendTxData( 0x41 );
```

```
while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
for(i = 0; i < 300; i++)
{

}
bData = SPIM_1_bReadRxData();
LCD_1_Position(1,5);
LCD_1_PrHexByte(bData);
UART_1_PutChar(0x65);
UART_1_PutChar(0x48);
UART_1_PutChar(0xB0);
UART_1_PutSHexByte(bData);

PRT2DR = 0b11111111;

PRT2DR = 0b11111110;
for(i = 0; i < 600; i++)
{

}

bData = SPIM_1_bReadRxData();
while( ! (SPIM_1_bReadStatus() &
            SPIM_1_SPIM_TX_BUFFER_EMPTY  ) );
SPIM_1_SendTxData( 0x41 );
while( ! (SPIM_1_SPIM_RX_BUFFER_FULL ));
for(i = 0; i < 300; i++)
{

}
bData = SPIM_1_bReadRxData();
LCD_1_Position(1,10);
LCD_1_PrHexByte(bData);
UART_1_PutChar(0x66);
UART_1_PutChar(0x48);
UART_1_PutChar(0xB0);
```

```
                UART_1_PutSHexByte(bData);
                UART_1_PutCRLF();

                PRT2DR = 0b11111111;
                }
}
```

プログラムリスト2

SPI 通信　スレーブ用プログラム

```
#include <m8c.h>
#include "PSoCAPI.h"

int KEISAN(void), x;
char A, B, C, D, Z;
unsigned int count;

void main(void)
{
    int i;
    char theStr[] = "PSoC LCD";
    BYTE  bData, count1, count2, a;
    SPIS_1_Start(SPIS_1_SPIS_MODE_0);

    x = 0;
    while(1)
    {
            C = PRT0DR & 0b00100000 ;          //A'Š,Ì"Ç,Ý Ž,Ý
            D = PRT0DR & 0b01000000 ;          //B'Š,Ì"Ç,Ý Ž,Ý
            count = KEISAN();
            count1 = count / 256;
            count2 = (count - ((count1)*256));

           if( SPIS_1_bReadStatus() & SPIS_1_SPIS_SPI_COMPLETE )
            {
                  bData = SPIS_1_bReadRxData();
                  switch(x)
```

```
{
        case 0:
       SPIS_1_SetupTxData(count2);
        x = 1;
        break;

        case 1:
        SPIS_1_SetupTxData(bData);
        x = 2;
        break;

        case 2:
        SPIS_1_SetupTxData(bData);
        x = 3;
        break;

        case 3:
        SPIS_1_SetupTxData(bData);
        x = 4;
        break;

        case 4:
        SPIS_1_SetupTxData(bData);
        x = 5;
        break;

        case 5:
        SPIS_1_SetupTxData(bData);
        x = 6;
        break;

        case 6:
        SPIS_1_SetupTxData(count1);
        x = 7;
        break;
```

```
                        case 7:
                        SPIS_1_SetupTxData(bData);
                        x = 8;
                        break;

                        case 8:
                        SPIS_1_SetupTxData(bData);
                        x = 9;
                        break;

                        case 9:
                        SPIS_1_SetupTxData(bData);
                        x = 10;
                        break;

                        case 10:
                        SPIS_1_SetupTxData(bData);
                        x = 11;
                        break;

                        case 11:
                        SPIS_1_SetupTxData(bData);
                        x = 0;
                        break;

                    }
                }
        }
}

int KEISAN (void)
{
        Z = PRT0DR&0b10000000;
                if(Z == 128)
                {
                        count = 0;
```

```
        }


if(A!=C || B!=D)
{
        if(B == 64)
        {
                if(C == 32)
                {
                        count += 1;
                        if(count == 14400)
                        {
                                count = 0;
                        }

                }
                else
                {
                        count -= 1;
                        if(count == -1)
                        {
                                count = 14399;
                        }

                }
        }
        else
        {
                if(C == 32)
                {
                        count -= 1;
                        if(count == -1)
                        {
                                count = 14399;
                        }
```

```
                        }
                        else
                        {
                                count += 1;
                                if(count == 14400)
                                {
                                        count = 0;
                                }

                        }
                }
        }
        A = C;
        B = D;
        return(count);
}
```

プログラムリスト3

座標計算とシリアル通信用プログラム

```
#include <ansi_c.h>
/*---------------------------------------------------------------------------*/
/* FILE:    serial.c                                                         */
/* PURPOSE: This example illustrates how to use the RS232 Library to send and*/
/* receive commands to and from a device via a serial port.  First,          */
/*you must configure the port, then send a command to the device,            */
/*then receive a command from the device.                                    */
/*It is important to select the correct terminator on sending and            */
/*reading, which depends upon your particular RS232 device.  This           */
/*program gives the option of none, line feed or carriage return             */
/*for terminators.  It is also important to make sure you have the           */
/*correct type of cable.  See the LabWindows/CVI documenation for            */
/*more details on cabling and handshaking.                                   */
/*---------------------------------------------------------------------------*/

/*-----------------------------------------------------------------*/
/* Include files                                                   */
```

```
/*---------------------------------------------------------------------------*/
#include <cvirte.h>
#include <userint.h>
#include <rs232.h>
#include <utility.h>
#include <formatio.h>
#include <string.h>
#include "serial.h"
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>

/*---------------------------------------------------------------------------*/
  /* 変数宣言                                                                 */
  /*---------------------------------------------------------------------------*/
  int panel_handle,
  config_handle,
  comport,
  baudrate,
  portindex,
  parity,
  databits,
  stopbits,
  inputq,
  outputq,
  xmode,
  ctsmode,
  stringsize,
  bytes_sent,
  bytes_read,
  RS232Error,
  config_flag,
  breakstatus,
  port_open,
  com_status,
  send_mode,
```

```
send_byte,
send_term_index,
read_term_index,
read_term,
inqlen,
outqlen,
  zahyou;
  i,
  x,
  y,
  suuchi,
  ABC,
  m,
  an;
  int a_num,b_num,c_num,d_num,e_num,f_num;
  float an1, an2 ,an3 ,an4 ,an5, an6;
  double X, Y, Z, A;
  short read_cnt;
  double timeout;
  char devicename[30],
  send_data[2000],
  read_data[2000],
  tbox_read_data[2000],
  com_msg[500],
  msg[100],
  zahyou_1[2000],
  suuchi_1[2000],
  AN[6],
  a[5],b[5],c[5],d[5],e[5],f[5],
  arg[6];
  float ARG[6];
  const char str[5];

  #define QuitHelp        1
  #define InputqHelp      2
  #define Pi              3.141592
```

```
/*---------------------------------------------------------------------*/
/* プロトタイプ宣言                                                    */
/*---------------------------------------------------------------------*/
void DisplayRS232Error (void);
void SetConfigParms (void);
void GetConfigParms (void);
void DisplayHelp (int);
void EnablePanelControls (int);
void DisplayComStatus (void);
void ActivateSendControls (int);
void SendAscii (void);
void SendByte (void);
int moji(int AN);
void radkeisan ( float an1, float an2, float an3, float an4, float an5, float an6,
                 float AN[]);
unsigned long ToDec(const char str[ ]);
void Gattai(char r_data[]);
float kakudo (int a);
void Z_kaiten(float arg, float x_0, float y_0, float z_0, float l_x, float l_y, float l_z,
                 float ans_z[]);
void Y_kaiten(float arg, float x_0, float y_0, float z_0, float l_x, float l_y, float l_z,
                 float ans_y[]);
void X_kaiten(float arg, float x_0, float y_0, float z_0, float l_x, float l_y, float l_z,
                 float ans_x[]);


/*---------------------------------------------------------------------*/
/* メイン関数                                                          */
/*---------------------------------------------------------------------*/
int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)
        return -1;
    panel_handle = LoadPanel (0, "serial.uir", SERIAL);
    DisplayPanel (panel_handle);
```

```
    RunUserInterface ();
    DiscardPanel (panel_handle);
    CloseCVIRTE ();
    return 0;
}

/*---------------------------------------------------------------------*/
/* ポート構成の設定                                                    */
/*---------------------------------------------------------------------*/
void SetConfigParms (void)
{
    SetCtrlVal (config_handle, CONFIG_COMPORT, comport);
    SetCtrlVal (config_handle, CONFIG_BAUDRATE, baudrate);
    SetCtrlVal (config_handle, CONFIG_PARITY, parity);
    SetCtrlVal (config_handle, CONFIG_DATABITS, databits);
    SetCtrlVal (config_handle, CONFIG_STOPBITS, stopbits);
    SetCtrlVal (config_handle, CONFIG_INPUTQ, inputq);
    SetCtrlVal (config_handle, CONFIG_OUTPUTQ, outputq);
    SetCtrlVal (config_handle, CONFIG_CTSMODE, ctsmode);
    SetCtrlVal (config_handle, CONFIG_XMODE, xmode);
    SetCtrlIndex (config_handle, CONFIG_COMPORT, portindex);
}

/*---------------------------------------------------------------------*/
/*ポート構成パラメータの情報を得る                                     */
/*---------------------------------------------------------------------*/
void GetConfigParms (void)
{
    GetCtrlVal (config_handle, CONFIG_COMPORT, &comport);
    GetCtrlVal (config_handle, CONFIG_BAUDRATE, &baudrate);
    GetCtrlVal (config_handle, CONFIG_PARITY, &parity);
    GetCtrlVal (config_handle, CONFIG_DATABITS, &databits);
    GetCtrlVal (config_handle, CONFIG_STOPBITS, &stopbits);
    GetCtrlVal (config_handle, CONFIG_INPUTQ, &inputq);
    GetCtrlVal (config_handle, CONFIG_OUTPUTQ, &outputq);
    GetCtrlIndex (config_handle, CONFIG_COMPORT, &portindex);
```

```
    #ifdef _NI_unix_
        devicename[0]=0;
    #else
        GetLabelFromIndex (config_handle, CONFIG_COMPORT, portindex,
                           devicename);
    #endif
}

/*---------------------------------------------------------------------*/
/* ユーザーによるポート設定                                            */
/*---------------------------------------------------------------------*/
int CVICALLBACK ConfigCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            config_handle = LoadPanel (panel_handle, "serial.uir", CONFIG);
            InstallPopup (config_handle);

            if (config_flag)
                SetConfigParms ();
            else
                config_flag = 1;
            break;
        case EVENT_RIGHT_CLICK :
            break;
        }
    return(0);
}

/*---------------------------------------------------------------------*/
/* 構成パネルのクローズ                                                */
/*---------------------------------------------------------------------*/
int CVICALLBACK CloseConfigCallback (int panel, int control, int event,
                                     void *callbackData, int eventData1,
```

```
                                        int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT :
            port_open = 0;
            GetConfigParms ();
            DisableBreakOnLibraryErrors ();
            RS232Error = OpenComConfig (comport, devicename, baudrate,
                parity, databits, stopbits, inputq, outputq);
            EnableBreakOnLibraryErrors ();
            if (RS232Error) DisplayRS232Error ();
            if (RS232Error == 0)
                {
                port_open = 1;
                GetCtrlVal (config_handle, CONFIG_XMODE, &xmode);
                SetXMode (comport, xmode);
                GetCtrlVal (config_handle, CONFIG_CTSMODE, &ctsmode);
                SetCTSMode (comport, ctsmode);
                GetCtrlVal (config_handle, CONFIG_TIMEOUT, &timeout);
                SetComTime (comport, timeout);
                EnablePanelControls (0);
                }
            else
                EnablePanelControls (1);
            DiscardPanel (config_handle);
            break;
        }
    return(0);
}

/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/
void EnablePanelControls (int enable)
{
    SetCtrlAttribute (panel_handle, SERIAL_SEND, ATTR_DIMMED, enable);
```

```
    SetCtrlAttribute (panel_handle, SERIAL_READ, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, SERIAL_READ_COUNT, ATTR_DIMMED,
            enable);
    SetCtrlAttribute (panel_handle, SERIAL_TBOX_READ, ATTR_DIMMED,
            enable);
    SetCtrlAttribute (panel_handle, SERIAL_BYTES, ATTR_DIMMED,
            enable);
    SetCtrlAttribute (panel_handle, SERIAL_ERROR, ATTR_DIMMED,
            enable);
    SetCtrlAttribute (panel_handle, SERIAL_FLUSHINQ, ATTR_DIMMED,
            enable);
    SetCtrlAttribute (panel_handle, SERIAL_FLUSHOUTQ, ATTR_DIMMED,
            enable);
    SetCtrlAttribute (panel_handle, SERIAL_GETINQ, ATTR_DIMMED,
            enable);
    SetCtrlAttribute (panel_handle, SERIAL_GETOUTQ, ATTR_DIMMED,
            enable);
    SetCtrlAttribute (panel_handle, SERIAL_COMSTATUS, ATTR_DIMMED,
            enable);
    SetCtrlAttribute (panel_handle, SERIAL_READTERM, ATTR_DIMMED,
            enable);
    SetCtrlAttribute (panel_handle, SERIAL_SENDMODE, ATTR_DIMMED,
            enable);
    SetCtrlAttribute (panel_handle, SERIAL_RCV_HELP_MSG,
            ATTR_DIMMED, enable);

    SetCtrlAttribute (panel_handle, SERIAL_CLEARBOX, ATTR_DIMMED,
            enable);
    ActivateSendControls (enable);
}

/*---------------------------------------------------------------------*/
/* テキストボックスなどの表示設定                                      */
/*---------------------------------------------------------------------*/
void ActivateSendControls (int enable)
{
```

```
    GetCtrlVal (panel_handle, SERIAL_SENDMODE, &send_mode);
    if (send_mode)
        {
        SetCtrlAttribute        (panel_handle,      SERIAL_TBOX_SEND,
            ATTR_DIMMED,
                                enable);
        SetCtrlAttribute        (panel_handle,      SERIAL_SENDTERM,
            ATTR_DIMMED, enable);
        SetCtrlAttribute        (panel_handle,      SERIAL_SENDBYTE,
            ATTR_DIMMED,
                                !enable);
        }
    else
        {
        SetCtrlAttribute        (panel_handle,      SERIAL_SENDBYTE,
            ATTR_DIMMED, enable);
        SetCtrlAttribute        (panel_handle,      SERIAL_TBOX_SEND,
            ATTR_DIMMED,
                                !enable);
        SetCtrlAttribute        (panel_handle,      SERIAL_SENDTERM,
            ATTR_DIMMED,
                                !enable);
        }
}

/*-------------------------------------------------------------------------*/
/* キャラクタディスプレイのクリア                                          */
/*-------------------------------------------------------------------------*/
int CVICALLBACK ClearBoxCallBack (int panel, int control, int event,
                                  void *callbackData, int eventData1,
                                  int eventData2)
{
    if (event == EVENT_COMMIT)
        ResetTextBox (panel_handle, SERIAL_TBOX_READ, "¥0");
    return 0;
}
```

```
/*---------------------------------------------------------------------*/
/* 送信の設定                                                          */
/*---------------------------------------------------------------------*/
int CVICALLBACK SendModeCallBack (int panel, int control, int event,
                                  void *callbackData, int eventData1,
                                  int eventData2)
{
    if (event == EVENT_COMMIT)
        ActivateSendControls (0);
    return 0;
}

/*---------------------------------------------------------------------*/
/*入力文字列のクリア                                                   */
/*---------------------------------------------------------------------*/
int CVICALLBACK FlushInCallBack (int panel, int control, int event,
                                 void *callbackData, int eventData1,
                                 int eventData2)
{
    if (event == EVENT_COMMIT)
        {
        FlushInQ (comport);
        MessagePopup ("RS232 Message", "Input queue flushed.");
        }
    return 0;
}

/*---------------------------------------------------------------------*/
/* 出力文字列のクリア                                                  */
/*---------------------------------------------------------------------*/
int CVICALLBACK FlushOutQCallBack (int panel, int control, int event,
                                   void *callbackData, int eventData1,
                                   int eventData2)
{
    if (event == EVENT_COMMIT)
```

```
        {
        FlushOutQ (comport);
        MessagePopup ("RS232 Message", "Output queue flushed.");
        }
    return 0;
}

/*---------------------------------------------------------------------*/
/* 入力文字列のバイト数を得る                                          */
/*---------------------------------------------------------------------*/
int CVICALLBACK GetInQCallBack (int panel, int control, int event,
                                void *callbackData, int eventData1,
                                int eventData2)
{
    if (event == EVENT_COMMIT)
        {
        inqlen = GetInQLen (comport);
        Fmt (msg, "%s<Input queue length = %i", inqlen);
        MessagePopup ("RS232 Message", msg);
        }
    return 0;
}

/*---------------------------------------------------------------------*/
/*出力文字列のバイト数を得る                                           */
/*---------------------------------------------------------------------*/
int CVICALLBACK GetOutQCallBack (int panel, int control, int event,
                                 void *callbackData, int eventData1,
                                 int eventData2)
{
    if (event == EVENT_COMMIT)
        {
        outqlen = GetOutQLen (comport);
        Fmt (msg, "%s<Output queue length = %i", outqlen);
        MessagePopup ("RS232 Message", msg);
        }
```

```
    return 0;
}

/*---------------------------------------------------------------------*/
/*COM ポートの状態を得て、それをユーザーに表示する      */
/*---------------------------------------------------------------------*/
int CVICALLBACK ComStatusCallBack (int panel, int control, int event,
                                   void *callbackData, int eventData1,
                                   int eventData2)
{
    if (event == EVENT_COMMIT)
        {
        com_status = GetComStat (comport);
        DisplayComStatus ();
        }
    return 0;
}

/*---------------------------------------------------------------------*/
/*ディスプレイ情報をユーザーに表示                                    */
/*---------------------------------------------------------------------*/
void DisplayComStatus ()
{
    com_msg[0] = '¥0';
    if (com_status & 0x0001)
        strcat (com_msg, "Input lost: Input queue"
                " filled and characters were lost.¥n");
    if (com_status & 0x0002)
        strcat (com_msg, "Asynch error: Problem "
                "determining number of characters in input queue.¥n");
    if (com_status & 0x0010)
        strcat (com_msg, "Parity error.¥n");
    if (com_status & 0x0020)
        strcat (com_msg, "Overrun error: Received"
                " characters were lost.¥n");
    if (com_status & 0x0040)
```

```
        strcat (com_msg, "Framing error: Stop bits were not received"
                " as expected.¥n");
    if (com_status & 0x0080)
        strcat (com_msg, "Break: A break signal was detected.¥n");
    if (com_status & 0x1000)
        strcat (com_msg, "Remote XOFF: An XOFF character was received."
                "¥nIf XON/XOFF was enabled, no characters are removed"
                " from the output queue and sent to another device "
                "until that device sends an XON character.¥n");
    if (com_status & 0x2000)
        strcat (com_msg, "Remote XON: An XON character was received."
                "¥nTransmisson can resume.¥n");
    if (com_status & 0x4000)
        strcat (com_msg, "Local XOFF: An XOFF character was sent to¥n"
                " the other device.  If XON/XOFF was enabled, XOFF is¥n"
                " transmitted when the input queue is 50%, 75%, and 90%¥n"
                " full.¥n");
    if (com_status & 0x8000)
        strcat (com_msg, "Local XON: An XON character was sent to¥n"
                " the other device.  If XON/XOFF was enabled, XON is¥n"
                " transmitted when the input queue empties after XOFF¥n"
                " was sent.  XON tells the other device that it can ¥n"
                " resume sending data.¥n");
    if (strlen (com_msg) == 0)
        strcat (com_msg, "No status bits are set.");
    MessagePopup ("RS232 Message", com_msg);
}

/*---------------------------------------------------------------------*/
/* ポートにデータを送信する                                            */
/*---------------------------------------------------------------------*/
int CVICALLBACK SendCallBack (int panel, int control, int event,
                              void *callbackData, int eventData1,
                              int eventData2)
{
    if (event == EVENT_COMMIT)
```

```
        {
        GetCtrlVal (panel_handle, SERIAL_SENDMODE, &send_mode);
        if (send_mode)
            SendAscii ();
        else
            SendByte ();
        RS232Error = ReturnRS232Err ();
        if (RS232Error)
            DisplayRS232Error ();
        SetCtrlAttribute (panel_handle, SERIAL_BYTES, ATTR_DIMMED, 0);
        SetCtrlVal (panel_handle, SERIAL_BYTES, bytes_sent);
        }
    return 0;
}

/*---------------------------------------------------------------------------*/
/* ポートに 1 バイト送信する                                                 */
/*---------------------------------------------------------------------------*/
void SendByte (void)
{
    GetCtrlVal (panel_handle, SERIAL_SENDBYTE, &send_byte);
    bytes_sent = ComWrtByte (comport, send_byte);
}

/*---------------------------------------------------------------------------*/
/* ポートに ASCII 文字を送信する                                             */
/*---------------------------------------------------------------------------*/
void SendAscii (void)
{
    GetCtrlVal (panel_handle, SERIAL_TBOX_SEND, send_data);
    GetCtrlIndex (panel_handle, SERIAL_SENDTERM, &send_term_index);
    switch (send_term_index)
        {
        case 1:
            strcat(send_data, "¥r");
            break;
```

```
            case 2:
                strcat(send_data, "¥n");
                break;
            }
        stringsize = StringLength (send_data);
        bytes_sent = ComWrt (comport, send_data, stringsize);
}

/*-------------------------------------------------------------------------*/
/*関係するエラー情報を表示                                                 */
/*-------------------------------------------------------------------------*/
int  CVICALLBACK  ErrorCallBack  (int  panel,  int  control,  int  event,
      void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            RS232Error = ReturnRS232Err ();
            DisplayRS232Error ();
            break;
        case EVENT_RIGHT_CLICK :
            break;
        }
    return 0;
}

/*-------------------------------------------------------------------------*/
/* COM ポートからデータを読み込む                                           */
/*-------------------------------------------------------------------------*/
int CVICALLBACK ReadCallBack (int panel, int control, int event,
                              void *callbackData, int eventData1,
                              int eventData2)
{
 unsigned long a_num, b_num,c_num,d_num,e_num,f_num;
 int a01,a02,a03,a04,a05,a06,i;
 float an1,an2,an3,an4,an5,an6;
```

```
float l_x, l_y, l_z;
float z_0,y_0,x_0, arg;
float ans_y[4], ans_x[3], ans_z[3],ARG[6], AN[6];


ans_z[0] = ans_z[1] = ans_z[2] = 0.0;

switch (event)
{
case EVENT_COMMIT:
while(1)
{
        read_data[0] = '¥0';
        GetCtrlVal (panel_handle, SERIAL_READ_COUNT, &read_cnt);
        GetCtrlIndex (panel_handle, SERIAL_READTERM,
            &read_term_index);
        switch (read_term_index)
            {
            case 0:
                read_term = 0;
                break;
            case 1:
                read_term = 13;
                break;
            case 2:
                read_term = 10;
                break;
            }
        if (read_term)
                {
                while(1)
                {
                  bytes_read = ComRdTerm (comport,
                      read_data,read_cnt,read_term);
                  CopyString (tbox_read_data, 0, read_data, 0,
                      bytes_read);
```

```
SetCtrlVal(panel_handle,SERIAL_TBOX_READ,
tbox_read_data);
RS232Error = ReturnRS232Err ();
Gattai(read_data);
a_num = ToDec(a);
a01 = (int) a_num;
an1 = kakudo(a01);
b_num = ToDec(b);
a02 = (int) b_num;
an2 = kakudo(a02);
c_num = ToDec(c);
a03 = (int)   c_num;
an3 = kakudo(a03);

d_num = ToDec(d);
a04 = (int) d_num;
an4 = kakudo(a04);

e_num = ToDec(e);
a05 = (int) e_num;
an5 = kakudo(a05);

f_num = ToDec(f);
a06= (int) f_num;
an6 = kakudo(a06);

radkeisan (an1, an2, an3, an4, an5, an6, AN);

l_x = 0.0;
l_y = 30.0;
l_z = 0.0;

x_0 = 0.0;
y_0 = 0.0;
z_0 = 0.0;
```

```
arg = AN[5];

Y_kaiten(arg, x_0, y_0, z_0, l_x, l_y, l_z, ans_y);

l_x = 0.0;
l_y = 0.0;
l_z = 0.0;

x_0 = ans_y[0];
y_0 = ans_y[1];
z_0 = ans_y[2];

arg = AN[4];

X_kaiten(arg, x_0, y_0, z_0, l_x, l_y, l_z, ans_x);

l_x = 0.0;
l_y = 100.0;
l_z = 0.0;

x_0 = ans_x[0];
y_0 = ans_x[1];
z_0 = ans_x[2];

arg = AN[3];

Y_kaiten(arg, x_0, y_0, z_0, l_x, l_y, l_z, ans_y);

l_x = 0.0;
l_y = 0.0;
l_z = 28.3;

x_0 = ans_y[0];
y _0 = ans_y[1];
z_0 = ans_y[2];
```

```
        arg = AN[2];

        X_kaiten(arg, x_0, y_0, z_0, l_x, l_y, l_z, ans_x);

        l_x = 0.0;
        l_y = 0.0;
        l_z = 76.67;

        x_0 = ans_x[0];
        y_0 = ans_x[1];
        z_0 = ans_x[2];

        arg = AN[1];

        X_kaiten(arg, x_0, y_0, z_0, l_x, l_y, l_z, ans_x);

        l_x = 0.0;
        l_y = 33.3;
        l_z = 100.0;

        x_0 = ans_x[0];
        y_0 = ans_x[1];
        z_0 = ans_x[2];

        arg = AN[0];

        Z_kaiten(arg, x_0, y_0, z_0, l_x, l_y, l_z, ans_z);


    SetCtrlVal(panel_handle,SERIAL_ans_x,ans_z[0]);
    SetCtrlVal(panel_handle,SERIAL_ans_y, ans_z[1]);
    SetCtrlVal (panel_handle, SERIAL_ans_z, ans_z[2]);

            }
    }
```

```
                    else
                    {
                        while(1)
                        {
                        bytes_read = ComRd (comport,
                            read_data,read_cnt);
                        CopyString (tbox_read_data, 0, read_data, 0,
                            bytes_read);
                        SetCtrlVal(panel_handle,
                            SERIAL_TBOX_READ,
                            tbox_read_data);
                         RS232Error = ReturnRS232Err ();
                        }
                    }
                if (RS232Error)
                    DisplayRS232Error ();
                break;
                    }
            case EVENT_RIGHT_CLICK :
                break;
            }
    return 0;
}

/*---------------------------------------------------------------------*/
/*ヘルプの表示                                                         */
/*---------------------------------------------------------------------*/
int CVICALLBACK InputQCallBack (int panel, int control, int event,
                                void *callbackData, int eventData1,
                                int eventData2)
{
    if (event == EVENT_RIGHT_CLICK)
        DisplayHelp (InputqHelp);
    return 0;
}
```

```
/*-------------------------------------------------------------------*/
/*文字列の状態をチェック                                             */
/*-------------------------------------------------------------------*/
int CVICALLBACK QuitCallBack (int panel, int control, int event,
                              void *callbackData, int eventData1,
                              int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT :
            if (port_open)
                {
                outqlen = GetOutQLen (comport);
                if (outqlen > 0)
                    {
                    MessagePopup ("RS232 Message", "The output queue
                      has¥n" "data in it. Wait for device to receive¥n"
                                    "the data or flush the queue.¥n");
                    break;
                    }
                RS232Error = CloseCom (comport);
                if (RS232Error)
                    DisplayRS232Error ();
                }
            QuitUserInterface (0);
            break;
        case EVENT_RIGHT_CLICK :
            DisplayHelp (QuitHelp);
            break;
        }
    return 0;
}

/*-------------------------------------------------------------------*/
/* エラー情報をユーザーに表示                                        */
/*-------------------------------------------------------------------*/
```

```
void DisplayRS232Error (void)
{
    char ErrorMessage[200];
    switch (RS232Error)
        {
        default :
            if (RS232Error < 0)
                {
                Fmt (ErrorMessage, "%s<RS232 error number %i",
                    RS232Error);
                MessagePopup ("RS232 Message", ErrorMessage);
                }
            break;
        case 0  :
            MessagePopup ("RS232 Message", "No errors.");
            break;
        case -2 :
            Fmt (ErrorMessage, "%s", "Invalid port number (must be in the "
                                    "range 1 to 8).");
            MessagePopup ("RS232 Message", ErrorMessage);
            break;
        case -3 :
            Fmt (ErrorMessage, "%s", "No port is open.¥n"
                "Check COM Port setting in Configure.");
            MessagePopup ("RS232 Message", ErrorMessage);
            break;
        case -99 :
            Fmt (ErrorMessage, "%s", "Timeout error.¥n¥n"
                "Either increase timeout value,¥n"
                "       check COM Port setting, or¥n"
                "       check device.");
            MessagePopup ("RS232 Message", ErrorMessage);
            break;
        }
}
```

```
/*---------------------------------------------------------------------------*/
/* ヘルプ情報をユーザーに表示                                                */
/*---------------------------------------------------------------------------*/
void DisplayHelp (int HelpId)
{
    switch (HelpId)
        {
        case 1 :
            MessagePopup ("Quit Help",
                          "The Quit button closes the current COM port,¥n"
                          "checks and displays any error messages, ¥n"
                          "and then exits this program.");
            break;
        case 2 :
            MessagePopup ("Input Queue Help",
                          "Specifies the size of the input queue for the "
                          "selected port.¥n"
                          "Default Value:  512¥n"
                          "Valid Range:     28 to 65516");
            break;
        }
}

/*---------------------------------------------------------------------------*/
/*文字列変換                                                                 */
/*---------------------------------------------------------------------------*/

unsigned long ToDec(const char str[ ])
{
    short i = 0;          /* 配列の添字として使用 */
    short n;
    unsigned long x = 0;
    char c;

    while (str[i] != '¥0')
 {
```

```
            if ('0' <= str[i] && str[i] <= '9')
                n = str[i] - '0';
             else if ('a' <= (c = tolower(str[i])) && c <= 'f')
                n = c - 'a' + 10;

            else {
                printf("無効な文字です。¥n");
                exit(0);
            }
            i++;

            x = x *16 + n;
        }
        return (x);
}

/*-------------------------------------------------------------------------*/
/*配列合体                                                                 */
/*-------------------------------------------------------------------------*/
void Gattai(char r_data[])
 {
   a[0] = r_data[33];
   a[1] = r_data[34];
   a[2] = r_data[3];
   a[3] = r_data[4];
   a[4] = 0;


   b[0] = r_data[38];
   b[1] = r_data[39];
   b[2] = r_data[8];
   b[3] = r_data[9];
   b[4] = 0;

   c[0] = r_data[43];
   c[1] = r_data[44];
```

```
  c[2] = r_data[13];
  c[3] = r_data[14];
  c[4] = 0;

  d[0] = r_data[48];
  d[1] = r_data[49];
  d[2] = r_data[18];
  d[3] = r_data[19];
  d[4] = 0;

  e[0] = r_data[53];
  e[1] = r_data[54];
  e[2] = r_data[23];
  e[3] = r_data[24];
  e[4] = 0;

  f[0] = r_data[58];
  f[1] = r_data[59];
  f[2] = r_data[28];
  f[3] = r_data[29];
  f[4] = 0;
 }

/*-------------------------------------------------------------------------*/
/*角度計算                                                                 */
/*-------------------------------------------------------------------------*/
float kakudo(int a)
{
 float arg, arg_1, an;
 arg_1 = (a / 14400.0)*360.0;
 arg = arg_1;
 if(arg > 180)
 {
          arg = -(360-arg);
 }
```

```
 return arg;
}
/*-------------------------------------------------------------------*/
/*rad 計算                                                          */
/*-------------------------------------------------------------------*/
void radkeisan ( float an1, float an2, float an3, float an4, float an5, float an6,
    float AN[])
{
 AN[0] = an1 * Pi /180.0;
 AN[1] = an2 * Pi /180.0;
 AN[2] = an3 * Pi /180.0;
 AN[3] = an4 * Pi /180.0;
 AN[4] = an5 * Pi /180.0;
 AN[5] = an6 * Pi /180.0;
}

/*-------------------------------------------------------------------*/
/*座標計算 Z                                                        */
/*-------------------------------------------------------------------*/
void Z_kaiten(float arg, float x_0, float y_0, float z_0, float l_x, float l_y, float l_z,
    float ans_z[])
{

 ans_z[0] = x_0 * cos(arg) - y_0 * sin(arg) + l_x;
 ans_z[1] = x_0 * sin(arg) + y_0 * cos(arg) + l_y;
 ans_z[2] = z_0 +l_z;
}

/*-------------------------------------------------------------------*/
/*座標計算 Y                                                        */
/*-------------------------------------------------------------------*/
void Y_kaiten(float arg, float x_0, float y_0, float z_0, float l_x, float l_y, float l_z,
    float ans_y[])
{

 ans_y[0] = x_0 * cos(arg) - z_0 * sin(arg) + l_x;
```

```
  ans_y[1] = y_0 + l_y;
  ans_y[2] = z_0 * cos (arg) + x_0 * sin (arg) +l_z;
}

/*-------------------------------------------------------------------------*/
/*座標計算  X                                                              */
/*-------------------------------------------------------------------------*/
void X_kaiten(float arg, float x_0, float y_0, float z_0, float l_x, float l_y, float l_z,
   float ans_x[])
{

  ans_x[0] = x_0 + l_x;
  ans_x[1] = y_0 * cos(arg) - z_0 * sin(arg) + l_y;
  ans_x[2] = y_0 * sin(arg) + z_0 * cos(arg) + l_z;
}
```

１１．　参考文献

1)　石田晴久　後藤良和　高田大二　中島寛和：入門 ANSI-C，実教出版，2004 年

2)　米田　完　坪内孝司　大隈　久：はじめてのロボット創造設計，講談社，2006 年

3)　下嶋　浩　佐藤　治：ロボット工学，森北出版，1999 年

4)　桑野雅彦：はじめての PSoC マイコン，CQ 出版社，2004 年

5)　桑野雅彦：PSoC マイコン・トレーニング・キット，CQ 出版社，2008 年

6)　林晴比古：新訂 新 C 言語入門 ビギナー編，ソフトバンク パブリッシング，1991 年

１２．　謝辞

本研究を行うにあたり、終始ご指導してくださった、本校電子制御工学科の由井四海教官、実習工場の技官の方々、専攻科生の方々には心から感謝いたします。