1　はじめに

1.1　背景

エアロゾル濃度によって地球環境の状態を知ることが出来る。エアロゾルとは、黄砂や工場の排煙に含まれる微粒子のことを指す。地球の温暖化・寒冷化の双方に影響を及ぼすことが知られている。そのため、エアロゾルの濃度測定を行うことは今後の環境変化を知る上で重要である。

1.2　リモートセンシング

リモートセンシングとは、対象物に接触することなく、対象物の状態を計測・分析する技術のことである。一般的なリモートセンシングの概要を図 1-1 に示す。光源のスペクトルは大気成分の影響を受けた後、計測装置によって測定され、大気成分の状態を推測することが出来る。

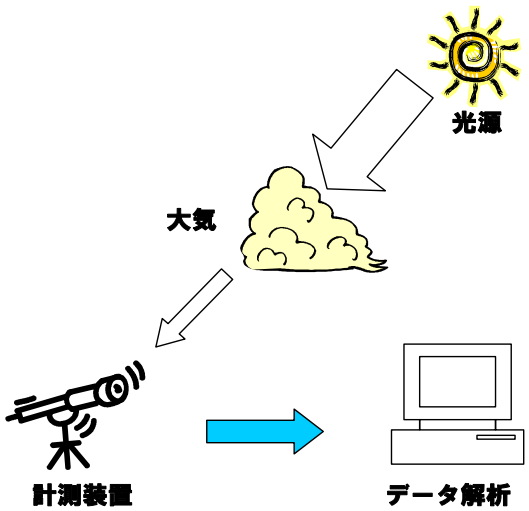

図 1-1　リモートセンシングによるデータ収集

1.2.1.1　　目的

エアロゾル濃度を光学的に測定する際に太陽を光源とすると、人工の光よりも遠方にあるため広域でのリモートセンシングが可能で，海上や山中でも観測が可能であるといった利点がある。しかし、太陽は地球から見て移動しているため、計測装置を常に太陽に向けながら、計測するサンフォトメーターが必要となる。

本研究は、太陽を光源としたリモートセンシングを行うために必要となるサンフォトメーターの製作を行う。

1.3　エアロゾル濃度測定原理

エアロゾル濃度測定原理の概要を図 1-2 に示す。エアロゾル濃度をリモートセンシングするために、太陽を光源とする。太陽から放射された光は大気分子やエアロゾルによって吸収、散乱され、地上で観測される放射照度が変化する。大気分子による影響で、地上で観測される太陽放射照度の変化は季節や緯度によって多少変化するが、時間的、空間的にほぼ一定として扱うことが出来る事が知られている。そのため、太陽の放射照度を計測し、その変化を連続的に観測することによって、エアロゾル濃度の増減を推測することが可能である。

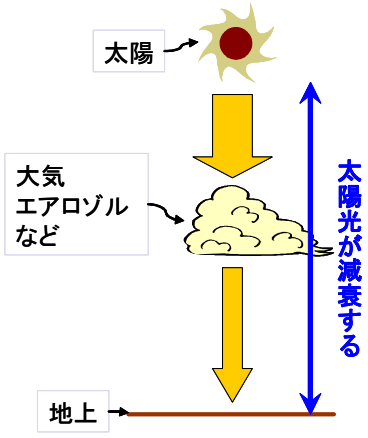

図 1-2　エアロゾル濃度測定原理

1.4　サンフォトメーター

サンフォトメーターの概要を図 1-3 に示す。サンフォトメーターは、太陽を追従するサントラッカー部分と光強度を計測するフォトセンサー部分から構成される。サントラッカーは PD として 4 分割フォトダイオードを用いて、太陽の位置をとらえて自動追尾を行う。

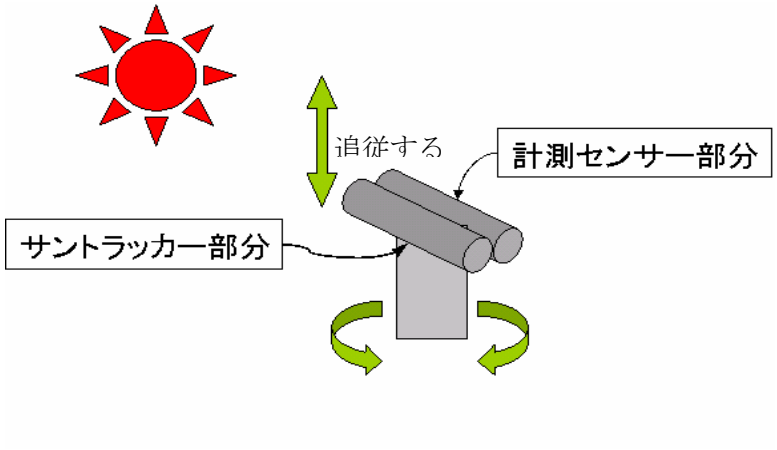

図 1-3　サンフォトメーターの概要

2　太陽エネルギー

太陽から地球に供給されるエネルギーは太陽定数と呼ばれ、太陽定数は太陽光線に対して垂直な 1 ㎡の面が 1 秒間に受け取るエネルギーである。その値は 1.37kW/m² である。太陽定数は太陽の活動によって異なり、一定の値ではないが、本研究では一定として扱った。しかし、実際には、地球は球体であり、図 1-4 のように太陽光線が地面に対して垂直に注ぐ地域は限られている。

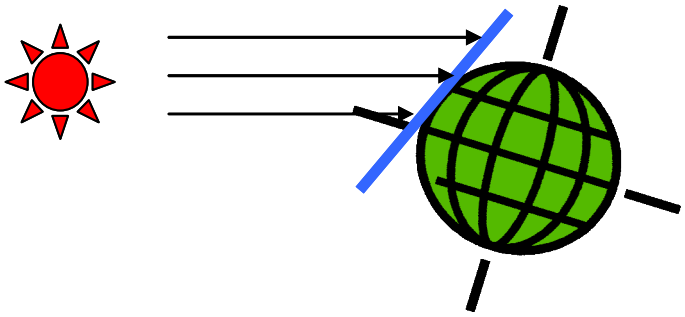

図 1-4　地球上に注ぐ太陽光

地球上の緯度 φ での 1 ㎡に入射する太陽光のエネルギーは次式 α と太陽定数の積によって求められる。

$$\alpha = (L_0 / L)^2 \cos(z)$$
$$\cos(z) = \sin\phi \sin\delta + \cos\phi \cos\delta \cos h$$

ここで、δ は太陽の赤緯、h は太陽の時角で、日時と時間による変数である。L は地球と太陽の実距離、L_0 は地球と太陽の平均距離である。富山（E137° 13′ N36° 42′ ）での南中時の一年を通じた太陽光のエネルギーの変化は約 0.71～1.28 kW/㎡になる。太陽光エネルギーの変化を図 1-5 に示す。

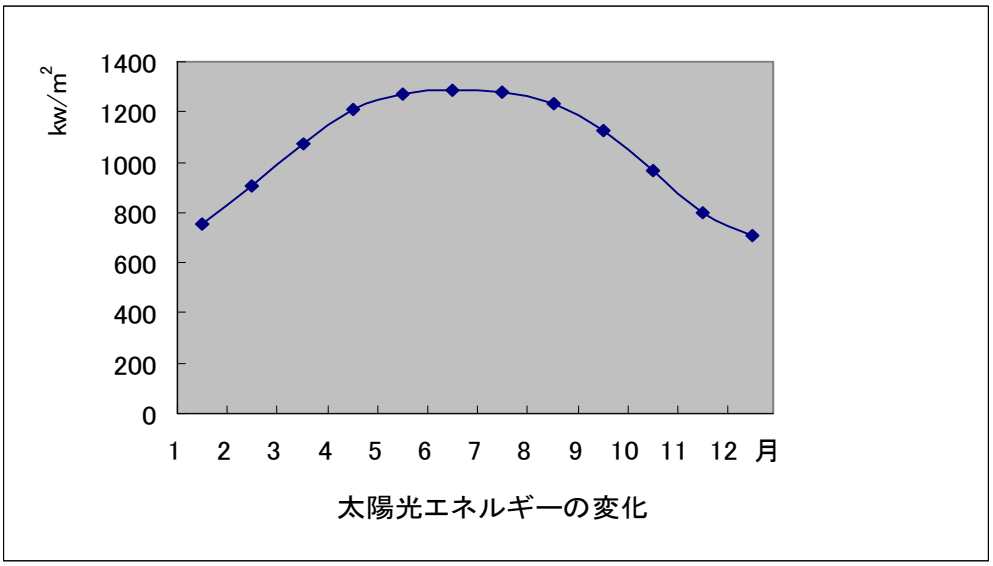

図 1-5　太陽光エネルギーの変化

2　サントラッカーの製作

2.1　サントラッカーのシステム概要

サントラッカーのシステム概要を図 2-1 に示す。4 分割フォトダイオード（PD）は受光した光エネルギーの強さに応じて電流が生じる。PD で発生した光起電流は電流・電圧変換回路へ入力され電圧に変換される。PD で発生した電流はマイナスの起電流であり、電流・電圧変換回路でマイナスの電圧に変換されているので、反転増幅回路を通して PIC へ入力される。PD から得られた 4 チャンネル分の電圧に基づき、必要なパルスが PIC からステッピングモーターに出力され、軸が回転し PD は常に光源の方向へ向けられる。光源の追従のためには方位角方向、高度角方向の 2 軸が必要となり、本研究では 2 軸システムを製作した。サントラッカーの簡単なフローチャートを図 2-2 に示す。

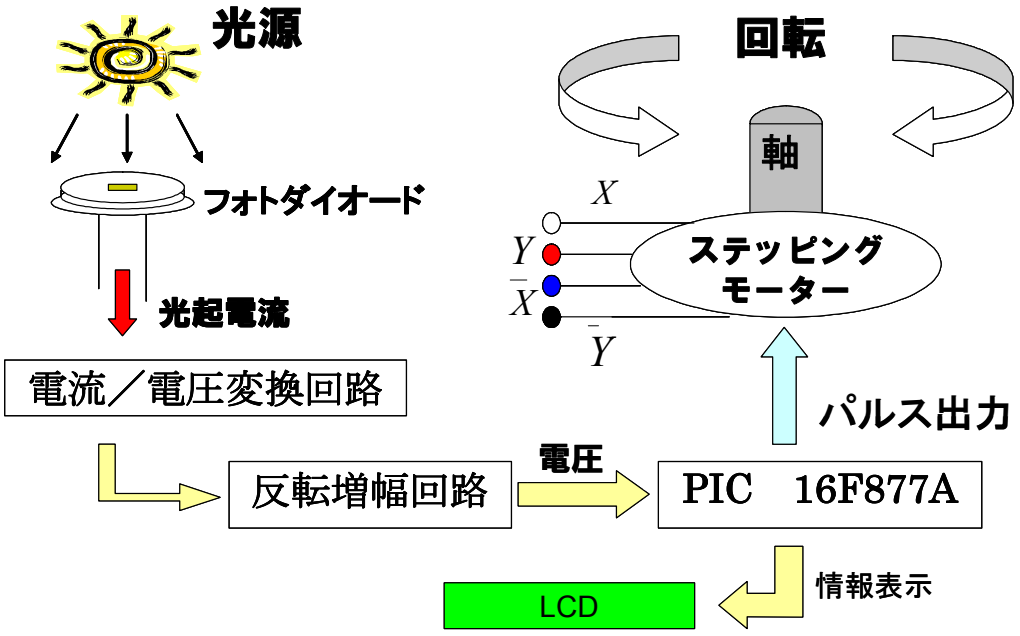

図 2-1　サントラッカーシステム概要

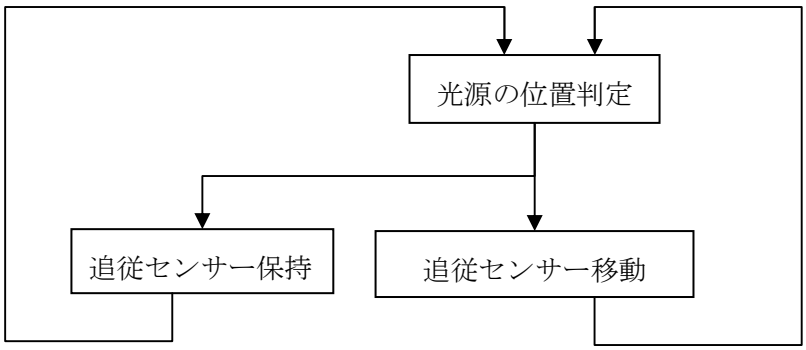

図 2-2　サントラッカーフローチャート

2.2　制御用マイクロコンピュータについて

サントラッカーの制御用マイコンとして、Microchip Technology 社製 PIC 16F877A を使用した。PIC とは、Peripheral Interface Controller の略称であり、周辺機器制御用のワンチップマイクロコンピュータである。16F877A は 40pin の DIP タイプである。16F877A のピン配置を図 2-3 に示す。

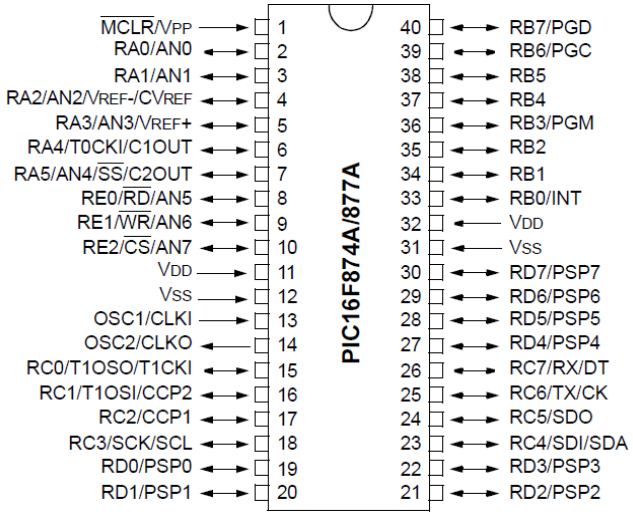

図 2-3　PIC16F877A　PIN 配置

16F877A の主な特徴を以下に示す。

・10bit　A/D コンバータ内臓

・PWM モジュール内蔵

・コンパレータ機能内蔵

・TIMER1,2 内蔵

・14K バイトのプログラムエリア

・シリアル通信サポート

・256 バイトの RAM

・256 バイトの EEPROM データメモリ内蔵

・ICD（インサーキットデバッガ）対応

PIC16F877A は 20MHz での動作に対応している。しかし、本研究では高速な動作は必要なく、安定した動作のため 4MHz で動作させた。サンフォトメーターでは、日の出から日の入りまでの長時間にわたる安定した動作が必要となるが、PIC には WDT（ウォッチ・ドグ・タイマ）、PWRT（パワー・アップ・タイマ）、OST（オシレータスタートアップタイマ）BOR（ブラウン・アウト・リセット）などの安定動作のための機能が標準で用意されている。

WDT はプログラムが暴走し正常に動作しなくなった場合、自動的に PIC をリセットする機能である。プログラム中に WDT を定期的にリセットする処理を入れておくと、正常に動作している場合は定期的に WDT がリセットされる。プログラムが暴走し、WDT がリセットされないと WDT がタイムアップして PIC が強制的にリセットされる。WDT 用のクロックは独立した RC 発振器から供給されているため、システムクロックが停止しても WDT は動作する。

PWRT は電源投入直後に電源電圧、周辺機器が安定するまでの時間（72ms）待つ機能である。PWRT は電源電圧が確実に立ち上がるまでの時間を PWRT で確認し、確実なオシレータの発振を OST で確認する。よって 72m s より十分短い時間で立ち上がる事が保証されている電源であれば、電源オン時のリセットはこの内蔵機能だけで実現できる。BOR は動作中に異常に電圧が低下した場合、PIC が暴走

するのを防ぐため自動的にリセットがかかる。

2.3　C 言語による PIC プログラミング

PIC はアセンブリ言語にて、プログラムされることが多い。PIC16F877A で使用するニーモニックは 35 種類しかなく、ごく単純な動作であれば、アセンブリ言語で簡単にプログラムを記述できる。しかし、少し複雑な動作をするプログラムをアセンブラ言語で作成するには行数が増え、手間がかかる。高級言語である C 言語によってプログラムを作成することによって、複雑な動作も簡単にプログラミングができる。C 言語の一般的な特徴として、

- 言語仕様が簡潔である。
- プログラムは関数の集合で構成される。
- 演算子の種類が多く、複雑な式が簡単にプログラムできる。

が挙げられる。また、WIZ-C（FED 社製）でのプログラミングではプログラマーは PIC 特有のバンクの切り替えなどを意識することなく、プログラミングが行える。しかし、C 言語のアセンブラ言語と C 言語の実行速度を比べると、C 言語では、機械語に変換するコンパイラの性能によって多少の違いはある物の、アセンブラ言語に比べると、無駄な処理が増え、動作速度の面では劣る。

本研究では ANSI(American National Standards Institute,アメリカ国内規格協会)準拠の WIZ-C を用いて、C 言語にて PIC プログラムを作成した。

WIZ-C による開発手順を簡単に説明する。まず、プロジェクトを新規作成し、アプリケーションデザイナと呼ばれる画面で、使用する I/O ポートや接続する LED（発光ダイオード）や LCD（液晶表示器）などの設定を行う。設定が終わると、xxx_User.c と xxx_main.c が自動生成される（xxx はプロジェクト名）。xxx_main.c 内には I/O ポートの設定や、接続する LCD などの初期化処理が組み込まれる。xxx_User.c 内の UserInterrupt 関数内にユーザー割り込み処理、UserInitialise 関

数内にユーザー初期化処理、UserLoop 関数内に目的のプログラムを組み込む。UserLoop 関数は xxx_main.c 内でループ処理される。プログラムが一通り完成したら WIZ-C からコンパイル操作を行う。コンパイルエラーを修正後、デバッガ画面にて、PIC に LED、LCD、可変抵抗などを接続し、パソコン上でプログラム動作のシミュレーションを行うことが出来る。

シミュレーションで問題がなければ、生成された HEX ファイルを PIC に書き込む。動作に問題があれば、プログラムを修正し、これを繰り返し、目的の動作をするプログラムを作成する。

2.4　光源の追従

2.4.1　追従原理

光源の追従は 4 分割フォトダイオードを用いて光源の方向を検出し行う。4 分割フォトダイオードは Si PIN 接合型の S4346（浜松ホトニクス製）を使用した（図 2-4)。

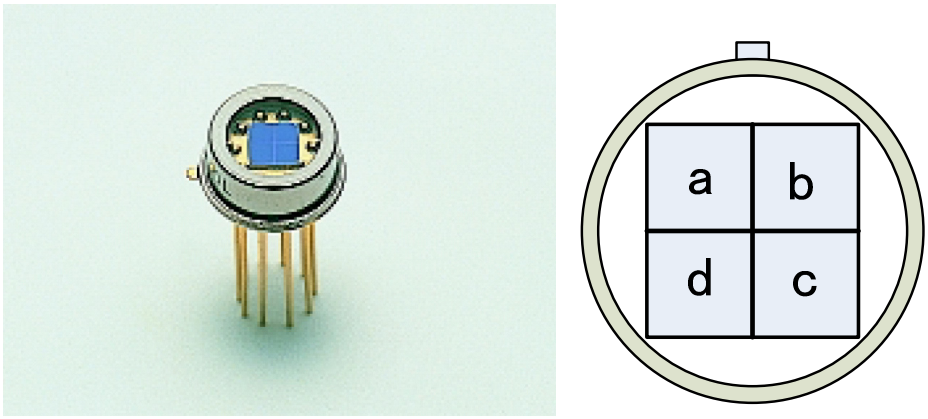

図 2-4　4 分割フォトダイオード（S4346）

受光感度は波長によって異なるが、波長 500nm において、0.28A/W である。

太陽の位置検出の為の PD 部は、四分割フォトダイオードと凸レンズを一直線に配置する。光源の位置検出では一軸につき、2 素子の電圧を使用するため、4 分割フォトダイオードでは 2 方向の光源の移動を検出することが出来る。θ 方向では素子番号 a と c、ϕ 方向では素子番号 b と d の電圧を比較する。凸レンズで集光された光は図 2-5 のように四分割フォトダイオードに像を作る。

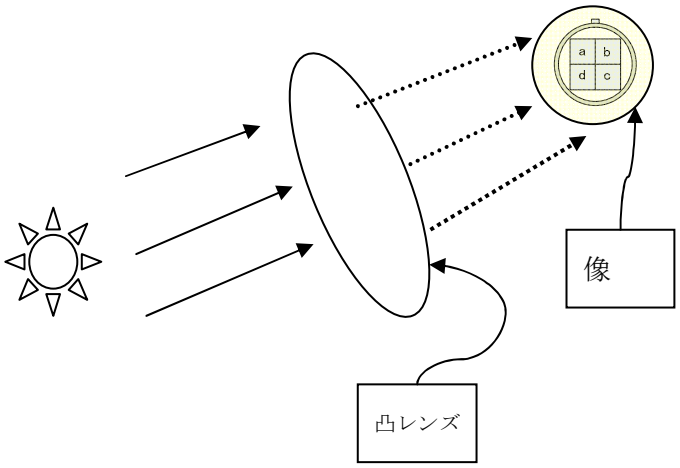

図 2-5　4 分割フォトダイオード上の像

光源が正面にある場合、2 素子の電圧は等しくなり（図 2-6）、光源が正面にあると判定できる。

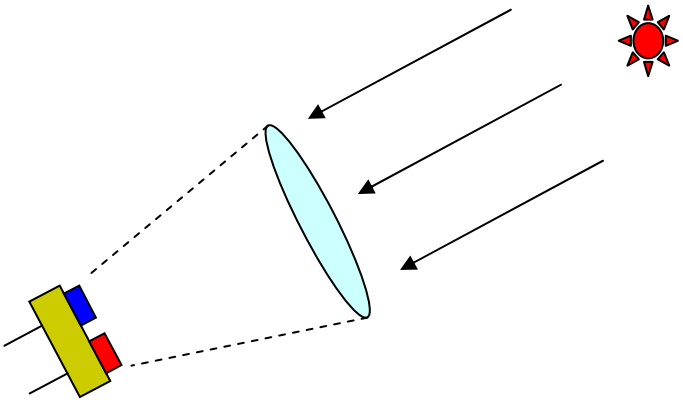

図 2-6　光源が正面の場合

光源が移動した場合、逆の素子の電圧が高くなり（図 2-7）、光源が移動したことが判定できる。

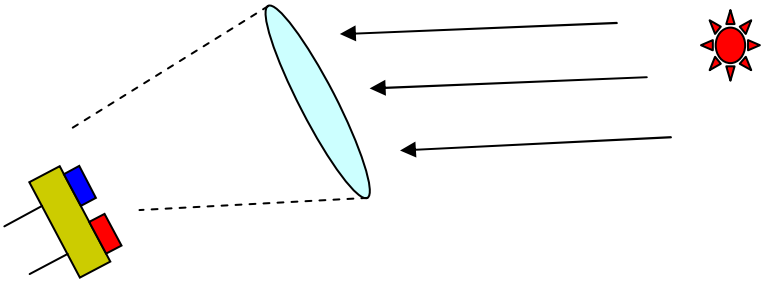

図 2-7　光源が正面でない場合

このような追従動作を行うプログラムのフローチャートを図 2-8 に示す。

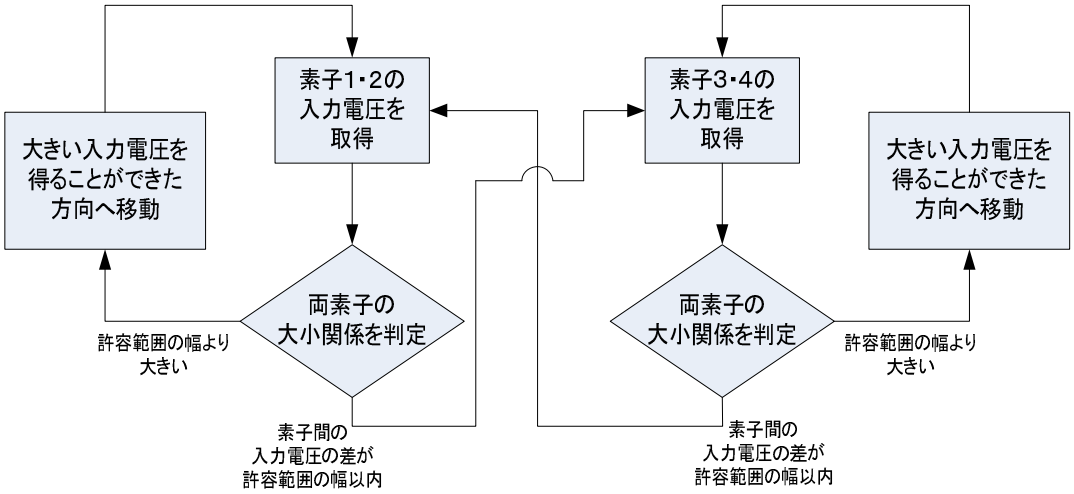

図 2-8　追従動作フローチャート

2.4.2　追従精度

4 分割フォトダイオードは 1 素子が一辺 1.5mm の正方形で、4 素子あわせると、一辺が 3mm の正方形となる。4 素子のフォトダイオードすべてに光が当たる状態になる、ように凸レンズとフォトダイオード間の距離を調整する。4 素子すべてに光が当たる状態で像が一番小さくなる像は一辺 3mm の正方形の対角線の長さが直径となる円形の像である。よって、4 素子すべてに光が当たるように、レンズと 4 分割フォトダイオード間の距離調整した場合、像の直径 4.24mm の円となる。光源の位置が移動し直径 4.24mm の円がフォトダイオード上を移動したとき、像の移動方向により受光面積が減った素子の電圧は減少する。像の移動したときの、受光面積 S は下式で表すことが出来る。

（素子の一辺の長さ：A、光源の移動距離：c、像の半径：r）

$$S = A \cdot Z + \int_0^{A-Z} Z dx \qquad (0 \le c \le Z)$$

$$S = \int_0^{X} X dx \qquad (Z < c \le A)$$

$$Z = \sqrt{r^2 - (c + A)^2} - c$$

$$X = \sqrt{r^2 - c^2} - c$$

この式より、光源が移動し像が移動したときの受光面積を計算したグラフを図 2-9 に示す。

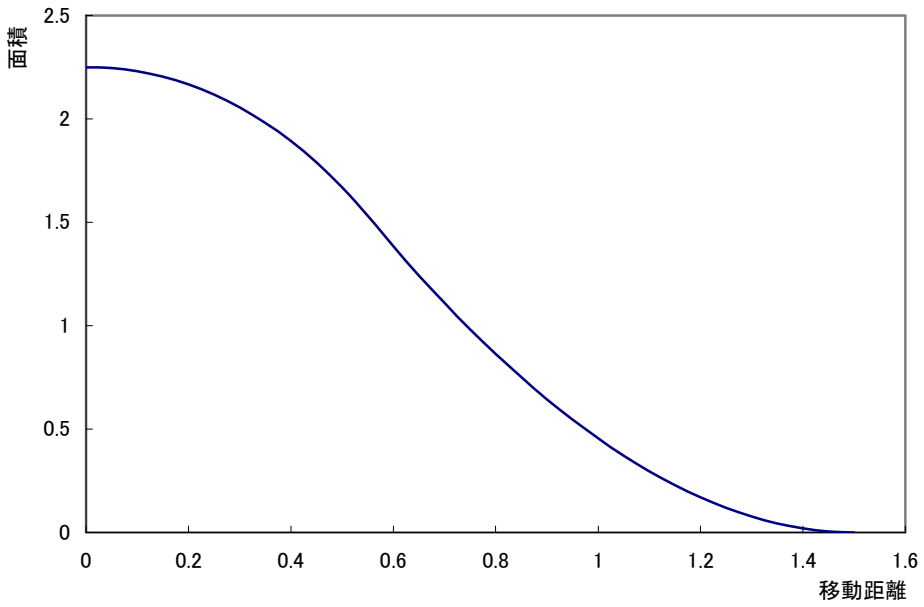

図 2-9　光源の移動距離-受光面積

フォトダイオードは受光面積に比例して起電力 P を発生するので、元の受光面積を S_0 とすると、

$$P = \frac{S}{S_0}$$

より元の起電力に対する割合がわかる。さらに、像の移動距離と光源の移動角度の関係から、光源と PD 部との変位角度と起電力変化の関係がわかる。光源が 4 分割フォトダイオードの正面にあり、その時の起電力が 5V だったと仮定し、4 分割フォトダイオードの正面から光源の変位角度と電圧の関係を図 2-10 に示す。

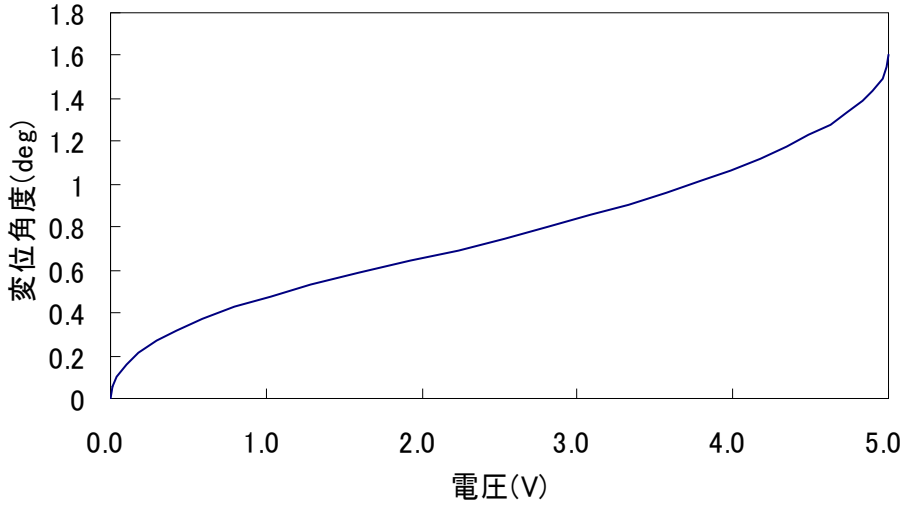

図 2-10　光源の変位角度-電圧

A/D 変換の最小分解能が 0.00488mV なので、検出できる光源の変位角度は 0.2° 以下となる。また、像が完全に 4 分割フォトダイオード上から外れる光源の変位角度は 3.2° であり、その範囲を外れると光源を見失うことになる。

2.4.3　しきい値

PD 部の 1 ステップ角度が無限小という理想的な機構の場合、光源が正面にある場合、両素子の電圧は等しくなる。しかし、ステッピングモーターを使用した本研究では最小回転角が無限小という機構の製作は難しい。本研究で製作した機構では、ステッピングモーターの 1 ステップ角度 1.8° を 1/9 に細分化し 1 ステップの最小回転角は 0.2° である。一方、4 分割フォトダイオードで検出できる PD 部と光源との変位角度は 0.2° 以下である。光源とのズレが 0.2° 以上の場合、PIC はステッピングモーターにパルスを出力し、PD 部を回転させることになるが、1 ステップ角が 0.2° のため、振動してしまう。よって、ソフトウェア的に、ある程度の変位角度を許容する必要がある。具体的には、比較する 2 つの入力電圧の差が設定した範囲内にある場合は、光源に正対

していると見なし、パルスを出力させない。この許容する PD 部と光源の変位角度のしきい値を適切に設定する必要がある。

2.4.4　レンズ

使用する PD の受光感度が 0.28A/W（波長 500nm）であり、必要な電流は 0.05A 程度である。よって、4 分割フォトダイオードには 0.17W 前後の太陽光を供給したい。よって、直径 2cm 前後の凸レンズが最適である。本研究では直径 2.5cm、焦点距離 5cm の凸レンズを使用した。

また、太陽を追従する場合、太陽光が強すぎて、4 分割フォトダイオードが過信号飽和する事が考えられる。そこで、凸レンズの前に透過率 40%の ND フィルター（Neutral Density Filter）を配置して、太陽光を弱めた。ND フィルターとは全波長においてほぼ均一に光強度を弱めるフィルターである。

2.5　電流・電圧変換回路および反転増幅回路

図 2-11 に電流・電圧変換回路と反転増幅回路を接続した回路図を示す。電流・電圧変換回路は入力された電流を変換係数-R_t で電圧に変換する。反転増幅回路は、入力された電圧を-R_2/R_1 の増幅率で反転増幅する。電流･電圧変換回路への入力電流と、反転増幅回路からの出力電圧の関係は、

$$V_{OUT} = \frac{R_2}{R_1} R_t I_{in}$$

で表すことができ、正電圧 V_{OUT} を得ることが出来る。オペアンプには 4 回路入っている LM324 を使用した。

反転増幅回路からの出力電圧はそのまま、PIC への入力電圧となる。PIC での A/D 変換は 0V から 5V の入力電圧に対応している。光源の強さによって、反転増幅回路の出力電圧が変化するため、反転増幅回路の出力電圧が、PIC の AD 変換モジュールの最大値を超えないような値となるように、電流電圧変換回路、反転増幅回路の抵抗 R_t,R_1,R_2 を調整する必要がある。白熱電球(74W)を用いた実験では、R_t=75kΩ,R_1=200kΩ,R_2=20kΩとした。太陽を用いた実験では、R_t=1.8kΩ,R_1=200Ω,R_2=20Ωとした。

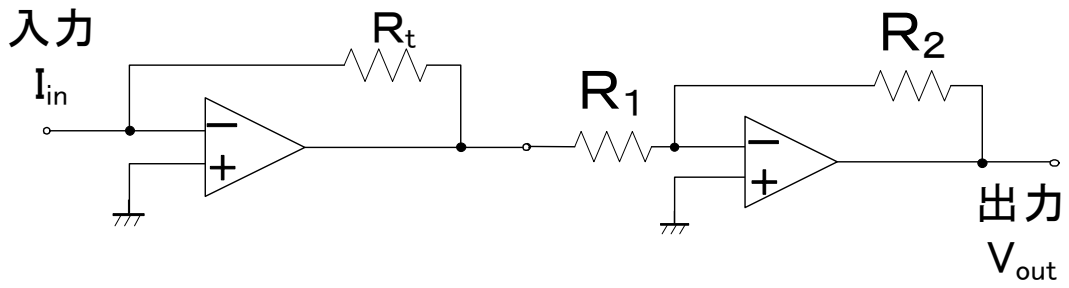

図 2-11　電流・電圧変換回路

2.6　駆動装置

太陽の追従のために、方角、高度の 2 軸制御が必要となる。軸の駆動源として、ステッピングモーターを使用した。ステッピングモーターは入力するパルスに応じて、固有のステップ角度で回転する。そのため、高い位置決め精度が必要とされる自動制御装置に使用されることが多い。

ステッピングモータードライブ回路を図 2-12 に示す。

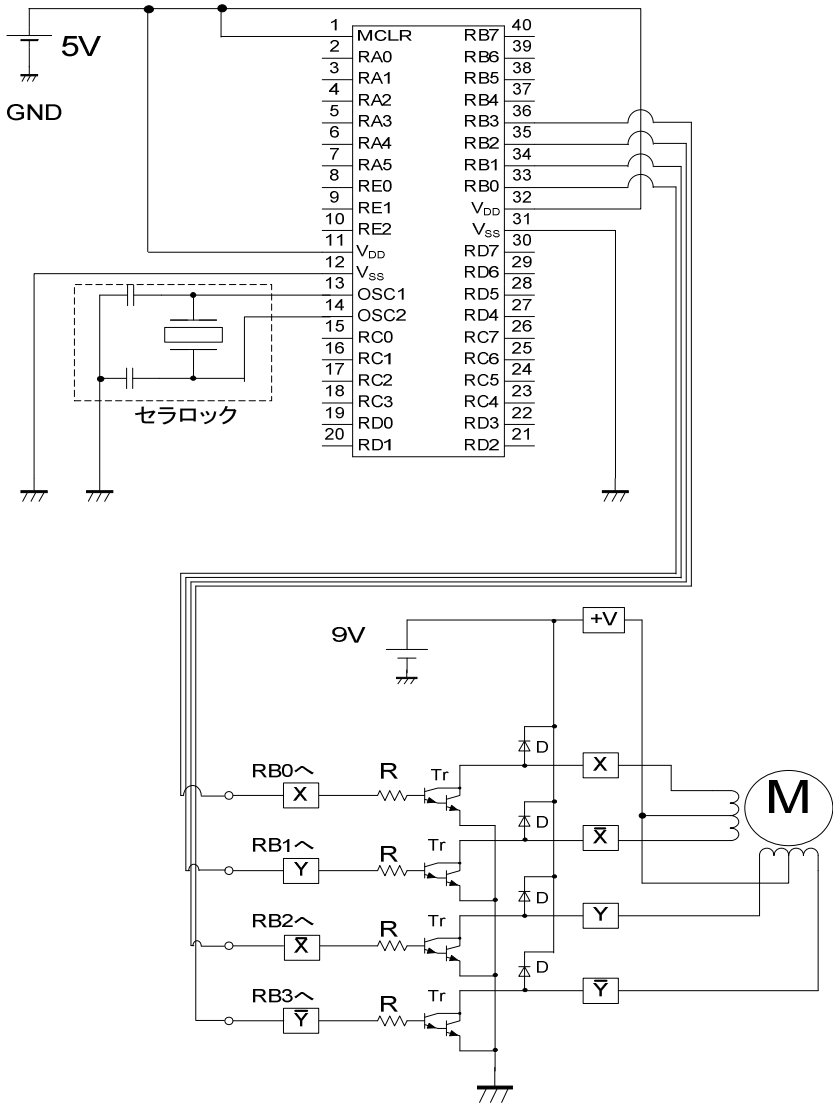

図 2-12　モータードライブ回路

使用したステッピングモーター（VEXTA 製　PH264-02）は 2 相励磁方式であ

る。回転させるためには、図 2-13 のように、4 本の電力線に位相を 1/4 ずつ遅らせたパルスを入力することで、ステッピングモーターの軸は回転する。

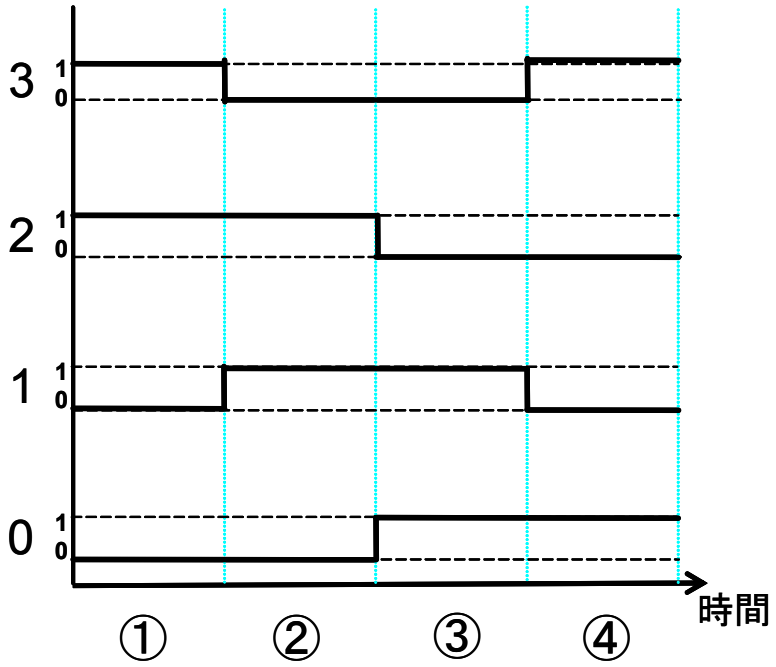

図 2-13　ステッピングモーター回転パルス

ステッピングモーターを回転させるのに必要なパルスを出力する MotorDrive 関数を作成した。MotorDrive 関数のフローチャートを図 2-14 に示す。

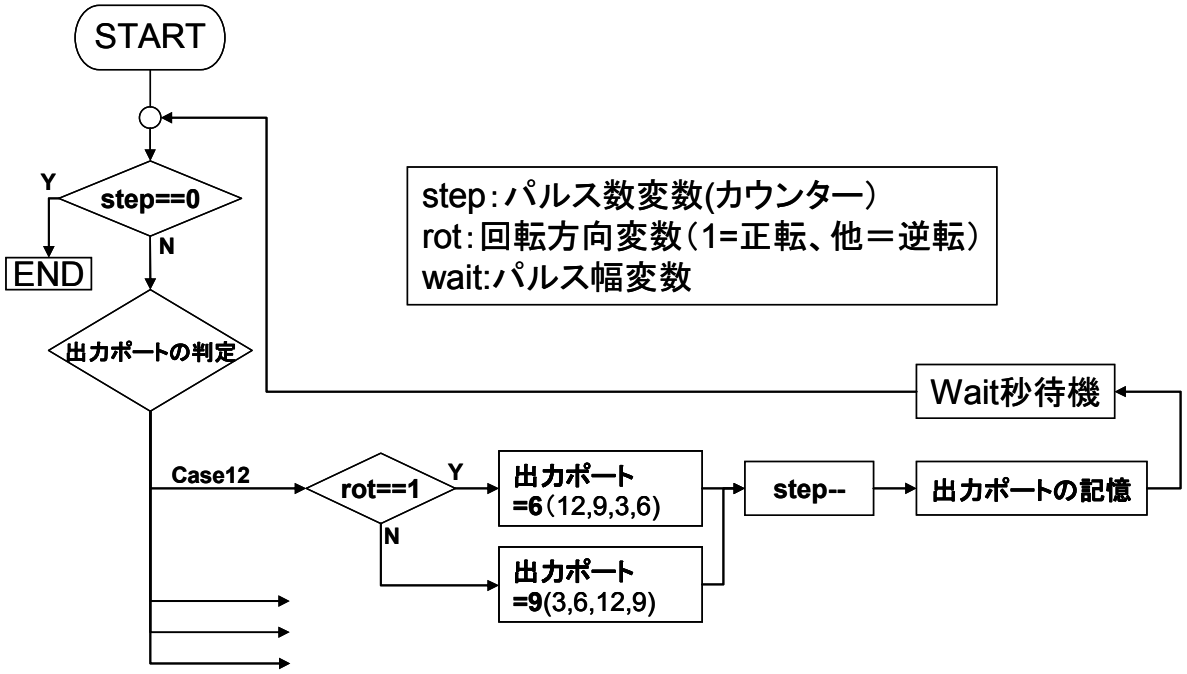

図 2-14　MotorDrive 関数フローチャート

MotorDrive 関数は、パルス数、パルス幅、回転方向を引数として MotorDrive 関数を呼び出すと、指定した、回転方向とパルス幅でパルスを指定回数、出力する。パルス幅 10ms の MotorDrive 関数の出力した波形を図 2-15 に示す。図 2-15 は上から、ポート 0,1,2,3 の順でならんでいる。オシロスコープで観測された波形は、プログラム通り、10ms のパルス幅で出力することが出来た。

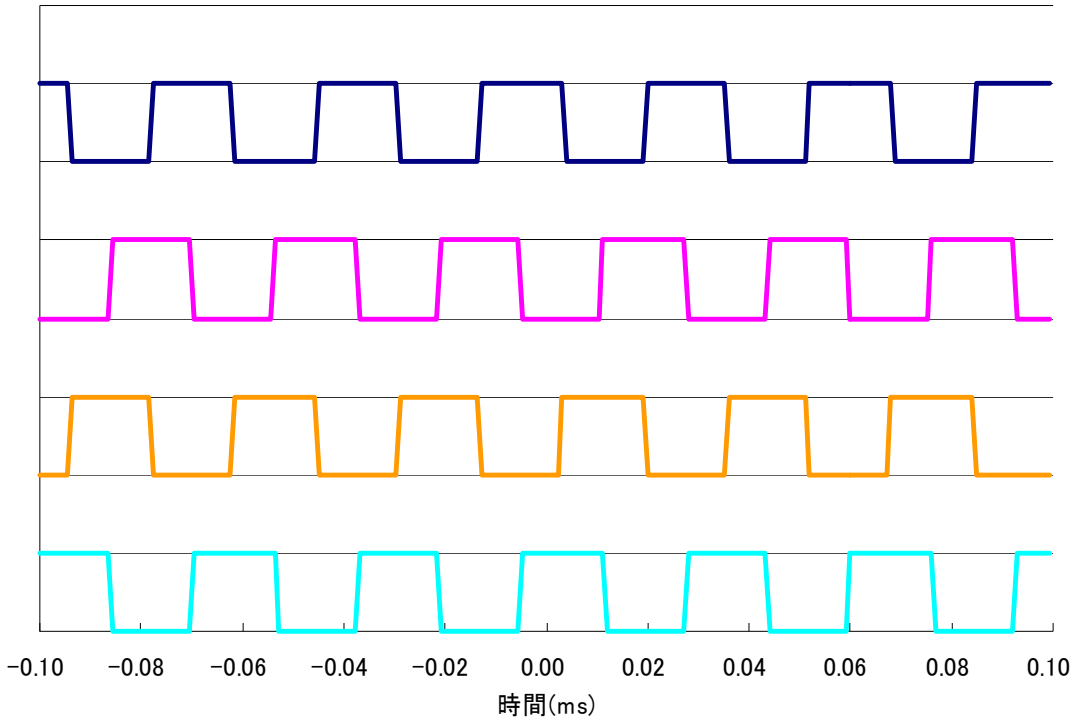

図 2-15　MotorDrive 関数出力パルス

今回使用したステッピングモーターの 1 ステップ角度は 1.8°である。1.8°は太陽の約 7.2 分間に移動する距離と同じである。このままでは、PD を正確に太陽へ向けることができない。そこで、タイミングベルトを使用して、ステップ角度の細分化を行った。1:3 のタイミングベルトを 2 段使用して、1:9 に減速した。よって、PD 部分の 1 ステップ角度は 0.2°となる。これは、太陽の 48 秒間の移動距離に相当する。

2.7　まとめ

サントラッカーの回路図を図 2-15 に示す。外観図を図 2-16 に示す。

4 分割フォトダイオードを用いた光源の位置検出する方法について検討し、プログラムを作成した。

ステッピングモーターを回転させるためのパルスを出力する関数をプログラムし、任意のパルスを出力出来ることを確認した。

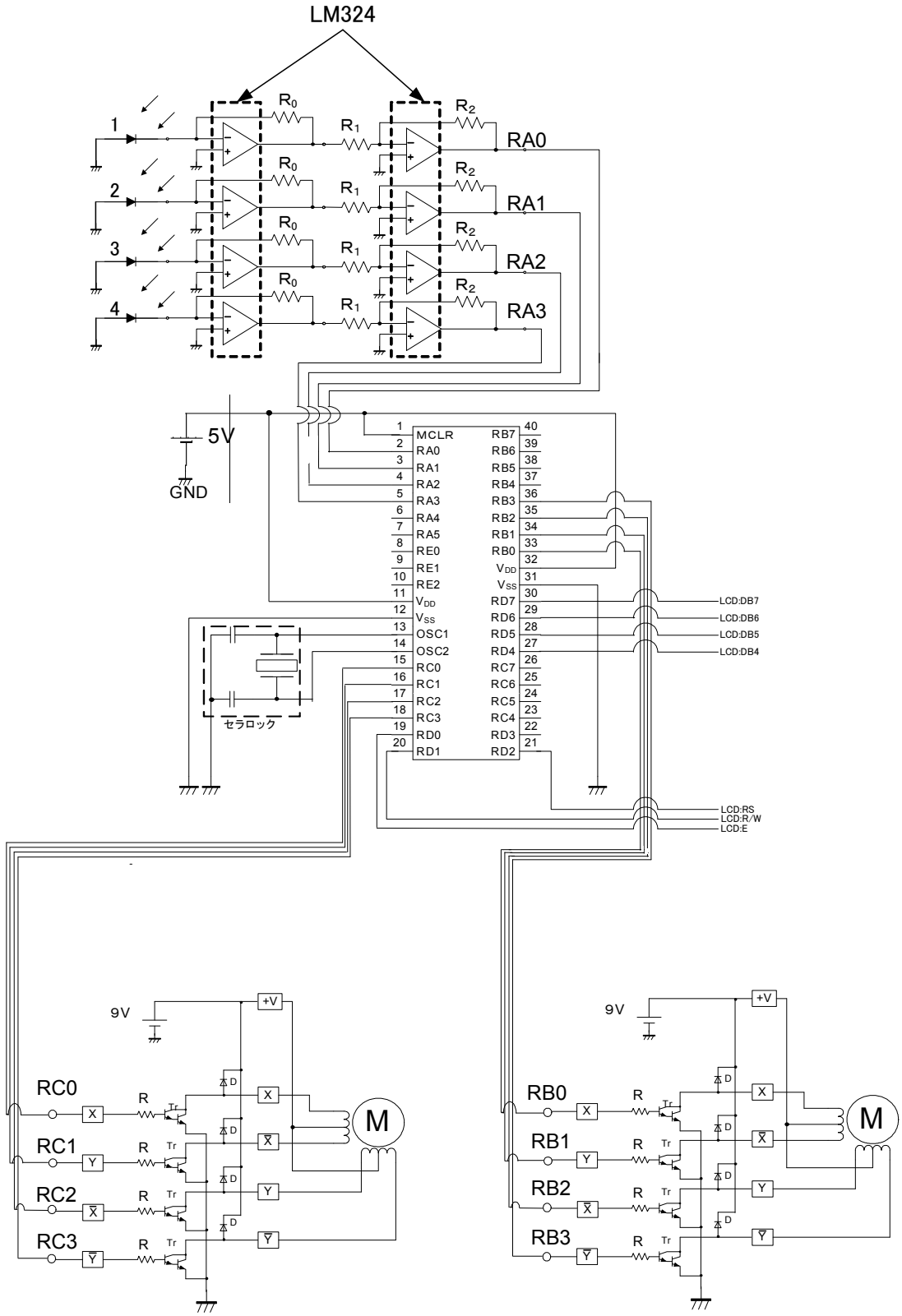

図 2-15　サントラッカー回路図

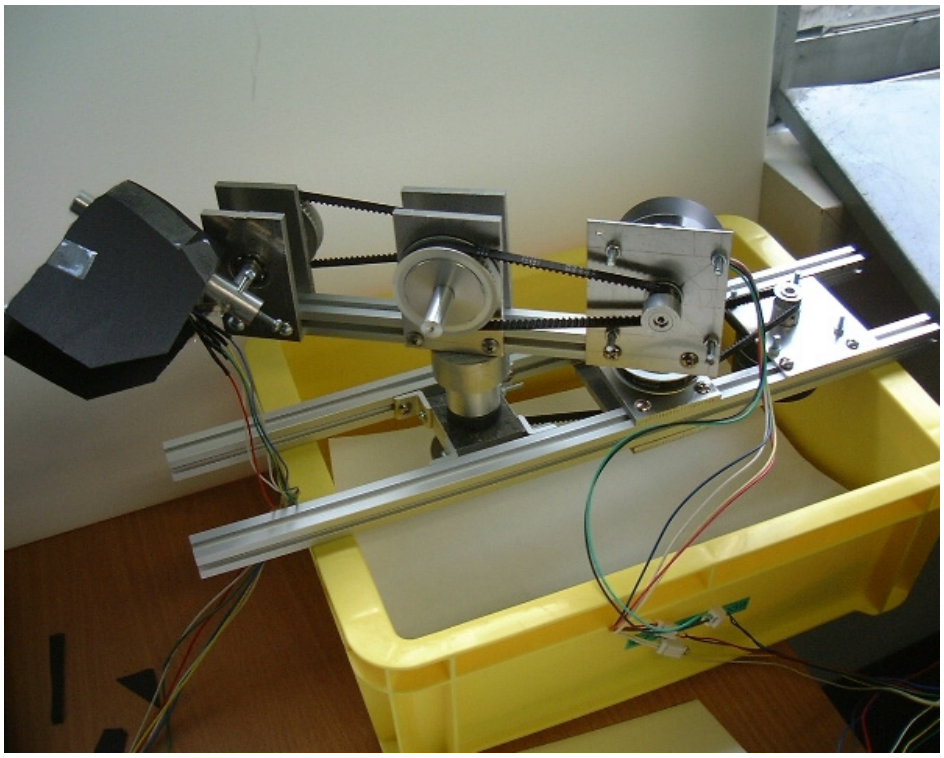

図 2-16　サンフォトメーター外観

3.　追従の実験

3.1.　白熱電球の追従実験

3.1.1.　最大追従速度

追従実験に先立ち、ステッピングモーターの最大回転速度を計測した。ステッピングモーターはその原理から、入力されるパルス周波数が早くなると、パルスに回転が追従できなくなり、脱調を起こす。最大回転数の計測では、パルス幅を 10ms から 1ms ずつ減少させる。モーター負荷によって脱調するパルス幅が変化することが考えられるため、ステッピングモーターには、タイミングベルトを取り付け、実際と同じ負荷で実験をした。ステッピングモーターが 10 回転する時間を計測し、その結果を表 3-1 に示す。表より、パルス幅が 6ms までは、理論値どおりの速度で回転しているが、パルス幅が 5ms の時に大幅に回転数が下がり、10 回転分のパルスを入力しているにもかかわらず、10 回転未満で停止した。この実験からパルス幅が 5ms で脱調している事がわかった。以上のことからステッピングモーターの 1 ステップ角度の 1.8° 回転する最短の時間は 5ms である。1 回転のパルス数は 200 パルスであり、最短のパルス幅 5ms より、最大回転数は 1rps である。ステッピングモーターにタイミングベルトで接続される、センサー部（4 分割フォトダイオードと凸レンズ）は回転速度を 1/9 に減速してあるため、センサー部の回転速度は約 0.2π [rad/s]となる。よって、製作したサントラッカーによって追従できる光源の移動角度は 0.2π [rad/s]以下であると考えられる。

表3-1　パルス幅ー回転時間

パルス幅	秒数	理論
50	96	100
30	58	60
20	39	40
10	20	20
9	18	18
8	16	16
7	14.2	14
6	12.3	12
5	10回転せず	10

3.1.2. 実験概要

製作したサントラッカーの動作を確認するために、白熱電球を追従させる実験を行った。実験の様子は図 3-1 に示す。光源には 74W の白熱電球を使用し、実験は一軸のみで行った。ギヤードモーターに 1m のアルミ棒を取り付け、アルミ棒の先端には白熱電球を取り付けた。ギヤードモーターは赤外線リモコンによって操作が行える。回転速度は PWM 制御によって行った。

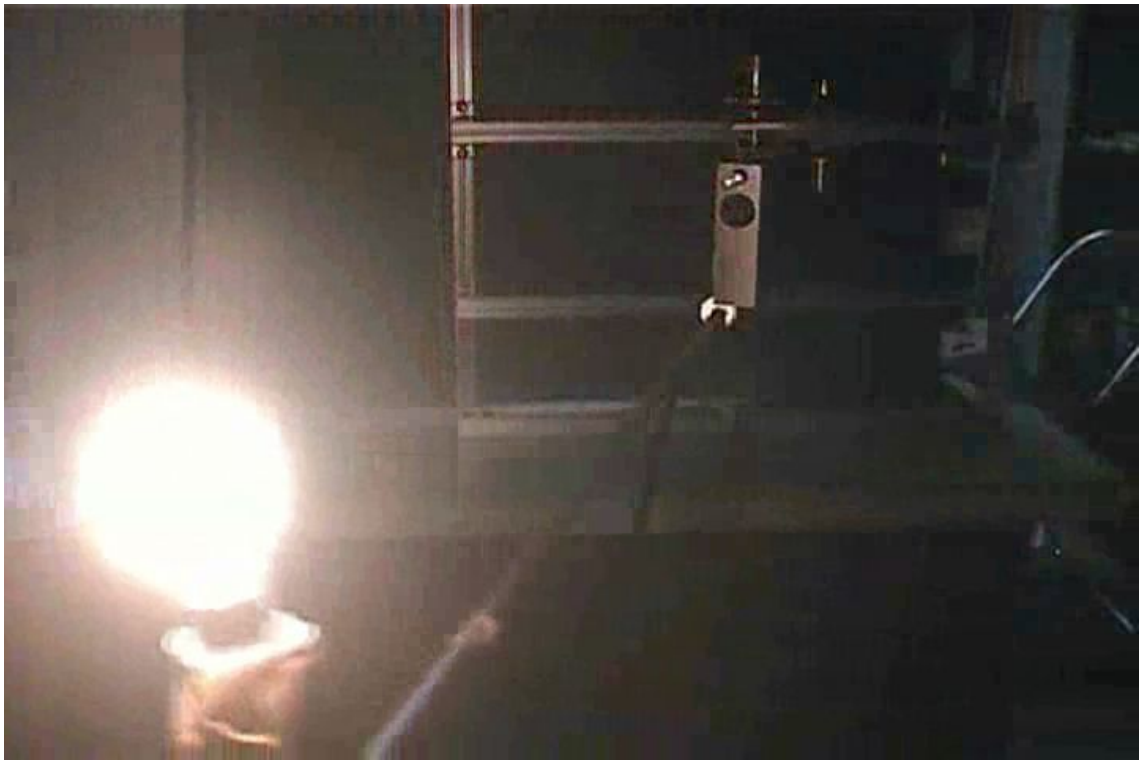

図 3-1　白熱電球による追従実験の様子

3.1.3. 実験結果

光源の移動に応じて、PD 部が回転し光源を追従しているのが確認できる。光源の移動速度を増やし、0.5rad/s で追従することが確認できた。それ以上の速度では、光源を見失うことが多かった。しかし、本研究では追従対象が太陽であるため、十分な追従速度である。

しきい値を 0V に設定すると、PD 部は振動するのが確認できた。しきい値を 0V から 1V ずつ、増やしていくと 1V で振動がなく追従出来ることが確認できた。しきい値を増やした時の問題として、光源の急激な動きに追従しにくくなる事が挙げられる。しかし、本研究では、追従対象が太陽であるため、問題ないと言える。

以上の結果により、製作したサントラッカーは太陽を追従するのに十分な性能を持つことが確認できた。

3.2. 太陽の追従実験

3.2.1. 実験方法

本校、電子機器実験室の窓際にて、太陽追従実験を行った。PD を手動にて太陽に正対させ、サントラッカーの電源を入れ、追従の様子を記録した。白熱電球の追従実験に使用したシステムでは 1 軸であるが、太陽を追従するために 2 軸システムに拡張した。PD を太陽に向け、最大電圧が 5V を超えることがないように、電流・電圧変換回路の変換率、増幅率はそれぞれ、1.8×10^3 と 10 に設定した。ステッピングモーターの回転パルス幅は θ 方向が 5ms、ϕ 方向が 10ms に設定した。

3.2.2. 実験結果

しきい値は追従しながら変化させ、振動することなく追従する最適値を模索したところ、太陽高度角が高い場合は 1V の時に振動することなく追従できた。

太陽を追従中の方位角方向(φ方向）の素子 a,d の電圧変化を 1 秒間隔で記録した結果を図 3-2 に示す。太陽の移動に対応して、移動方向の素子 d の電圧は下がり、移動方向とは逆の素子 b の電圧が上がっている。両素子の差がしきい値の 1V を超えたとき、PD が回転して太陽にほぼ正対し、両素子の電位差は縮まった。図 3-2 では、PD の回転の 1 ステップ回転する時間は約 40 秒であった。この時刻において、太陽が方位角方向へ 0.2° 移動する秒数は 40 秒であり、PD の追従速度と一致する。

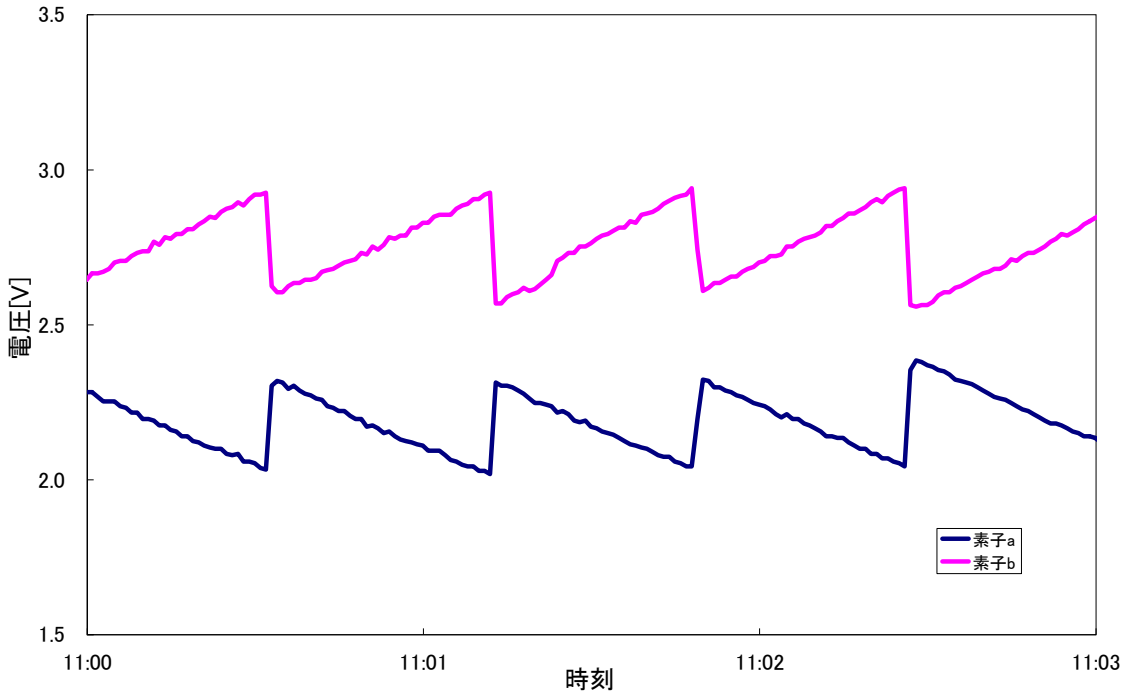

図 3-2　太陽追従中の電圧変化

3.2.3. 考察

太陽高度角が下がるに従って、地上で観測される太陽の放射照度が下がる。それに伴い、PD 出力電圧も低下する。PD 出力電圧が下がると、太陽の移動に対する PD 出力電圧の変化が鈍くなる。PD 出力電圧が 1V を下回った付近から、しきい値が 1V では太陽を見失うことが多く、うまく追従することが出来なかった。その対策として、電流・電圧変換回路の変換×増幅係数を変更し、増幅率を上げる手段がある。しかし、回路中の 4 素子分の抵抗値を動作中に等しく変更するのはあまり現実的ではない。もう一つの手段として、しきい値を小さくする方法がある。この実験では PIC のプログラムを変更して、しきい値のパラメータを変更して対応したが、太陽高度角やフォトセンサー電圧によって自動でしきい値が変化する制御プログラムが望ましい。

また、太陽を追従中に太陽が雲に隠れる事がある。太陽が雲に隠れると、エアロゾル濃度計測の為のフォトセンサーのデータとして無効になるばかりでなく、濃い雲では、サントラッカーが太陽を見失うことがあった。その場合、雲が移動し、追従が可能になった時に人間が再び太陽の方へ手動で PD を向ける必要が出てくる。太陽を追従中に太陽が薄い雲に隠れた場合の素子 b、d の電圧変化を 1 秒間隔で記録した結果を図 3-3 に示す。波形の形は図 3-2 に似ているが、波形全体に凸凹が多い事がわかる。太陽の移動に伴う両素子の電圧変化は逆の動きになるが、雲に伴う電圧変化は、両素子の電圧変化は同じ動きになる。電圧変化の微分値を比較することで、雲の検出を行う事が可能である。これによって、雲を検出したら軌道計算による追従に切り替えるなどの発展した制御プログラムが実現できる。

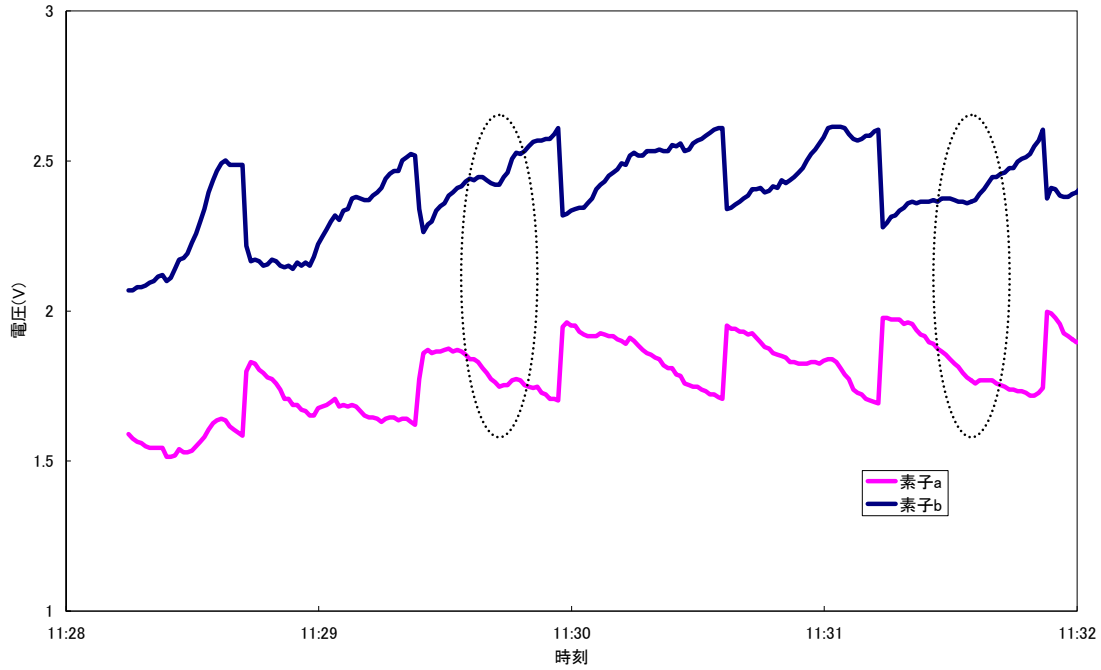

図 3-3　追従中の雲による影響

4.　放射照度観測結果

2007 年 2 月 7 日 10 時 18 分から 16 時 35 分まで、1 秒ごとにフォトセンサー出力電圧を記録した。この計測時間中に雲による影響はなかった。計測結果を図 4-1 に示す。図？？の縦軸は電圧であり、電流電圧変換回路の、変換率×増幅率よりフォトダイオードの出力電流を計算した。フォトダイオードの受光感度は波長により異なるが、太陽のスペクトルは 500nm~600nm 付近のスペクトル成分が強いため、500nm 付近の受光感度 0.28A/W を使用して、電流からエネルギー量、すなわち、太陽の放射照度へ変換した。図 4-2 に富山の緯度 N36° 42′ での大気圏外放射照度と計測した放射照度の変化を示す。大気圏外照度は大気がない場合での計算であり、地上での観測では大気による減衰がある。この二つの波形の傾向は似ていることがわかる。

大気の全光学的厚さτ(λ)を求めるためには大気がない状態での太陽放射照度 $I_0(\lambda)$ と観測した地上での太陽放射照度 I(λ)の差を求めればよい。大気の全光学的厚さからエアロゾル以外の要因を差し引くことで、エアロゾルによる光学的厚さを求めることが出来る。

透過率 T と光学的厚さτの関係はランバート・ビアの法則

$$T = \exp(-\tau)$$

により表される。

この式から透過率 T は $I_0(\lambda)$- I(λ)で求められ、ランバート・ビアの法則の両辺対数をとり、τについて変形すると

$$\tau = -\ln T$$

になり、光学的厚さτが求められる。実際の観測では太陽高度は時間とともに変化し、太陽光が通過する大気中の光路が変化する。そのため、どの時間でも光路の長さが一定となるように、補正する必要がある。天頂角 0° の時の光路の長さを 1 としたとき、天

$$m(\theta) = \frac{1}{\cos\theta}$$

頂角 θ の時の光路の長さ m(θ)は

で表される。m(θ)はエアマスと呼ばれる。エアマスを適用し、光路の長さを補正すると、τ は

$$\tau = \frac{-\ln T}{m(\theta)}$$

で表される。この式から観測したデータの τ を求めたものを図 4-3 に示す。

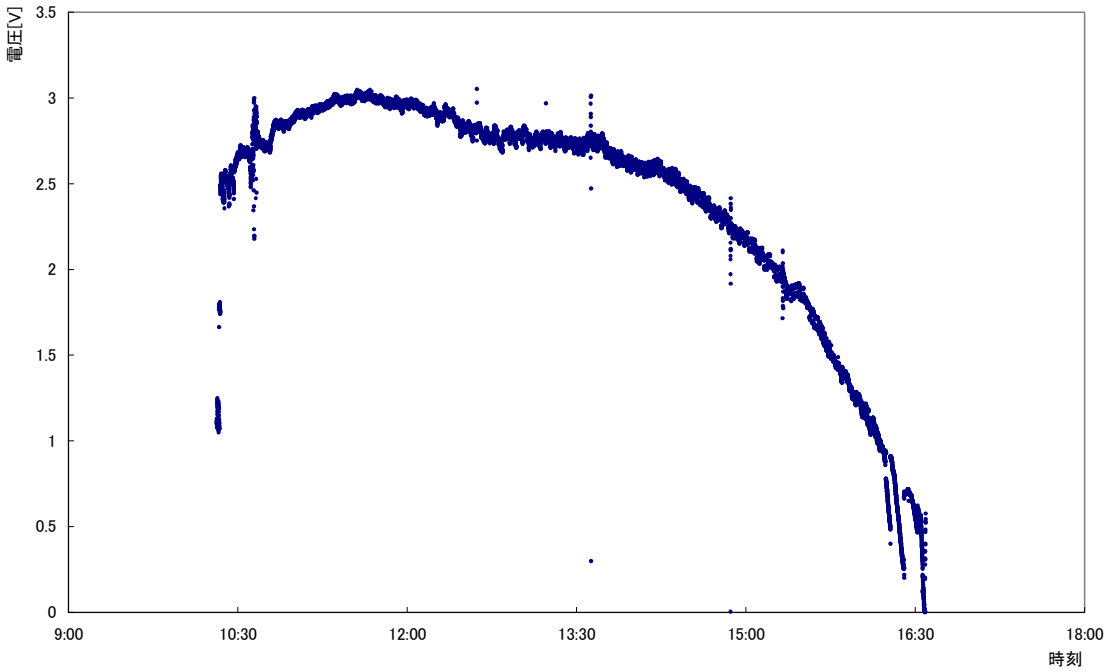

図 4-1　計測データ

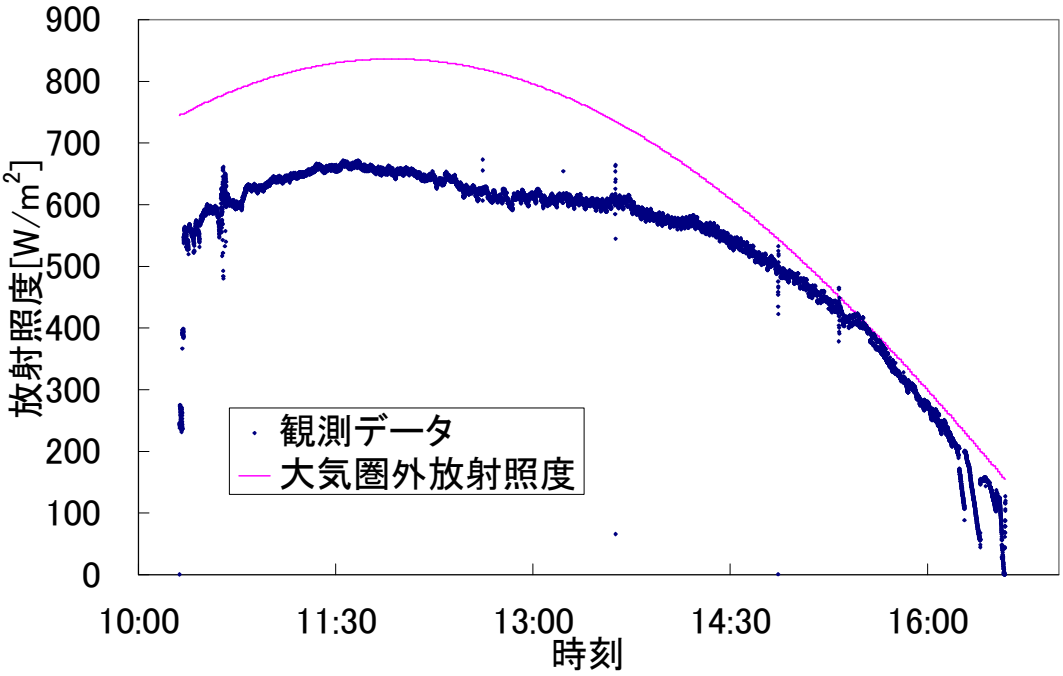

図 4-2　観測した太陽放射照度

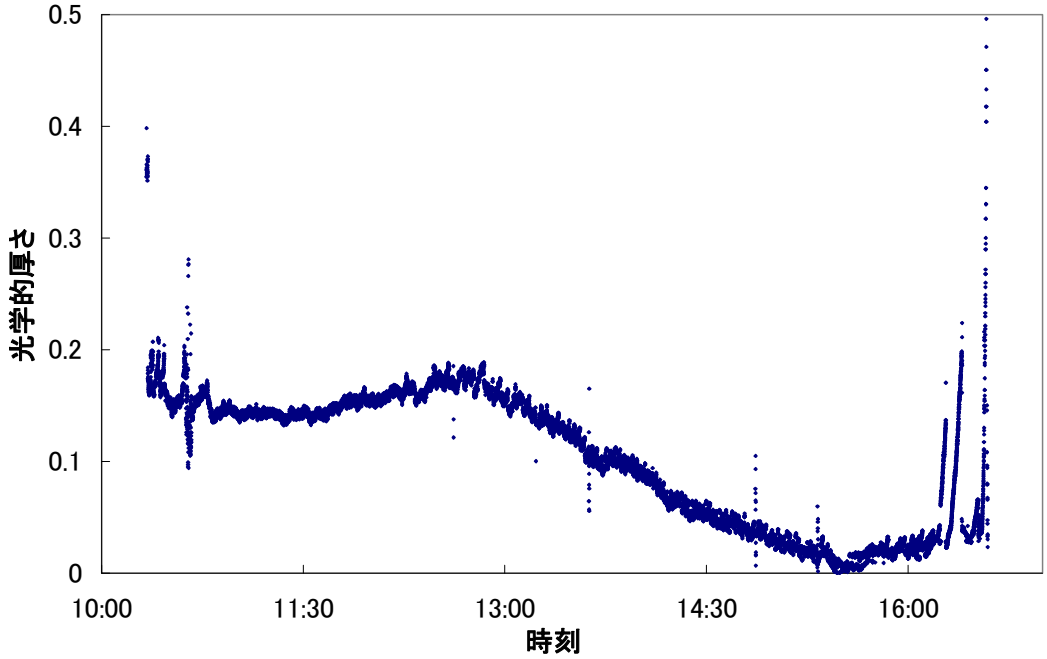

図 4-3　大気光学的厚さ

5.　まとめ・課題

サンフォトメーターの製作を行った。製作したサンフォトメーターで太陽を自動追尾

し、太陽の放射照度を計測することが可能になった。今後は継続的に太陽の放射照度を計測し、エアロゾル濃度変化を明らかにしていきたい。

しかし、製作したサンフォトメーターでは、観測の始め、雲により太陽を見失ったとき、太陽の高度角の変化に伴うしきい値の変更など、追尾中に人による手助けが必要となる。サンフォトメーターとして、以下のような機能があると実用性が高い。

- 日の出とともに自動的に追尾・観測を開始し、日の入りとともに追尾を終了する。
- 太陽の高度角に応じて、自動的に最適なしきい値が設定される。
- 雲に太陽が隠れて一時的に太陽を見失ったとしても、軌道計算により自動的に追尾を再開する。

このような点が今後の課題として挙げられる。

6. 参考文献

1)中尾　司：C 言語で作る PIC 電子工作，CQ 出版

2)日本リモートセンシング研究会：図解リモートセンシング，日本測量協会

7. 謝辞

本研究を行うにあたり、終始変わらぬご指導をしていただいた、本校電子制御工学科由井四海教員や実習工場の技官の方々に心から感謝いたします。

付録

・MotorDrive 関数

```
/*motor drive*/
void m1drive(int wtime,int step,int rot);
void abc(int wtime,int step,int rot);
void m1drive(int wtime,int step,int rot){
        int state2;
        static int position;
        while(step>0){
                switch(position){
                        case 12:
                        if(rot==1){
                        state2 = 9;
                        position = 9;
                        }
                        else{
                        state2 = 6;
                        position = 6;
                        }

                        PB = state2;
                        Wait(wtime);
                        step--;
                        break;
```

```
case 9:
if(rot==1){
state2 = 3;
position = 3;
}
else{
state2 = 12;
position = 12;
}
PB = state2;
Wait(wtime);
step--;
break;

case 3:
if(rot==1){
state2 = 6;
position = 6;
}
else{
state2 = 9;
position = 9;
}
```

```
PB = state2;
Wait(wtime);
step--;
break;

case 6:
if(rot==1){
state2 = 12;
position = 12;
}
else{
state2 = 3;
position = 3;
}
PB = state2;
Wait(wtime);
step--;
break;

default:
state2 = 9;
PB = state2;
Wait(wtime);
step--;
```

```
                position = 9;
                break;
            }
            if(step == 0)return;

        }


}


void abc(int wtime,int step,int rot){
        int state2;
        /*static*/ int position2;
        while(step>0){
            switch(position2){
                case 12:
                if(rot==1){
                state2 = 9;
                position2 = 9;
                }
                else{
                state2 = 6;
                position2 = 6;
```

```
}

PC = state2;
Wait(wtime);
step--;
break;

case 9:
if(rot==1){
state2 = 3;
position2 = 3;
}
else{
state2 = 12;
position2 = 12;
}
PC = state2;
Wait(wtime);
step--;
break;

case 3:
if(rot==1){
state2 = 6;
```

```
position2 = 6;
}
else{
state2 = 9;
position2 = 9;
}
PC = state2;
Wait(wtime);
step--;
break;

case 6:
if(rot==1){
state2 = 12;
position2 = 12;
}
else{
state2 = 3;
position2 = 3;
}
PC = state2;
Wait(wtime);
step--;
break;
```

```
                default:
                state2 = 9;
                PC = state2;
                Wait(wtime);
                step--;
                position2 = 9;
                break;
        }
        if(step == 0)return;

    }


}

/*motor2 drive*/

void m2drive(int wtime,int step,int rot);


void m2drive(int wtime,int step,int rot){
    int state;
    static int position2;
```

```
while(step>0){
        switch(position2){
                case 12:
                if(rot==1){
                state = 9;
                position2 = 9;
                }
                else{
                state = 6;
                position2 = 6;
                }

                PC = state;
                Wait(wtime);
                step--;
                break;

                case 9:
                if(rot==1){
                state = 3;
                position2 = 3;
                }
                else{
                state = 12;
```

```
position2 = 12;
}
PC = state;
Wait(wtime);
step--;
break;

case 3:
if(rot==1){
state = 6;
position2 = 6;
}
else{
state = 9;
position2 = 9;
}
PC = state;
Wait(wtime);
step--;
break;

case 6:
if(rot==1){
state = 12;
```

```
                position2 = 12;
                }
                else{
                state = 3;
                position2 = 3;
                }
                PC = state;
                Wait(wtime);
                step--;
                break;

                default:
                state = 9;
                PC = state;
                Wait(wtime);
                step--;
                position2 = 9;
                break;
        }
        if(step == 0)return;

}
```

```
}
```

・ AD 変換関数

```
/*ad_convert*/

#define ADC_START              ADCON0|=(1<<GO)                    // AD
#define TST_ADC_NOT_DONE (ADCON0&(1<<GO))                         // AD


void SelADCChannel(int ch);
WORD ReadADCValue(void);



WORD ad_convert(int port){
        WORD val;
        SelADCChannel(port);
        ADC_START;
        while(TST_ADC_NOT_DONE) {}
        val = ReadADCValue();

        return(val);
}
```

```
void SelADCChannel(int ch) {
        BYTE chn;
        BYTE reg;
        chn = (ch << 3) & 0x38;
        reg = ADCON0 & ~0x38;
        reg |= chn;
        ADCON0 = reg;
}



WORD ReadADCValue(void) {
        WORD adval;

        adval = (WORD)ADRESH << 8;
        adval = adval | ADRESL;
        return adval;
}
```

・User 関数

```
#include "Z:¥¥mydocument¥¥kaba¥¥step2¥¥step_Auto.h"
#include <maths.h>
#include<strings.h>
#__config 0x3F36
#include "ad_convert.h"
```

```
#include "m1drive.h"
int Compare(WORD val1,WORD val2);
//
// This file includes all user definable routines. It may be changed at will as
// it will not be regenerated once the application has been generated for the
// first time.
//

//*************************************************************************
***
//
// Insert your interrupt handling code if required here.
// Note quick interrupts are used so code must be simple
// See the manual for details of quick interrupts.
//

void UserInterrupt()
{
        // Insert your code here

// #asmline SETPCLATH UserIntReturn,-1    ; SETPCLATH for interrupt routine
  //#asmline goto UserIntReturn             ; Assembler - go back to interrupt routine
}
```

```
//*************************************************************************
***
//
// Insert your initialisation code if required here.
// Note that when this routine is called Interrupts will not be enabled - the
// Application Designer will enable them before the main loop
//

void UserInitialise()
{
}

//*************************************************************************
***
//
// Insert your main loop code if required here. This routine will be called
// as part of the main loop code
//

void UserLoop()
{
```

```
static int adval1;
static int adval2; //ad
static int adval3;
static int adval4;
static int com;
static int com2;
adval1=ad_convert(0);
adval2=ad_convert(1);



//com=2;
com=Compare(adval1,adval2);
//adval1=0;
//adval2=0;




if(com==2){
        m1drive(5,1,1);
}
else if(com==1){
        m1drive(5,1,0);
}
else{
```

```
        }
        adval3=ad_convert(2);
        adval4=ad_convert(3);
        com2=Compare(adval3,adval4);
        if(com2==2){
                m2drive(10,1,1);
                        }
        else if(com2==1){
                m2drive(10,1,0);
        }

        else{
        }
}
//
// User occurrence code
//
void adfinish(){
}
#define TH 1.0
int Compare(WORD val1,WORD val2){

        double a=0.004883,b1,b2;
        b1=a*val1;
```

```
        b2=a*val2;
        if(fabs(b1-b2) < TH){
                return(0);
        }
        else if(b1<b2){
                return(1);
        }
        else if(b1>b2){
                return(2);
        }
        return(0);
}
```