センサと制御　実験Ⅰ　実験マニュアル

1. 目的および概要

(1) Arduino IDE によるサンプル・プログラム動作の確認

(2) センサ（圧力センサ、赤外線距離センサ）の特性の測定

(3) サーボの駆動および特性の測定

(4) 応用課題

2. 使用機器および準備

2.1 使用機器

(1) 実験Ⅰ実験装置

(2) PC Dynabook R741D（東芝）

ソフトウェア開発環境：Arduino IDE

(3) オシロスコープ　DLM2024（横河）

(4) デジタルマルチメータ　VOAC7413（岩通）

(5) 電源　LX018-2A（高砂）

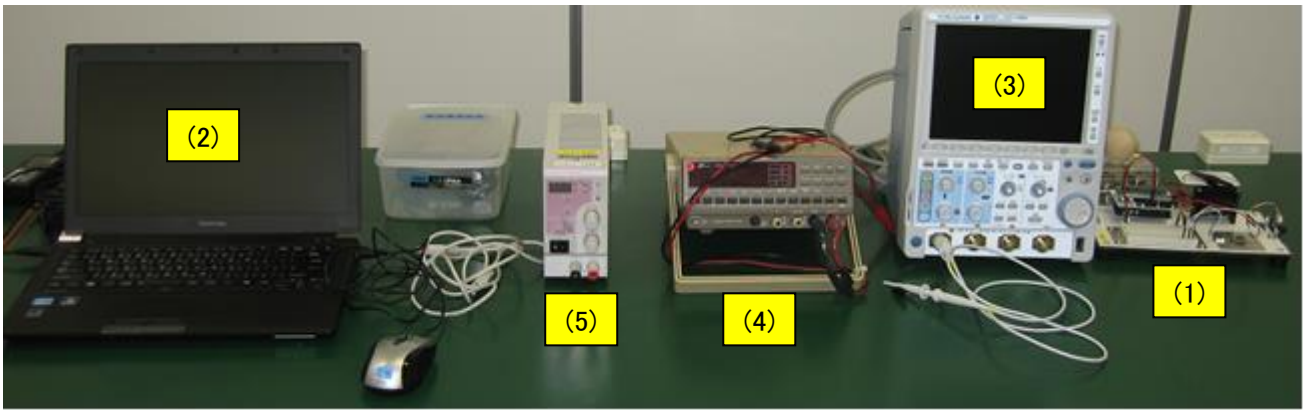


図 2-1　実験装置外観

実験 I 実験装置構成品（図 2−２）

(a) Arduino Uno + LCD シールド

(b) サーボ：HS-5496MH（HiTEC 社製）

(c) 圧力センサ：FSR-402（Interlink Electronics 社製）

(d) 赤外線距離センサ：GP2YOA21YK（シャープ製）

(e) プロトボード

(f) ボール（木製）

(g) 錘り （10g / 20g）

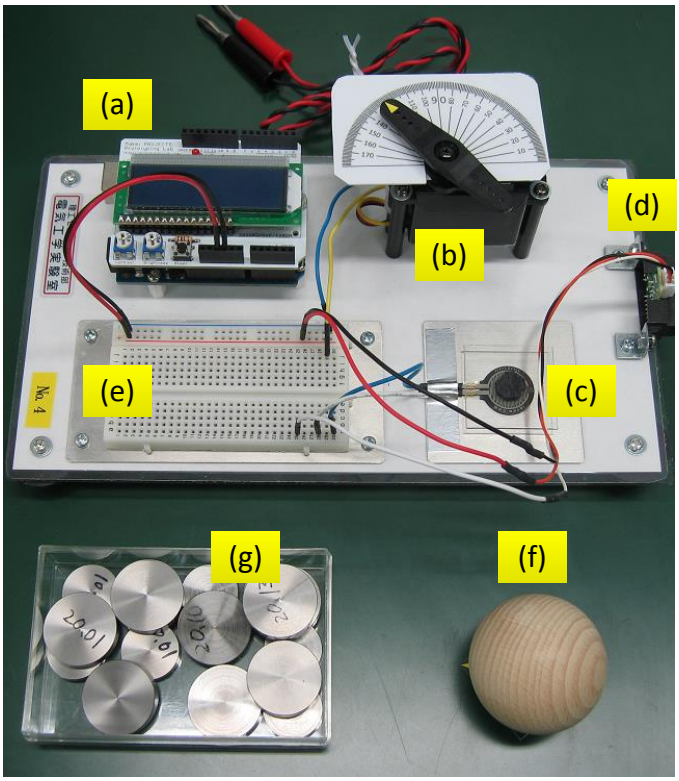


図 2-2

2.2 準備

サーボ用電源(5)の出力電圧を 6.5V に設定する。

ブレッドボードの使い方については、Appendix 3 を参照。

3. 実験手順

3.1 Arduino IDE による最初のプログラム動作の確認

以下の手順で（テキスト 3.(5)項参照）、LCD にメッセージを表示させる。

Arduino IDE をインストール済みの Windows PC と Arduino を USB ケーブルで接続し（図 3-1）、Windows を立上げ、デスクトップの Arduino の ICON をダブルクリックすると、図 3-2 のようにプログラム（スケッチ）を作成するウィンドウが現れる。

図 3-3 のプログラム（LCD にメッセージを表示するスケッチ）を入力する。

入力が終わった後、コンパイルアイコンを押す。問題がなければコンパイルが終了し、Window 下部のメッセージ領域に「コンパイル後のスケッチサイズ： ...」と表示される。

次にファイル転送アイコンを押すと、Arduino に実行ファイルが転送され、転送後に Arduino は自動的にプログラムを開始する。

なお、作成したプログラムは、以下のディレクトリに保存してよいが、最後に各自の USB メモリに保存して持ち帰ること。ライブラリ￥ドキュメント￥Work￥

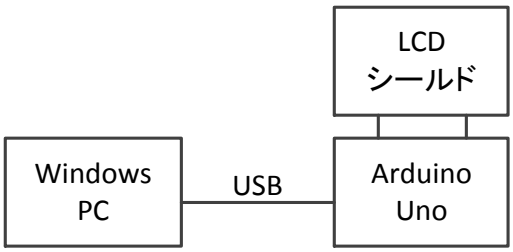


図 3-1　Arduino と PC との接続図

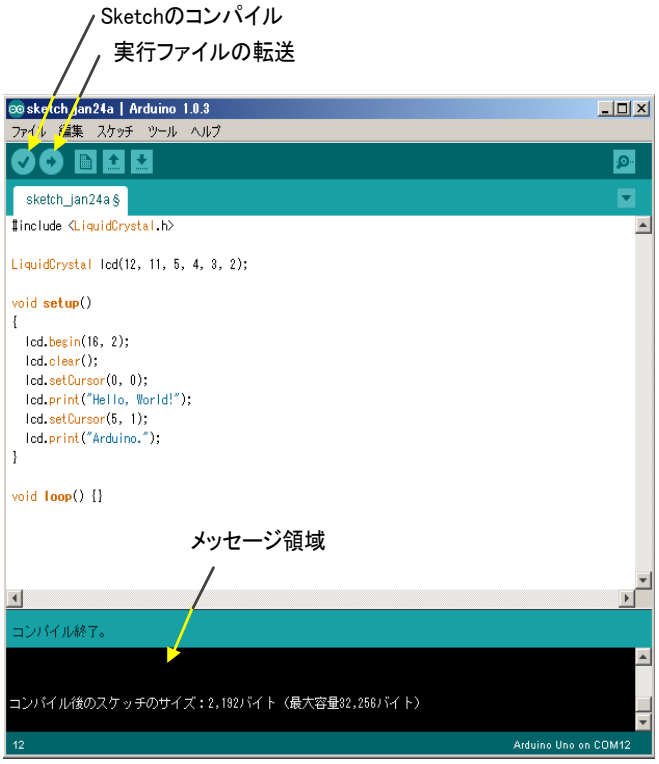


図 3-2　Arduino IDE の画面

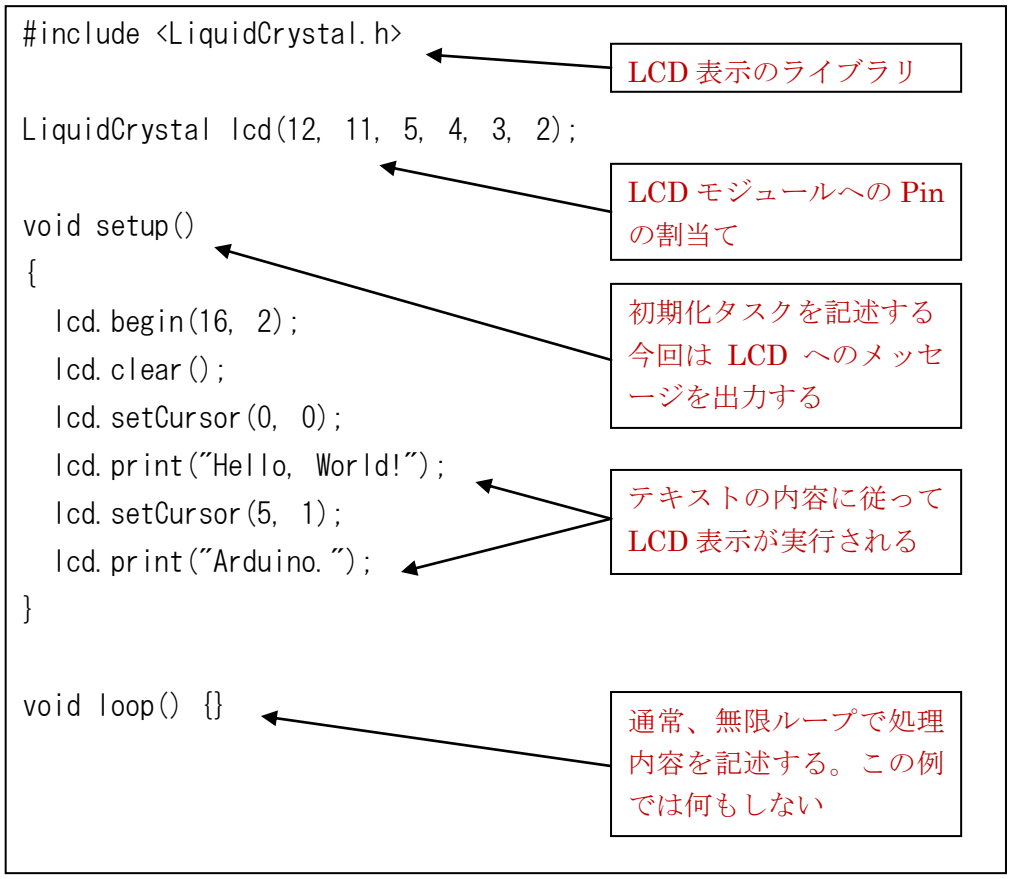


図 3-3　サンプル・プログラム（スケッチ）1

3.2 圧力センサの特性の測定

(1) センサ単体の測定

圧力センサの出力をマルチメータに接続し、錘を乗せた時の抵抗値を測定する。(図 3-4)

また、ボールを乗せた時の抵抗値を測定する。(表 3-1 参照)

質量：30 / 40 / 50 / 60 / 70 / 80 / 90 / 100 / 120 [g]

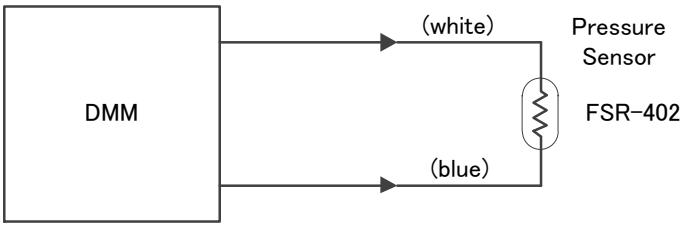


図 3-4

(2) Arduino Uno による測定

プロトボードに図 3-5（左）圧力センサの測定回路を組み、Arduino でアナログ電圧入力を行い、LCD に表示するプログラムを作成する。(1)項と同じ質量の錘を乗せて、電圧を測定する。

R = 47 kΩ とし、圧力センサの抵抗値を計算して、LCD に表示（単位：kΩ）すること。

サンプル・プログラム（スケッチ）2-1（図 3-6）を基にして、プログラムを作成すること。

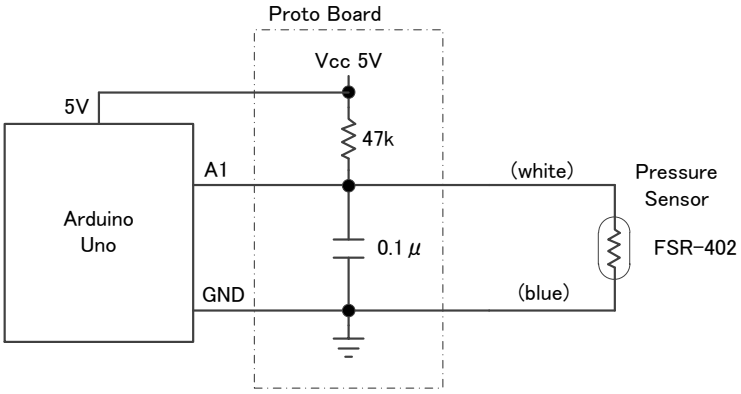
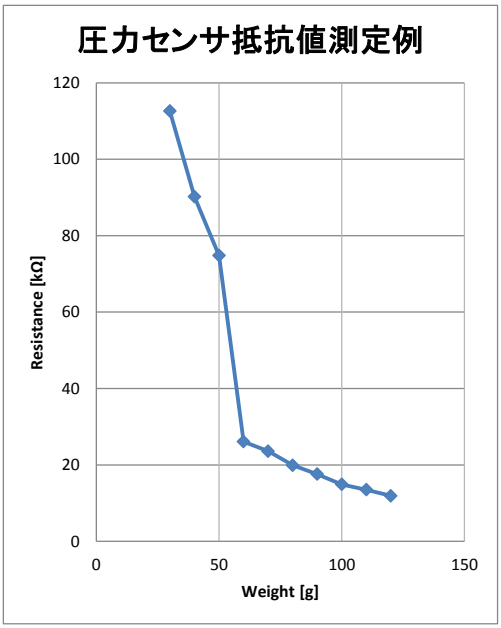


図 3-5

表 3-1　圧力センサ測定

質量 [g]	DMM 測定値 [kΩ]	Arduino 表示値[kΩ]
30	*	*
40		
50		
60		
70		
80		
90		
100		
120		
BALL		

*注：30g の場合、抵抗値は測定不能の場合がある。



```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);          // LCD のピンを指定

void setup()
{
  lcd.begin(16, 2);                              // LCD ライブラリ初期化
}

void loop() {
  float analogin1;
  float Rsensor;                                 // Rsensor: センサー抵抗値[kΩ]


  analogin1 = float(analogRead(1));              // アナログ入力　ピン A1

  Rsensor =
```

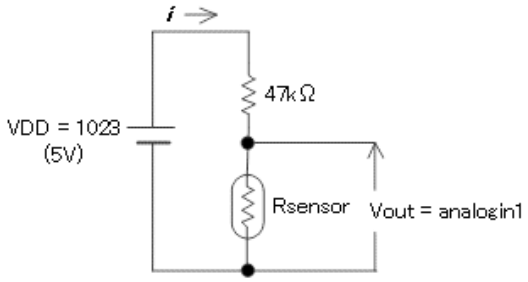
※ Analogin1 から、センサー抵抗値 Rsensor[kΩ]を求める式を書く

```
  lcd.clear();                                   //  LCD 表示クリア
  lcd.setCursor(0, 0);                           //  カーソルに移動
  lcd.print("Ain: ");                            //  "Ain: "を表示
  lcd.print(Rsensor);                            //  Rsensor を表示

  delay(250);                                    //  250ms ウェイト
}
```

図 3-6　サンプル・プログラム（スケッチ）2-1

※ センサー抵抗値の計算：下の関係から、Rsensor を計算する式を求める



analogin1 の入力電圧は AD 変換され、5V 入力時、値が 1023 になるようにスケーリングされる。

3.3 赤外線距離センサの特性の測定

プロトボードに図 3-7 の測定回路を組み、Arduino で、赤外線距離センサからのアナログ電圧入力を行い、**LCD に電圧値を表示**するプログラムを作成する。（図 3-9 サンプル・プログラム（スケッチ）2-2）

測定用シートを使用し、ボールを指定の位置に置いた時の、表示電圧を記録する。

（図 3-8 および表 3-2 参照）

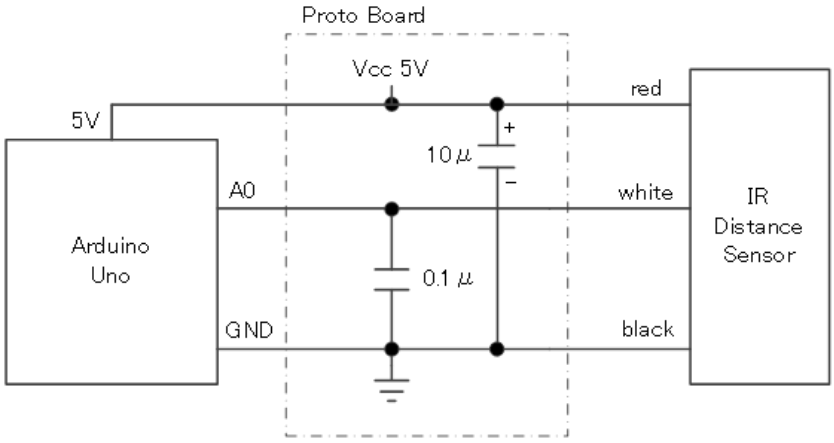
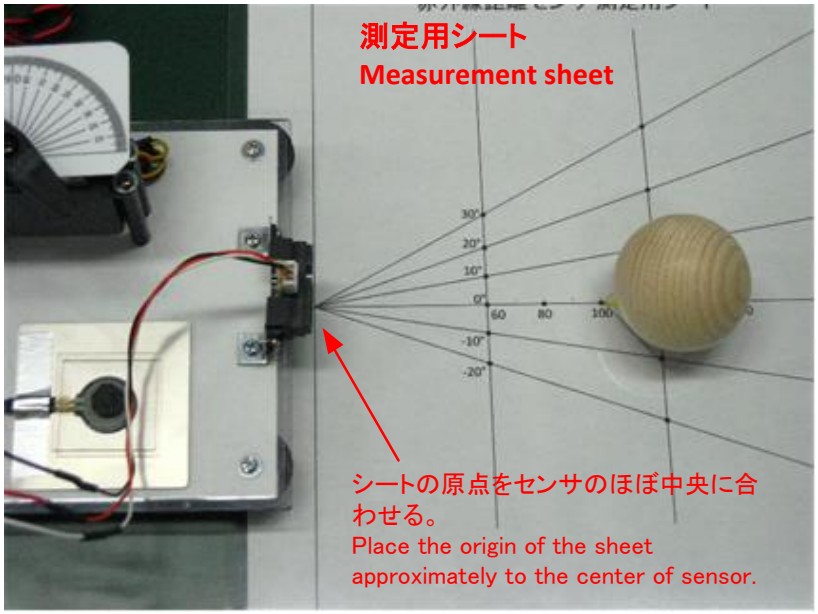
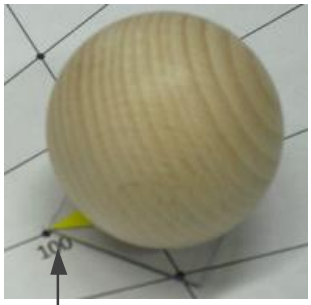


図 3-7





黄色い矢印の先端を測定ポイントに置く
Place the edge of yellow allow on the point to be measured.

図 3-8

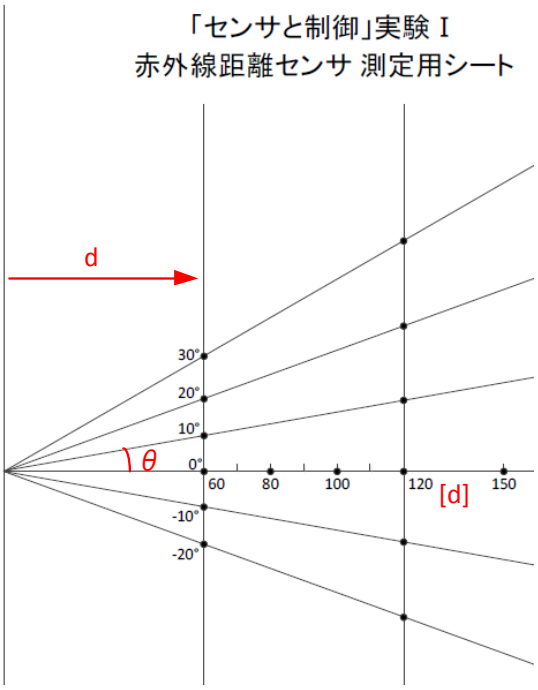
表 3-2　赤外線距離センサ　出力電圧測定

θ＝0°における距離－電圧特性

距離 d[mm]	電圧[V]
60	
80	
100	
120	
150	
200	
250	
300	

距離 d＝一定の時の角度θ依存性

θ [deg]	電圧[V]	
	d=60 mm	d=120 mm
30		
20		
10		
0		
−10		
−20		



```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);      // LCD のピンを指定

void setup()
{
  lcd.begin(16, 2);                          // LCD ライブラリ初期化
}

void loop() {
  float analogin0;
  float Vsensor;                             // Vsensor: センサー出力電圧[V]


  analogin0 = float(analogRead(0));          // アナログ入力　ピン A0

  Vsensor =   ※ Analogin0（Full Scale＝1023@5V）を
              電圧値に変換する式を書く。*

  lcd.clear();                               //  LCD 表示クリア
  lcd.setCursor(0, 0);                       //  カーソルに移動
  lcd.print("Ain: ");                        //  "Ain: "を表示
  lcd.print(Vsensor);                        //  Vsesnor を表示

  delay(250);                                //  250ms ウェイト
}
```

図 3-9　サンプル・プログラム（スケッチ）2-2

※赤字部分が、サンプル・プログラム（スケッチ）2-1 と異なる。

*今回は、5V 入力の時、値が 1023 になると仮定してよい。

実際には、Arduino の A/D 変換の基準電圧[default]は USB 電源の 5V のため、±10%程度の誤差がありうる。

3.4 サーボの駆動

プロトボードに図 3-10 のようにサーボを駆動する回路を作成する。

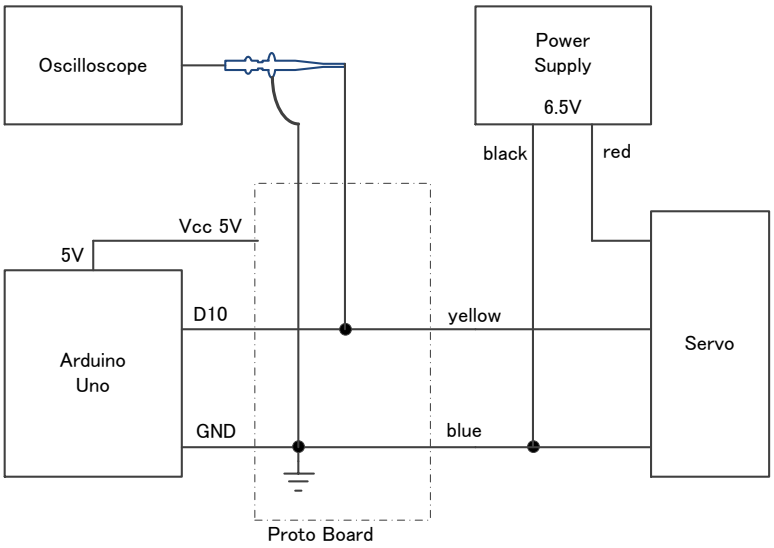


図 3-10

サンプル・プログラム（スケッチ）3（図 3-13）を完成させて、以下の角度に相当するパルス幅で駆動する。

(1) オシロスコープで以下の測定を行う。

a) 角度 90° にて、周期 T を測定し、画面を各自の USB メモリーに記録する。
画面コピーが正しく動作しない場合は、Appendix4 を参照。なお、紙に出力してもよい。

b) 角度 90° にて、パルス幅 T w を画面から読み取る。（画面のコピーは不要）

c) 同様に角度 30° にて、パルス幅 T w を読み取る。

d) 同様に角度 150° にて、パルス幅 T w を読み取る。

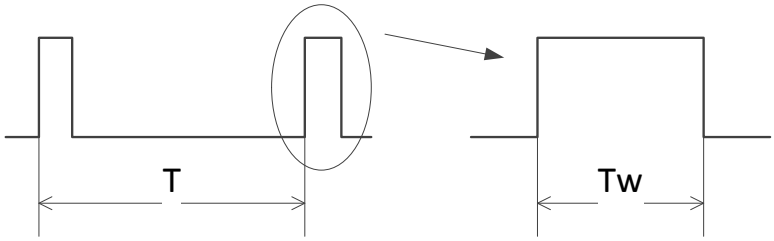


図 3-11　PWM 波形

(2) サーボの回転角をスケールで読み取る。（図 3-12 参照）

サーボ駆動角度：20° / 30° / 40° / 60° / 90° / 120° / 140° / 150° / 160°

図 3-12

表 3-3　サーボ測定

設定値		スケール指示値	オシロ測定値
角度 Angle[deg]	PWMout [us]	[deg]	Tw [us]
20	2200		―
30	2100		
40	2000		―
60	1800		―
90	1500		
120	1200		―
140	1000		―
150	900		
160	800		―

周期T測定値＝

サンプル・プログラム（スケッチ）3 の内容および修正方法：

サーボの角度を設定するために、PC からシリアルライン（USB）を通じて、コマンドを送り、コマンドに応じた角度相当のパルス幅を出力する。

シリアルラインの使用方法については、図 3-14 を参照のこと。

例えば、コマンドと角度の対応を表 3-4 のようにすればよい。サンプル・プログラム（スケッチ）3 では角度 Angle=90° の時のみ記述してあるので、他のコマンドについても追加して、完成させる。

なお、サンプル・プログラム（スケッチ）3 のひな形は PC の以下のディレクトリに予め用意してあるので、書き換えて（上書き可）使用してよい。

ライブラリ￥ドキュメント￥Work￥SampleSketch3￥SampleSketch3.ino

表 3-4

コマンド	角度 Angle[deg]	PWMout [us]
2	20	2200
3	30	2100
4	40	2000
6	60	1800
9	90	1500
c	120	1200
e	140	1000
f	150	900
g	160	800

★★★★★　このサンプルスケッチは、ひな形が PC にインストールされている。

→ ライブラリ￥ドキュメント￥Work￥SampleSketch3￥SampleSketch3.ino

```
#include <LiquidCrystal.h>
#include <Servo.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
Servo servo1;                          //  servo1 をサーボ出力とする
int Angle;                             //  角度[deg]
int PWM_out;                           //  PWM 出力パルス幅[us]

void setup()
{
  lcd.begin(16, 2);
  servo1.attach(10);                   //  servo1 を D10 ピンに割り当てる
  Serial.begin(9600);                  //  シリアルの baud rate 指定
  Angle = 90;
}

void loop() {
  char Chr_in;

  if(Serial.available() > 0){
    Chr_in = Serial.read();            //  シリアル入力があれば、Chr_in に入力

    switch (Chr_in){                   //  Chr_in の値に応じて、Angle を設定
      case '9':
        Angle = 90;
        break;

        他のコマンド（表 3-4 に示す文字）
        について記述する。

    }
  }
  PWM_out = 1500 - 10 * (Angle - 90);      //  出力パルス幅を設定

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Srv:");
  lcd.print(Angle);
  lcd.setCursor(8, 0);
  lcd.print(PWM_out);

  servo1.writeMicroseconds(PWM_out);

  delay(250);
}
```

図 3-13　サンプル・プログラム（スケッチ）3

シリアルモニタの使用方法

プログラムを Arduino に書き込んだ後、ツールメニュー→シリアルモニタを選択する。

下のように、ウィンドウが表示されるので、⇒の部分をクリックして文字を入力し、「送信」ボタンを押す。

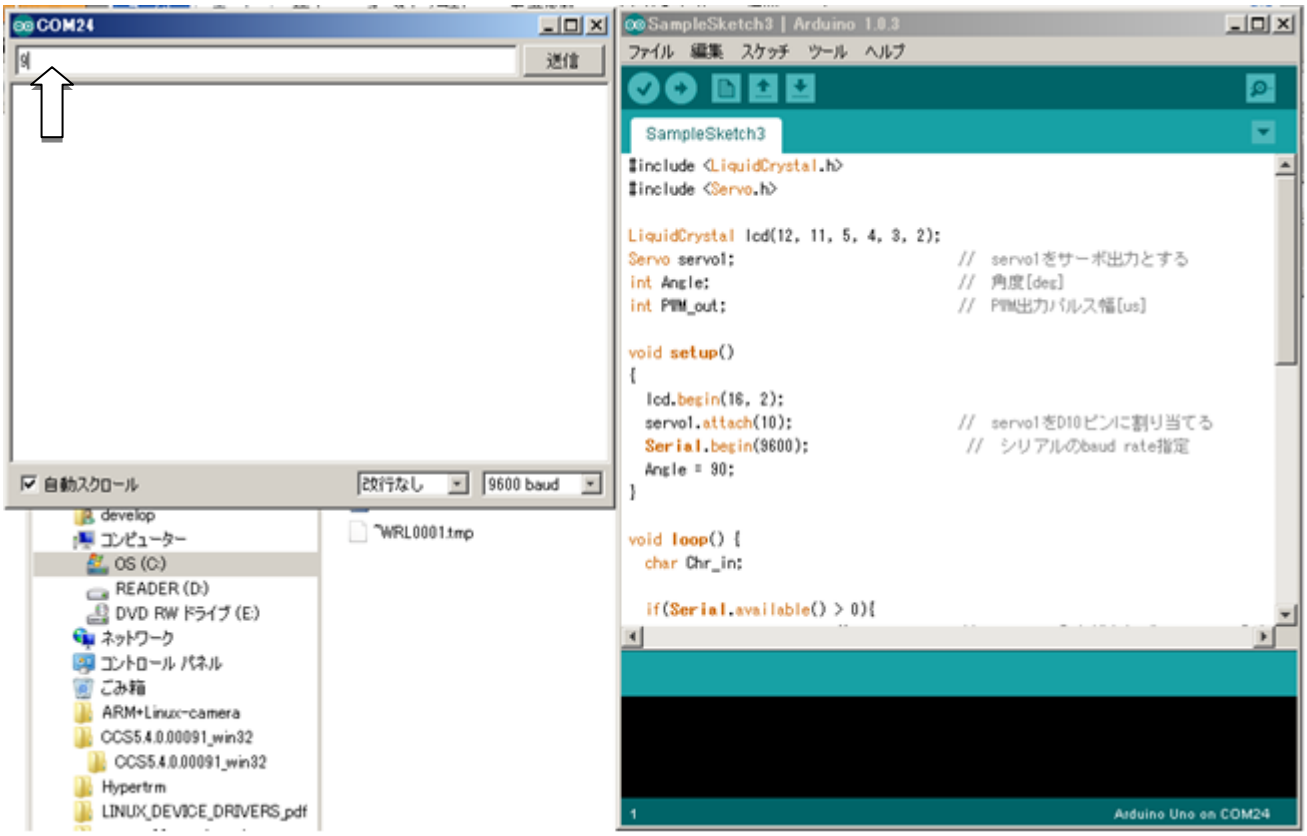


図 3-14

3.5 応用課題

(1) 圧力センサに錘を乗せた時の値[g]をサーボの角度で指示する秤プログラムの作成

例えば、50ｇ、60g・・・、100g の錘を乗せた時、サーボの角度が 50°、60° ・・・100° となるようにする。

(2) サーボの回転速度を制御するプログラム（スケッチ）の作成

サンプル・プログラム（スケッチ）4 は、メトロノームのように、サーボを左右に動かすプログラムである。

これを参考にして、サーボの動きを制御するプログラムを工夫して作成せよ。

例えば、

- 振り子のように、中央で速度が最大になるような動き。
- 時計の針のように移動と小休止を繰り返す。
- その他。

サンプル・プログラム（スケッチ）4 は、以下のディレクトリにあるので、これを修正して使用してよい。（上書き可）

ライブラリ￥ドキュメント￥Work￥SampleSketch4￥SampleSketch4.ino

4. 第 2 週目実験に向けての準備

第 1 週目実験が終わった後、テキスト p41　5.2.項「第 2 週目に向けての準備...」を**必ず事前に行って、第 2 週目実験に臨むこと**。

なお、計算に使用する座標は「センサと制御実験Ⅱ 実験マニュアル」図 3-3 を参照のこと。

アームの動きの制約から、Xoffset は、-30mm～-50mm くらいの範囲で選択する必要がある。

また、キャッチャ部の角度は水平でなく、図のようにわずかに傾ける（5° ~20° ）必要がある。

5. 報告事項

(1) 圧力センサの測定データ（表）およびグラフ（質量 vs 抵抗値）
(2) 同上プログラム・リスト*
(3) 赤外線距離センサの測定データ（表）およびグラフ（距離―電圧／角度―電圧）
(4) 同上プログラム・リスト*
(5) サーボ測定データ（オシロ画面ハードコピー、データ表、グラフ 角度－スケール指示値）
(6) 同上プログラム・リスト*
(7) 応用課題(1) プログラム・リスト*および説明（実現した機能、実現方法）
(8) 応用課題(2) 同上

プログラム（スケッチ）は、Arduino のソースファイル（.ino）を COPY して持ち帰ること。

サンプル・プログラム（スケッチ）4 の説明

(1) 機能

サンプル・プログラム（スケッチ）4 は、実験 I 用装置のサーボをメトロノームのように左右に振らせるものである。30° と 150° を境界とし、その間を往復する。

(2) 関数の機能

(a) setup

LCD、サーボのピン指定、変数の初期化を行う。DriveServo1 は構造体 ServoControl で定義され、以下の数値に初期化される。

・DriveServo1.Angle = 90　　サーボの初期角度[deg]（最初だけ指定する）

・DriveServo1.Target = 30　　サーボが（右に）回転する時の目標角度[deg]

・DriveServo1.Speed = 0.5　　10ms 毎の回転角[deg]

(b) loop

ここでは Task スケジュールを行い、以下の関数を定期的に実行する。

・Main_loop：100ms

・Message_out：100ms

・Move_servo：10ms

(c) Main_loop

時計回り（CW）に回転する State を 0 と定義し、反時計回り（CCW）に回転する State を 1 と定義して、状態遷移を制御する。(図 3-15)

State が変化する条件は、現在の角度 Angle と目標値 Angle_target との差分 Angle_move_remain が≒0 になることである。

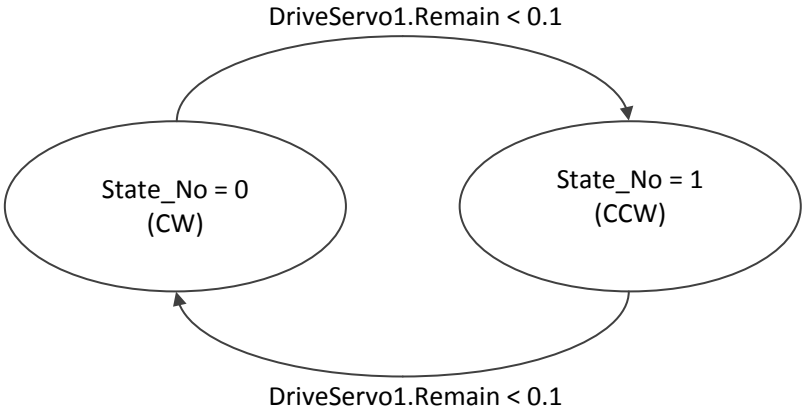


図 3-15

(d) Message_out：LCD 表示を行う。

(e) Move_servo

Servo の回転を制御する。現在の Angle から、目標値 Angle_target に向けて、増分 Angle_move_speed だけ近づけ、PWM 出力を実行する。

目標値との差分(Angle - Angle_target)の絶対値を Angle_move_remain として出力する。

```
// ****************************************************************
// サンプル・スケッチ 4 :　メトロノーム風 Servo 駆動スケッチ
// ****************************************************************
//          File Name: SampleSketch4       Ver.2
// ****************************************************************
#include <LiquidCrystal.h>
#include <Servo.h>
#include <MsTimer2.h>                  // タイマー2 割り込みを使用する
#include "types.h"                     // 構造体の宣言

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);   // LCD のピン設定

Servo servo1;                          // servo1=サーボモータ

int loop_count = 0;                    // 割込み(10ms)カウンタ
boolean t300ms_flag;                   // 300ms 毎にフラグが立つ
int State_No = 0;                      // ステート No [0 or 1]
int PWM_out;                           // サーボ駆動パルス幅 [us]

ServoControl DriveServo1;
// ServoControl 型である変数 DriveServo1 は以下のメンバー変数からなる
//     DriveServo1.Angle =  サーボの角度の現在値 [deg]
//     DriveServo1.Target =  サーボの目標値 [deg]
//     DriveServo1.Speed =  サーボが 10ms 毎に移動する角度 [deg]
//     DriveServo1.Remain =  目標値と現在地の差
void Move_servo(ServoControl DriveServo1, Servo servo1);

// ****************************************************************
// ******************** 10ms 割込み処理 ***************************
// ****************************************************************
void timer_int(){
  Move_servo(&DriveServo1, servo1); // サーボを駆動する

  loop_count++;                        // loop_count をカウントアップ
  if (loop_count == 30){               // 300ms になったら
    t300ms_flag = 1;                   // t300ms_flag を 1 にして（LCD 表示用）
    loop_count = 0;                    // loop_count を 0 にする
  }
}

// ****************************************************************
// ******************** SETUP 処理(初期設定) **********************
// ****************************************************************
void setup()
{
  lcd.begin(16, 2);                    // LCD のモード設定（16 行 2 列）
  lcd.clear();                         // LCD 表示をクリヤする
  servo1.attach(10);                   // D10 をサーボ出力に設定

  DriveServo1.Angle = 90;              // 始め 90°（中央）にする（実験 II では重要）
  DriveServo1.Target = 30;             // Target = 30°
  DriveServo1.Speed = 0.5;             // Speed = 0.5 [°/10ms]

  MsTimer2::set(10, timer_int);        // Timer2 を使用、10ms 毎に"timer_int"を呼び出す
  MsTimer2::start();                   // Timer2 の開始
```

```
}

void loop() {
  switch(State_No){
    case 0:                              // 右へ移動する State
      // *****  以下に State0 の処理を記述する



      // State0 終了の判断
      if (DriveServo1.Remain < 0.1){  // 目標角度との差分が 0.1deg 以下になったら
        State_No = 1;                 //    State_No を 1 にする
        DriveServo1.Target = 150;     //    Target 角度を設定
        DriveServo1.Speed = 0.5;      //    Speed を設定
        DriveServo1.Remain = 1;       //    この値は、0.1 より大きい任意の値とする
      }
      break;

    case 1:                              // 左へ移動する State
      // *****  以下に State1 の処理を記述する



      // State1 終了の判断
      if (DriveServo1.Remain < 0.1){  // 目標角度との差分が 0.1deg 以下になったら
        State_No = 0;                 //    State_No を 0 にする
        DriveServo1.Target = 30;      //    Target 角度を設定
        DriveServo1.Speed = 0.5;      //    Speed を設定
        DriveServo1.Remain = 1;       //    この値は、0.1 より大きい任意の値とする
      }
      break;
  }
  if (t300ms_flag == 1){               // t300ms_flag が 1 になったら
    Message_out();                     // LCD にメッセージを表示する
    t300ms_flag = 0;                   // t300ms_flag を 0 にする
  }
}

// ********************************************************************
// ********************** LCD メッセージ表示  ************************
// ********************************************************************
void Message_out()
{
    //lcd.clear();                     // (LCD 表示を毎回クリヤしたい時に実行する)
    lcd.setCursor(0, 0);               // カーソルを左上に移動する
    lcd.print("Angle: ");              // "Angle: "を表示
    lcd.print(DriveServo1.Angle);      // 現在のサーボ角度を表示
    lcd.print(" ");                    // (前回のごみを消去)
    lcd.setCursor(0, 1);               // カーソルを 2 行目に移動する
    lcd.print("State: ");              // "State: "を表示
    lcd.print(State_No);               // ステート No を表示
}
```

```
// ****************************************************************
// ****************** Move_servo(サーボ駆動) ********************
// ****************************************************************
void Move_servo(ServoControl *DSi, Servo servo_i){
//
// servo を指定の SPEED で移動（回転）する関数
// 引数：
//    DSi: ServoControl 型の構造体（参照渡し）
//       DSi.Angle: Servo に出力する角度[deg] -> 実行後、書換えられる
//          注：Angle の初期値は 90 にすること
//                （実験Ⅱで、アームの初期値を直立とするため）
//    DSi.Target: 目標角度[deg]              -> 実行後、影響されない
//       DSi.Speed: Servo 移動速度[deg/10ms]  -> 実行後、影響されない
//       DSi.Remain: 現在の角度（Angle）と目標角度との差
//                                            -> 実行後、書換えられる
//    servo_i: 対象サーボ
// ****************************************************************
   float Angle, Target, Speed, Remain;

   Angle = DSi->Angle;                  // 引数のメンバー変数を
   Target = DSi->Target;                // ローカル変数にコピーする
   Speed = DSi->Speed;

   Speed = abs(Speed);                  // Speed を常に正の値にする
   if (Target >= Angle){                // Target 角 >= 現在値(Angle)なら
      Angle = Angle + Speed;            // Angle に Speed を加算する
      if (Target < Angle){              // その結果、現在値が Target を超えたら
         Angle = Target;                // 現在値を Target とする
      }
      Remain = Target - Angle;          // Target と現在値の差を Remain とする
   }else{                               // Target 角 < 現在値なら
      Angle = Angle - Speed;            // 上記と符号を変えて、同じ演算を行う
      if (Target > Angle){
         Angle = Target;
      }
      Remain = Angle - Target;
   }
   PWM_out = 1500 - 10 * (Angle - 90); // PWM_out(サーボ駆動バルス幅)を計算
   servo_i.write(PWM_out);              // サーボを駆動する

   DSi->Angle = Angle;                  // 現在角(Angle)の計算値を引数に反映
   DSi->Remain = Remain;                // Remain の計算値を引数に反映
}
```

APPENDIX　1　　Arduino Uno ピン配置

ARDUINO

search

Buy | Download | Getting Started | Learning | Reference | Products | FAQ | Contact Us

Reference Language | Libraries | Comparison | Changes

Language Reference

Arduino programs can be divided in three main parts: *structure*, *values* (variables and constants), and *functions*.

Structure

- setup()
- loop()

Control Structures

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return
- goto

Further Syntax

- ; (semicolon)
- {} (curly braces)
- // (single line comment)
- /* */ (multi-line comment)
- #define
- #include

Arithmetic Operators

- = (assignment operator)
- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- % (modulo)

Comparison Operators

- == (equal to)
- != (not equal to)
- < (less than)
- > (greater than)
- <= (less than or equal to)
- >= (greater than or equal to)

Boolean Operators

- && (and)
- || (or)
- ! (not)

Pointer Access Operators

- * dereference operator
- & reference operator

Bitwise Operators

- & (bitwise and)
- | (bitwise or)
- ^ (bitwise xor)
- ~ (bitwise not)
- << (bitshift left)
- >> (bitshift right)

Compound Operators

- ++ (increment)
- -- (decrement)
- += (compound addition)
- -= (compound subtraction)
- *= (compound multiplication)
- /= (compound division)
- &= (compound bitwise and)
- |= (compound bitwise or)

Variables

Constants

- HIGH | LOW
- INPUT | OUTPUT| INPUT_PULLUP
- true | false
- integer constants
- floating point constants

Data Types

- void
- boolean
- char
- unsigned char
- byte
- int
- unsigned int
- word
- long
- unsigned long
- short
- float
- double
- string - char array
- String - object
- array

Conversion

- char()
- byte()
- int()
- word()
- long()
- float()

Variable Scope & Qualifiers

- variable scope
- static
- volatile
- const

Utilities

- sizeof()

Functions

Digital I/O

- pinMode()
- digitalWrite()
- digitalRead()

Analog I/O

- analogReference()
- analogRead()
- analogWrite() - *PWM*

Due only

- analogReadResolution()
- analogWriteResolution()

Advanced I/O

- tone()
- noTone()
- shiftOut()
- shiftIn()
- pulseIn()

Time

- millis()
- micros()
- delay()
- delayMicroseconds()

Math

- min()
- max()
- abs()
- constrain()
- map()
- pow()
- sqrt()

Trigonometry

- sin()
- cos()
- tan()

Random Numbers

- randomSeed()
- random()

Bits and Bytes

- lowByte()
- highByte()
- bitRead()
- bitWrite()
- bitSet()
- bitClear()
- bit()

External Interrupts

- attachInterrupt()
- detachInterrupt()

Interrupts

- interrupts()
- noInterrupts()

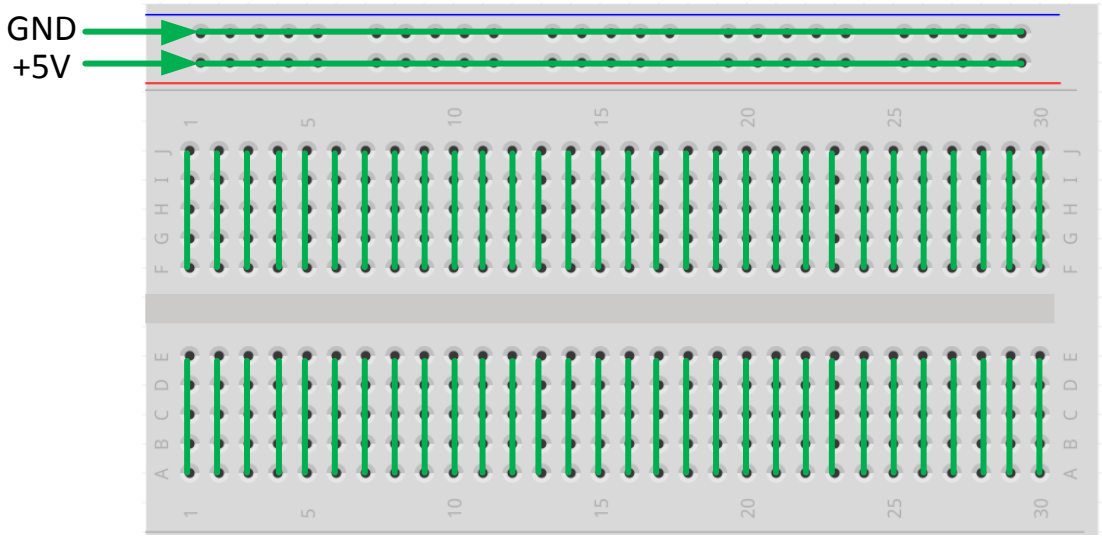
Communication

- Serial
- Stream

USB (Leonardo and Due only)

- Keyboard
- Mouse

図の緑色の部分が接続されている。
なお、上の青（GND）および赤（+5V）は、あらかじめ Arduino の電源に接続されている。
配線時にショートさせないように気をつけること。



Appendix 4　USB メモリーへの書込みおよび紙に印刷する設定について

オシロの設定で、PRINT 先の Default が「USB メモリー」になっています。

(1) 画像ファイルとして USB メモリーに書き込むには、写真 1 のようにファイル出力が選択されていることを確認し、PRINT ボタンを押します。

写真 1

写真 2

写真 3

PC で正常に画像が書き込めているかどうか確認して下さい。

なお、万一画像の背景が黒くなっている場合は（このまま印刷するとインクを大量に消費するので）以下の操作を行って、再度取り込んで下さい。

(a) オシロ中央の「Shift」ボタンを押してから、「PRINT」ボタンを押す。

(b) 写真 2 のメニューが現れる。項目「Color」は「ON(Rev.)」以外の設定になっているはずなので、「Color」ボタンを押すと、写真 3 のメニューが現れる。

(c) 「ON(Rev.)」ボタンを押す。

(d) Menu を消すために、「ESC」ボタンを押す。

(2) 紙に印刷したい場合

(a) オシロ中央の「Shift」ボタンを押してから、「PRINT」ボタンを押す。

(b) 写真 2 のメニューが現れるので、「Print To」ボタンを押す。

(c) 写真 4 のメニューが現れるので、「BuiltIn」ボタンを押す。

(d) 「PRINT」ボタンを押して、ハードコピーを取る。

(e) ハードコピーを取り終わったら、上記(a)(b)を行い、写真 4 の状態で「File」ボタンを押すと、元の状態に復帰する。（必ずここまで行って下さい。）

写真 4