観客賞　賞金５万円

ジキル博士と hide(high voltage demonstrator)氏

香川高等専門学校高松キャンパス

お天気実況アイス、スーちゃん

大阪教育大学付属高等学校天王寺校舎

磁気ル博士
と
hide 氏

香川高等専門学校　4 年　岩井　亮大
笠松　雅史
村田　優斗

1. 目的

周辺の磁界と雷との関連性を示しかつ電磁誘導による過電流を防止する。

2. 導入

図 2-1 のように変電所から各施設に電力が供給されていることを考える。

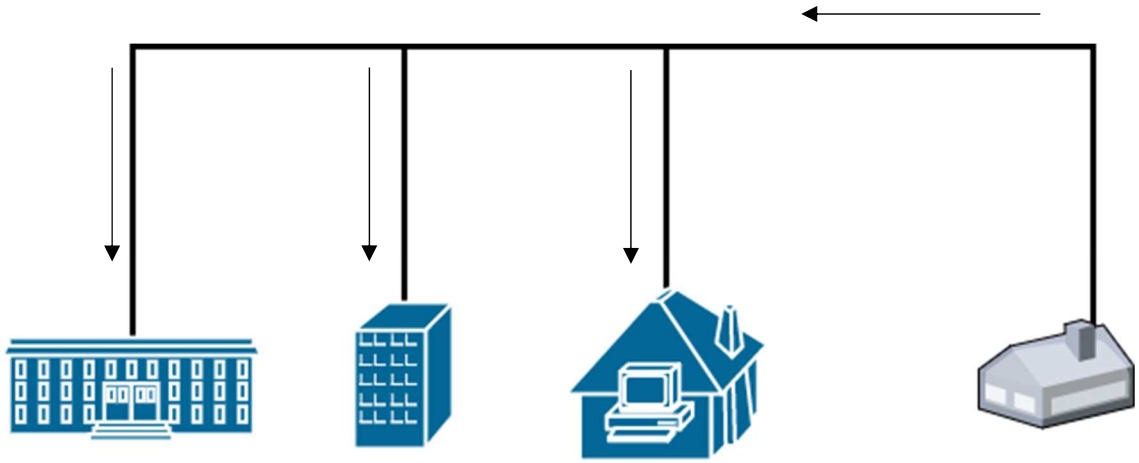


図 2-1 配電図

仮に送電線付近に雷が落ちたと仮定する。ここで送電線又は配線系統の物は全て抵抗、誘導性リアクタンス、容量性リアクタンスの複合素子と考えられるので周辺に高圧がかかり電流が流れた場合には電磁誘導により送電線にも電流が流れる。このときに過電流によりあたかも落雷が送電線に落ちたかのように配線している機器が故障したりする。これが誘導雷というものである。(図 2-2)

このときの磁界を検出し過電流を GND（地表）に逃がすことができれば誘導雷による被害は防げる。(図 2-3)

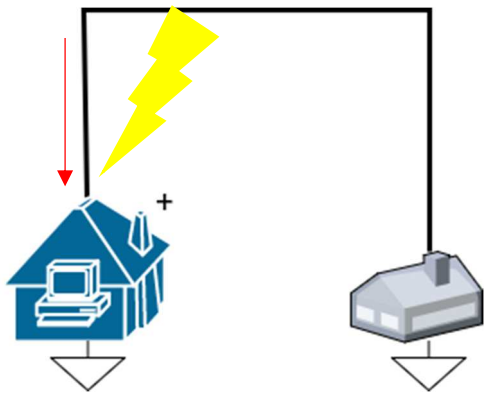


図 2-2 誘導雷

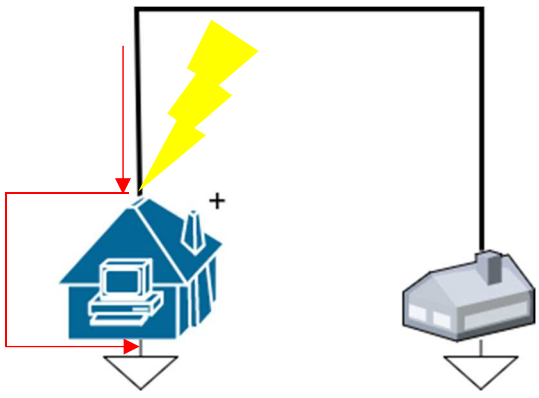


図 2-3 避雷回路

実際に機器が損傷する程の誘導雷を観測することは確率的に現実的ではないので高圧発生装置(high voltage demonstrator:hide: hide 氏)を用いることで仮想的に雷（電力は極小）を発生させその磁場を検出した。

3.　磁気ル博士

3.1 概要

磁気ル博士（磁気観測装置）の外形を図 3.1-1 に示す。磁気ル博士はセンサ部と制御部に分かれ、センサ部には 3 軸磁気センサ（HMC5883L※今回は X,Y の 2 軸のみ使用）及びキャリブレーション用のモータがあり制御部には arduino、モバイルバッテリー、L6470 モータ制御 IC がある。（図 3.1-2）

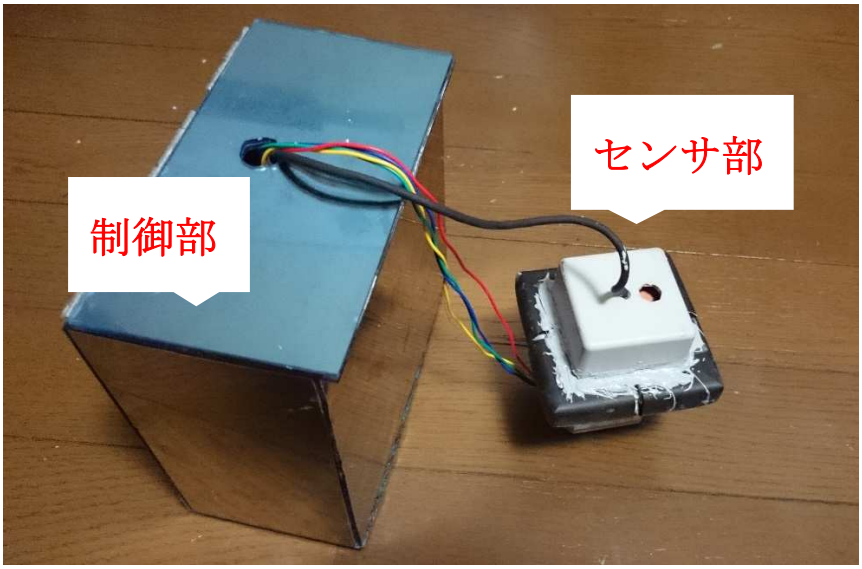


図 3.1-1 外形

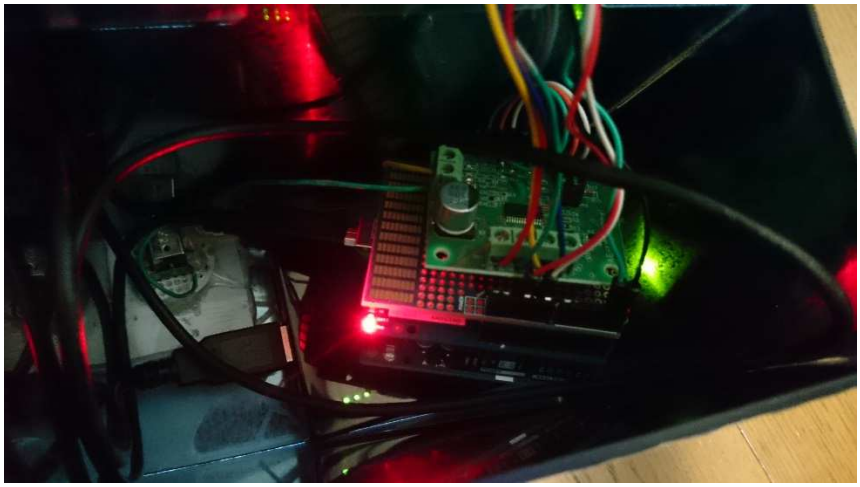


図 3.1-2 制御部内部

3.2 センサ部

磁気センサをそのまま使うと電波や微量の磁気ノイズも検出する恐れがあるので周囲の安定磁界のみを計測するために高周波シールドの中にセンサを設置した。（図 3.2-1）

今回高周波シールドに用いた素材は銅で（図 3.2-2）、制御部のシールドには磁性体であるステンレスを用いた。またセンサ部と制御部をつなぐ I2C 用のケーブルには同軸ケーブルを用いて信号ノイズを減衰させた。

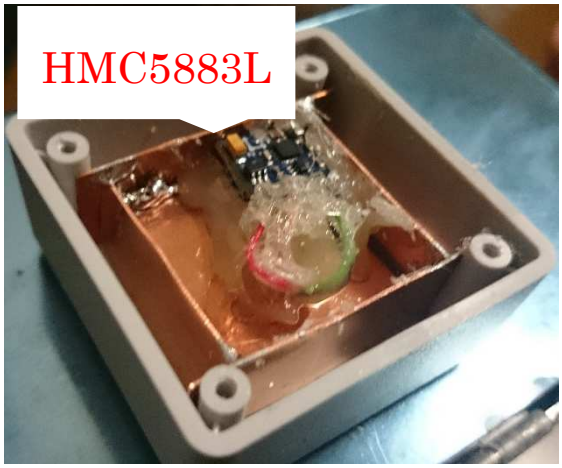


図 3.2-1 センサ部

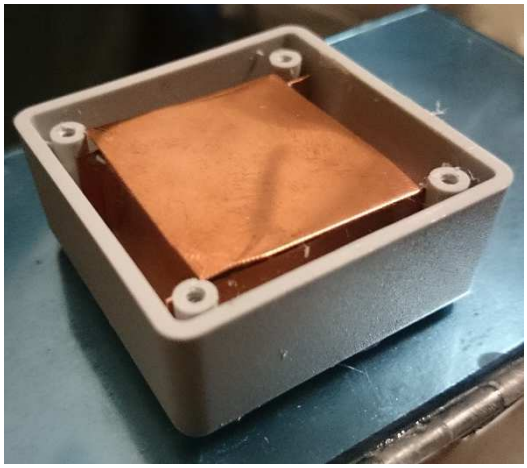


図 3.2-2 高周波シールド

3.3 制御部

制御部ではセンサを校正（キャリブレーション）するためにステッピングモータを使用した。ステッピングモータを arduino から直接制御することは難しいので、L6470（モータ制御 IC）を用いた。（図 3.3-1）

PC と arduino での通信には Xbee を用いて行った。また、L6470 と arduino の通信は SPI 通信を用いて行った。これらの接続を簡易モデル化したものを図 3.3-2 に示した。

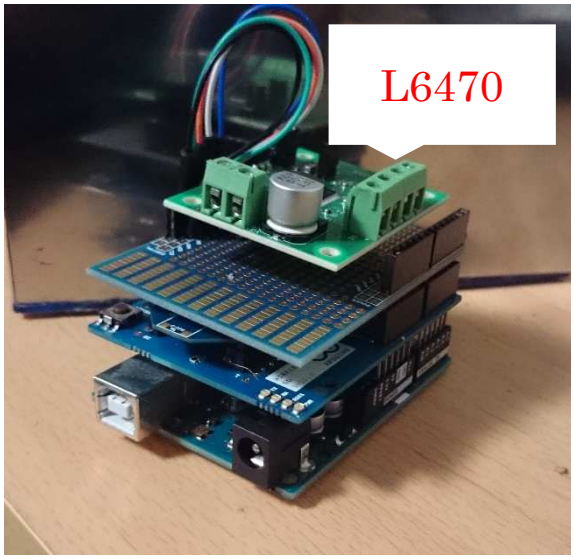


図 3.3-1 arduino と L6470

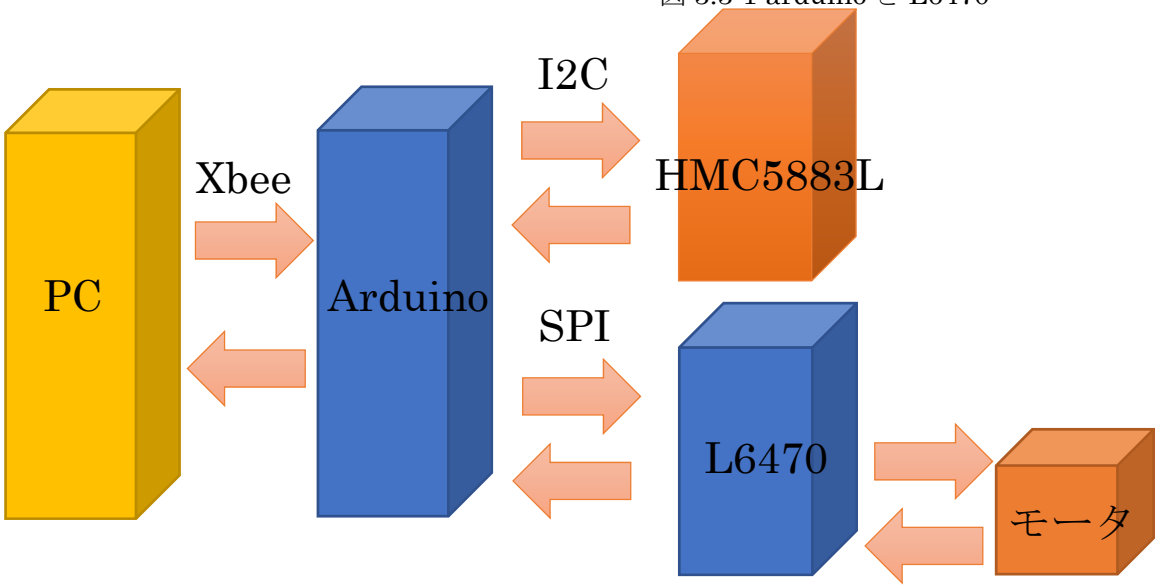


図 3.3-2 配線図

3.4 校正

磁気センサは自身の回路内部の磁界による影響でセンサを回転させたとき方向によって誤差が生じる。それを改善するためにはセンサを校正（キャリブレーション）しなければならない。（図 3.4-1）

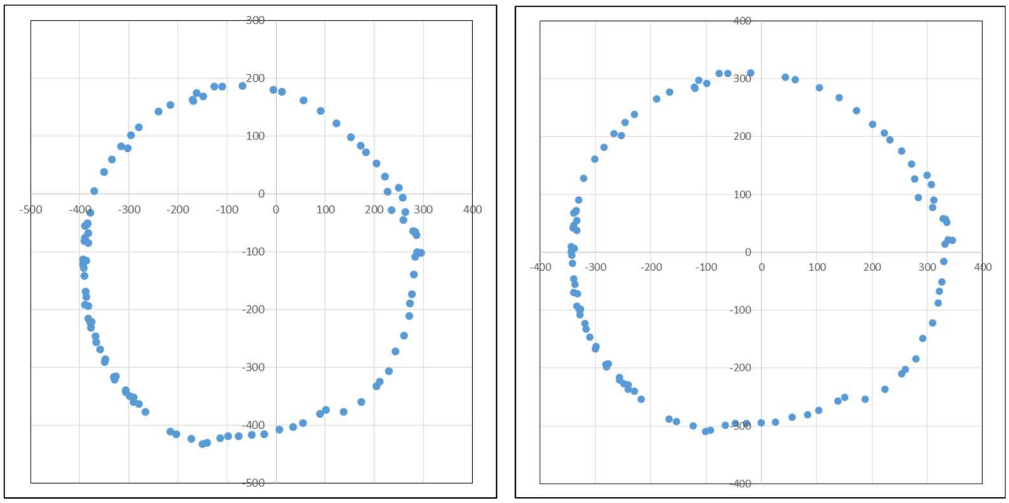


図 3.4-1 キャリブレーション前（左）と後(右)の例

このキャリブレーションを正確に行うためにステッピングモータを用いた。手動で回転させた場合、中心がずれ、誤差を生じる場合があるため、モータを用いた ACS(Auto Calibration System)で精度を高めた。

モータの中心にセンサを置き回転させデータ取得することで X,Y の最大値 X_Max,Y_Max と最小値 X_Min,Y_Min を求めそれから円の中心座標を求めた。キャリブレーション前の円の中心座標(X,Y)の値を(X_OFFSET,Y_OFFSET)とすると以下の式で表すことができる。

$$X\_OFFSET = (X\_Max + X\_min)/2 \quad (1)$$

$$Y\_OFFSET = (Y\_Max + Y\_min)/2 \quad (2)$$

上式(1)(2)で求めた値を用いて測定値を(X,Y)を以下の式で補正した。

$$X = X - X\_OFFSET \quad (3)$$

$$Y = Y - Y\_OFFSET \quad (4)$$

機器を動作させる前に PC から校正コマンドを Xbee 通信で送ることによりキャリブレーションを行うことができる。

なお、円は 3 点によって求めることができるが、そうするとデータのばらつき具合によっては誤差が大きくなるためデータ量を増やす必要があるため上記の方法で行った。

今回は基本ステップ角 1.8° のステッピングモータを用いた。

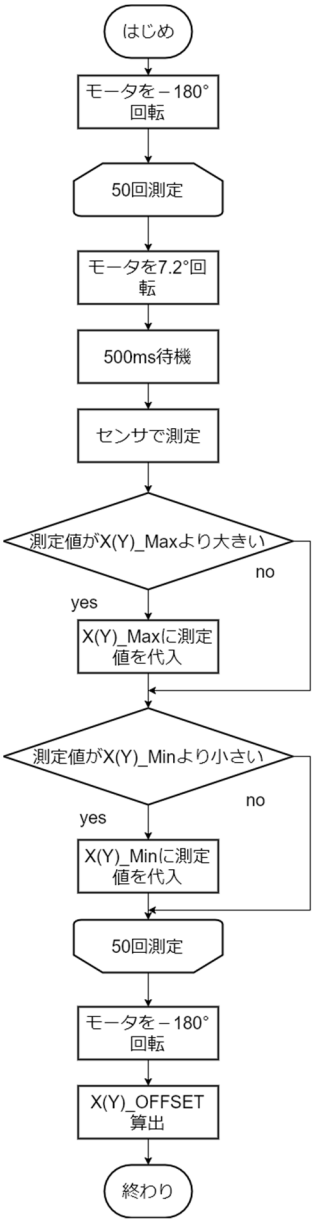


図 3.4-2 フローチャート

校正プログラムのフローチャートを図 3.4-2 に示した。

このプログラムはモータを 7.2° ずつ動かして 50 回（360° ）測定を行うものである。モータは電磁石によって構成されているためモータを回転させながら測定すると、電磁石によるノイズを拾ってしまい正しいデータが測定できなくなる。そこでモータを停止させ 500ms 待機した後センサで測定することによりモータの磁界による影響を受けなくしている。また、センサで測定するたびに X_Max, Y_Max,X_Min,Y_Min の値を判定して代入し 50 回（360° ）測定を行った後 X_OFFSET,Y_OFFSET の値を算出している。

なお、モータを－180° 回転させていることでセンサ部と制御部をつなぐ同軸ケーブルが絡まらないようにするためである。

4.　高圧発生装置(high voltage demonstrator　：　hide　氏)

hide 氏は 6 個の乾電池からの電圧を冷陰極間インバータ回路を用いて昇圧した交流電源回路を構成し、さらにそれをコッククロフトウォルトン回路により昇圧することで 30kV~40kV 程度の高圧を発生させる装置である。(図 4-1)

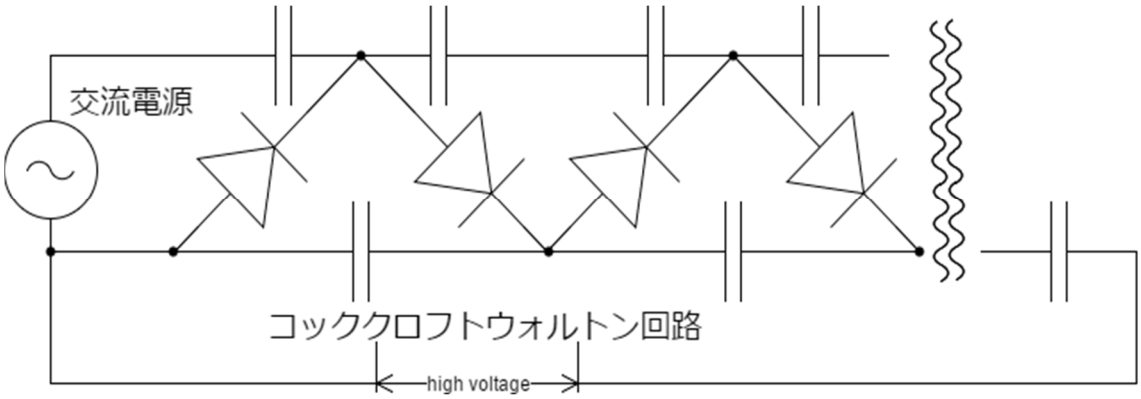


図 4-1　高圧発生回路

図 4-2 hide 氏の外形

なお安全のため放電は空気中ではなくカメラのストロボに用いられるキセノンランプ内で放電させた。(図 4-2)

5. GUI

arduino と PC との通信において GUI による操作法を導入した。（図 5-1）これにより磁気ル博士の操作が簡易でかつ複雑なものを行えるようになった。

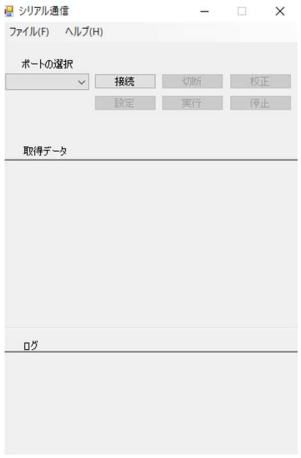


図 5-1 GUI 画面

6. 実験

6.1 ACS(Auto Calibration System)

ACS 実行結果を以下に示した。

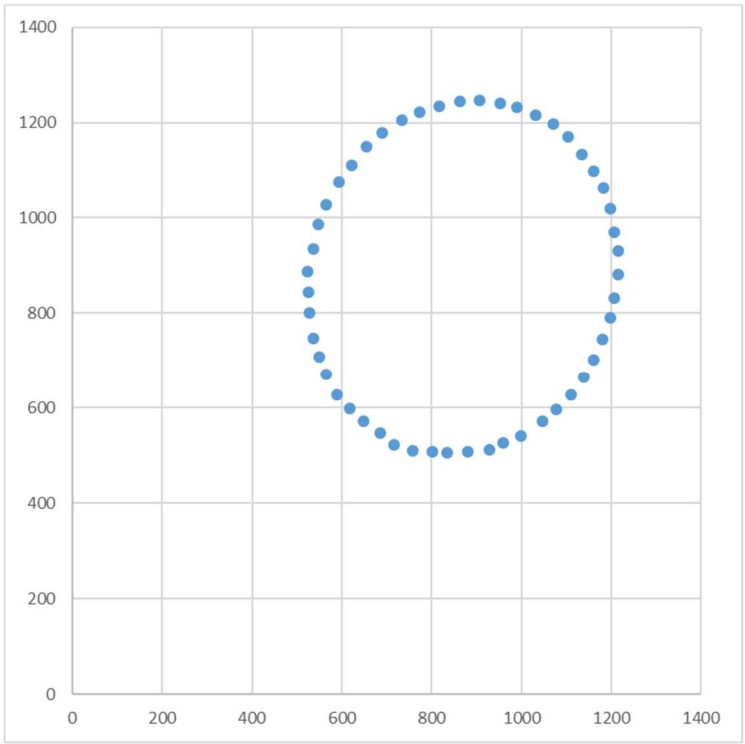


図 6.1-1 キャリブレーション前

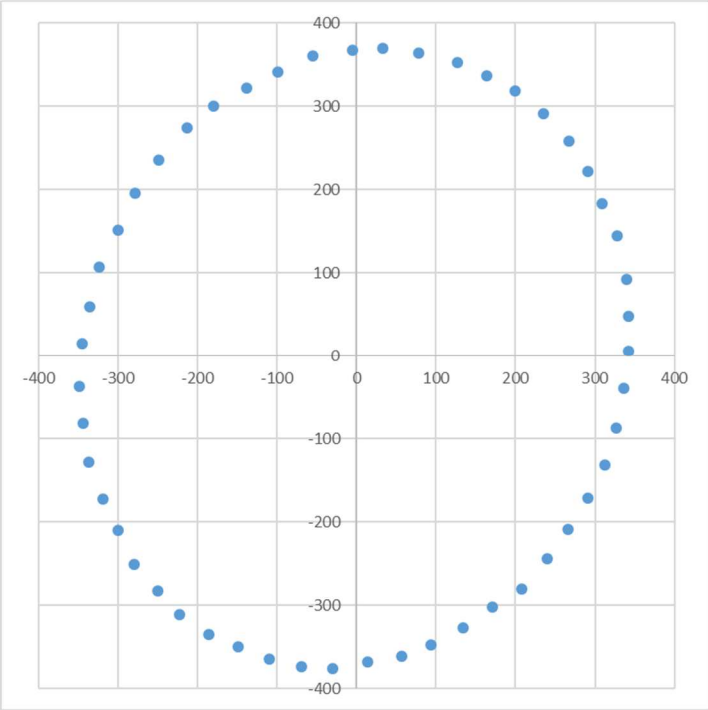


図 6.1-2 キャリブレーション後

6.2 サージノイズ

磁気ル博士を用いて hide 氏からのノイズを検出した。実験はジキル博士の測定モード 75Hz、±0.88G のレンジで 3 回にわたり測定し結果を以下に記した。なお hide 氏のスイッチは 1~5 番目のデータで OFF、6~10 番目のデータで ON となっている。

表 6.2-1 実験 1 回目結果

n番目	磁界[mG]	平均値[mG]
1	273	272
2	273	
3	272	
4	271	
5	271	
6	272	274
7	274	
8	276	
9	275	
10	274	

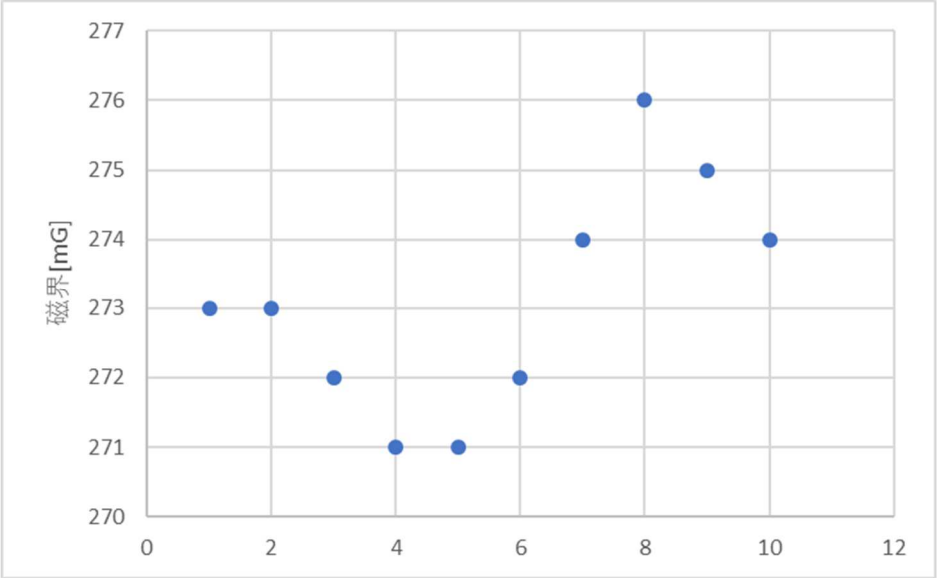


図 6.2-1 実験 1 回目結果

表 6.2-2 実験 2 回目結果

n番目	磁界[mG]	平均値[mG]
1	269	269
2	268	
3	268	
4	269	
5	269	
6	272	272
7	274	
8	272	
9	271	
10	272	

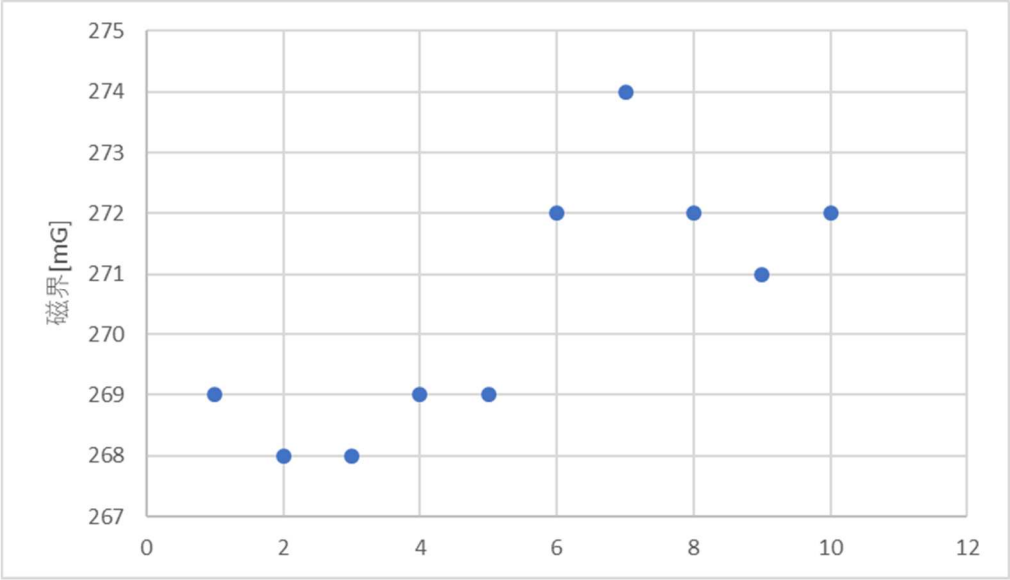


図 6.2-2 実験 2 回目結果

表 6.2-3 実験 3 回目結果

n番目	磁界[mG]	平均値[mG]
1	275	275
2	274	
3	276	
4	275	
5	274	
6	270	272
7	272	
8	273	
9	272	
10	272	

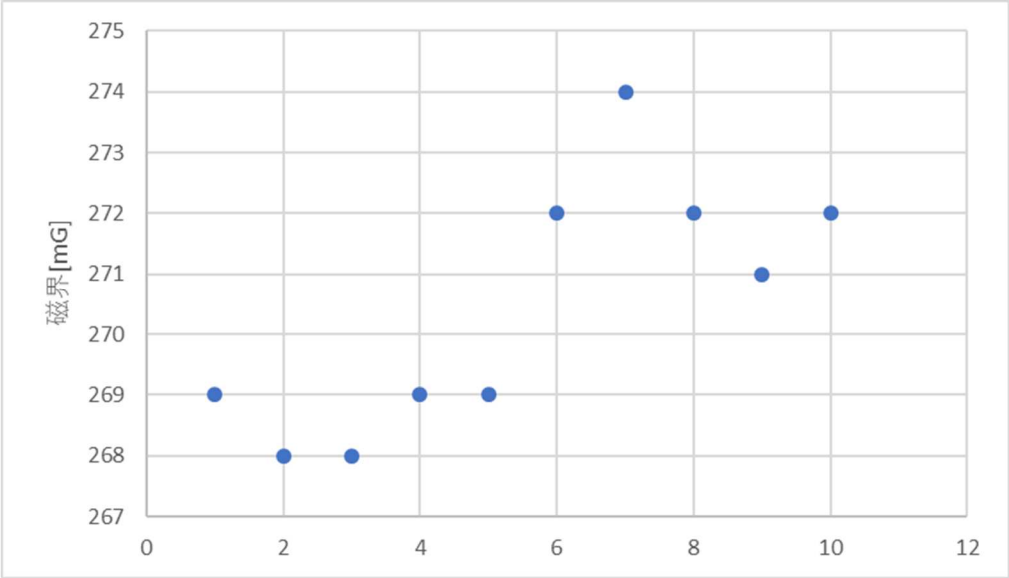


図 6.2-3 実験 3 回目結果

7. 考察

7.1 ACS

6.1 の図 6.1-1、6.1-2 からキャリブレーション前は原点から大きく離れた円になっているがキャリブレーション後には原点を中心とする円となっていて ACS が正確に動作していることが分かる。ただ、測定点が正確な円ではなく楕円となるのはセンサ内部による回路ノイズが原因だと考えられる。

7.2 サージノイズ

6.2 の結果から hide 氏のスイッチが ON の状態と OFF の状態での平均値が 2mG 以上の差分として表れている。これは hide 氏の電源回路で用いたコックロフトウォルトン回路の昇圧性能が交流電源（冷陰極管インバータ）の周波数に依存するためサージが発生するまでに誤差が生じたと考えられる。

8. 課題

8.1 避雷回路の構成

今回の空中放電では電流がそれほど流れておらずサージノイズが小さいため定常磁界の変化と重なる部分があったために確実にサージノイズをカットする回路を構成できなかった。しかし、実際の誘導雷では大電流が流れるため、より大きなノイズになるであろうと予想した。

8.2 シリアル通信

arduino に PC からデータ停止コマンドを送信する際に受信側と送信側で干渉してしまい停止コマンドが反応しない場合があった。これについては今後タイミングを統一することで改善する。

9.　ソースコード

今回用いた GUI 以外のソースコードを以下に記載した。

9.1 arduino

```
#include <JEKYLL.h>
#include <SPI.h>
#include <Wire.h>

JEKYLL jekyll;

void setup()
{
  jekyll.init();
}

void loop()
{
  jekyll.run();
}
```

9.2 JEKYLL.h

```
#include <Arduino.h>

#include <L6470.h>
#include <HMC5883L.h>

class JEKYLL
{
        public:
                JEKYLL();

                void init(void);

                void run(void);
                void calibration(void);

        private:
                void serialEvent(void);
                void measure(void);

        private:
                L6470 stepper;
                HMC5883L compass;

                boolean measureFlag;
};
```

9.3 HMC5883L.h

```
#include <Arduino.h>

typedef enum
{
        HMC5883L_ADDRESS
= 0x1E,
        HMC5883L_REG_CONFIG_A
= 0x00,
        HMC5883L_REG_CONFIG_B
= 0x01,
        HMC5883L_REG_MODE
= 0x02,
        HMC5883L_REG_OUT_X_M
= 0x03,
        HMC5883L_REG_OUT_X_L
= 0x04,
        HMC5883L_REG_OUT_Z_M
= 0x05,
        HMC5883L_REG_OUT_Z_L
= 0x06,
        HMC5883L_REG_OUT_Y_M
= 0x07,
        HMC5883L_REG_OUT_Y_L
= 0x08,
        HMC5883L_REG_STATUS
= 0x09,
        HMC5883L_REG_IDENT_A
= 0x0A,
        HMC5883L_REG_IDENT_B
= 0x0B,
        HMC5883L_REG_IDENT_C
= 0x0C,
} hmc5883l_address;

typedef enum
{
        HMC5883L_RANGE_0_88GA
= 0b000,
        HMC5883L_RANGE_1_3GA
= 0b001,
        HMC5883L_RANGE_1_9GA
= 0b010,
        HMC5883L_RANGE_2_5GA
= 0b011,
        HMC5883L_RANGE_4GA
= 0b100,
        HMC5883L_RANGE_4_7GA
= 0b101,
        HMC5883L_RANGE_5_6GA
= 0b110,
        HMC5883L_RANGE_8_1GA
= 0b111
} hmc5883l_range;

typedef enum
{
        HMC5883L_CONTINOUS
= 0b00,
        HMC5883L_SINGLE          =
0b01,
        HMC5883L_IDLE            =
0b10
} hmc5883l_mode;

typedef enum
{
        HMC5883L_DATARATE_0_75_
HZ      = 0b000,
        HMC5883L_DATARATE_1_5HZ
= 0b001,
        HMC5883L_DATARATE_3HZ
```

```
= 0b010,
        HMC5883L_DATARATE_7_5HZ
= 0b011,
        HMC5883L_DATARATE_15HZ
= 0b100,
        HMC5883L_DATARATE_30HZ
= 0b101,
        HMC5883L_DATARATE_75HZ
= 0b110
} hmc5883l_dataRate;

typedef enum
{
        HMC5883L_SAMPLES_1      =
0b00,
        HMC5883L_SAMPLES_2      =
0b01,
        HMC5883L_SAMPLES_4      =
0b10,
        HMC5883L_SAMPLES_8      =
0b11
} hmc5883l_samples;

struct Vector
{
    float XAxis;
    float YAxis;
    float ZAxis;
};

class HMC5883L
{
    public:
        HMC5883L();

        void init(void);

        Vector readRaw(void);
        Vector readNormalize(void);

                void setOffSet(Vector
vec);
                void setOffSet(float x,
float y, float z);
                Vector getOffSet(void);

        void setRange(hmc5883l_range
range);
                void setRange(int
range);
        hmc5883l_range getRange(void);

        void
setMeasurementMode(hmc5883l_mode
mode);
                void
setMeasurementMode(int mode);
        hmc5883l_mode
getMeasurementMode(void);

        void
setDataRate(hmc5883l_dataRate rate);
                void setDataRate(int
rate);
        hmc5883l_dataRate
getDataRate(void);

        void
setSamples(hmc5883l_samples samples);
                void setSamples(int
samples);
        hmc5883l_samples
```

```
getSamples(void);

                    void setSettings(String
str);
        void printSettings(void);
                    String
getSettings(void);

        void writeRegister8(byte reg,
byte value);
        unsigned char
readRegister8(byte reg);

    private:
        Vector data;
                    Vector offSet;
        float mgPerDigit;
};
```

9.4 L6470.h

```
#include <Arduino.h>

typedef enum
{

L6470_PIN_SS
= 10,

L6470_PIN_MOSI
= 11,

L6470_PIN_MISO
= 12,

L6470_PIN_SCK
= 13,

L6470_PIN_RESET
=  6,

L6470_PIN_BUSYN
=  4,

L6470_PIN_STAT1
= 14,

L6470_PIN_STAT2
= 15,

L6470_PIN_SWITCH
=  8
} l6470_pin;

typedef enum
{

L6470_STEP_MODE_SYNC_0_5_STEP_
1                   = 0x0000,

L6470_STEP_MODE_SYNC_0_5_STEP_
2                   = 0x0001,

L6470_STEP_MODE_SYNC_0_5_STEP_
4                   = 0x0002,

L6470_STEP_MODE_SYNC_0_5_STEP_
8                   = 0x0003,

L6470_STEP_MODE_SYNC_0_5_STEP_
16                  = 0x0004,

L6470_STEP_MODE_SYNC_0_5_STEP_
32                  = 0x0005,

L6470_STEP_MODE_SYNC_0_5_STEP_
64                  = 0x0006,

L6470_STEP_MODE_SYNC_0_5_STEP_
```

128 = 0x0007,

L6470_STEP_MODE_SYNC_1_STEP_2 = 0x0011,

L6470_STEP_MODE_SYNC_1_STEP_4 = 0x0012,

L6470_STEP_MODE_SYNC_1_STEP_8 = 0x0013,

L6470_STEP_MODE_SYNC_1_STEP_16 = 0x0014,

L6470_STEP_MODE_SYNC_1_STEP_32 = 0x0015,

L6470_STEP_MODE_SYNC_1_STEP_64 = 0x0016,

L6470_STEP_MODE_SYNC_1_STEP_128 = 0x0017,

L6470_STEP_MODE_SYNC_2_STEP_4 = 0x0022,

L6470_STEP_MODE_SYNC_2_STEP_8 = 0x0023,

L6470_STEP_MODE_SYNC_2_STEP_16 = 0x0024,

L6470_STEP_MODE_SYNC_2_STEP_32 = 0x0025,

L6470_STEP_MODE_SYNC_2_STEP_64 = 0x0026,

L6470_STEP_MODE_SYNC_2_STEP_128 = 0x0027,

L6470_STEP_MODE_SYNC_4_STEP_8 = 0x0033,

L6470_STEP_MODE_SYNC_4_STEP_16 = 0x0034,

L6470_STEP_MODE_SYNC_4_STEP_32 = 0x0035,

L6470_STEP_MODE_SYNC_4_STEP_64 = 0x0036,

L6470_STEP_MODE_SYNC_4_STEP_128 = 0x0037,

L6470_STEP_MODE_SYNC_8_STEP_16 = 0x0044,

L6470_STEP_MODE_SYNC_8_STEP_32 = 0x0045,

L6470_STEP_MODE_SYNC_8_STEP_64 = 0x0046,

L6470_STEP_MODE_SYNC_8_STEP_128 = 0x0047,

L6470_STEP_MODE_SYNC_16_STEP_32 = 0x0055,

L6470_STEP_MODE_SYNC_16_STEP_6

```
4                       = 0x0056,

L6470_STEP_MODE_SYNC_16_STEP_1
28                      = 0x0057,

L6470_STEP_MODE_SYNC_32_STEP_6
4                       = 0x0066,

L6470_STEP_MODE_SYNC_32_STEP_1
28                      = 0x0067,

L6470_STEP_MODE_SYNC_64_STEP_1
28                      = 0x0077,

L6470_STEP_MODE_SYNC_DISABLE
= 0x0000,

L6470_STEP_MODE_SYNC_ENABLE
= 0x0080
} l6470_step_mode;

typedef enum
{

L6470_ALARM_EN_OVERCURRENT
= 0x0001,

L6470_ALARM_EN_THERMAL_SHUTD
OWN                     = 0x0002,

L6470_ALARM_EN_THERMAL_WARNI
NG                      = 0x0004,

L6470_ALARM_EN_UNDER_VOLTAGE
= 0x0008,

L6470_ALARM_EN_STALL_DET_A
= 0x0010,

L6470_ALARM_EN_STALL_DET_B
= 0x0020,

L6470_ALARM_EN_SW_TURN_ON
= 0x0040,

L6470_ALARM_EN_WRONG_NPERF_C
MD                      = 0x0080,

L6470_ALARM_EN_ALL
= 0x00FF
} l6470_alarm_en;

typedef enum
{

L6470_CONFIG_OSC_INT_16MHZ
= 0x0000,

L6470_CONFIG_OSC_INT_16MHZ_OU
T_2MHZ                  = 0x0008,

L6470_CONFIG_OSC_INT_16MHZ_OU
T_4MHZ                  = 0x0009,

L6470_CONFIG_OSC_INT_16MHZ_OU
T_8MHZ                  = 0x000A,

L6470_CONFIG_OSC_INT_16MHZ_OU
T_16MHZ                 = 0x000B,

L6470_CONFIG_OSC_EXT_8MHZ_OUT
_XTAL                   = 0x0004,
```

L6470_CONFIG_OSC_EXT_16MHZ_OUT_XTAL = 0x0005,

L6470_CONFIG_OSC_EXT_24MHZ_OUT_XTAL = 0x0006,

L6470_CONFIG_OSC_EXT_32MHZ_OUT_XTAL = 0x0007,

L6470_CONFIG_OSC_EXT_8M_OUT_INVERT = 0x000C,

L6470_CONFIG_OSC_EXT_16M_OUT_INVERT = 0x000D,

L6470_CONFIG_OSC_EXT_24M_OUT_INVERT = 0x000E,

L6470_CONFIG_OSC_EXT_32M_OUT_INVERT = 0x000F,

L6470_CONFIG_SW_MODE_DISABLE = 0x0000,

L6470_CONFIG_SW_MODE_ENABLE = 0x0010,

L6470_CONFIG_EN_VSCOMP_DISABLE = 0x0000,

L6470_CONFIG_EN_VSCOMP_ENABLE = 0x0020,

L6470_CONFIG_OC_SD_DISABLE = 0x0000,

L6470_CONFIG_OC_SD_ENABLE = 0x0080,

L6470_CONFIG_POW_SR_320V_us = 0x0000,

L6470_CONFIG_POW_SR_75V_us = 0x0100,

L6470_CONFIG_POW_SR_110V_us = 0x0200,

L6470_CONFIG_POW_SR_260V_us = 0x0300,

L6470_CONFIG_F_PWM_DEC_0_625 = 0x0000,

L6470_CONFIG_F_PWM_DEC_0_75 = 0x0400,

L6470_CONFIG_F_PWM_DEC_0_875 = 0x0800,

L6470_CONFIG_F_PWM_DEC_1 = 0x0C00,

L6470_CONFIG_F_PWM_DEC_1_25 = 0x1000,

L6470_CONFIG_F_PWM_DEC_1_5 = 0x1400,

L6470_CONFIG_F_PWM_DEC_1_75 = 0x1800,

```
L6470_CONFIG_F_PWM_DEC_2
= 0x1C00,

L6470_CONFIG_F_PWM_INT_1
= 0x0000,

L6470_CONFIG_F_PWM_INT_2
= 0x2000,

L6470_CONFIG_F_PWM_INT_3
= 0x4000,

L6470_CONFIG_F_PWM_INT_4
= 0x6000,

L6470_CONFIG_F_PWM_INT_5
= 0x8000,

L6470_CONFIG_F_PWM_INT_6
= 0xA000,

L6470_CONFIG_F_PWM_INT_7
= 0xC000
} l6470_config;

typedef enum
{

L6470_STATUS_HIZ
= 0x0001,

L6470_STATUS_BUSY
= 0x0002,

L6470_STATUS_SW_F
= 0x0004,

L6470_STATUS_SW_EVN
= 0x0008,

L6470_STATUS_DIR
= 0x0010,

L6470_STATUS_MOT_STATUS_STOPP
ED                  = 0x0000,

L6470_STATUS_MOT_STATUS_ACCEL
ERATION             = 0x0020,

L6470_STATUS_MOT_STATUS_DECEL
ERATION             = 0x0040,

L6470_STATUS_MOT_STATUS_CONST
_SPD                = 0x0060,

L6470_STATUS_NOTPERF_CMD
= 0x0080,

L6470_STATUS_WRONG_CMD
= 0x0100,

L6470_STATUS_UVLO
= 0x0200,

L6470_STATUS_TH_WRN
= 0x0400,

L6470_STATUS_TH_SD
= 0x0800,

L6470_STATUS_OCD
```

```
= 0x1000,

L6470_STATUS_STEP_LOSS_A
= 0x2000,

L6470_STATUS_STEP_LOSS_B
= 0x4000,

L6470_STATUS_SCK_MOD
= 0x8000
} l6470_status;

typedef enum
{

L6470_PARAM_ABS_POS
= 0x0001,

L6470_PARAM_EL_POS
= 0x0002,

L6470_PARAM_MARK
= 0x0003,

L6470_PARAM_SPEED
= 0x0004,

L6470_PARAM_ACC
= 0x0005,

L6470_PARAM_DEC
= 0x0006,

L6470_PARAM_MAX_SPEED
= 0x0007,

L6470_PARAM_MIN_SPEED
= 0x0008,

L6470_PARAM_FS_SPD
= 0x0015,

L6470_PARAM_KVAL_HOLD
= 0x0009,

L6470_PARAM_KVAL_RUN
= 0x000A,

L6470_PARAM_KVAL_ACC
= 0x000B,

L6470_PARAM_KVAL_DEC
= 0x000C,

L6470_PARAM_INT_SPEED
= 0x000D,

L6470_PARAM_ST_SLP
= 0x000E,

L6470_PARAM_FN_SLP_ACC
= 0x000F,

L6470_PARAM_FN_SLP_DEC
= 0x0010,

L6470_PARAM_K_THERM
= 0x0011,

L6470_PARAM_ADC_OUT
= 0x0012,
```

```
L6470_PARAM_OCD_TH
= 0x0013,

L6470_PARAM_STALL_TH
= 0x0014,

L6470_PARAM_STEP_MODE
= 0x0016,

L6470_PARAM_ALARM_EN
= 0x0017,

L6470_PARAM_CONFIG
= 0x0018,

L6470_PARAM_STATUS
= 0x0019
} l6470_param;

typedef enum
{

L6470_COMMAND_NOP
= 0x0000,

L6470_COMMAND_SET_PARAM
= 0x0000,

L6470_COMMAND_GET_PARAM
= 0x0020,

L6470_COMMAND_RUN
= 0x0050,

L6470_COMMAND_STEP_CLOCK
= 0x0058,

L6470_COMMAND_MOVE
= 0x0040,

L6470_COMMAND_GOTO
= 0x0060,

L6470_COMMAND_GOTO_DIR
= 0x0068,

L6470_COMMAND_GO_UNTIL
= 0x0082,

L6470_COMMAND_RELEASE_SW
= 0x0092,

L6470_COMMAND_GO_HOME
= 0x0070,

L6470_COMMAND_GO_MARK
= 0x0078,

L6470_COMMAND_RESET_POS
= 0x00D8,

L6470_COMMAND_RESET_DEVICE
= 0x00C0,

L6470_COMMAND_SOFT_STOP
= 0x00B0,

L6470_COMMAND_HARD_STOP
= 0x00B8,

L6470_COMMAND_SOFT_HIZ
= 0x00A0,
```

```
L6470_COMMAND_HARD_HIZ
= 0x00A8,

L6470_COMMAND_GET_STATUS
= 0x00D0
} l6470_command;

typedef enum
{

L6470_DIRECTION_FWD
= 0x0001,

L6470_DIRECTION_REV
= 0x0000
} l6470_deirection;

typedef enum
{

L6470_ACTION_RESET
= 0x0000,

L6470_ACTION_COPY
= 0x0008
} l6470_action;

class L6470 {

public:


L6470();
```

```
void init(void);

boolean isBusy(void);

void delayBusy(void);

void delayBusy(long);

void commandNOP(void);

void commandSetParam(l6470_param
param, long value);

long commandGetParam(l6470_param
param);

void commandRun(l6470_deirection dir,
long spd);

void
commandStep_Clock(l6470_deirection
dir);

void commandMove(l6470_deirection dir,
```

long n_step);

void commandGoTo(long pos);

void commandGoTo_DIR(l6470_deirection dir, long pos);

void commandGoUntil(l6470_action act, l6470_deirection dir, long spd);

void commandReleaseSW(l6470_action act, l6470_deirection dir);

void commandGoHome(void);

void commandGoMark(void);

void commandResetPos(void);

void commandResetDevice(void);

void commandSoftStop(void);

void commandHardStop(void);

void commandSoftHiZ(void);

void commandHardHiZ(void);

long commandGetStatus(void);

long calcRegValSPEED(float stepsPerSec);

long calcRegValACC(float stepsPerSecPerSec);

long calcRegValDEC(float stepsPerSecPerSec);

long calcRegValMAX_SPEED(float stepsPerSec);

long calcRegValMIN_SPEED(float stepsPerSec);

long calcRegValFS_SPD(float stepsPerSec);

long calcRegValINT_SPEED(float

```
stepsPerSec);

long calcRegValOCD_TH(float milliAmpere);

long calcRegValSTALL_TH(float milliAmpere);

float calcUnitSPEED(long speed);

float calcUnitACC(long acc);

float calcUnitDEC(long dec);

float calcUnitMAX_SPEED(long max_speed);

float calcUnitMIN_SPEED(long min_speed);

float calcUnitFS_SPD(long fs_spd);

float calcUnitINT_SPEED(long int_speed);

//private:

void setABS_POS(long abs_pos);

void setEL_POS(long el_pos);

void setMARK(long mark);

void setACC(long acc);

void setDEC(long dec);

void setMAX_SPEED(long max_speed);

void setMIN_SPEED(long min_speed);

void setFS_SPD(long fs_spd);

void setKVAL_HOLD(long kval_hold);

void setKVAL_RUN(long kval_run);

void setKVAL_ACC(long kval_acc);
```

void setKVAL_DEC(long kval_dec);

long getEL_POS(void);

void setINT_SPEED(long int_speed);

long getMARK(void);

void setST_SLP(long st_slp);

long getSPEED(void);

void setFN_SLP_ACC(long slp_acc);

long getACC(void);

void setFN_SLP_DEC(long slp_dec);

long getDEC(void);

void setK_THERM(long k_therm);

long getMAX_SPEED(void);

void setOCD_TH(long ocd_th);

long getMIN_SPEED(void);

void setSTALL_TH(long stall_th);

long getFS_SPD(void);

void setSTEP_MODE(long step_mode);

long getKVAL_HOLD(void);

void setALARM_EN(long alarm_en);

long getKVAL_RUN(void);

void setCONFIG(long config);

long getKVAL_ACC(void);

long getABS_POS(void);

long getKVAL_DEC(void);

```
long getINT_SPEED(void);

long getST_SLP(void);

long getFN_SLP_ACC(void);

long getFN_SLP_DEC(void);

long getK_THERM(void);

long getADC_OUT(void);

long getOCD_TH(void);

long getSTALL_TH(void);

long getSTEP_MODE(void);

long getALARM_EN(void);

long getCONFIG(void);

long getSTATUS(void);
```

```
private:

long checkOverflow(long data, long range);

long checkOverflow(long data, long max_range, long min_range);

byte Xfer(byte data);
};
```

9.5 JEKYLL.cpp

```
#include "JEKYLL.h"
#include "SPI.h"
#include "Wire.h"


JEKYLL::JEKYLL() {}

void JEKYLL::init(void)
{
	Serial.begin(9600);
	measureFlag = false;
	compass.init();

	stepper.init();

	compass.writeRegister8(0x02,
0x00);

	pinMode(LED_pin_1, OUTPUT);
}

void JEKYLL::run(void)
{
	if (Serial.available() > 0)
	{
		serialEvent();
	}
	if (measureFlag)
	{
		measure();
	}
}

void JEKYLL::calibration(void)
{
	Vector vec;
	float max_X, min_X, max_Y,
min_Y;
	int data_num = 50;

	stepper.commandMove(L6470_D
IRECTION_REV, 100);
	stepper.delayBusy(1000);

	delay(500);
	vec = compass.readRaw();
	max_X = min_X = vec.XAxis;
	max_Y = min_Y = vec.YAxis;

	for (int i = 0; i < data_num; i++)
	{

	stepper.commandMove(L6470_D
IRECTION_FWD, (200 / data_num));
		stepper.delayBusy();

		delay(500);
		vec =
compass.readRaw();

		if (max_X < vec.XAxis)
		{
			max_X =
vec.XAxis;
		}
		if (min_X > vec.XAxis)
		{
			min_X =
vec.XAxis;
		}
		if (max_Y < vec.YAxis)
```

```
                {
                        max_Y =
vec.YAxis;
                }
                if (min_Y > vec.YAxis)
                {
                        min_Y =
vec.YAxis;
                }
        }
        stepper.commandMove(L6470_D
IRECTION_REV, 100);
        stepper.delayBusy(1000);

        compass.setOffSet((max_X +
min_X) / 2, (max_Y + min_Y) / 2, 0);

}

void JEKYLL::serialEvent(void)
{
        String code =
Serial.readStringUntil('@');

        if (code.indexOf("set") != -1)
        {
                measureFlag = false;

        compass.setSettings(code);

        Serial.print("###received set " +
compass.getSettings() + "@");
        }
        else if (code.indexOf("stop") != -
1)
        {
                measureFlag = false;

        Serial.print("###received
stop@");
        }
        else if
(code.indexOf("measure") != -1)
        {
                measureFlag = true;

        Serial.print("###received
measure@");
        }
        else if
(code.indexOf("calibration") != -1)
        {
                measureFlag = false;
                calibration();

        Serial.print("###received
calibration@");
        }
        else

        Serial.print("###MESSAGE
ERROR!@");
}

void JEKYLL::measure(void)
{
        Vector vec;
        int VEC;

        vec = compass.readNormalize();
        VEC = int(sqrt(pow(vec.XAxis,
2.0) + pow(vec.YAxis, 2.0)));
```

```
            Serial.print("Vec =" +
String(VEC_before));
            Serial.print("Vec =" +
String(VEC_after));
            Serial.print("@");
}
```

9.6 HMC5883L.cpp

```
#include "HMC5883L.h"
#include "Wire.h"

HMC5883L::HMC5883L() {}

void HMC5883L::init()
{
        Wire.begin();


setRange(HMC5883L_RANGE_1_3GA);

setMeasurementMode(HMC5883L_CONTINOUS);

setDataRate(HMC5883L_DATARATE_15HZ);

setSamples(HMC5883L_SAMPLES_1);
        setOffSet(0.0, 0.0, 0.0);
}

Vector HMC5883L::readRaw(void)
{
    data.XAxis = ((readRegister8(HMC5883L_REG_OUT_X_M) << 8) & 0xFF00) | readRegister8(HMC5883L_REG_OUT_X_L);
        data.XAxis -= offSet.XAxis;
        data.YAxis = ((readRegister8(HMC5883L_REG_OUT_Y_M) << 8) & 0xFF00) | readRegister8(HMC5883L_REG_OUT_Y_L);
        data.YAxis -= offSet.YAxis;
        data.ZAxis = ((readRegister8(HMC5883L_REG_OUT_Z_M) << 8) & 0xFF00) | readRegister8(HMC5883L_REG_OUT_Z_L);
        data.ZAxis -= offSet.ZAxis;

    return data;
}

Vector HMC5883L::readNormalize(void)
{
    readRaw();
        data.XAxis *= mgPerDigit;
        data.YAxis *= mgPerDigit;
        data.ZAxis *= mgPerDigit;

    return data;
}

void HMC5883L::setOffSet(Vector vec)
{
        offSet = vec;
}

void HMC5883L::setOffSet(float x, float y, float z)
{
        offSet.XAxis = x;
        offSet.YAxis = y;
        offSet.ZAxis = z;
}

Vector HMC5883L::getOffSet(void)
{
```

```
        return offSet;
}

void
HMC5883L::setRange(hmc5883l_range
range)
{
    switch(range)
    {
        case
HMC5883L_RANGE_0_88GA:
            mgPerDigit = 0.73f;
            break;

        case
HMC5883L_RANGE_1_3GA:
            mgPerDigit = 0.92f;
            break;

        case
HMC5883L_RANGE_1_9GA:
            mgPerDigit = 1.22f;
            break;

        case
HMC5883L_RANGE_2_5GA:
            mgPerDigit = 1.52f;
            break;

        case HMC5883L_RANGE_4GA:
            mgPerDigit = 2.27f;
            break;

        case
HMC5883L_RANGE_4_7GA:
            mgPerDigit = 2.56f;
            break;

        case
HMC5883L_RANGE_5_6GA:
            mgPerDigit = 3.03f;
            break;

        case
HMC5883L_RANGE_8_1GA:
            mgPerDigit = 4.35f;
            break;

        default:
            break;
    }


writeRegister8(HMC5883L_REG_CONFI
G_B, range << 5);
}

void HMC5883L::setRange(int range)
{
    setRange((hmc5883l_range)range);
}

hmc5883l_range
HMC5883L::getRange(void)
{
    return
(hmc5883l_range)((readRegister8(HMC5
883L_REG_CONFIG_B) >> 5));
}

void
HMC5883L::setMeasurementMode(hmc5
```

```
883l_mode mode)
{
    byte value;

    value =
readRegister8(HMC5883L_REG_MODE)
;
    value &= 0b11111100;
    value |= mode;


writeRegister8(HMC5883L_REG_MODE
, value);
}

void
HMC5883L::setMeasurementMode(int
mode)
{

setMeasurementMode((hmc5883l_mode)
mode);
}

hmc5883l_mode
HMC5883L::getMeasurementMode(void)
{
    byte value;

    value =
readRegister8(HMC5883L_REG_MODE)
;
    value &= 0b00000011;

    return (hmc5883l_mode)value;
}
```

```
void
HMC5883L::setDataRate(hmc5883l_data
Rate rate)
{
    byte value;

    value =
readRegister8(HMC5883L_REG_CONFI
G_A);
    value &= 0b11100011;
    value |= (rate << 2);


writeRegister8(HMC5883L_REG_CONFI
G_A, value);
}

void HMC5883L::setDataRate(int rate)
{

setDataRate((hmc5883l_dataRate)rate);
}

hmc5883l_dataRate
HMC5883L::getDataRate(void)
{
    byte value;

    value =
readRegister8(HMC5883L_REG_CONFI
G_A);
    value &= 0b00011100;
    value >>= 2;

    return (hmc5883l_dataRate)value;
```

```
}

void HMC5883L::setSamples(hmc5883l_samples samples)
{
    byte value;

    value = readRegister8(HMC5883L_REG_CONFIG_A);
    value &= 0b10011111;
    value |= (samples << 5);

    writeRegister8(HMC5883L_REG_CONFIG_A, value);
}

void HMC5883L::setSamples(int samples)
{
    setSamples((hmc5883l_samples)samples);
}

hmc5883l_samples HMC5883L::getSamples(void)
{
    byte value;

    value = readRegister8(HMC5883L_REG_CONFIG_A);
    value &= 0b01100000;
    value >>= 5;

    return (hmc5883l_samples)value;
}

void HMC5883L::setSettings(String str)
{
        String sNum;

        str = str.substring(str.indexOf("set") + 3);
        sNum = str.substring(0, str.indexOf(","));
        setRange(sNum.toInt());

        str = str.substring(str.indexOf(",") + 1);
        sNum = str.substring(0, str.indexOf(","));

        setMeasurementMode(sNum.toInt());

        str = str.substring(str.indexOf(",") + 1);
        sNum = str.substring(0, str.indexOf(","));
        setDataRate(sNum.toInt());

        str = str.substring(str.indexOf(",") + 1);
        setSamples(str.toInt());

        }
```

```
void HMC5883L::printSettings(void)
{
    Serial.print("Selected range: ");
    switch (getRange())
    {
        case
HMC5883L_RANGE_0_88GA:
Serial.println("0.88 Ga"); break;
        case
HMC5883L_RANGE_1_3GA:
Serial.println("1.3 Ga");   break;
        case
HMC5883L_RANGE_1_9GA:
Serial.println("1.9 Ga");   break;
        case
HMC5883L_RANGE_2_5GA:
Serial.println("2.5 Ga");   break;
        case HMC5883L_RANGE_4GA:
Serial.println("4 Ga");     break;
        case
HMC5883L_RANGE_4_7GA:
Serial.println("4.7 Ga");   break;
        case
HMC5883L_RANGE_5_6GA:
Serial.println("5.6 Ga");   break;
        case
HMC5883L_RANGE_8_1GA:
Serial.println("8.1 Ga");   break;
        default:
Serial.println("Bad range!");
    }

    Serial.print("Selected Measurement
Mode: ");
    switch (getMeasurementMode())
    {
        case HMC5883L_IDLE:
Serial.println("Idle mode");
break;
        case HMC5883L_SINGLE:
Serial.println("Single-Measurement");
break;
        case HMC5883L_CONTINOUS:
Serial.println("Continuous-
Measurement"); break;
        default:
Serial.println("Bad mode!");
    }

    Serial.print("Selected Data Rate: ");
    switch (getDataRate())
    {
        case
HMC5883L_DATARATE_0_75_HZ:
Serial.println("0.75 Hz"); break;
        case
HMC5883L_DATARATE_1_5HZ:
Serial.println("1.5 Hz");   break;
        case
HMC5883L_DATARATE_3HZ:
Serial.println("3 Hz");     break;
        case
HMC5883L_DATARATE_7_5HZ:
Serial.println("7.5 Hz");   break;
        case
HMC5883L_DATARATE_15HZ:
Serial.println("15 Hz");    break;
        case
HMC5883L_DATARATE_30HZ:
Serial.println("30 Hz");    break;
        case
HMC5883L_DATARATE_75HZ:
```

```
Serial.println("75 Hz");     break;
            default:
Serial.println("Bad data rate!");
      }

      Serial.print("Selected number of
samples: ");
      switch (getSamples())
      {
            case HMC5883L_SAMPLES_1:
Serial.println("1"); break;
            case HMC5883L_SAMPLES_2:
Serial.println("2"); break;
            case HMC5883L_SAMPLES_4:
Serial.println("4"); break;
            case HMC5883L_SAMPLES_8:
Serial.println("8"); break;
            default:
Serial.println("Bad number of samples!");
      }
}

String HMC5883L::getSettings(void)
{
            String str = "";

            switch (getRange())
            {
            case
HMC5883L_RANGE_0_88GA: str += "0";
break;
            case
HMC5883L_RANGE_1_3GA:    str += "1";
break;
            case
HMC5883L_RANGE_1_9GA:    str += "2";
break;
            case
HMC5883L_RANGE_2_5GA:    str += "3";
break;
            case HMC5883L_RANGE_4GA:
str += "4"; break;
            case
HMC5883L_RANGE_4_7GA:    str += "5";
break;
            case
HMC5883L_RANGE_5_6GA:    str += "6";
break;
            case
HMC5883L_RANGE_8_1GA:    str += "7";
break;
            default:                        str
+= "-1";
            }

            str += ",";
            switch (getMeasurementMode())
            {
            case HMC5883L_CONTINOUS:
str += "0"; break;
            case HMC5883L_SINGLE:
str += "1"; break;
            case HMC5883L_IDLE:
str += "2"; break;
            default:                        str +=
"-1";
            }

            str += ",";
            switch (getDataRate())
            {
            case
```

```
HMC5883L_DATARATE_0_75_HZ: str += "0"; break;
        case HMC5883L_DATARATE_1_5HZ: str += "1"; break;
        case HMC5883L_DATARATE_3HZ: str += "2"; break;
        case HMC5883L_DATARATE_7_5HZ: str += "3"; break;
        case HMC5883L_DATARATE_15HZ: str += "4"; break;
        case HMC5883L_DATARATE_30HZ: str += "5"; break;
        case HMC5883L_DATARATE_75HZ: str += "6"; break;
        default: str += "-1";
        }

        str += ",";
        switch (getSamples())
        {
        case HMC5883L_SAMPLES_1: str += "0"; break;
        case HMC5883L_SAMPLES_2: str += "1"; break;
        case HMC5883L_SAMPLES_4: str += "2"; break;
        case HMC5883L_SAMPLES_8: str += "3"; break;
        default: str += "-1";
        }

        str += ",";
        str += String(getOffSet().XAxis);
        str += ",";
        str += String(getOffSet().YAxis);
        str += ",";
        str += String(getOffSet().ZAxis);

        return str;
}

void HMC5883L::writeRegister8(byte reg, byte value)
{

Wire.beginTransmission(HMC5883L_ADDRESS);
    Wire.write(reg);
    Wire.write(value);
    Wire.endTransmission(true);
}

byte HMC5883L::readRegister8(byte reg)
{
    byte value;

Wire.beginTransmission(HMC5883L_ADDRESS);
    Wire.write(reg);
    Wire.endTransmission(false);

Wire.requestFrom(HMC5883L_ADDRESS, 1);
```

```
    value = Wire.read();

    return value;
}
```

9.7 L6470.cpp

```
#include "L6470.h"
#include "SPI.h"

L6470::L6470() {}

void L6470::init(void)
{

pinMode(L6470_PIN_SS, OUTPUT);

digitalWrite(L6470_PIN_SS, HIGH);

pinMode(L6470_PIN_MOSI, OUTPUT);

pinMode(L6470_PIN_MISO, INPUT);

pinMode(L6470_PIN_SCK, OUTPUT);


SPI.begin();

SPI.setBitOrder(MSBFIRST);

SPI.setClockDivider(SPI_CLOCK_DIV16
);

SPI.setDataMode(SPI_MODE3);


commandNOP();

commandNOP();

commandNOP();

commandNOP();

commandResetDevice();


setMAX_SPEED(calcRegValMAX_SPEE
D(200));

setMIN_SPEED(0x1000);

setKVAL_HOLD(0x20);

setKVAL_RUN(0x40);

setKVAL_ACC(0x40);

setKVAL_DEC(0x40);

setSTEP_MODE(0x00);


commandHardStop();
}

boolean L6470::isBusy(void)
{

return !bitRead(commandGetStatus(), 1);
}

void L6470::delayBusy(void)
{

while (isBusy()) {}
}
```

```
void L6470::delayBusy(long milliSec)
{

while (isBusy()) {}

delay(milliSec);
}


void L6470::commandNOP(void)
{

Xfer((byte)L6470_COMMAND_NOP);
}

void
L6470::commandSetParam(l6470_param
param, long value)
{

switch (param)

{


case L6470_PARAM_ABS_POS:
setABS_POS(value); break;


case L6470_PARAM_EL_POS:
setEL_POS(value); break;


case L6470_PARAM_MARK:
setMARK(value); break;


case L6470_PARAM_SPEED: /*
read only */ break;


case L6470_PARAM_ACC:
setACC(value); break;


case L6470_PARAM_DEC:
setDEC(value); break;

case L6470_PARAM_MAX_SPEED:
setMAX_SPEED(value); break;

case L6470_PARAM_MIN_SPEED:
setMIN_SPEED(value); break;

case L6470_PARAM_FS_SPD:
setFS_SPD(value); break;


case L6470_PARAM_KVAL_HOLD:
setKVAL_HOLD(value); break;


case L6470_PARAM_KVAL_RUN:
setKVAL_RUN(value); break;
```

```
case L6470_PARAM_KVAL_ACC: setKVAL_ACC(value); break;

case L6470_PARAM_KVAL_DEC: setKVAL_DEC(value); break;

case L6470_PARAM_INT_SPEED: setINT_SPEED(value); break;

case L6470_PARAM_ST_SLP: setST_SLP(value); break;

case L6470_PARAM_FN_SLP_ACC: setFN_SLP_ACC(value); break;

case L6470_PARAM_FN_SLP_DEC: setFN_SLP_DEC(value); break;

case L6470_PARAM_K_THERM: setK_THERM(value); break;

case L6470_PARAM_ADC_OUT: /* read only */ break;

case L6470_PARAM_OCD_TH: setOCD_TH(value); break;

case L6470_PARAM_STALL_TH: setSTALL_TH(value); break;

case L6470_PARAM_STEP_MODE: setSTEP_MODE(value); break;

case L6470_PARAM_ALARM_EN: setALARM_EN(value); break;

case L6470_PARAM_CONFIG: setCONFIG(value); break;

case L6470_PARAM_STATUS: /* read only */ break;

default:
break;

}
}

long L6470::commandGetParam(l6470_param param)
{

long value = 0;

switch (param)
```

```
{                                                   = getFS_SPD();        break;

case L6470_PARAM_ABS_POS:      value      case        L6470_PARAM_KVAL_HOLD:
= getABS_POS();       break;                        value = getKVAL_HOLD();   break;

case L6470_PARAM_EL_POS:       value      case        L6470_PARAM_KVAL_RUN:
= getEL_POS();        break;                        value = getKVAL_RUN();    break;

case L6470_PARAM_MARK:         value      case        L6470_PARAM_KVAL_ACC:
= getMARK();          break;                        value = getKVAL_ACC();    break;

case L6470_PARAM_SPEED:        value      case        L6470_PARAM_KVAL_DEC:
= getSPEED();         break;                        value = getKVAL_DEC();    break;

case L6470_PARAM_ACC:          value      case        L6470_PARAM_INT_SPEED:
= getACC();           break;                        value = getINT_SPEED();   break;

case L6470_PARAM_DEC:          value      case L6470_PARAM_ST_SLP:      value
= getDEC();           break;                        = getST_SLP();        break;

case        L6470_PARAM_MAX_SPEED:         case        L6470_PARAM_FN_SLP_ACC:
value = getMAX_SPEED();   break;                      value = getFN_SLP_ACC(); break;

case        L6470_PARAM_MIN_SPEED:         case        L6470_PARAM_FN_SLP_DEC:
value = getMIN_SPEED();   break;                      value = getFN_SLP_DEC(); break;

case L6470_PARAM_FS_SPD:       value      case        L6470_PARAM_K_THERM:
```

```
value = getK_THERM();       break;

case              L6470_PARAM_ADC_OUT:
value = getADC_OUT();       break;

case              L6470_PARAM_OCD_TH:
value = getOCD_TH();        break;

case              L6470_PARAM_STALL_TH:
value = getSTALL_TH();      break;

case          L6470_PARAM_STEP_MODE:
value = getSTEP_MODE();     break;

case          L6470_PARAM_ALARM_EN:
value = getALARM_EN();      break;

case L6470_PARAM_CONFIG:        value
= getCONFIG();        break;

case L6470_PARAM_STATUS:        value
= getSTATUS();        break;

default:                        value = -1;
break;

}
```

```
return value;
}

void
L6470::commandRun(l6470_deirection
dir, long spd)
{

spd = checkOverflow(spd, 0x0FFFFF);

Xfer((byte)L6470_COMMAND_RUN      |
dir);

Xfer((byte)(spd >> 16));

Xfer((byte)(spd >> 8));

Xfer((byte)(spd));
}

void
L6470::commandStep_Clock(l6470_deirec
tion dir)
{

Xfer((byte)L6470_COMMAND_STEP_CL
OCK | dir);
}

void
L6470::commandMove(l6470_deirection
dir, long n_step)
{

n_step        =        checkOverflow(n_step,
```

```
0x3FFFFF);

Xfer((byte)L6470_COMMAND_MOVE   |
dir);

Xfer((byte)(n_step >> 16));

Xfer((byte)(n_step >> 8));

Xfer((byte)(n_step));
}

void L6470::commandGoTo(long pos)
{

pos = checkOverflow(pos, 0x3FFFFF);

Xfer((byte)L6470_COMMAND_GOTO);

Xfer((byte)(pos >> 16));

Xfer((byte)(pos >> 8));

Xfer((byte)(pos));
}

void
L6470::commandGoTo_DIR(l6470_deirect
ion dir, long pos)
{

pos = checkOverflow(pos, 0x3FFFFF);

Xfer((byte)L6470_COMMAND_GOTO_D
IR | dir);

Xfer((byte)(pos >> 16));

Xfer((byte)(pos >> 8));

Xfer((byte)(pos));
}

void
L6470::commandGoUntil(l6470_action
act, l6470_deirection dir, long spd)
{

spd = checkOverflow(spd, 0x0200);

Xfer((byte)L6470_COMMAND_GO_UNT
IL | act | dir);

Xfer((byte)(spd >> 16));

Xfer((byte)(spd >> 8));

Xfer((byte)(spd));
}

void
L6470::commandReleaseSW(l6470_actio
n act, l6470_deirection dir)
{

Xfer((byte)L6470_COMMAND_RELEAS
E_SW | act | dir);
}

void L6470::commandGoHome(void)
{
```

```
Xfer((byte)L6470_COMMAND_GO_HOM
E);
}

void L6470::commandGoMark(void)
{

Xfer((byte)L6470_COMMAND_GO_MAR
K);
}

void L6470::commandResetPos(void)
{

Xfer((byte)L6470_COMMAND_RESET_P
OS);
}

void L6470::commandResetDevice(void)
{

Xfer((byte)L6470_COMMAND_RESET_
DEVICE);
}

void L6470::commandSoftStop(void)
{

Xfer((byte)L6470_COMMAND_SOFT_ST
OP);
}

void L6470::commandHardStop(void)
{

Xfer((byte)L6470_COMMAND_HARD_S
TOP);
}

void L6470::commandSoftHiZ(void)
{

Xfer((byte)L6470_COMMAND_SOFT_HI
Z);
}

void L6470::commandHardHiZ(void)
{

Xfer((byte)L6470_COMMAND_HARD_H
IZ);
}

long L6470::commandGetStatus(void)
{

long status = 0;


Xfer((byte)L6470_COMMAND_GET_STA
TUS);

status |= Xfer(0) << 8;

status |= Xfer(0);


return status;
}
```

```
long L6470::calcRegValSPEED(float stepsPerSec)
{

long speed = stepsPerSec * 67.108864;

return speed;
}

long L6470::calcRegValACC(float stepsPerSecPerSec)
{

long acc = stepsPerSecPerSec * 0.068719;

return acc;
}

long L6470::calcRegValDEC(float stepsPerSecPerSec)
{

long dec = stepsPerSecPerSec * 0.068719;

return dec;
}

long L6470::calcRegValMAX_SPEED(float stepsPerSec)
{

long max_speed = stepsPerSec * 0.065536;

return max_speed;
}

long L6470::calcRegValMIN_SPEED(float stepsPerSec)
{

long min_speed = stepsPerSec * 4.194304;

return min_speed;
}

long L6470::calcRegValFS_SPD(float stepsPerSec)
{

long fs_spd = (stepsPerSec * 0.065536) - 0.5;

return fs_spd;
}

long L6470::calcRegValINT_SPEED(float stepsPerSec)
{

long int_speed = stepsPerSec * 4.194304;

return int_speed;
}

long L6470::calcRegValOCD_TH(float milliAmpere)
{

long ocd_th = ocd_th = ceil(milliAmpere /
```

```
375);

return ocd_th;
}

long L6470::calcRegValSTALL_TH(float milliAmpere)
{

long stall_th = ceil(milliAmpere / 31.25);

return stall_th;
}



float L6470::calcUnitSPEED(long speed)
{

float stepsPerSec = speed * 0.015;

return stepsPerSec;
}

float L6470::calcUnitACC(long acc)
{

float stepsPerSecPerSec = acc * 14.552;

return stepsPerSecPerSec;
}

float L6470::calcUnitDEC(long dec)
{

float stepsPerSecPerSec = dec * 14.552;

return stepsPerSecPerSec;
}

float L6470::calcUnitMAX_SPEED(long max_speed)
{

float stepsPerSec = max_speed * 15.259;

return stepsPerSec;
}

float L6470::calcUnitMIN_SPEED(long min_speed)
{

float stepsPerSec = (min_speed & 0x0FFF) * 0.238;

return stepsPerSec;
}

float L6470::calcUnitFS_SPD(long fs_spd)
{

float stepsPerSec = (fs_spd + 0.5) * 15.259;

return stepsPerSec;
}

float L6470::calcUnitINT_SPEED(long int_speed)
```

```
{

float stepsPerSec = int_speed * 0.238;

return stepsPerSec;
}



void L6470::setABS_POS(long abs_pos)
{

abs_pos = checkOverflow(abs_pos,
0x3FFFFF, -0x3FFFFF);

Xfer((byte)L6470_PARAM_ABS_POS);

Xfer((byte)(abs_pos >> 16));

Xfer((byte)(abs_pos >> 8));

Xfer((byte)(abs_pos));
}

void L6470::setEL_POS(long el_pos)
{

el_pos = checkOverflow(el_pos, 0x01FF);

Xfer((byte)L6470_PARAM_EL_POS);

Xfer((byte)(el_pos >> 8));

Xfer((byte)(el_pos));
}

void L6470::setMARK(long mark)
{

mark = checkOverflow(mark, 0x3FFFFF,
-0x3FFFFF);

Xfer((byte)L6470_PARAM_MARK);

Xfer((byte)(mark >> 16));

Xfer((byte)(mark >> 8));

Xfer((byte)(mark));
}

void L6470::setACC(long acc)
{

acc = checkOverflow(acc, 0x0FFF);

Xfer((byte)L6470_PARAM_ABS_POS);

Xfer((byte)(acc >> 8));

Xfer((byte)(acc));
}

void L6470::setDEC(long dec)
{

dec = checkOverflow(dec, 0x0FFF);

Xfer((byte)L6470_PARAM_DEC);
```

```
Xfer((byte)(dec >> 8));

Xfer((byte)(dec));
}

void L6470::setMAX_SPEED(long max_speed)
{

max_speed = checkOverflow(max_speed, 0x03FF);

Xfer(L6470_PARAM_MAX_SPEED);

Xfer((byte)(max_speed >> 8));

Xfer((byte)(max_speed));
}

void L6470::setMIN_SPEED(long min_speed)
{

if (min_speed <= 0x0010)

{


min_speed |= 0x1000;

}

min_speed = checkOverflow(min_speed, 0x0FFF);

Xfer((byte)L6470_PARAM_MAX_SPEED
);

Xfer((byte)(min_speed >> 8));

Xfer((byte)(min_speed));
}

void L6470::setFS_SPD(long fs_spd)
{

fs_spd = checkOverflow(fs_spd, 0x03FF);

Xfer((byte)L6470_PARAM_FS_SPD);

Xfer((byte)(fs_spd >> 8));

Xfer((byte)(fs_spd));
}

void L6470::setKVAL_HOLD(long kval_hold)
{

kval_hold = checkOverflow(kval_hold, 0xFF);

Xfer((byte)L6470_PARAM_KVAL_HOLD)
;

Xfer((byte)kval_hold);
}

void L6470::setKVAL_RUN(long kval_run)
{
```

```
kval_run = checkOverflow(kval_run, 0xFF);

Xfer((byte)L6470_PARAM_KVAL_RUN);

Xfer((byte)kval_run);
}

void L6470::setKVAL_ACC(long kval_acc)
{

kval_acc = checkOverflow(kval_acc, 0xFF);

Xfer((byte)L6470_PARAM_KVAL_ACC);

Xfer((byte)kval_acc);
}

void L6470::setKVAL_DEC(long kval_dec)
{

kval_dec = checkOverflow(kval_dec, 0xFF);

Xfer((byte)L6470_PARAM_KVAL_DEC);

Xfer((byte)kval_dec);
}

void L6470::setINT_SPEED(long int_speed)
{

int_speed = checkOverflow(int_speed, 0x3FFF);

Xfer((byte)L6470_PARAM_INT_SPEED);

Xfer((byte)(int_speed >> 8));

Xfer((byte)(int_speed));
}

void L6470::setST_SLP(long st_slp)
{

st_slp = checkOverflow(st_slp, 0xFF);

Xfer((byte)L6470_PARAM_ST_SLP);

Xfer((byte)st_slp);
}

void L6470::setFN_SLP_ACC(long fn_slp_acc)
{

fn_slp_acc = checkOverflow(fn_slp_acc, 0xFF);

Xfer((byte)L6470_PARAM_FN_SLP_ACC);

Xfer((byte)fn_slp_acc);
}

void L6470::setFN_SLP_DEC(long fn_slp_dec)
{
```

```
fn_slp_dec = checkOverflow(fn_slp_dec,
0xFF);

Xfer((byte)L6470_PARAM_FN_SLP_DEC
);

Xfer((byte)fn_slp_dec);
}

void L6470::setK_THERM(long k_therm)
{

k_therm = checkOverflow(k_therm,
0x0F);

Xfer((byte)L6470_PARAM_K_THERM);

Xfer((byte)(k_therm));
}

void L6470::setOCD_TH(long ocd_th)
{

ocd_th = checkOverflow(ocd_th, 0x0F);

Xfer((byte)L6470_PARAM_OCD_TH);

Xfer((byte)(ocd_th));
}

void L6470::setSTALL_TH(long stall_th)
{

stall_th = checkOverflow(stall_th, 0x7F);

Xfer((byte)L6470_PARAM_STALL_TH);
Xfer((byte)(stall_th));
}

void L6470::setSTEP_MODE(long
step_mode)
{

step_mode = checkOverflow(step_mode,
0xF7);

Xfer((byte)L6470_PARAM_STEP_MODE
);

Xfer((byte)(step_mode));
}

void L6470::setALARM_EN(long alarm)
{

alarm = checkOverflow(alarm, 0xFF);

Xfer((byte)L6470_PARAM_ALARM_EN);

Xfer((byte)(alarm));
}

void L6470::setCONFIG(long config)
{

config = checkOverflow(config, 0xFFBF);

Xfer((byte)L6470_PARAM_CONFIG);

Xfer((byte)(config >> 8));
```

```
Xfer((byte)(config));
}



long L6470::getABS_POS(void)
{

long abs_pos = 0;


Xfer((byte)L6470_PARAM_ABS_POS);

abs_pos |= Xfer(0) << 16;

abs_pos |= Xfer(0) << 8;

abs_pos |= Xfer(0);

if (bitRead(abs_pos, 21))

{


abs_pos = ~abs_pos;

}


return abs_pos;
}

long L6470::getEL_POS(void)
{

long el_pos = 0;


Xfer((byte)L6470_PARAM_EL_POS);

el_pos |= Xfer(0);


return el_pos;
}

long L6470::getMARK(void)
{

long mark = 0;


Xfer((byte)L6470_PARAM_MARK);

mark |= Xfer(0) << 16;

mark |= Xfer(0) << 8;

mark |= Xfer(0);

if (bitRead(mark, 21))

{


mark = ~mark;

}
```

```
return mark;
}

long L6470::getSPEED(void)
{

long speed = 0;


Xfer((byte)L6470_PARAM_SPEED);

speed |= Xfer(0) << 16;

speed |= Xfer(0) << 8;

speed |= Xfer(0);


return speed;
}

long L6470::getACC(void)
{

long acc = 0;


Xfer((byte)L6470_PARAM_ACC);

acc |= Xfer(0) << 8;

acc |= Xfer(0);


return acc;
}
```

```
long L6470::getDEC(void)
{

long dec = 0;


Xfer((byte)L6470_PARAM_DEC);

dec |= Xfer(0) << 8;

dec |= Xfer(0);


return dec;
}

long L6470::getMAX_SPEED(void)
{

long max_speed = 0;


Xfer((byte)L6470_PARAM_MAX_SPEED
);

max_speed |= Xfer(0) << 8;

max_speed |= Xfer(0);


return max_speed;
}

long L6470::getMIN_SPEED(void)
{
```

```
long min_speed = 0;

Xfer((byte)L6470_PARAM_MIN_SPEED)
;

min_speed |= Xfer(0) << 8;

min_speed |= Xfer(0);


return min_speed;
}

long L6470::getFS_SPD(void)
{

long fs_spd = 0;


Xfer((byte)L6470_PARAM_FS_SPD);

fs_spd |= Xfer(0) << 8;

fs_spd |= Xfer(0);


return fs_spd;
}

long L6470::getKVAL_HOLD(void)
{

long kval_hold = 0;

Xfer((byte)L6470_PARAM_KVAL_HOLD)
;

kval_hold |= Xfer(0);


return kval_hold;
}

long L6470::getKVAL_RUN(void)
{

long kval_run = 0;


Xfer((byte)L6470_PARAM_KVAL_RUN);

kval_run |= Xfer(0);


return kval_run;
}

long L6470::getKVAL_ACC(void)
{

long kval_acc = 0;


Xfer((byte)L6470_PARAM_KVAL_ACC);

kval_acc |= Xfer(0);


return kval_acc;
```

```
}

long L6470::getKVAL_DEC(void)
{

long kval_dec = 0;


Xfer((byte)L6470_PARAM_KVAL_DEC);

kval_dec |= Xfer(0);


return kval_dec;
}

long L6470::getINT_SPEED(void)
{

long int_speed = 0;


Xfer((byte)L6470_PARAM_INT_SPEED);

int_speed |= Xfer(0) << 8;

int_speed |= Xfer(0);


return int_speed;
}

long L6470::getST_SLP(void)
{

long st_slp = 0;

Xfer((byte)L6470_PARAM_ST_SLP);

st_slp |= Xfer(0);


return st_slp;
}

long L6470::getFN_SLP_ACC(void)
{

long fn_slp_acc = 0;


Xfer((byte)L6470_PARAM_FN_SLP_ACC
);

fn_slp_acc |= Xfer(0);


return fn_slp_acc;
}

long L6470::getFN_SLP_DEC(void)
{

long fn_slp_dec = 0;


Xfer((byte)L6470_PARAM_FN_SLP_DEC
);

fn_slp_dec |= Xfer(0);
```

```
return fn_slp_dec;
}

long L6470::getK_THERM(void)
{

long k_therm = 0;


Xfer((byte)L6470_PARAM_K_THERM);

k_therm |= Xfer(0);


return k_therm;
}

long L6470::getADC_OUT(void)
{

long adc_out = 0;


Xfer((byte)L6470_PARAM_ADC_OUT);

adc_out |= Xfer(0);


return adc_out;
}

long L6470::getOCD_TH(void)
{

long ocd_th = 0;


Xfer((byte)L6470_PARAM_OCD_TH);

ocd_th |= Xfer(0);


return ocd_th;
}

long L6470::getSTALL_TH(void)
{

long stall_th = 0;


Xfer((byte)L6470_PARAM_STALL_TH);

stall_th |= Xfer(0);


return stall_th;
}

long L6470::getSTEP_MODE(void)
{

long step_mode = 0;


Xfer((byte)L6470_PARAM_STEP_MODE
);

step_mode |= Xfer(0);
```

```
return step_mode;
}

long L6470::getALARM_EN(void)
{

long alarm_en = 0;


Xfer((byte)L6470_PARAM_ALARM_EN);

alarm_en |= Xfer(0);


return alarm_en;
}

long L6470::getCONFIG(void)
{

long config = 0;


Xfer((byte)L6470_PARAM_CONFIG);

config |= Xfer(0) << 8;

config |= Xfer(0);


return config;
}

long L6470::getSTATUS(void)
{

long status = 0;


Xfer((byte)L6470_PARAM_STATUS);

status |= Xfer(0) << 8;

status |= Xfer(0);


return status;
}

long L6470::checkOverflow(long data,
long range)
{

if (range > 0)

{


return checkOverflow(data, range, 0);

}

else if (range < 0)

{


return checkOverflow(data, 0, range);
```

```
}

else

{

return 0;

}
}

long L6470::checkOverflow(long data, long max_range, long min_range)
{

if (data > max_range)

{

data = max_range;

}

if (data < min_range)

{


data = min_range;

}
```

```
return data;
}


byte L6470::Xfer(byte data)
{

byte data_out;


digitalWrite(L6470_PIN_SS, LOW);

data_out = SPI.transfer(data);

digitalWrite(L6470_PIN_SS, HIGH);


return data_out;
}
```

気象観測機器スーちゃん

アイスのスーちゃんとお天気の物語

稲葉毬恵
金澤涼夏
鈴木万結子
中江雪乃
大阪教育大学附属高等学校天王寺校舎 地学部
2017年10月20日

要旨

この研究では、一目見て、気温と紫外線量、湿度を目に見える形で教えてくれる「スライム」を利用し、アイスクリームを模したちょっと可愛い気象観測機器の作製を試みた。

本機器に最適な材料を実験により見出すことができたため、画像やデータを掲載した。しかし、機器としての実用性については調査が不足しており、課題も多く抱えている。

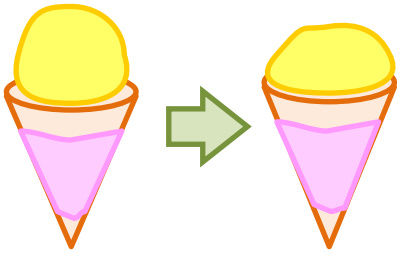
はじめに

スライムは暑くなるとドロドロに溶けてしまう。そんなスライムの溶け具合で温度が計測できるのではないかと考え、今回の気象観測機器を考案した。

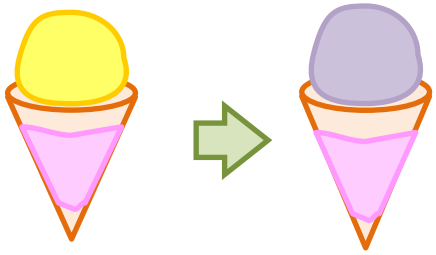
気象観測機器：スーちゃん

特徴

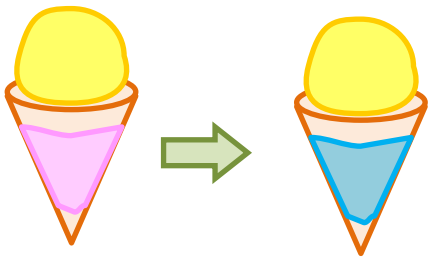
特徴①　暑さでアイスが溶ける



特徴② 紫外線量でアイスが変色する



特徴③ 湿度で巻紙が変色する



「スーちゃん」は

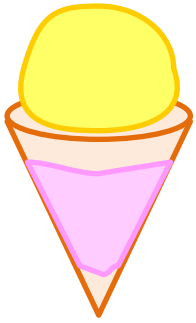
- **気温**
- **紫外線量**
- **湿度**

の三つの気象要素を簡単に視覚的に判断できる気象観測機器である。

構造

コーン部分
- 紙粘土

巻紙部分
- 湿度で色変化する紙



アイス部分
- 紫外線変色塗料を混ぜ込んだスライム

スーちゃん作製に伴う実験

実験１:スライムの硬度変化実験

目的

スーちゃんのアイス部分に最適な硬度のスライムを作製する。

方法

ホウ砂、水、洗濯のりの量を変化させ、適切な硬度のスライムを作製し(表2-1)、①を基準として、弾力、伸縮、実用性の３つの面で比較を行った。弾力とは机にスライムを落とした際の跳ね返り具合、伸縮とは手で上下に引っ張った際の伸び具合(図3-2)、実用性とは手に乗せた際の形の崩れにくさ(図3-3)を示す。

(表2-1)　スライム硬度変化実験 材料配分表

	①	②	③	④	⑤
ホウ砂(g)	2.5	2.5	3.5	2.5	3.5
水(g)	75	75	75	95	75
洗濯のり(g)	50	70	50	50	70

実験２:紫外線変色スライム作成実験

目的

スライムを紫外線が照射されたときに視覚的な変化が起こるものにする。

方法

文献調査で得た紫外線で変色するものの一つ、紫外線増白剤を含む洗剤をスライムに混ぜ込む、塗るの2つの方法で実験を行った。

実験３:塩化コバルト紙の反応度実験

目的

通常の塩化コバルト紙は低湿度でも反応してしまい本研究には不向きのため、湿度の変化がよりわかりやすく、色変化の時間が短い、最適な紙を作製する。

方法

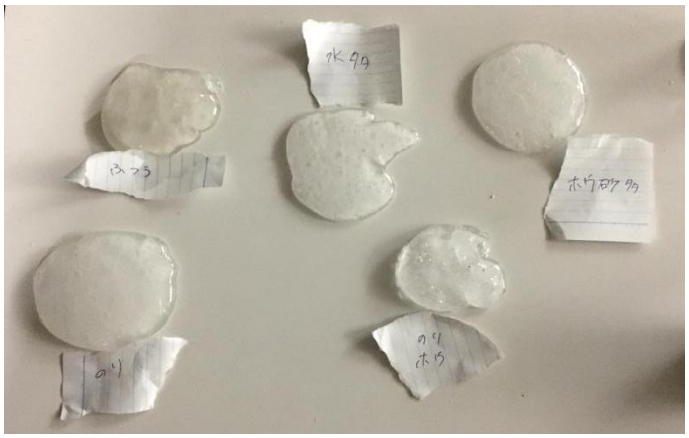
濾紙に含ませる塩化コバルトの量を変化させ、適切な塩化コバルト紙を作成し(表2-2）、高湿度の浴室で塩化コバルト紙が水に反応し、色が変化する速度と、色の彩度を目視で①と比較した。

(表2-２)　コバルト紙の反応実験 材料配分表

	①	②	③	④
塩化コバルト(g)	0.4	0.2	0.6	0.4
塩化カルシウム(g)	0.2	0.2	0.2	0.2
水(g)	3.4	3.4	3.4	1.7

実験結果１：スライム硬度変化実験

実験結果



(図3-1)　制作したスライム一覧

(図3-2)　スライムの伸縮調査の様子

(表3-1)　スライム硬度変化実験　実験結果一覧

	②	③	④	⑤
弾力性	◎	○	○	◎
伸縮性	△	◎	◎	△
固まりかた	○	△	× 固まらない	◎

(図3-３)　スライム固まり具合調査の様子

実験結果(表3-1)は①と比較して、弾力はよく跳ねた方から、伸縮性はよく伸びた方から、固まり方はより形が崩れにくかったものから、◎、○、△、×と四段階で評価する。この結果から、弾力性、実力性の面で⑤（ホウ砂3.5g,水75g,のり70g）のスライムがスーちゃんに適していると判断した。よってこのスライムをスーちゃんに使用することとした。

考察・課題

結果②⑤より全体分量に対しのりが多いものほど固まりやすく、弾力性の強いものができているが、スライムは洗濯のりに含まれるポリビニルアルコール(PVA)がホウ砂イオンを介して水素結合してできているため、洗濯のりが多く全体量に対しホウ砂が少なくなる②や⑤はより結合しにくくなるはずなので不思議な結果となった。

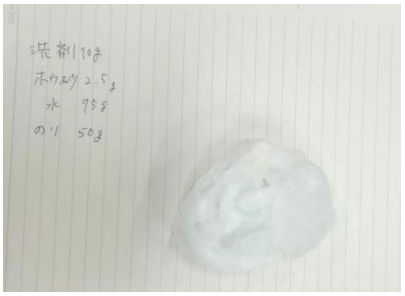
また前述を考慮すると、ホウ砂の割合の高い③において、スライムが固まりにくいという結果になっているため、これらの要因について、文献、実験による調査を深め、原因を調べていきたい。

実験結果２：紫外線変色スライム作成実験

実験結果

結果①

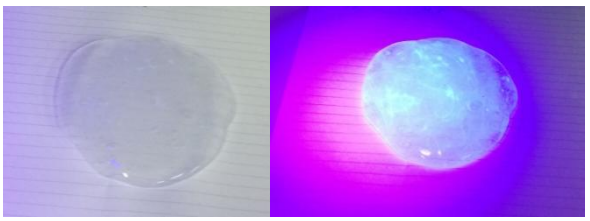
スライム作成時に、紫外線増白剤の入った洗剤を混ぜ込むと弾力性を失い、硬い個体になってしまった。（図4-1）



(図4-1)　硬化したスライム

結果②

作成済みのスライムの周りに洗剤をぬり、395nmのブラックライトを照射したところ、光る様子が確認された。この結果より、スーちゃんにはこの方法を使用することにした。（図4-2）



(図4-2)　紫外線を受けて発光するスライム

考察・課題

結果①でスライムが弾力性を失い硬くなった原因として、洗剤に含まれる塩化ナトリウムが影響を与えたと考え、文献調査を行った。その結果、スライムと塩化ナトリウム間で濃度の差異が生じ、高張液となっている塩化ナトリウムとの濃度の差を埋めるためにスライム内の水分が出てくることによって、構造が壊れてしまった可能性が考えられた。

結果②において、使用したブラックライトの波長は395nmであるが、日常生活で太陽から照射される紫外線の主な波長はUV-Aで、波長は310nm~380nmと、ブラックライトに比べて短いため、実用的な面で正確なデータを得られなかったことが課題としてあげられる。

また、洗剤をスライムの周囲に塗ることでスライムにべたつきが生じるため対策を講じる必要がある。

また、蛍光増白剤は紫外線を受けると光るものであるが、白昼に光っていても周囲が明るいために光っていることが分からないのではないかという非常に重大な問題が考えられた。しかし、天候に恵まれず、雨天が続いたために野外実験を行うことができておらず、使用可能かも不明である。

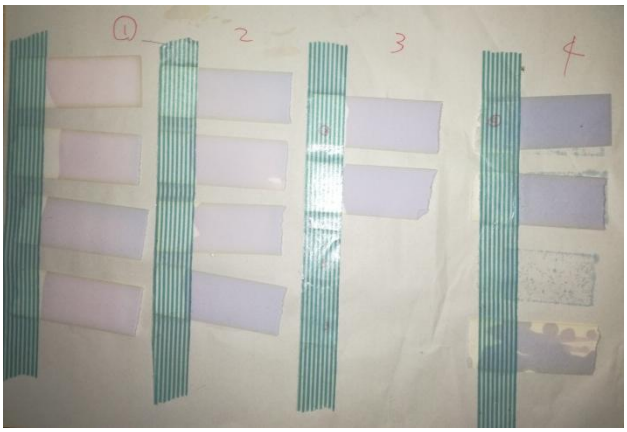
今回は紫外線に反応し光るものを使用したが、実用性の低さも目立つため、今後は、紫外線で日焼けをして変色するバナナの皮の性質や、紫外線変色ビーズなどの成分を調査し、応用することで、より視覚的かつ実用的な紫外線変色塗料を作成していきたいと考えている。

実験結果３：塩化コバルト紙の反応度実験

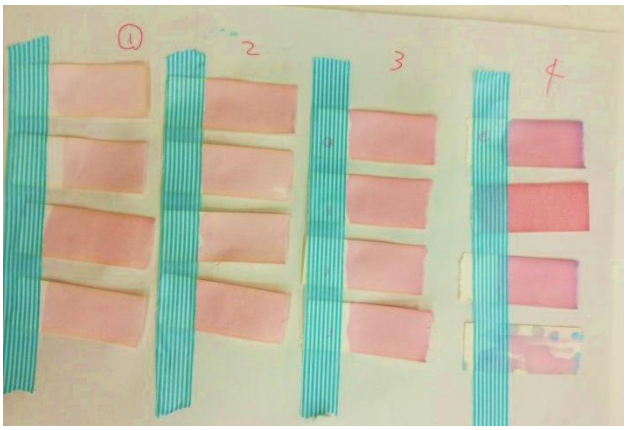
実験結果

(表5-1)　塩化コバルト紙の反応実験　結果一覧

	②	③	④
彩度	△	○	◎
色が変化する速度	△	○	○



(図5-1)　青く変色する塩化コバルト紙



(図5-2)　赤く変色する塩化コバルト紙

実験結果(表5-1)は①と比較し、彩度は色鮮やかな方から、色が変化する速度は変色速度の速い方から、◎、○、△、×の四段階で評価する。この結果より、④の塩化コバルト紙が彩度、色が変化する速度の点でスーちゃんに適していると判断した。よって、この塩化コバルト紙をスーちゃんに使用することとした。

考察

結果①②③より、水の量に対し塩化コバルトの量を増加させるほど、紙が変色する速度が速くなった。この原因としては、全体量に対して、色変化を起こす塩化コバルトの量の割合が多いことが考えられた。

今回の実験では、湿度や気温管理が非常に曖昧な環境での調査となってしまったため、今後は厳密に湿度を管理し、変色時間なども測定し、精密なデータも収集したい。

④は水の量を基準となる①から半減させて制作した結果であるが、これも上記の考察同様、全体量に対し、塩化コバルトの量が多く、変色速度が速くなっている。また彩度が高いのは、水の量が極端に少ないため、塩化コバルトの色が薄められなかったためだと考えられた。

結果：完成品

運用面

利点
・簡易的に使用できる。
・見た目が可愛い。
・視覚的に利用できるため、教材等にも使用できる。

課題点
・コーン部分に使用した粘土が重い
・スライムが溶けたときにコーンからあふれてしまう。
・長期間スライムを置いていると乾燥で固化してしまう。

詳細

コーン部分
・紙粘土

巻紙部分
巻紙に作成した塩化コバルト紙を装飾として使用。
塩化コバルト紙
・塩化コバルト0.4g
・塩化カルシウム0.2g
・水1.7ｇ
・濾紙

アイス部分
スライム
・ホウ砂3.5g
・水75g
・のり70g
紫外線で変色するよう蛍光増白剤(洗剤)を塗装。

考察

私たちの生活に身近な気象を、簡易的かつ視覚的に表現するものが制作できたため、まだ数値を読めない小さな子供などにも理科の世界への興味を誘うようなものにもなると考える。
また、視覚的にも可愛いので、インテリアとしても使用しやすく、あらゆる層の人に使用されやすいと思う。さらに今後、ストラップなど携帯式のスーちゃんを作成するなど、より気象を身近に観測できるものにしたい。

終わりに

今後の課題等

今回の実験を経て残った課題である。

・蛍光増白剤が白昼に発光していても誰も気付かない。
・蛍光増白剤が、波長の長い太陽光の紫外線に反応するのかが不明。
・スライムが洗剤でべた付く。
・湿度変化による塩化コバルト紙の色変化のデータが無い。
・スーちゃんが重い。
・アイスが溶けて溢れ出す。
・アイスが固化するため長期間の使用ができない。

全体として実験の試行量が非常に少ないため、どの実験においてもあまり正確とは言えない結果になり、ありとあらゆる問題が浮上したが、上記の課題を参考に、研究を深め、さらに便利で精密なアイスクリームの気象観測機器を作っていきたいと思う。

感想

研究開始が遅れたことに加え、学校行事やテストが重なり部員の確保が難しく思うように進めることができず、実験は、データ量及び試行回数が非常に少なくなり、満足の結果を得ることができなかった。

しかし、自分たちで観測機器を作るという点では、未知の経験で、またアイスクリームを模した簡易気象観測機器という実用性が高く、かわいく素敵な自慢の機器が製作できて非常に楽しい研究ができた。製作費用を頂いてありがとうございました。

謝辞

実験材料の準備や、実験の相談を度々受けて頂いた、地学部顧問の井村有里先生。ご協力ありがとうございました。

参考文献

・「スライムができる原理」http://slimeeee.com/about/principle.html(参照2017,10,4)
・「紫外線で浮かび上がる絵」https://site.ngk.co.jp/lab/no125/index.html(参照2017,10,10)
・山田真実,吉川廉「スライムの性質変化とその利用について」
・福島県立福島高等学校サイエンス探究クラス化学班スライムグループ「スライムの化学構造に関する一考察」
・朝日奈優衣,竹島亜実,東山つぼみ「スライム」
・「非電化湿度計『紫陽花』」http://www.hidenka.net/hidenkaseihin/ajisai/(参照2017,10,10)