# D-Egg Scan Box 説明書

Yuya Takemasa

2022 年 6 月 10 日

## Change Log

- 2022-03-09：Created the original manual (Y. Takemasa)

## 内容

この文書は D-Egg Scan Box の取扱説明書です。D-Egg Scan Box の使い方を中心に書いてあります。詳細は次のページの目次から飛んでください。この文書は overleaf にて作成しました。新たに仕様が変わり説明書を変更したい場合はリンクから編集し PDF ファイルをアップロードして下さい。またソースコードに関しては github に上がっていますので合わせながら読むといいでしょう。その他不明点があれば Slack の Yuya Takemasa まで連絡ください。

# 目 次

# Listings

# 第1章　はじめに

## 1.1　D-Egg Scan Boxの目的

D-Egg Scan Box は D-Egg に向けてコリメートされた光を打ち D-Egg 全体を走査して D-Egg のガラスやゲル中の光伝搬を含めた光電子増倍管の応答を評価する目的で製作を行いました。また今後 D-Egg 内の較正用 LED の光分布の測定を行ったり、次世代検出器の測定にも使用できるように応用可能な装置になっています。

## 1.2　D-Egg Scan Boxの全体図

D-Egg Scan Box は図 1.1 に示すような高さ 130 cm, 縦横 110 cm の直方体になっています。Box 内部には 2 つの回転ステージとファイバが取り付けられた 4 つの一軸ステージががあり、これにより D-Egg 全体を走査する仕組みになってます。4 つのスキャンを行うことで全体を走査でき、それぞれのスキャンを b-r scan(bottom, r 方向), b-z scan, t-r scan, t-z scan と呼ぶ。これらのスキャンを同時に行うことができず、測定したい scan のファイバに光を送るように設定する必要がある (2.2 節)。

図 1.2 に D-Egg Scan Box のダイアグラムを示す。このシステムの詳細は 2 章で説明する。

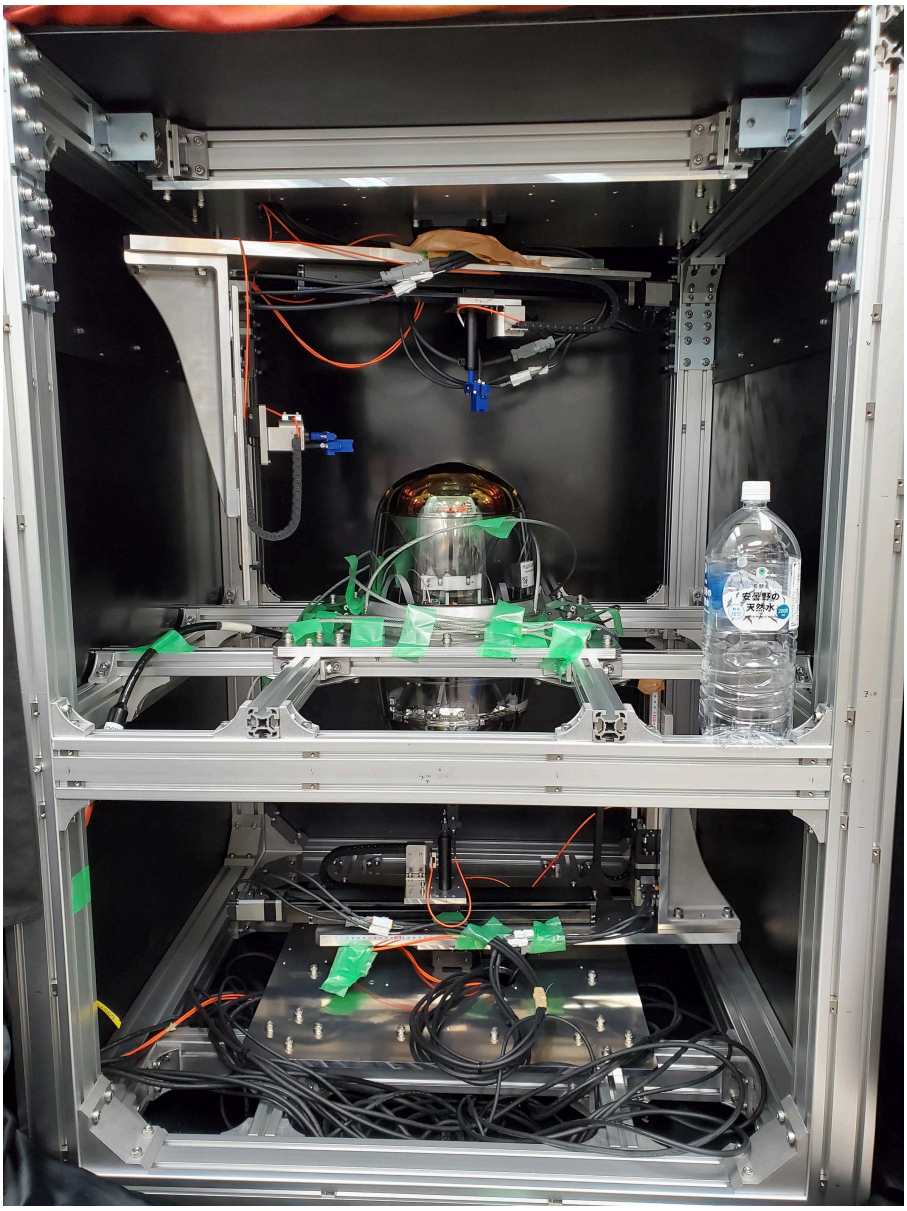


図 1.1: D-Egg Scan Box

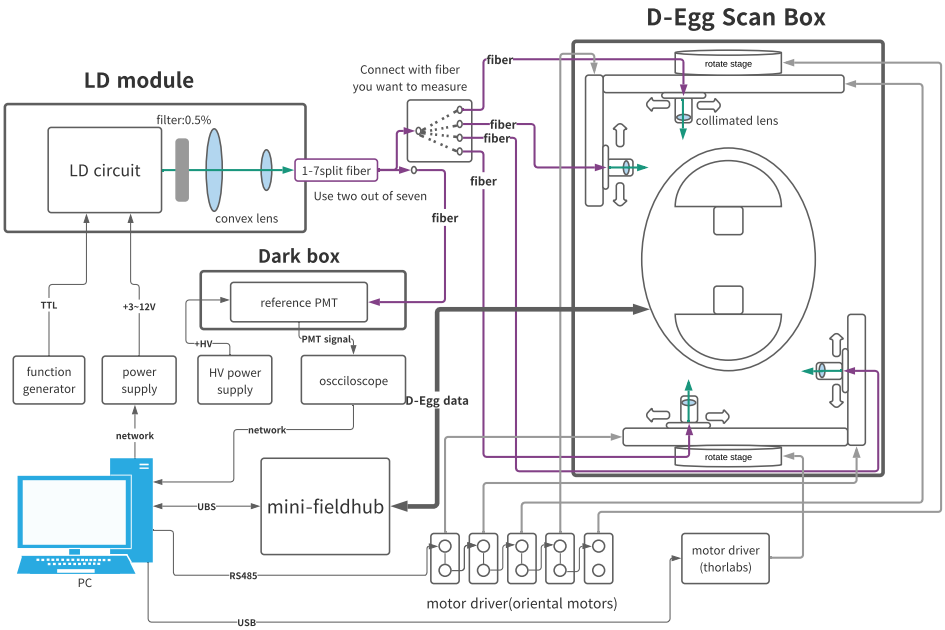



図 1.2: D-Egg Scan Box のダイアグラム

# 第2章　D-Egg Scan Boxの詳細

図 1.2 を基に D-Egg Scan Box の詳細を説明する。

## 2.1　LD module

LD module は 405 nm のピーク波長を持つ LD(L405G2) とそれを駆動させるための基盤、0.5 %のフィルタ、2 つの凸レンズ、ファイバのコネクタが 3D プリンタで製作されたホルダーによって固定されている (図 2.1)。

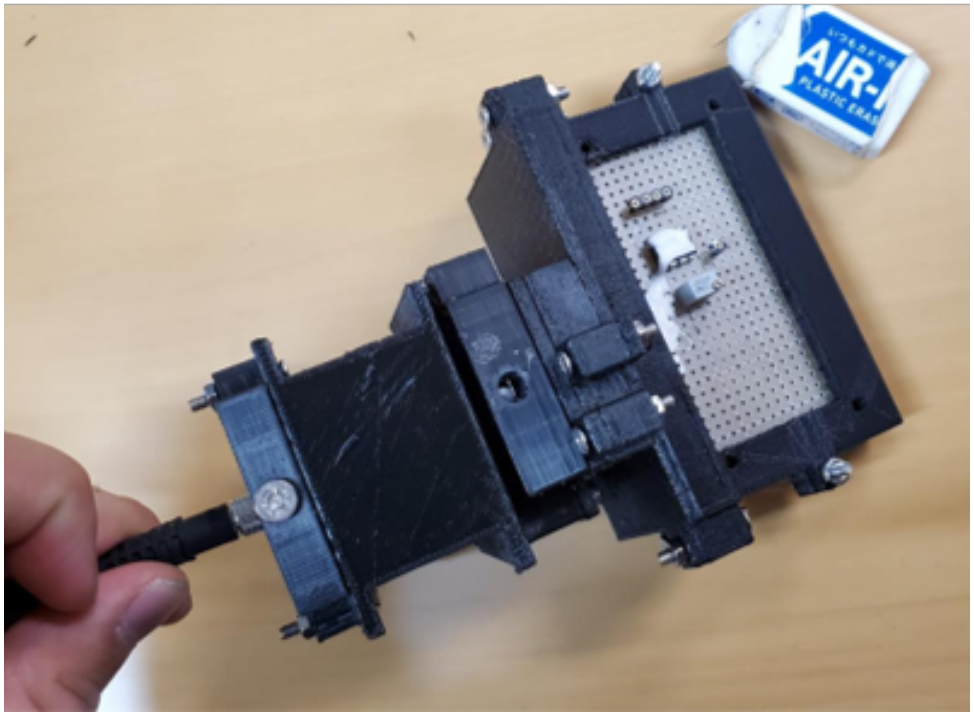


図 2.1: LD module

また LD の回路は図 2.2 のようになっている。この回路図の C1 と R4 で構成される微分回路によって短パルスを生成しトランジスタで整形されていくことで LD の光は 10 ns から 20 ns の短いパルス光を放出している。この回路には TTL 信号と電源 (2 V から 15 V) を供給する必要があり、Function Generator と電源電圧を使用する。また電源電圧の値を変化させることで光量を強めたり弱めたりすることができる。LD の制御は次章の 3.2 節で説明する。

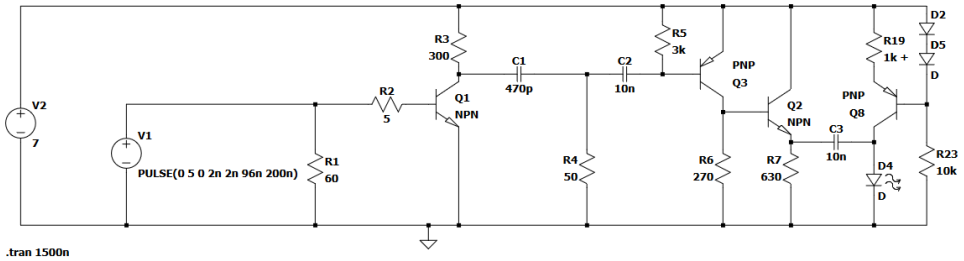



図 2.2: LD の回路図

## 2.2 Fiber

LD module から出た光は 1 つの入力に対して 7 つの出力をもつ 1 分岐ファイバに入光する。この測定では 7 つの出力の内 2 つのみを使う。分岐の 1 つは単独のファイバーを接続し reference PMT に光を運ぶ。残りの 1 つは測定したいスキャンに用いる一軸ステージにあるファイバと接続をし Scan Box 内に光を送る。

## 2.3 reference PMT

reference PMT は測定中の LD の光量のモニターに用いる。他にも各一軸ステージから放出されている光量の確認にも用いる。この PMT は Scan Box 外部にある暗箱内に設置してあり、分岐ファイバからの光を観測している。PMT の信号は同軸ケーブルを通りオシロスコープへと運ばれ波形を取得する。また PMT への印加電圧は浜松製の高電圧電源を使用している。PMT のゲインカーブは図 2.3 のようになっており、常に一定のゲインで測定をおこなうために印加電圧を 1361V に設定する必要がある。D-Egg も同様だが PMT の扱いには最大の注意を払う必要がある。

- 光電面を明るいところに曝さない
- 電圧印加中は必ず暗箱内に置いておく (ソフトで制限するようにしてもいいかも)
- 落とさない

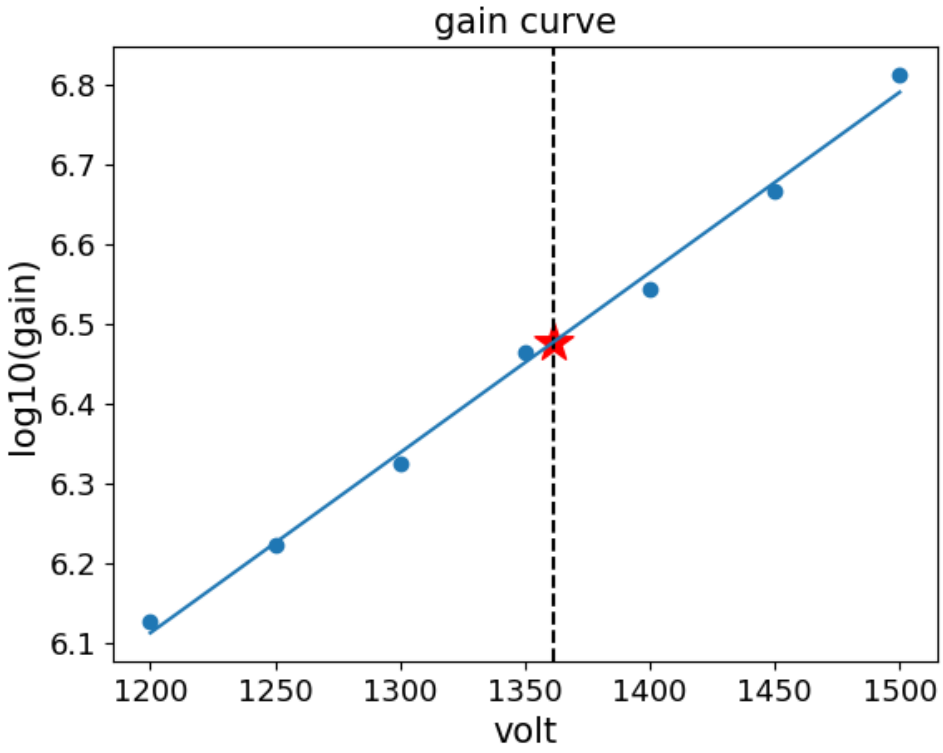



図 2.3: LD の回路図

## 2.4 可動ステージ

この装置には 2 つの回転ステージユニットと 4 つの一軸ステージユニットが用いられている。6 つのうち 1 つの回転ステージユニットは Thorlabs 社製の回転ステージとドライバで構成されていて、もう一つの回転ステージは Orientalmotor 社製の回転ステージとドライバである。4 つの一軸ステージは THK 社製のスライダと orientalmotor 社製のモータとドライバで構成されている。Orientalmotor のドライバは同じ製品であり、単相/三相 200-240 V の電源電圧と 24 V の制御電圧が必要である。各ステージの制御は次章の 3.3 節、3.4 節で説明する。

## 2.5 D-Egg と MiniFieldHub

D-Egg とのやり取りは MiniFieldHub(MFH) を介して PC で行う。D-Egg の制御は次章の 3.5 節で設明する。

## 2.6 メイン PC

このシステムでは多くの装置を PC によって制御することでほぼ自動で測定を行うことができる。また D-Egg や PMT からの信号も PC に送られデータの解析にも用いられる。これらを行う PC は Scan Box 横のテーブル下に置かれている。この PC のディレクトリ構成は次節の 3.1 節で説明する。

## 2.7 まとめ

表 2.1 にこのシステムに用いている装置の一覧を示す。リンクが貼れるものは貼ってい置いたのでより詳細な情報が必要な場合参考にするといいだろう。

| 型番 | 名前 | メーカー | 個数 | その他 |
|---|---|---|---|---|
| L405G2 | LD | Thorlabs | 1 | 仕様書 |
| ?? | Function Generator | ?? | 1 | |
| PMX70-1A | Power Supply | Kikusui | 1 | 製品ページ |
| ?? | Filter | Thorlabs | 1 | |
| ?? | Convex Lens | Thorlabs | 1 | |
| ?? | Convex Lens | Thorlabs | 1 | |
| BF74HS01 | Split Fiber | Thorlabs | 1 | 製品ページ |
| M28L05 | Fiber | Thorlabs | 5 | 製品ページ |
| | reference PMT | 浜松フォトニクス | 1 | |
| | HV Power Supply | 浜松フォトニクス | 1 | |
| | オシロスコープ | ?? | 1 | |
| AZD-AD | ドライバ | Orientalmotor | 5 | 製品ページ |
| AZM46MOC | 一軸ステージ用モータ | Orientalmotor | 4 | 製品ページ |
| SKR3306A | 水平方向スライダ | THK | 2 | 製品ページ |
| SKR2602A | 垂直方向スライダ | THK | 2 | 製品ページ |
| DGM130R | 上部回転アクチュエータ | Orientalmotor | 1 | 製品ページ |
| BSC201 | ドライバ | Thorlabs | 1 | 製品ページ |
| HDR50/M | 下部回転アクチュエータ | Thorlabs | 1 | 仕様書 |
| ??? | Collimated Lens | ?? | 4 | ここで購入 |
| PH75/M | 垂直方向用ポストホルダー | Thorlabs | 2 | 製品ページ |
| PH100/M | 水平方向用ポストホルダー | Thorlabs | 2 | 製品ページ |

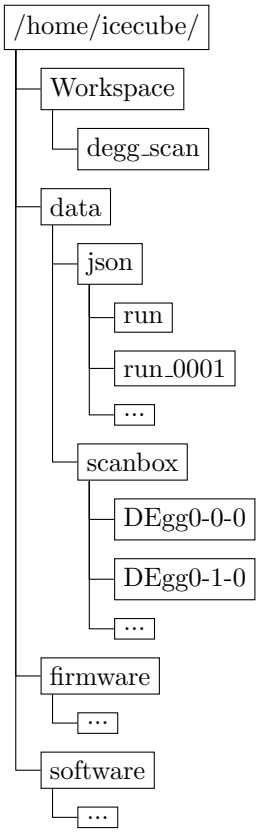
表 2.1: 装置一覧

# 第3章　装置の制御

## 3.1　PC

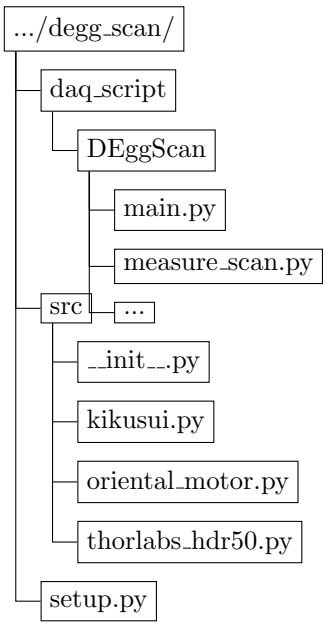
### 3.1.1　ディレクトリ構成

D-Egg Scan Box を用いた測定のほとんどの作業が PC から命令を送ることで完結する。そこでまずこの測定に用いる PC のディレクトリ構成について説明する。




**degg_scan**

測定に関するスクリプトは “/home/icecube/Workspace/degg_scan/”内に入っている。このディレクトリ内の必要な項目だけ記述する。




すべてのコードは python で記述されている。DEggScan を行う場合は現在 “daq_script/DEggScan/main.py” を実行する。main.py は measure_scan.py 内の関数を使用しているため measure_scan.py は必ず main.py 以下の階層に配置しておく必要がある。このコードの内容は章 4.2 で説明する。

“src/” 内にはモータの制御するクラスを記述したコード (oriental_motor.py, thorlabs_hdr50.py) と LD への供給電圧を制御するクラスを記述したコード (kikusui.py) がある。また__init__.py(コード 3.1) と setup.py(コード 3.2) の記述によりあらゆる場所で src 内のクラスを再利用できるようになる。src 内のコードを変更した場合 setup.py がある場所で “pip install -e .” を実行する必要がある。また src 内に新たにクラスを作る場合は新たに__init__.py の__all__内に追記し setup.py がある場所で “pip install -e .” を実行する必要がある。setup.py の詳しい説明はここを参考にしてください。

```
from .thorlabs_hdr50 import *
from .kikusui import *
from .oriental_motor import *

__all__ = [ 'HDR50', 'PMX70_1A', 'AZD_AD']

```

Listing 3.1: init.py

```
from setuptools import setup

setup(

    name='degg_scan',
    version='0.1.0',
    description='Collection of scripts for DEgg Scan system',
    author='Icehap',
    packages=['src'],

)

```

Listing 3.2: setup.py

### data

data 内には Degg の情報を記述している json ファイルと DEggScan での測定データがある。

/data/json/run/内には D-EggID の情報が保管されているファイルが配置されている (例 3.3)。新しい DEgg を設置したときそのたびに json ファイルを作成する必要がある。今のところ筆者は作り方が分からないので Colton に聞いてください。

/data/json/run_00001/内の json ファイルには HV の情報や D-Egg に搭載されている機器の ID、測定のログなどが記載されている (例 3.4)。

```
{
    "DEgg2020-2-058": "/home/icecube/data/json/run_00001/DEgg2020-2-058.json",
    "comment": "test run",
    "date": "2022-01-12"
}
```

Listing 3.3: run_00001.json

```
{
    "DEggSerialNumber": "DEgg2020-2-058",
    "UpperGlassSerialNumber": "degg-top-20210129-2U",
    "LowerGlassSerialNumber": "degg-bot-20210129-2L",
    "UpperGlass": "4108",
    "LowerGlass": "4109",
    "UpperPmt": {
        "SerialNumber": "SQ0556",
        "HVB": "20135c_068",
        "HV1e7Gain": 1482.1818401882272,
        "HV1e7GainDefault": 1500,
        "BaselineFilename": "/home/icecube/data/scanbox/baseline/20220128_03/SQ0556.hdf5",
        "GainMeasurement_00": {
            "GitActiveBranch": "install_restructure",
            "GitShortSHA": "d173e72",
            "GitUncommittedChanges": true,
            "Folder": "None",
            "Comment": "test in dark box"
        },
        "GainMeasurement_01": {
            "GitActiveBranch": "install_restructure",
            "GitShortSHA": "d173e72",
            "GitUncommittedChanges": true,
            "Folder": "None",
            "Comment": "test in dark box"
        },
.....
    },
    "PenetratorType": -1,
    "PenetratorNumber": "179-1_13_D13",
    "SealingDate": "2021-02-24",
    "ArrivalDate": "2021-03-15",
    "flashID": -1,
    "ICMID": "1e00000f17de482d",
    "fpgaVersion": 270,
    "IcebootVersion": 49,
    "BoxNumber": "L-B-1",
    "Port": 5007,
    "ICM": "0201",
    "FlasherID": "REV2-117",
    "MainboardNumber": "4.1-154",
    "ElectricalInspectionNME": "",
    "FlasherNumber": "REV2-117",
    "CameraNumber": "R488623632D",
    "Constants": {
        "Samples": 128,
        "Events": 10000,
        "DacValue": 30000
    },
    "GitShortSHA": "d173e72"
}
```

Listing 3.4: DEgg2020-2-058.json

/data/scanbox/(DEgg の ID)/内の “/各スキャン (top-r, top-z, bottom-r, bottom-z)/日付フォルダー/”に測定データがある。その中にも sig/と ref/に分かれている。sig/charge_stamp.hdf5 内に DEggScan の結果がデータフレームとして保存されている (例 3.5)。ref/内には各φの時のモニタ用 PMT での光量の波形データが保存されている。

| | timestamp | charge | channel | event_num | r_point | t_point |
|---|---|---|---|---|---|---|
| 0 | 457506007360 | 123.426117 | 1 | 0 | 0 | 0 |
| 1 | 457506007874 | 12.731033 | 1 | 0 | 0 | 0 |
| 2 | 457506140665 | 27.651093 | 1 | 0 | 0 | 0 |
| 3 | 457506487359 | 162.728497 | 1 | 0 | 0 | 0 |
| 4 | 457506792815 | 3.330186 | 1 | 0 | 0 | 0 |

```
...              ...           ...       ...       ...       ...       ...
2995  7681220178394  256.346583         0        46       138       354
2996  7681274092243  314.668680         0        46       138       354
2997  7681274491714  327.192051         0        46       138       354
2998  7681332473295  303.506997         0        46       138       354
2999  7681333570228  622.160545         0        46       138       354
```

Listing 3.5: charge_stamp.hdf5

**firmware & software**

firmware と software 内には D-Egg とやり取りするためのモジュールが入っています。詳しくは colton か max に聞くといいでしょう。

## 3.2 LD

LD の動作には 2 から 15 V の直流電源と TTL 信号を送るための Function Generator が必要です。使用する直流電源と Function Generator は表 2.1 の通りです。現状 Kikusui の直流電源は PC で電圧値の制御ができるようになっています。将来的には Function Generator も PC からコントロールできるようにするのが好ましい。

直流電源の制御には src 内の kikusui.py”にある PMX70_1A というクラスを使用します (コード 3.6)。このクラスは測定で用いる LD を使用する設定で作っており LD の許容電流や電圧を超える場合にはエラーを出すように set_volt_current や change_volt_current の関数内で設定している。コード自体は汚いのでなおしてくれるとありがたい。このクラスを使用するサンプルコードを示す (コード 3.7)。

```
import vxi11
import time
import sys

class PMX70_1A:

    def __init__(self, ip):

        self._ip = ip

    def connect_instrument(self):
        try:
            ps = vxi11.Instrument(self._ip)
            print('Get this power supply\n -->> ' + ps.ask('*IDN?') + '\n')

        except:
            print('IP address ERROR.\nPlease check PY File you ran.')
            sys.exit()

        time.sleep(2)

    def set_volt_current(self, volt, current):

        if volt <= 10 and current <= 0.1:
            print('Voltage & Currnt setting is GOOD.  KEEP GOING!!')

        else:
            print('Voltage & Currnt setting is BAD!!\nPlease check PY File you ran!!')
            sys.exit()

        try:
            ps = vxi11.Instrument(self._ip)

        except:
            print('IP address ERROR.\nPlease check PY File you ran.')
            sys.exit()

        print(ps.ask("*IDN?"))
        ps.write("VOLT " + str(volt))
        ps.write("CURR " + str(current))
        ps.write("OUTP 1")
        time.sleep(3)
```

```
            res = ps.ask("MEAS:ALL?")
            print('current and volt -->> ' + res + '\n')
            return res

        def change_volt_current(self, volt, current):

            if volt <= 10 and current <= 0.1:
                print('Voltage & Currnt setting is GOOD.  KEEP GOING!!')

            else:
                print('Voltage & Currnt setting is BAD!!\nPlease check PY File you ran!!')
                sys.exit()

            try:
                ps = vxi11.Instrument(self._ip)

            except:
                print('IP address ERROR.\nPlease check PY File you ran.')
                sys.exit()

            ps = vxi11.Instrument(self._ip)
            ps.write("VOLT " + str(volt))
            ps.write("CURR " + str(current))
            time.sleep(1)
            res = ps.ask("MEAS:ALL?")
            print(res + '\n')
            return res

        def turn_off(self):

            try:
                ps = vxi11.Instrument(self._ip)

            except:
                print('IP address ERROR.\nPlease check PY File you ran.')
                sys.exit()

            ps = vxi11.Instrument(self._ip)
            ps.write("OUTP 0")
            print("turn off the device\n")

```

Listing 3.6: kikusui.py

```
from .kikusui import * #import the kikusui module

LD = PMX70_1A('10.25.123.249') #PMX70_1A(IP address)
LD.connect_instrument()
LD.set_volt_current(6, 0.02) #set_volt_current(voltage(V), current(A)

```

Listing 3.7: sample.py

## 3.3 可動ステージ (Thorlabs)

6 つの可動ステージのうち下側にある回転ステージは Thorlabs 製の HDR50/M という回転ステージを用いている。詳細はホームページをみるといいでしょう。

この回転ステージは Kinesis という専用ソフトウェアか python のスクリプトで操作することができる。測定では python スクリプトを用いて制御しているが、試験的に回転させたいときは Kinesis を用いるとよい。

### 3.3.1 python での制御

まず python コードを用いた制御を説明する。この回転台の制御には src 内の thorlabs_hdr50.py にある HDR50 というクラスを使用する (コード 3.8)。このクラスは thorlabs_apt_device という外部ライブラリの BSC201 を継承して作られている。このクラスの注意点としては連続して move_なんたらという関数を使用しないことである。これをすると回転が完了する前に次の回転命令を行ってしまい期待通りに回転しなくなる。そのため move_なん

たらを使用した次に wait_up 関数を動かし動作が終わるのを待つ必要がある。(クラスの中身を変更し move_なんたらの関数の最後に wait_up() を入れればいいだけかも？)

このクラスを使用した回転台の試運転用のスクリプトは motor 内の move_thorlabs.py にある (コード 3.9)。このコードは “python3 move_thorlabs.py [角度] (-d negative)” と実行する。角度の所に回したい角度を入れることで回転台が時計周りに回転する (相対的に)。またオプションで “-d negative” と入れることで回転方向が反時計回りになる。また motor 内の thorlab_home.py(コード 3.10) を実行することで回転台が原点位置に戻る。ただしモータの位置により回転方向が変わり、ケーブルベアの限界もあるため使わないほうがよい。

```
from thorlabs_apt_device import BSC201
import time

class HDR50(BSC201):

    def __init__(self, serial_port=None, vid=None, pid=None, manufacturer=None, product=
None, serial_number="40", location=None, home=True, invert_direction_logic=False,
swap_limit_switches=True):

        super().__init__(serial_port=serial_port, vid=vid, pid=pid, manufacturer=
manufacturer, product=product, serial_number=serial_number, location=location, home=home,
invert_direction_logic=invert_direction_logic, swap_limit_switches=swap_limit_switches)

        self.set_velocity_params(acceleration=4506, max_velocity=8987328)
        self.set_jog_params(size=75091, acceleration=4506, max_velocity=8987328)
        self.set_home_params(velocity=8987328, offset_distance=0)

    def move_absolute(self, degree=None, now=True, bay=0, channel=0):

        position = degree * 75091

        return super().move_absolute(position=position, now=now, bay=bay, channel=channel)

    def move_jog(self, step=None, direction="forward", bay=0, channel=0):

        step = step * 75091

        if(step!=None):
            self.set_jog_params(size=step, acceleration=4030885, max_velocity=4030885)

        return super().move_jog(direction=direction, bay=bay, channel=channel)

    def move_relative(self, degree=None, now=True, bay=0, channel=0):

        distance = degree * 75091

        return super().move_relative(distance=distance, now=now, bay=bay, channel=channel)

    def get_positoin_status(self):

        angle = self.status["position"]/75091

        return angle


    def wait_up(self):
        pos = int(self.status["position"])

        while True:
            print('Moving now ...(^_^)...')
            time.sleep(2)
            now = pos - int(self.status["position"])
            if(now==0):
                print(self.status["position"])
                break
            pos = int(self.status["position"])

    def turn_on(self):

        self.set_enabled(True)

        return 0

    def turn_off(self):

        self.set_enabled(False)
```

```
63
64              return 0
65
```

Listing 3.8: thorlabs_hdr50.py

```
1   from src.thorlabs_hdr50 import *
2   import click
3
4
5   @click.command()
6   @click.argument('distance')
7   @click.option('--direction', '-d', default='positive')
8   def main(distance, direction):
9       stage = HDR50(serial_number="40106754", home=False, swap_limit_switches=False)
10      if(direction=='negative'):
11          stage.move_relative(-int(distance))
12      else:
13          stage.move_relative(int(distance))
14      stage.wait_up()
15      print(stage.status)
16
17  if __name__ == '__main__':
18      main()
19
```

Listing 3.9: move_thorlabs.py

```
1   from src.thorlabs_hdr50 import *
2
3   stage = HDR50(serial_number="40106754", home=True, swap_limit_switches=False)
4   stage.wait_up()
5   print(stage.status)
6   stage.close()
7
```

Listing 3.10: thorlab_home.py

### 3.3.2 Kinesis ソフトウェア

Kinesis ソフトウェアに関してはここのリンクを見ればわかるでしょう。

ただし回転台には一軸モータが 2 つ乗っており、かなりの荷重がかかっているので動かす速さには注意しないといけない。動かす速さは設定でき、基本的に 10m/s 以下に設定する必要がある。

### 3.3.3 トラブルシューティング

理由はいまだわからないが thorlabs のモータを python で動かすとエラー (エラーが返されたり、正常に動作しなかったり) が起こることがある (結構ある)。

現状確認されているバグは

1. Received unknown event notification from APT device 0 と大量に返される。
2. 動かす動作をさせたはずなのに “Moving now ...(^_^)...” と一回しか出てこず、その後に出てくる position の値が前と変更されない。

の 2 点である。

どちらも対処方法は同じであり以下の方法をテキトーに試すのみである。

1. thorlabs のドライバの電源を入れなおす。
2. ケーブルの挿しなおしを行う。
3. ドライバの電源を切り Kinesis が入っている PC に USB ケーブルを挿す load,unload を行いデスクトップに挿しなおす。

4. ドライバの電源を切り Kinesis が入っている PC に USB ケーブルを挿し load を行い、USB ケーブルをデスクトップに挿しなおす。

これらのいずれかの方法 (もしくは組み合わせ) を行うとエラーは出なくなるはずである。本当に理由はわからないので適切な対処法が分かり次第この節書き直してほしい。

## 3.4 可動ステージ (Orientalmotor)

下側の回転ステージ以外の 5 つが orientalmotor のドライバとモータを使用している。

これらのステージは MEXE02 という専用ソフトウェアか python のスクリプトで操作することができる。測定では python スクリプトを用いて制御しているが、試験的に回転させたいときは MEXE02 を用いるとよい。

### 3.4.1 python での制御

まず python を使った 5 つのモータの制御を説明する。このモータのための 5 つのドライバは数珠繋ぎで接続し (図 3.1)、それぞれに slave address を設定することで 1 本の RS-485 ケーブルのみで 5 つのモータの制御が可能になる。ドライバの slave address やその他の細かい設定はここのリンクを参考にするといいでしょう。現在、各ドライバの slave address は下側横方向が “1”、下側縦方向が “2”、上側横方向が “3”、上側縦方向が “4”、上側回転ステージが “5”、と設定されている。またこのドライバは任意に原点位置を決めることができる。これも先ほどのリンクをもとに設定するといいでしょう。

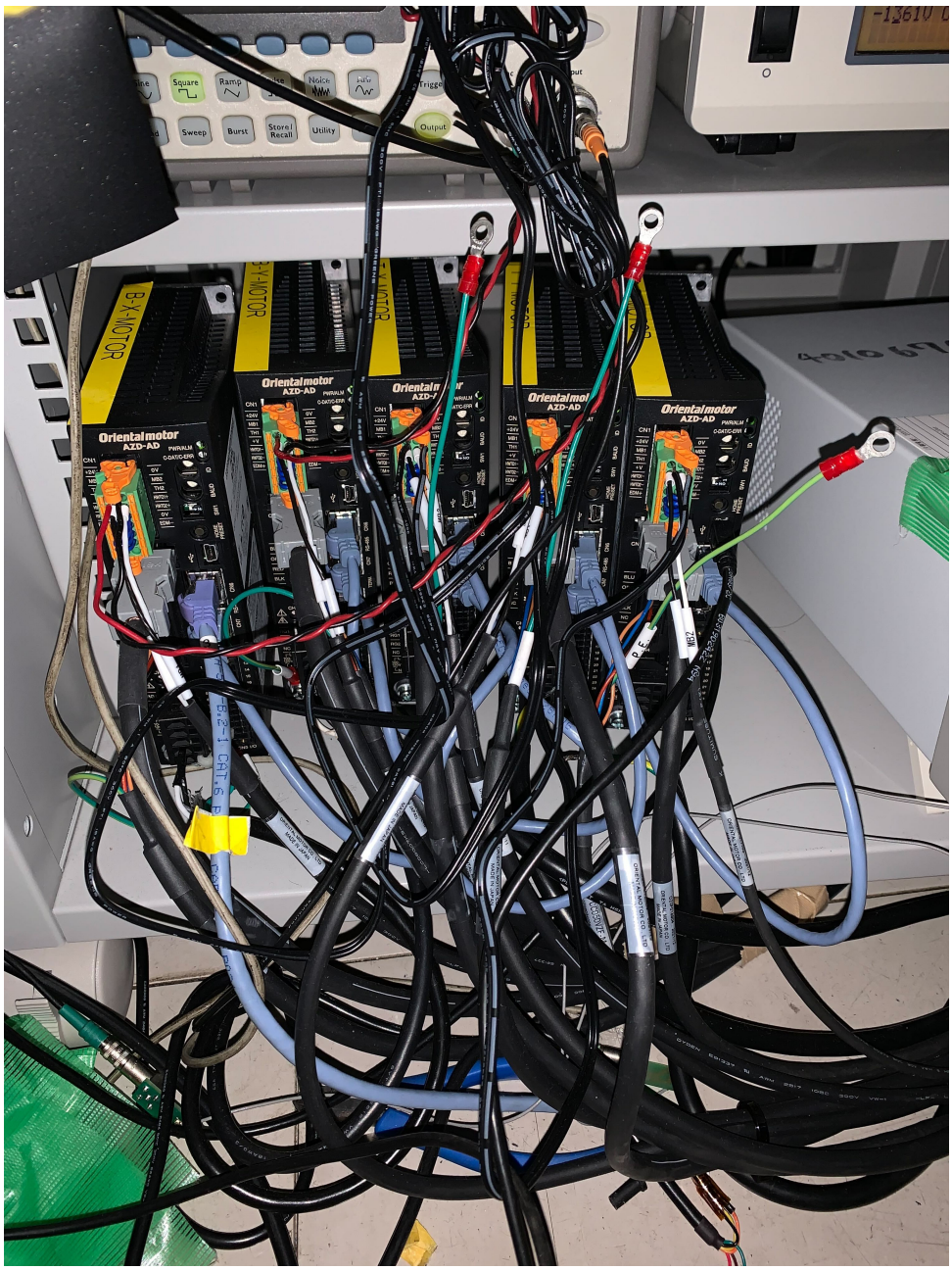



図 3.1: orientalmotor のドライバ。水色の LAN ケーブルによって数珠繋ぎで接続されている。

このように接続された Orientalmotor のステージは src 内の oriental_motor.py にある AZD_AD というクラスを使用する (コード 3.11)。横方向に用いられているモータは 1 ステップで 3/500 mm、縦方向に用いられているモータは 1 ステップで 1/500 mm、上側回転モータは 1 ステップで 1/100 度で動くため moveRelative の関数内でモータのステップ数に置き換えることをしている。これにより各モータのステップ数ではなく変位させたい距離 (mm, 度) で関数を用いることができる。

このクラスを使用したモータの試運転用のスクリプトは motor 内の move_oriental.py にある (コード 3.12)。このコードは “python3 move_oriental.py [slave address] [変位 “変位”] (-d negative)” と実行する。slave address の所に動かしたいモータの address、変位の所に動かしたい距離 (mm, 度) を入れることでステージが相対的に動く。またオプションで “-d negative” と入れることで逆方向に動く (回転ステージの場合時計回り (下側回転ステージと逆なので注意!!))。また motor 内の oriental_home.py(コード 3.13) を実行することですべての orientalmotor のステージが任意に指定した原点位置に戻る。ただしこの時の速度はとても早く、特に回転ステージに関しては壊れそうなぐらい速いのであらかじめ回転ステージを原点位置付近にある場合に実行してください (あまり使わないほうがいい)。

```
1    import serial
```

```
    import time

    class AZD_AD():

        def __init__(self, port=None, bps=115200, t_out=0.01, size=64):

            self._driver = serial.Serial(port, bps, timeout=t_out, parity=serial.PARITY_EVEN,
    stopbits=serial.STOPBITS_ONE)

            self.size = 64

        def to2Int(self, x):
            # argument: int (>=0, <65536)
            # return upper int and lower int
            if (x < 0 or x >= 16**4):
                ...

        def to4Int(self, x):
            # argument: int (>= -16**8/2, < 16**8/2)
            if (x < -16**8/2 or x >= 16**8/2):
                print("error: cannot convert " + str(x) + " into 4 bytes")
                ...

        def calcCRC(self, command):
            # calculate last 2 bytes of command (= CRC-16 error check)
            # argument is command without error check (type: bytes)
            res = 0xFFFF

            for byte in command:
                ...

        def genCommand(self, slaveAddress, functionCode, dataStart, dataNum, data):
            # generate command

            # array of int
            res = []
            res += [slaveAddress]
            ...

        def genCommand2(self, slaveAddress, functionCode, dataStart, data):
            # generate command (ZHOME)

            # array of int
            res = []
            res += [slaveAddress]
            ...

        def moveRelative(self, slaveAddress, dist):
            data = [dist]
            command = self.genCommand(slaveAddress, 10, 0, 2, data)
            self._driver.write(command)
            self._driver.read(self.size)
            return command

        def ZHOMEOn(self, slaveAddress):
            functinoCode = 0x06
            dataStart = 0x007D
            data = [0x0010]
            command = self.genCommand2(slaveAddress, functinoCode, dataStart, data)
            self._driver.write(command)
            self._driver.read(self.size)
            return command

        def ZHOMEOff(self, slaveAddress):
            functinoCode = 0x06
            dataStart = 0x007D
            data = [0x0000]
            command = self.genCommand2(slaveAddress, functinoCode, dataStart, data)
            self._driver.write(command)
            self._driver.read(self.size)
            return command

        def moveToHome(self, slaveAddress):
            self.ZHOMEOn(slaveAddress)
            time.sleep(5)
            self.ZHOMEOff(slaveAddress)

```

```
78          def moveRelative(self, slaveAddress, distance):
79
80              if(slaveAddress==1 or slaveAddress==3):
81                  displacement = int(distance * 500/3)
82              elif(slaveAddress==2 or slaveAddress==4):
83                  displacement = int(distance * 500)
84              elif(slaveAddress==5):
85                  displacement = int(distance * 100)
86
87              functionCode = 0x10
88              dataStart = 0x0058
89              dataNum = 16
90
91              driveData = 0 # No.
92              driveWay = 2 # 2: relative
93              velocity = 500
94              startRate = 400
95              stopRate = 400
96              electricCurrent = 1000 # >=0, <=1000
97              reflection = 1 # 1: reflect all data
98
99              data = [driveData, driveWay, displacement, velocity, startRate, stopRate,
    electricCurrent, reflection]
100
101             command = self.genCommand(slaveAddress, functionCode, dataStart, dataNum, data)
102             self._driver.write(command)
103             self._driver.read(self.size)
104             return command
105
```

Listing 3.11: oriental_motor.py

```
1   from src.oriental_motor import AZD_AD
2   import click
3
4   @click.command()
5   @click.argument('slave_address')
6   @click.argument('distance')
7   @click.option('--direction', '-d', default='positive')
8   def main(slave_address, distance, direction):
9       driver = AZD_AD(port='/dev/ttyUSB2')
10      if(direction=='positive'):
11          driver.moveRelative(int(slave_address), float(distance))
12      if(direction=='negative'):
13          driver.moveRelative(int(slave_address), -float(distance))
14  if __name__ == '__main__':
15      main()
16
```

Listing 3.12: move_oriental.py

```
1   from src.oriental_motor import AZD_AD
2
3   driver = AZD_AD(port='/dev/ttyUSB2')
4   driver.moveToHome(1)
5   driver.moveToHome(2)
6   driver.moveToHome(3)
7   driver.moveToHome(4)
8   driver.moveToHome(5)
9
```

Listing 3.13: oriental_home.py

### 3.4.2 MEXE02 ソフトウェア

MEXE02 ソフトウェアに関してはここのリンクを見ればわかるでしょう。

ただし python での制御の様に同時にすべてのモータをコントロールすることは不可能なため動かしたいモータに USB ケーブルを挿す必要がある。

### 3.4.3 トラブルシューティング

ケーブルベアの限界を超えた動作やステージの限度以上の動作によりドライバの右上に赤ランプが点滅することがある。これは主にモータの過負荷によって起こる。この場合赤点滅しているドライバを MEXE02 が入っている PC に接続しアラームリセットをすることで解決する。アラームリセットに関しては MEXE02 の取扱説明書を参考にしてください。将来的には python でアラームリセットできるといいでしょう (何かしらあるはず)。

## 3.5 MiniFieldHub

MiniFieldHub の電源ボタンは裏側にあり、電源を入れると前面パネルが光る。D-Egg を明るいところに曝すときは必ず電源を切るように注意する必要がある。そのほか詳しいことは colton,Max, 永井さんに聞くといいでしょう。D-Egg への接続などに関しては測定準備 (4.1) で説明する。

# 第4章　測定手順

## 4.1　測定準備

ここではメインの測定 (D-Egg Scan) を始めるまでの流れを説明する。

### 4.1.1　D-Egg の設置

上部の 3 つのモータが取り付けられている蓋はスライドできるようになっている。D-Egg を設置するにはまず前面パネルと裏面 (窓側) の上部のパネルを外す。その後、裏面から蓋のスライド機構を止めている部品 (2 つ) を六角レンチを使い取り外す (図 4.1)。そして蓋を窓側へスライドさせる。あとは D-Egg を入れるだけである。

D-Egg は最低 2 人以上で設置する。まず D-Egg をバケツに入れ D-Egg をセットするための丸い穴付近に持っていく (森井さんと何度か設置しているので聞くとよい)。この穴は図**??**のようになっている。その後 D-Egg の 3 本の下側のワイヤーとペネトレータケーブルを穴に通す。1 人は D-Egg を下から、もう 1 人は上側のワイヤーを持ち D-Egg を吊るすようにしガラス面をぶつけないようにそっと D-Egg を下ろしていく。この時図 4.2 の左下あたりにある大きめの窪みにペネトレータケーブルが落ちていくように工夫する。最終的に D-Egg のハーネス部分が穴の縁にピッタリ合うようにセットする。この時なるべく D-Egg が水平になるようにする。その後ワイヤーがステージなどと干渉しないように養生テープで固定する (図 4.3). またペネトレータケーブルは MiniFieldHub からのケーブルとしっかり接続し、ステージと干渉しない場所に設置する。

D-Egg を設置したら回転台が原点位置に来ているか確認する。回転台の原点位置は、下側が縦方向のステージが右側 (棚側)、上側は縦方向のステージが奥側 (窓側) である。特に上側の回転ステージはケーブルベアの制限を十分考慮する必要がある。これは小さな隙間から覗いて確認するしかない。これらの確認を完了したのち、パネルを取り付け、カーテンを被せしっかりと遮光する。この時特にケーブル類の出入り口を遮光するように気を付ける。

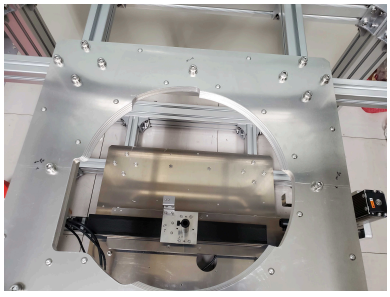


図 4.1: D-Egg を設置する穴。

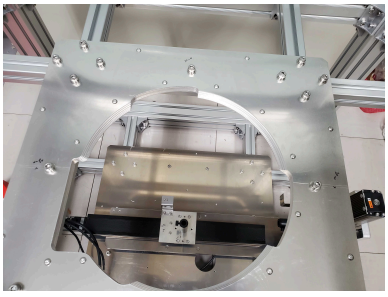


図 4.2: D-Egg を設置する穴。

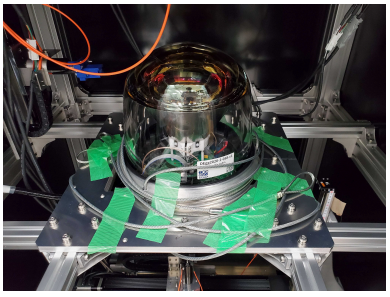


図 4.3: D-Egg 設置後。

### 4.1.2　USB の設定

LinuxPC に USB を挿すと/dev/ttyUSB0 のように最後に数字がついて認識される。この数字は基本的に挿した順によって決定される。この測定では MiniFieldHub からの USB を ttyUSB0 と ttyUSB1、orientalmotor のドライバからの USB を ttyUSB2、thorlabs のドライバからの USB を ttyUSB3 に設定する必要がある。これが一致していない場合、どこかでエラーが起こる。もし USB によるエラーが起こった場合は USB を全て抜き USB に関係ある電源を入れなおしてから順に挿しなおす必要がある。

### 4.1.3 D-Egg の設定

D-Egg は MinifieldHub を介しデスクトップでやり取りを行う。D-Egg の設定には

- json ファイルの作成
- ゲインの測定

の 2 つがある。正直わからないので colton に聞いてください。その後この節を書き直してください。

### 4.1.4 その他

他に必要な設定は

- モニター用 PMT への高電圧の印加
- Function Generator の設定
- オシロスコープの設定

である。モニター用 PMT は棚にある Hamamatsu の高圧電源を使用する。印加電圧は 1361V にすることで PMT の gain が $5\times10^6$ となる。Function Generator は square pulse を指定し、frequency=500 Hz,offset=2.5 V,amplitude=5 V にし output ボタンを押す。パルスが出ているときは output ボタンが光っている。オシロスコープは基本的に ON にしたままにしておく。測定中に取得する波形データはオシロスコープ上に表示されているものなのであらかじめ縦横のスケールを決定し、波形全体が表示されるようにする必要がある。

## 4.2 測定

測定準備が完了すればいつでも測定できる。まず D-Egg とのやり取りを ON にするために、/home/icecube/-software/degg_measurement/degg_measurement/utils/にある setup_degg.py を実行する。何度か y or n を聞かれるので y-y-n-n-y の順番で回答する。すると図 4.4 のようになる。ここで True になっていれば成功である。

次に分岐ファイバーを測定したいファイバーと接続する。

最後に.../Workspace/degg_scan/daq_script/DEggScan/内にある main.py を実行する。このファイルを実行すると図 4.5 のような選択画面が表示される。先ほど接続したファイバーを選択することで測定が開始される。

main.py のコードの内容を簡単に説明するがコード自体は長いためここには書かないので github を見てほしい。コード内の nevent はある点でのチャージスタンプの数となっている (現在は 3000)。r_step や z_step、t_step で各モータのステップ数を設定する。すでに横方向の一軸ステージは D-Egg のほぼ中心、縦方向のステージは D-Egg の最下 (上) 点より 2 cm ほどずれた位置に原点を設定しているため、max の値を変えることで測定範囲を指定できる。またすべての測定は縦方向のステージが奥 (窓側) の場所から時計回りで回るように測定を行う。上部は原点で設定してあるが、下側は原点を設定できないので原点に戻してから 90 度反時計回りに回転させ測定をスタートさせる (measure_scan.py 内の setup_bottom_devices で記述)。あとは一軸ステージをステップごとに動かし t_step 回転し一軸ステージ... を繰り返し 360 度行う。測定後はモータが原点付近に戻るようにしてある。(説明が下手で申し訳ありません)

測定で得られたデータは/home/icecube/data/scanbox/[D-Egg の ID]/[測定の種類]/[日付＋番号]/以下に、モニタ用 PMT の波形データのファイル ([φの値].hdf5) がある ref というフォルダとチャージスタンプのファイル (charge_stamp.hdf5) がある sig というフォルダに格納される。

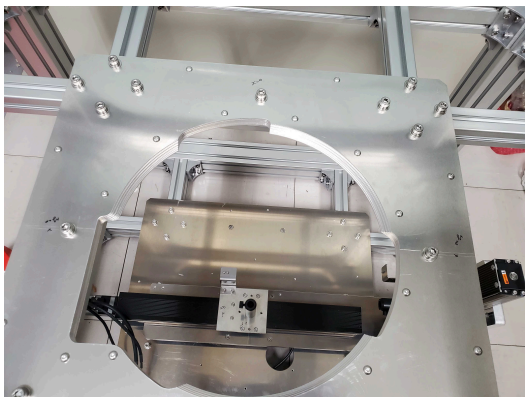


図 4.4: setup_degg.py を実行した結果。

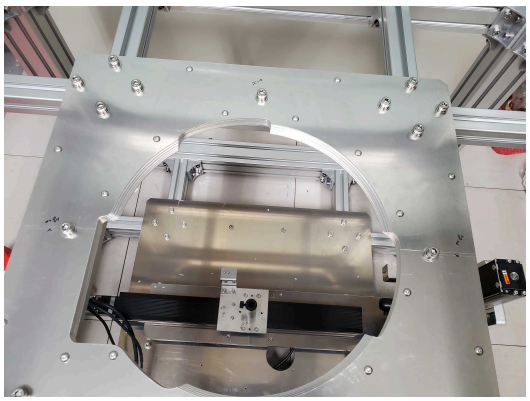


図 4.5: main.py 実行後の選択画面。

### 4.2.1 トラブルシューティング

現状、確認できるエラーは以下の通りである。

1. setup_degg.py でエラーが起こる。
2. 測定中 Killed と表示され強制終了する。
3. 下側の測定で回転ステージが回転しない。

1 のエラーには 2 種類ある。1 つは y か n をエラーメッセージが表示され終了する場合と最後に True でなく False が表示される場合である。前者は基本的に USB の挿す順の問題である。そのため前節の USB の設定を実行すれば解決する。後者は MiniFieldHub の電源を入れなおして再実行すると解決することが多い。理由はわからないがずっと False が表示される場合は colton に聞くとよい。

2 は処理が多すぎることによるメモリ不足で起こる。これは measure_scan.py の dataframe を作っている付近で起こっている (コード 4.1)。簡単に解決するには main.py 内の nevent の数を減らせばいい。ただこの行為はデータ数を減らしているだけでありあまりよくない。上手く measure_scan.py の中身を改善して処理を軽くする必要がある。

3 はよく起こる。理由はわかっていない。このエラーが起こっていない場合は下側の Scan を選択後 “Moving now ...(^_^)...” というのが 20 回ほど表示されモータが 90 度反時計回りで回る。“Moving now ...(^_^)...”が一度のみで position の値が 0 になっている場合はエラーである。この場合一度 ctrl+c で実行と止め、motor_thorlabs.py を 5 度ぐらいの角度で実行する。その後再び main.py を実行すると直ることが多い。理由はわからない。

```
    def measure_degg_charge_stamp(degg, nevents=100, event_num=0, r_point=0, t_point=0,
    data_dir=''):
        infoval = []
        num_retry = 0
        retry = True
        while retry == True:
            try:
                block = degg.session.DEggReadChargeBlock(10, 15, 14*nevents, timeout=200)
                channels = list(block.keys())
                for channel in channels:
                    charges = [(rec.charge * 1e12) for rec in block[channel] if not rec.flags]
                    timestamps = [(rec.timeStamp) for rec in block[channel] if not rec.flags]
                    for ts, q in zip(timestamps, charges):
                        info = infoContainer(ts, q, channel, event_num, r_point, t_point)
                        try:
                            infoval.append([ts, q, channel, event_num, r_point, t_point])
                        except:
                            continue
                        degg.addInfo(info, channel)
        #### ここらへんでされているKilled########################################
                try:
                    dfs = pd.read_hdf(f'{data_dir}/charge_stamp.hdf5')
```

```
                    df = pd.DataFrame(data=infoval, columns=["timestamp", "charge", "channel",
    "event_num", "r_point", "t_point"])
                    df_total = pd.concat([dfs, df])
                except:
                    df_total = pd.DataFrame(data=infoval, columns=["timestamp", "charge", "
    channel", "event_num", "r_point", "t_point"])
                df_total.to_hdf(f'{data_dir}/charge_stamp.hdf5', key='df')
                retry = False

            ##########################################

            except:
                print(f'no measure {r_point}: {t_point} - retry {num_retry}')
                retry = True
                num_retry += 1

                if num_retry > 5:
                    info = infoContainer(-1, -1, -1, -1, r_point, t_point)
                    infoval.append([-1, -1, -1, -1, r_point, t_point])
                    degg.addInfo(info, -1)
                    try:
                        dfs = pd.read_hdf(f'{data_dir}/charge_stamp.hdf5')
                        df = pd.DataFrame(data=infoval, columns=["timestamp", "charge", "
    channel", "event_num", "r_point", "t_point"])
                        df_total = pd.concat([dfs, df])
                    except:
                        df_total = pd.DataFrame(data=infoval, columns=["timestamp", "charge",
    "channel", "event_num", "r_point", "t_point"])
                    df_total.to_hdf(f'{data_dir}/charge_stamp.hdf5', key='df')
                    retry = False


```

Listing 4.1: measure_scan.py

## 4.3 データ解析

デスクトップでデータ解析のコードを実行するとなぜか PC がフリーズする事案が多く発生したので現在は grappa 上でデータ解析を行う。解析コードは grappa 上の /home/yuya_takemasa/degg_scan/degg_scan/analysis/deggscan_main.py である。scp で grappa にデータディレクトリ (/home/icecube/data/scanbox/[D-Egg の ID]/[測定の種類]/[日付＋番号]) をコピーし “python3 deggscan_main.py [data_dir]” を実行する。data_dir にはコピーしたディレクトリ (.../[日付＋番号]/) を指定する。すると /home/yuya_takemasa/fig/[D-Egg の ID]/[測定の種類]/[日付＋番号] 以下にモニター用 PMT で測定された LD の強度の変化を示すグラフ (図 4.6)、各点でのチャージのヒストグラム (図 4.7)、相対的な感度のヒートマップが作られる (図 4.8)。

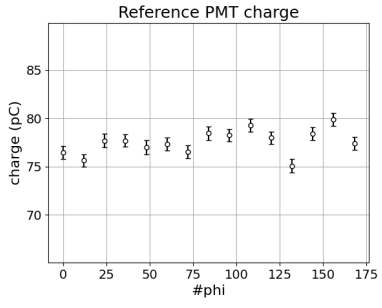



図 4.6: モニタ用 PMT での LD 強度の変化を示したグラフ。

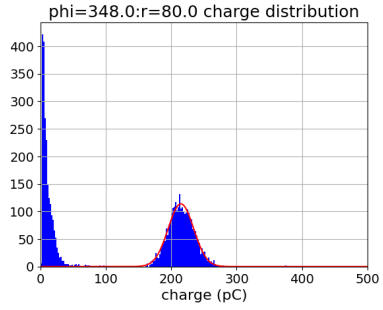



図 4.7: ある点でのチャージのヒストグラム。

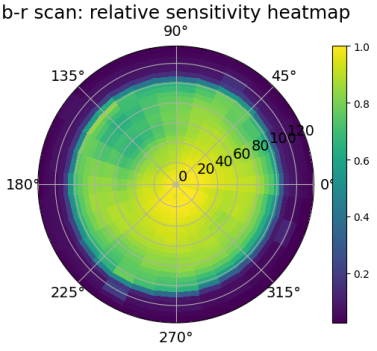



図 4.8: 相対的な検出感度のヒートマップ。

### 4.3.1 トラブルシューティング

チャージのヒストグラムをガウスフィットする際、たまにパラメータエラーが起こる。少ないときは無視していいが極力パラメータの初期値を適切に決定するようなプログラムにしたほうがいい。

# 第5章　シミュレーション

# 第6章　今後の課題