# D-Egg Scan Box Instruction Manual

Yuya Takemasa (Translated by Kareem Farrag)

This document is the instruction manual for the D-Egg Scan Box, focusing on how to use the D-Egg Scan Box. For details, please refer to the table of contents on the next page. This document was created by overleaf. If you want to change the manual due to new specifications, please edit the link and upload the PDF file. The source code is available on github. If you have any other questions, please contact Yuya Takemasa on Slack.

Yuya Takemasa (Translated by Kareem Farrag).

# D-Egg Scan Box Instruction Manual

# Contents

# Figures

# Introduction 1

The D-Egg Scan Box was manufactured to evaluate the response of photomultiplier tubes, including light propagation in the glass and gel of the D-Egg, by scanning the entire D-Egg with collimated light directed toward the D-Egg. The system can also be used to measure the light distribution of the calibration LEDs in the D-Egg and to measure next-generation detectors.

The D-Egg Scan Box is 130 cm high and 110 cm wide, as shown in Figure 1.1. Each scan is called a b-r scan (in the bottom, r direction), b-z scan, t-r scan, or t-z scan. These scans cannot be performed simultaneously and must be set to send light to the fiber of the scan you wish to measure (Section 2.2). Figure 1.2 shows a diagram of the D-Egg Scan Box. Details of this system are explained in Chapter 2.

## 1.1 More about D-Eggs

Figure 1.2 shows the details of the D-Egg Scan Box.

## 1.2 LD module

The LD module consists of an LD with a peak wavelength of 405 nm (L405G2), a substrate for driving the LD, a 0.5% filter, two convex lenses, and a fiber connector, all fixed by a holder fabricated by a 3D printer (Figure 2.1).

The LD circuit is shown in Figure 2.2. In this circuit diagram, the differential circuit composed of C1 and R4 generates short pulses, which are shaped by the transistor to emit short pulses of 10 to 20 ns. This circuit requires a TTL signal and a power supply (2 V to 15 V), and uses a Function Generator and a power supply voltage. The control of the LD is described in section 3.2 in the next chapter.

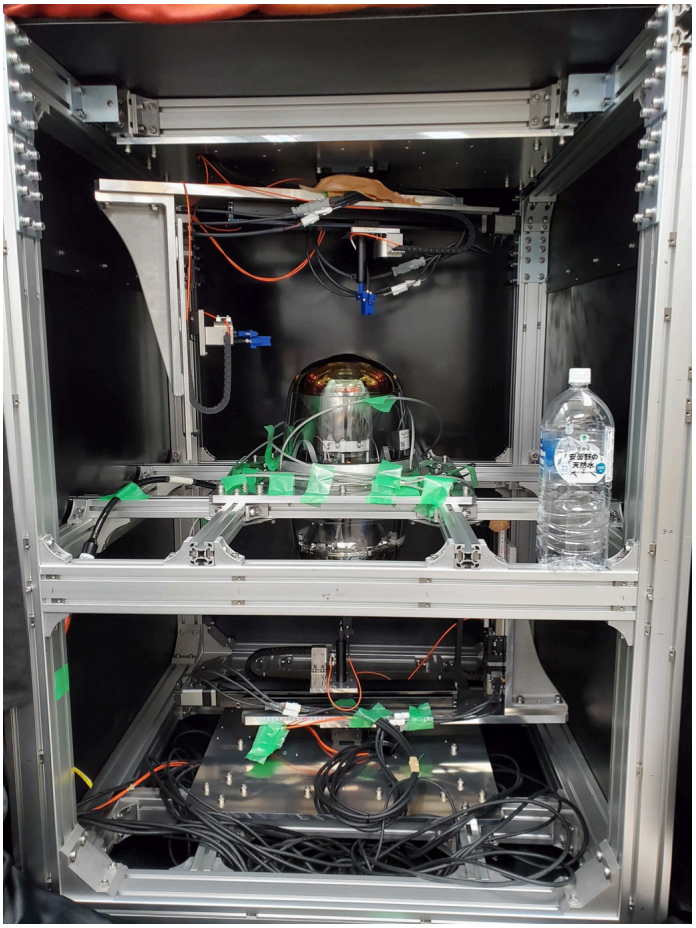


Figure 1.1: D-Egg Scan Box

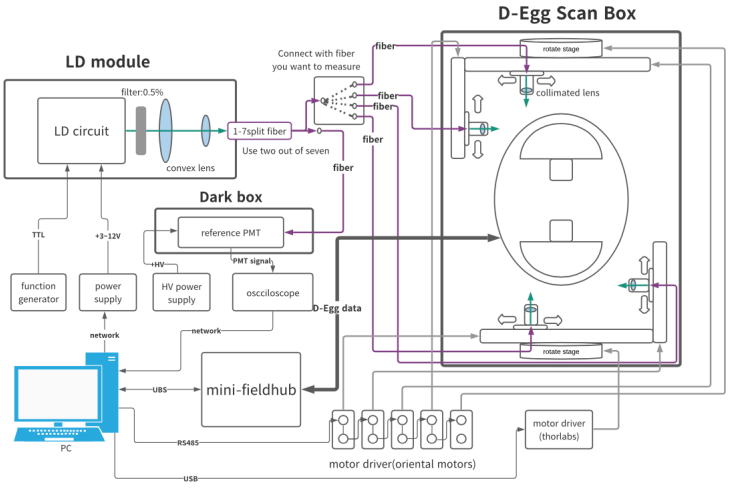



Figure 1.2: D-Egg Scan Box diagram

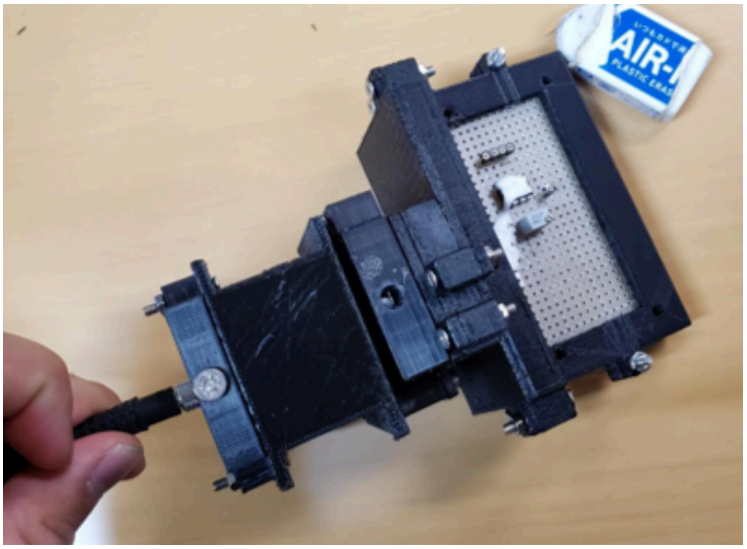


Figure 1.3: LD Module

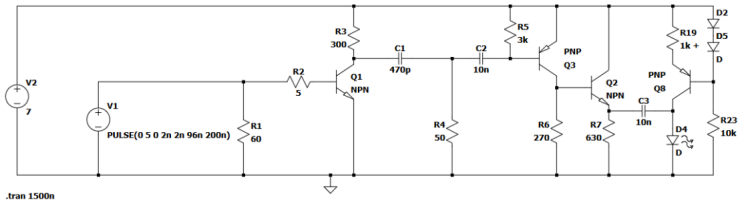



Figure 1.4: LD Schematic

## 1.3 Fiber

The light emitted from the LD module enters a single-branch fiber with seven outputs for one input. In this measurement, only two of the seven outputs are used. One of the branches is connected to a single fiber and carries light to the reference PMT. The remaining one is connected to a fiber in the uniaxial stage used for the scan to be measured and sends light into the Scan Box.

## 1.4 Reference PMT

The reference PMT is used to monitor the light intensity of the LD during measurement. It is also used to check the amount of light emitted from each uniaxial stage. The PMT signal is transmitted through a coaxial cable to an oscilloscope to obtain waveforms. The PMT gain curve is shown in Figure 2.3, and the applied voltage must be set to 1361 V to ensure constant gain measurement.

- ★ Do not expose the photocathode to bright light
- ★ Always keep it in a dark box while voltage is being applied (you may want to use software to limit this)

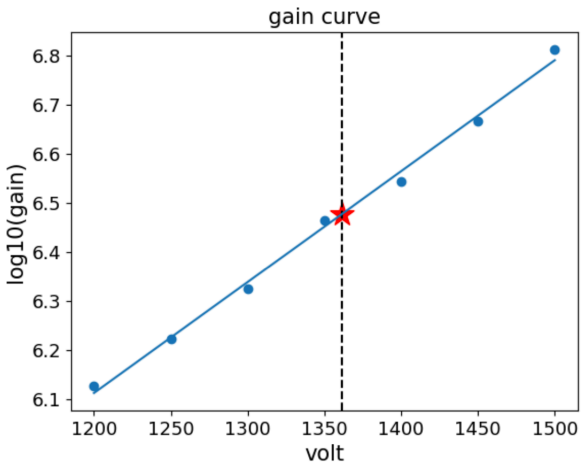



Figure 1.5: Circuit diagram of LD that will not be dropped

★ Don't drop it!

## 1.5 Movable stage

The system uses two rotary stage units and four single-axis stage units; one of the six rotary stage units consists of a Thorlabs rotary stage and driver, and the other rotary stage consists of an Orientalmotor rotary stage and driver. The four single-axis stages consist of THK sliders and orientalmotor motors and drivers. orientalmotor drivers are the same product and require single-phase/three-phase 200-240 V supply voltage and 24 V control voltage. The control of each stage is described in sections 3.3 and 3.4 in the next chapter.

## 1.6 D-Egg and MiniFieldHub

The control of D-Egg is described in section 3.5 in the next chapter.

## 1.7 Main PC

In this system, many devices are controlled by a PC, allowing almost automatic measurement. Signals from D-Egg and PMT are also sent to the PC for data analysis. The PC that performs these functions is located under the table next to the Scan Box. The directory structure of this PC is described in section 3.1.

## 1.8 Summary

Table 2.1 lists the equipment used in this system. If you need more detailed information, please refer to Table 2.1. If you need more detailed information, please refer to this table.

| Model Number | Name | Maker | Quantity | Other |
|---|---|---|---|---|
| LF052 | LD | Thorlabs | 1 | Specification |
| 33250A | Function Generator | Agilent | 1 | Data Sheet |
| PMX70-1A | Power Supply | Kikusui | 1 | Product page |
| ?? | Filter | Thorlabs | 1 | |
| ?? | Convex Lens | Thorlabs | 1 | |
| ?? | Convex Lens | Thorlabs | 1 | |
| BF74HS01 | Split Fibre | Thorlabs | 1 | |
| M28L05 | Fiber | Thorlabs | 5 | Product page |
| | Reference PMT | Hamamatsu | 1 | |
| | HV Power Supply | Hamamatsu | 1 | |
| RTB2004 | Oscilloscope | Rode & Schwartz | 1 | Product page |
| AZD-AD | Driver | Orientalmotor | 5 | Product page |
| AZM46MOC | Motor for Single axis stage | Orientalmotor | 4 | Product page |
| SKR3306A | Horizontal slider | THK | 2 | Product Page |
| SKR2602A | Vertical Slider | THK | 2 | Product Page |
| DGM130R | Upper Rotary Actuator | Orientalmotor | 1 | Product Page |
| BSC201 | Driver | Thorlabs | 1 | Product Page |
| HDR50/M | Lower rotary actuator | Thorlabs | 1 | Specification |
| ??? | Collimated Lens | ?? | 4 | Product Page |
| PH75/M | Hex-Locking Thumbscrew | Thorlabs | 2 | Product Page |
| PH100/M | Hex-Locking Thumbscrew (100mm) | Thorlabs | 2 | Product Page |

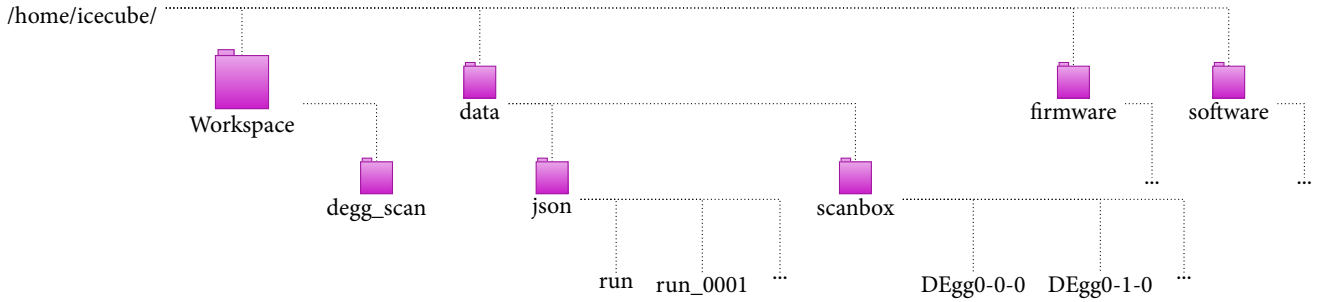
Table 1.1: Equipment List

# Controls 2

## 2.1 PC

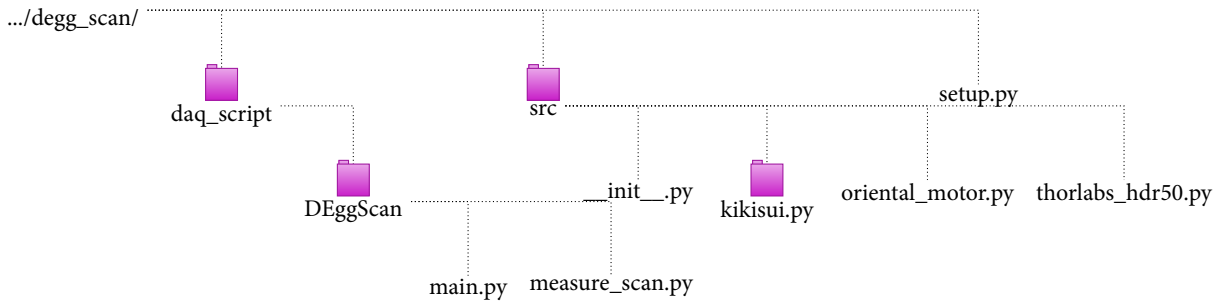
### directory structure

Most of the measurement tasks using the D-Egg Scan Box are completed by sending commands from a PC. First, we will explain the directory structure of the PC used for this measurement.




degg_scan

Scripts for measurement are located in **/home/icecube/Workspace/degg_scan/**. Only necessary items in this directory should be written.




All code is written in **python**. Since main.py uses functions in measure scan.py, measure scan.py must be placed under main.py. The contents of this code are

described in Chapter 4.2. In ”src/”, there are codes (oriental motor.py, thorlabs hdr50.py) describing classes to control motors and (kikusui.py) describing classes to control the voltage supplied to LDs. If you change the code in src, you can run ”pip install -e .” in the location where setup.py is located. If you create a new class in src, you must add it to all in init .py and run ”pip install -e .” at the location where setup.py exists. See here for a detailed description of setup.py.

Listing 2.1: __init__.py

```
from .thorlabs_hdr50 import * from .kikusui import *
from .oriental_motor import *
__all__ = [ 'HDR50', 'PMX70_1A', 'AZD_AD']
```

Listing 2.2: setup.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'],
)
```

## data

In /data/json/run/, there is a json file describing the Degg information and the measurement data from DEggScan. In /data/json/run/, there is a file that stores the D-EggID information (Example 3.3). The json file needs to be created each time a new DEgg is installed. I don't know how to do this at the moment, so please ask Colton. The json file in /data/json/run_00001/ contains information about the HV, IDs of the devices installed in D-Egg, measurement logs, etc. (Example 3.4).

Listing 2.3: run_00001.json

```
{
"DEgg2020 -2-058": "/home/icecube/data/json/run_00001/DEgg2020 -2-058.json",

"comment": "test run",

"date": "2022-01-12"

}
```

Listing 2.4: DEgg2020-2-058.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"
}
```


In /data/scanbox/(ID of DEgg)/ there is the measurement data in "/each scan (top-r, top-z, bottom-r, bottom-z)/date folder/". In sig/charge stamp.hdf5, the results of DEggScan are stored as a data frame (Example 3.5). ref/ contains the waveform data of the light intensity in PMT for monitoring at each $\phi$.

Listing 2.5: charge_stamp.hdf5

```
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
```

### firmware & software

Within the firmware and software are modules for interacting with D-Egg. You can ask colton or max for more information.

## 2.2 LD

LD operation requires a DC power supply of 2 to 15 V and a Function Generator to send TTL signals. The DC power supply and Function Generator used are shown in Table 2.1. Currently, Kikusui's DC power supply can be controlled by a PC. In the future, it is desirable to be able to control the Function Generator from a PC. The class PMX70 1A in "kikusui.py" in src is used to control the DC power supply (Code 3.6). This class is designed to use the LD used for measurement, and is configured in the set volt current and change volt current functions to generate an error if the allowable current or voltage of the LD is exceeded. The code itself is dirty, so it would be appreciated if you could fix it. A sample code that uses this class is shown below (Code 3.7).

Listing 2.6: kikusui.py

```
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 2.7: sample.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)
```

## 2.3 Movable Stage (Thorlabs)

The rotation stage on the lower side of the six movable stages is a Thorlabs HDR50/M rotation stage. See the website for details. The rotation stage can be controlled by dedicated software called Kinesis or by python scripts. The measurement is controlled by a python script, but you can use Kinesis if you want to test rotation.

### Control in python

First, the control using python code is described. The class HDR50 in thorlabs hdr50.py in src is used to control this turntable (code 3.8). This class is made by inheriting BSC201 from an external library called thorlabs apt device. One thing to note about this class is that it does not use the ”move something” function in succession. If you do this, the next rotation command is executed before the rotation is completed, and the rotation will not be executed as expected. Therefore, it is necessary to wait for the wait up function to finish the operation after using move something. (Maybe we just need to change the contents of the class and put wait up() at the end of the move something function?) The script for the test run of the turntable using this class is in move thorlabs.py in the motor (code 3.9). The code is ”python3 move thorlabs.py [angle] (-d negative)”. By entering the desired angle in the ”angle” field, the turntable will rotate clockwise (relatively

speaking). By entering "-d negative" as an option, the direction of rotation becomes counterclockwise. Executing thorlab home.py (code 3.10) in the motor will return the turntable to the home position. However, it should not be used because the direction of rotation changes depending on the position of the motor and because of the limitation of cable bearers.

Listing 2.8: thorlabs_hdf50.py

```
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)
return 0
```

Listing 2.9: move_thorlabs.py

```
from src.thorlabs_hdr50 import * import click
@click.command() @click.argument(' distance' )
@click.option(' --direction' , ' -d' , default=' positive' ) def main(distance , direction):
stage = HDR50(serial_number="40106754", home=False , swap_limit_switches=False) if(direction==' neg
stage.move_relative(-int(distance)) else:
```

```
stage.move_relative(int(distance)) stage.wait_up()
print(stage.status)
if __name__ == '__main__': main()
```

Listing 2.10: thorlab_home.py

```
from src.thorlabs_hdr50 import *
stage = HDR50(serial_number="40106754", home=True, swap_limit_switches=False) stage.wait_up()
print(stage.status) stage.close()
```

### Kinesis Software

As for the Kinesis software, you can see the link here. However, the rotating stand has two single-axis motors on it, and a considerable load is applied, so one must be careful about the speed at which it is moved. The speed of movement can be set, and should basically be set to 10 m/s or less.

### Troubleshooting

I still don't know why, but when I run thorlabs motors with python, I sometimes (quite often) get errors (return errors, not working properly, etc.).

The currently confirmed bugs are the following two points

1. Received unknown event notification from APT device 0 と大量に返される。(Received unknown event notification from APT device 0 and returned large numbers)

2. 動かす動作をさせたはずなのに“Moving now ...(ˆ ˆ)...”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。(I would have made the motion to move it, but ”Moving now ... (ˆ ˆ)...” appears only once, and the subsequent position value does not change from the previous one.)

The solution is the same in both cases, and the only thing to do is to try the following methods in a tentative manner.

1. Turn thorlabs driver back on.

2. Re-insert the cable.

3. Turn off the driver, plug the USB cable into the PC containing Kinesis, load and unload the driver, and plug it back into the desktop.

4. Turn off the driver, connect the USB cable to the PC with Kinesis and load, then connect the USB cable back to the desktop.

If you do any of these things (or a combination of them), the error should disappear. We really don't know the reason, so please rewrite this section as soon as you find an appropriate solution.

## 2.4 Movable Stage (Orientalmotor)

Except for the lower rotation stage, the other five stages use orientalmotor drivers and motors. These stages can be controlled by MEXE02 software or python scripts. In the measurements, the stages are controlled by python scripts, but you can use MEXE02 for experimental rotations.

### Controls in Python

First, we will explain how to control five motors using python. The five drivers for these motors are connected by beads (Fig. 3.1), and by setting a slave address for each driver, it is possible to control all five motors with only one RS-485 cable. For slave addresses and other detailed settings of the drivers, refer to this link. Currently, the slave address for each driver is ”1” for the bottom horizontal direction, ”2” for the bottom vertical direction, ”3” for the top horizontal direction, ”4” for the top vertical direction, and ”5” for the top horizontal direction. The slave address of the driver is set to ”1” for the bottom horizontal direction, ”2” for the bottom vertical direction, ”3” for the top horizontal direction, ”4” for the top vertical direction, and ”5” for the top rotation stage. This driver can also arbitrarily set the origin position. This can also be set based on the previous link.

The stages of the orientalmotor connected in this way use the class AZD AD in oriental motor.py in src (code 3.11). The motor used in the horizontal direction moves at 3/500 mm per step, the motor used in the vertical direction moves at 1/500 mm per step, and the upper rotation motor moves at 1/100 degree per step, so the number of motor steps is replaced in the moveRelative function. This allows the function to use the distance (mm, degree) to be displaced instead of the number of steps for each motor. The script for trial run of a motor using this class is in

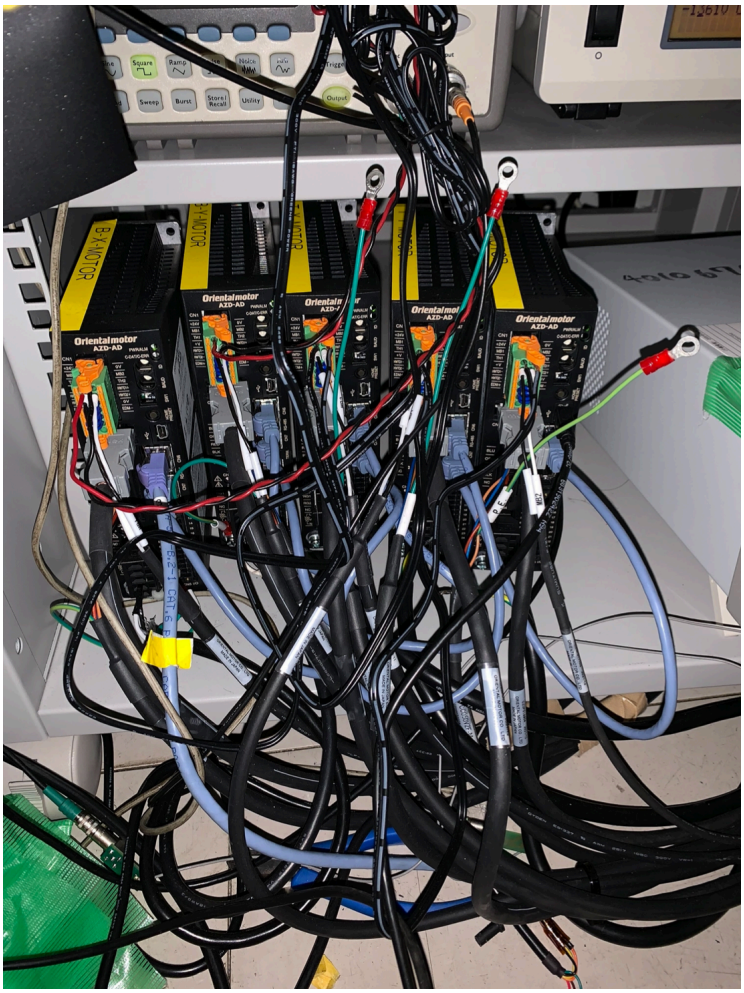



Figure 2.1: orientalmotor driver. They are connected by light blue LAN cables.

move oriental.py in motor (code 3.12). The code is ”python3 move oriental.py [slave address] [displacement ”displacement”] (-d negative)”. is the address of the motor you want to move, and ”displacement” is the distance you want to move (mm, degree). If you enter ”-d negative” as an option, the stage moves in the opposite direction (clockwise in the case of a rotation stage (note that this is the opposite of the lower rotation stage!)). The ”-d negative” option will move in the opposite direction. Also, by executing oriental home.py (code 3.13) in the motor, all orientalmotor stages will return to the arbitrarily specified origin position. However, the speed is very fast, especially for the rotation stage, so it should be done only when the rotation stage is near the home position (it is better not to use it too often).

Listing 2.11: oriental_motor.py

```
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)

def moveRelative(self, slaveAddress, distance):
if(slaveAddress==1 or slaveAddress==3):
displacement = int(distance * 500/3) elif(slaveAddress==2 or slaveAddress==4):
displacement = int(distance * 500) elif(slaveAddress==5):
displacement = int(distance * 100)
functionCode = 0x10 dataStart = 0x0058
dataNum = 16
driveData = 0 # No. driveWay = 2 # 2: relative
velocity = 500 startRate = 400
stopRate = 400
electricCurrent = 1000 # >=0, <=1000
reflection = 1 # 1: reflect all data
data = [driveData , driveWay , displacement , velocity , startRate , stopRate , electricCurrent , reflection]
command = self.genCommand(slaveAddress , functionCode , dataStart , dataNum , data)
self._driver.write(command) self._driver.read(self.size)
return command
```

Listing 2.12: move_oriental.py

```
from src.oriental_motor import AZD_AD import click
@click.command()
@click.argument('slave_address') @click.argument('distance')
@click.option('--direction', '-d', default='positive') def main(slave_address , distance , direction):
driver = AZD_AD(port='/dev/ttyUSB2') if(direction=='positive'):
driver.moveRelative(int(slave_address), float(distance)) if(direction=='negative'):
driver.moveRelative(int(slave_address), -float(distance)) if __name__ == '__main__' :
main ()
```

Listing 2.13: oriental_home.py

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

## 2.5 MEXE02 Software

You can find the MEXE02 software at this link. However, it is not possible to control all motors at the same time like python control, so you need to plug the USB cable

to the motor you want to move.

### Troubleshooting

Movement beyond the limits of the cable bear or beyond the limits of the stage may cause the red light to blink in the upper right corner of the driver. This is mainly caused by an overloaded motor. In this case, the problem can be solved by connecting the driver that is blinking red to the PC containing the MEXE02 and performing an alarm reset. Please refer to the MEXE02 instruction manual for details on alarm resetting. In the future, it would be nice to be able to use python for alarm resetting (there should be something like that).

### MiniFieldHub

The power button on the MiniFieldHub is located on the back of the MiniFieldHub, and the front panel glows when the power is turned on. For more details, please ask Colton, Max, or Ryo.

# Measurement Procedure 3

## 3.1 Preparation for measurement

This section describes the process of starting the main measurement (D-Egg Scan).

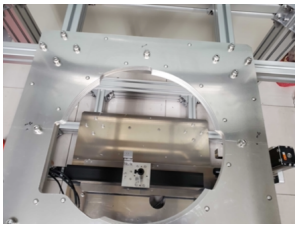
## 3.2 Installation of D-Egg

To install the D-Egg, first remove the front panel and the top panel on the back (window side). Then, use an Allen key to remove the two parts that hold the lid sliding mechanism from the back side (Figure 4.1). Then slide the lid toward the window. Now all that remains is to insert the D-Egg. **The D-Egg should be installed by at least two people.**

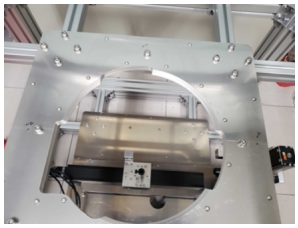
First, place the D-Egg in the bucket and bring it to the round hole where the D-Egg is to be set (you may want to ask Mr. Morii as he has set up the D-Egg several times). This hole is shown in Fig. ? The hole is as shown in Fig. 3.1. One person holds the D-Egg from the bottom, and the other person holds the upper wire, and gently lowers the D-Egg so that the D-Egg is suspended and does not hit the glass surface. At this time, the penetrator cable should fall into the large depression in the lower left corner of Figure 4.2. Finally, set the D-Egg so that the harness part of the D-Egg fits the edge of the hole. At this point, the D-Egg should be horizontal as much as possible. After that, fix the wires with curing tape so that they do not interfere with the stage (Figure 4.3). Connect the penetrator cable to the cable from the MiniFieldHub and place it in a location where it will not interfere with the stage.

After installing the D-Egg, check to see if the turntable is at the home position. The origin position of the turntable is the right side of the vertical stage (shelf side) on the lower side, and the back side (window side) on the upper side. In particular, the upper rotary stage must take into account the limitations of the cable bearers.

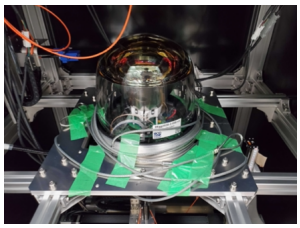
The only way to check this is to look through a small gap. After completing these checks, install the panels, cover them with curtains, and shade them from the light. At this time, special care should be taken to shade the entrance and exit of cables.



(a) Holes for installing D-Egg.



(b) Holes for installing D-Egg.



(c) After installation of D-Egg

Figure 3.1: Figures about the D-Egg installation

## 3.3 USB Configuration

When a USB device is plugged into a LinuxPC, it is recognized with a number at the end, such as /dev/ttyUSB0. This number is basically determined by the order of insertion. In this measurement, USB from MiniFieldHub should be set to ttyUSB0 and ttyUSB1, USB from oriientalmotor driver to ttyUSB2, and USB from thorlabs driver to ttyUSB3. If they do not match, an error will occur somewhere. If a USB error occurs, you need to unplug all USB devices, turn on the power related to USB, and then plug them back in.

### D-Egg Settings

D-Egg communicates with the desktop via MinifieldHub.

- ★ Creating json files
- ★ Gain measurement

I honestly don't know, so please ask Colton. Then rewrite this section.

### Other

Other necessary settings are The PMT for the monitor uses the Hamamatsu high-voltage power supply on the shelf.

- ★ Applying high voltage to PMT for monitoring

★ Function Generator settings

★ Setting up the oscilloscope

For the Function Generator, set square pulse, frequency=500 Hz, offset=2.5 V, amplitude=5 V, and press the output button. The output button glows when a pulse is being emitted. Basically, leave the oscilloscope on. The waveform data to be acquired during measurement is the one displayed on the oscilloscope, so it is necessary to determine the vertical and horizontal scales in advance so that the entire waveform is displayed.

## 3.4 measurement

Once the measurement preparation is complete, measurements can be taken at any time. First, run setup degg.py in /home/icecube/- software/degg measurement/degg measurement/utils/ to turn on communication with D-Egg. You will be asked for y or n several times, so answer in the order y-y-n-n-n-y. The result is as shown in Fig. 4.4. If the answer is true, it is a success.

Next, connect the branch fiber to the fiber to be measured. Finally, connect the branch fiber to the fiber you want to measure. Execute main.py in /Workspace/degg scan/daq script/DEggScan/. After executing this file, the selection screen as shown in Figure 4.5 will be displayed. By selecting the fiber you have just connected, the measurement will start.

The contents of the main.py code are briefly explained, but the code itself is too long to be described here. In the code, nevent is the number of charge stamps at a certain point (currently 3000), and r step, z step, and t step are used to set the number of steps for each motor. Since the origin is already set at approximately the center of the D-Egg for the horizontal uniaxial stage and at 2 cm off the bottom (top) point of the D-Egg for the vertical stage, the measurement range can be specified by changing the value of max. All measurements are taken in a clockwise direction, starting from the back (window side) of the vertical stage. The top stage is set at the origin, but the bottom stage cannot be set at the origin, so the measurement is started by returning to the origin and rotating 90 degrees counterclockwise (as described in setup bottom devices in measure scan.py). Then, move the single-axis stage step by step, rotate t steps, rotate the single-axis stage..., and repeat the process

for 360 degrees. Repeat the process 360 degrees. After the measurement, the motor returns to the origin. (Sorry for the poor explanation).

The data obtained from the measurement is stored under /home/icecube/data/scanbox/[ID of D-Egg]/[type of measurement]/[date+number]/ in a folder named ”ref” that contains the PMT waveform data file for the monitor ([φ value].hdf5) and the charge stamp file (charge stamp.hdf5) is stored in a folder named ”sig”.

### Troubleshooting

Currently, the errors that can be identified are as follows:

- ★ Error in setup_degg.py.
- ★ “Killed” is displayed during measurement and forced termination.
- ★ Rotation stage does not rotate in the lower measurement.

There are two types of error 1: one is when an error message is displayed indicating y or n, and the other is when False is displayed instead of True at the end of the error message. The former is basically a problem with the order in which the USB devices are inserted. The former is basically a problem with the order in which the USB cables are inserted, and can be resolved by following the USB settings described in the previous section. The latter is often solved by turning the MiniFieldHub back on and running it again. If you are not sure why, but False is always displayed, ask Colton.

The second problem is caused by insufficient memory due to too much processing. This happens near the dataframe in measure scan.py (code 4.1). A simple solution is to reduce the number of nevents in main.py. However, this only reduces the number of dataframes, which is not a good thing. It is necessary to improve the contents of measure scan.py to lighten the processing.

(3) happens frequently. The reason is not known. If this error is not happening, select Scan on the bottom and press ”Moving now .... Moving now ... (ˆ ˆ)...” is displayed about 20 times and the motor turns 90 degrees counterclockwise. ”Moving now ... Moving now ... (ˆ ˆ)...” If ”Moving now ... (ˆ ˆ)...” is displayed only once and the value of position is 0, it is an error.

In this case, stop execution with ctrl+c and run motor thorlabs.py at an angle of about 5 degrees. After that, running main.py again often fixes the problem. I don't know why.

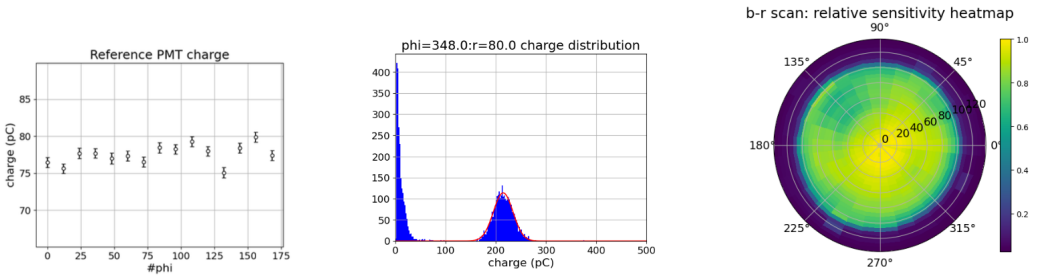
Listing 3.1: measure_scan.py

```
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
```

## 3.5 data analysis

Since there have been many cases where the PC freezes for some reason when executing the data analysis code on the desktop, data analysis is now performed on grappa. The analysis code is /home/yuya takemasa/degg scan/degg scan/analysis/deggscan main.py on grappa. (/home/icecube/data/scanbox/[ID of D-Egg]/[type of measurement]/[date+number Copy the data directory (/home/icecube/data/scanbox/[ID of D-Egg]/[type of measurement]/[date

+ number]) to pppa and execute ”python3 deggscan main.py [data dir]”. /[date+number]/) in the ”data dir”. Then you will find the following files under /home/yuya takemasa/fig/[ID of D-Egg]/[type of measurement]/[date+number]. A graph (Figure 4.6), a histogram of the charge at each point (Figure 4.7), and a heat map of the relative sensitivity (Figure 4.8) are generated.




(a) Graph showing change in LD intensity in PMT for monitoring

(b) Histogram of charges at a point

(c) Heatmap of relative detection sensitivity

Figure 3.2: Data recorded from measurement on box scanner

## 3.6 Troubleshooting

When Gaussian fitting a histogram of charges, sometimes a parameter error occurs. If the number of such errors is small, they can be ignored. However, it is recommended that the program be designed to determine the initial values of the parameters appropriately as much as possible.

# Simulation | 4

# Future Issues | 5