Data_processing_tools

Welcome to use Data processing tools designed for RflyMAD dataset ! With this toolkit, you can select your interested topics and files in RflyMAD dataset, and then generate processed file at a certain frequency as you want !

Quick Start

Download the toolkit into the local folder as follows:

```
cd <Your local folder>
git clone https://github.com/lerlis/Data_processing_tools.git
cd ./fault_data_process
```

To avoid unnecessary trouble, we strongly recommend you to use the same version of python and related libraries. The detailed information about the version is in Requirements.

In order to use this toolkit as easy as possible, the dataset structure should be as follows:

```
\RflyMAD dataset
    \SIL
        \acce <Flight status>
            \accelerometer <Fault type>
                ...
                <Exact flight cases>
                ...
            \barometer
            \GPS
            \gyroscope
            \load_lose
            \low_voltage
            \magnetometer
            \motor
            \no_fault
            \propeller
            \wind_affect
        \circling
        \dece
        \hover
        \velocity
        \waypoint
    \HIL
        ... <Flight status>
    \Real
        ... <Flight status>
```

And that is the same structure in our data paper and the introduction website of RflyMAD. With this structure, toolkits could play their role in data processing.

Note: The RflyMAD dataset is able to be downloaded from here. And the data formats in `<Exact flight cases>` could be seen at Generate your own JSON.

Usage

If you want to use RflyMAD as soon as possible and learn how to use this toolkit quickly, please follow the instructions below.

1. Select Data Type

First of all, you need to select which sub-dataset, flight status and fault types you want. Besides, in each occasion, how many flight cases you want to get and the data is processed in what kinds of frequency. We have integrated the functionality mentioned above by using `argparse` with Python. Here is the detailed introduction:

```python
def get_parse():
    # parse parameters
    parser = argparse.ArgumentParser(description='Dataset process tools')

    parser.add_argument('--original_path', type=str, default='./SampleData',
                        help='original data restore path')

    parser.add_argument('--restore_path', type=str, default='./ProcessData',
                        help='process data restore path')
    # sub-dataset:
    # default = 0, trans data in the SIL, HIL and Real folder
    # 1 for SIL, 2 for HIL, and 3 for Real
    # Note: if you only choose one sub-dataset type, please input as [X]
    parser.add_argument('--sub_dataset', type=int, nargs='+', default=[0],
                        choices=[0, 1, 2, 3],
                        help='select the sub_dataset you want')

    # fault type:
```

```
    # default = 0, trans all the fault type in the
dataset
    # others occasions, please see readme.md
    # Note: if you only choose one sub-dataset type,
please input as [X]
    parser.add_argument('--fault_type', type=int,
nargs='+', default=[0],
                        choices=range(0, 12),
                        help='select the fault type you
want')

    # flight status:
    # default = 0, trans all the flight type in the
dataset
    # other occasions, please see readme.md
    # Note: if you only choose one sub-dataset type,
please input as [X]
    parser.add_argument('--flight_status', type=int,
nargs='+', default=[0],
                        choices=range(0, 7),
                        help='select the flight status
you need')

    # trans num:
    # default = -1, trans all the flight cases in the
dataset
    # if input other numbers, the program will change
the transferred files.
    parser.add_argument('--trans_num', type=int,
default=-1,
                        help='the number of cases to
transfer')

    # trans frequency:
    # default = 20, users could change the frequency as
they want
```

```
    parser.add_argument('--trans_freq', type=int,
default=20,
                        help='the data frequency in
processed files')
    return parser
```

With `argparse`, you can modify the default settings in `./Data_processing_tools/get_parse.py` or run following example command in `cmd` window to realize the function of select certain data type:

```
python Rflytool_main.py --fault_type 5 8 6 --
flight_status 1
```

And the exact meaning of each parser argument is explained in the following table.

- TABLE Argparse Parameter Description

Name	Meaning	Value
--original_path	RflyMAD original dataset restore path	Relative path or Absolute path
--restore_path	RflyMAD processed data restore path	Relative path or Absolute path
--sub_dataset	Sub-dataset used in process	0 for all, 1 for SIL, 2 for HIL, 3 for Real
--fault_type	Fault type selected in process	See TABLE Fault Type Description
--flight_status	Flight status selected in process	See TABLE Flight Status Description
--trans_num	Flight cases is the combination of above situations	-1 for all, and must be int type
--trans_freq	The processed file frequency	Default: 20Hz

- TABLE Fault Type Description

Input number	Meaning
0	all
1	motor
2	propeller
3	low voltage
4	wind affect
5	load lose
6	accelerometer
7	gyroscope
8	magnetometer
9	barometer
10	GPS
11	no fault

- TABLE Flight Status Description

Input number	Meaning
0	all
1	hover
2	waypoint
3	velocity control
4	circling
5	acceleration
6	deceleration

With a series of selections above, you can decide the data type. And the range of the processed files will be limited within the regions you have selected.

2. Select Data Topic

After choose the data type you want, the next step is to select the message topic in each files, and that is also a key step to prepare the data which are used for research. As mentioned in our data paper and the introduction website of RflyMAD, each flight within the dataset contains four types of raw data and relevant processed files. They could be described as follows:

Name	Meaning	Exists in
Flight Information	Contains flight command, fault type and fault parameter	SIL, HIL and Real
ULog	Data recorded by autopilot	SIL, HIL and Real
Telemetry Log	Information communicated between a multicopter and its corresponding QGroundControl	SIL, HIL and Real
Ground Truth Data	Generated by RflySim platform during the simulation	SIL and HIL
BAG	Generated by the ROS system in real flight	Real flight

In order to make the selection process simpler and clearer, we have generated `json` files for you to choose the files and related topics in advance. There are six files in total, and they could be described as follows.

- `data_SIL_GTD.json`. Used for SIL simulation data, extract data in `Ground Truth Data`.
- `data_SIL_PX4.json`. Used for SIL simulation data, extract data in `ULog`.
- `data_HIL_GTD.json`. Used for HIL simulation data, extract data in `Ground Truth Data`.
- `data_HIL_PX4.json`. Used for HIL simulation data, extract data in `ULog`.
- `data_real_ROS.json`. Used for Real flight data, extract data in `ROS BAG`.
- `data_real_PX4.json`. Used for Real flight data, extract data in `ULog`.

Take `data_real_PX4.json` as an example, if you want to choose a message topic in a certain file, you can just set the dictionary key-value pair for this topic to 1. And set that to 0 means this topic is not selected, and this topic won't appeared in the processed files.

```
{
    "Real_PX4": {
        "_actuator_armed_0": {
            "armed": 0,
            "prearmed": 0,
            "ready_to_arm": 0,
            "lockdown": 0,
            "manual_lockdown": 0,
            "force_failsafe": 0,
            "in_esc_calibration_mode": 0,
            "soft_stop": 0
        },
        "_actuator_controls_0_0": {
            "timestamp_sample": 0,
            "control[0]": 1,
            "control[1]": 1,
            "control[2]": 1,
            "control[3]": 1,
            "control[4]": 0,
            "control[5]": 0,
            "control[6]": 0,
            "control[7]": 0
        },
        ...
    }
}
```

As shown above, `control[0]`, `control[1]`, `control[2]`, `control[3]` are selected and will be processed by this toolkit, and other message topics in this file `_actuator_controls_0_0.csv` won't be processed. Besides, all dictionary key-value pair of message topics in file `_actuator_armed_0.csv` are set to 0, which means this entire file won't be processed or even be read by programs.

Note: `Flight Information` contains a concise summary information of a single flight case, so there is no need to process this file. `Telemetry Log` have not been taken into consideration in this toolkit version, and the toolkit which could process `Telemetry Log` will be developed and released in the future.

2.5 [Additional] Generate your own JSON for data topic selection

In this toolkit, we have generated `JSON` files by using RflyMAD data which are under data formats we designed. If you want to transfer your own data, such as ULog and Rosbag with different message topics, you can use `./Data_processing_tools/read_contents.py` to generate `JSON` file that is suitable for your data, and then follow the above steps to get the processed files.

Before using the function, you need to adjust `mode`, and your own data restore formats need to de adjusted as follows:

```
...
\<Exact flight cases>
    \PX4_path
        \log_xx_20xx_xx_xx.ulg
        \log_xx_20xx_xx_xx_<topic_name1>.csv
        ...
        \log_xx_20xx_xx_xx_<topic_nameN>.csv
    \TLog
    \GTD_path(For SIL and HIL, doesn't exist in real flight data)
        \TrueState_data.xlsx
        \UAVState_data.xlsx
    \ROS_path(For real flight, doesn't exist in simulation data)
        \rfly_real_20xx_xx_xx.bag
        \_slash_mavlink_slash_xxx.csv
        \_slash_mavros_slash_<topic_name1>.csv
        ...
        \_slash_mavros_slash_<topic_nameN>.csv
    \TestInfo.csv(Flight Information)
...
```

Next code shows how to run `./Data_processing_tools/read_contents.py` to generate `JSON` files. `/Log` and `/TrueData` mean `PX4_path` and `GTD_path` in HIL simulation data, `/log_6_2023-5-17-15-43-36` and `/rfly_real_2023-05-17-15-41-51` mean `PX4_path` and `ROS_path` in real flight data. `generate_path` could be set as you want.

```
if __name__ == "__main__":
    """
    mode = 0 refers to HIL or SIL,
    and mode = 1 refers to Real
    """
    mode = 0
    if mode == 0:  # Generate data dict used for SIL and
HIL flight data
        PX4_path =
'./SampleData/HIL/acce/TestCase_1_2400000000/Log'
        GTD_path =
'./SampleData/HIL/acce/TestCase_1_2400000000/TrueData'
        generate_path = './'
        Generate_SHIL_flight_dict(PX4_path, GTD_path,
generate_path)
    elif mode == 1:  # Generate data dict used for real
flight data
        PX4_path =
'./SampleData/Real/hover/12_1/log_6_2023-5-17-15-43-36'
        ROS_path =
'./SampleData/Real/hover/12_1/rfly_real_2023-05-17-15-
41-51'
        generate_path = './'
        Generate_real_flight_dict(PX4_path, ROS_path,
generate_path)
```

When set `mode=0`, the program will generate `JSON` files for PX4 and Ground truth data in SIL and HIL simulation data. For the reason that `JSON` files in HIL and SIL have similar formats, you can just choose one of them. When set `mode=1`, `JSON` files for PX4 and Rosbag in real flight data will be generated.

3. Get Processed Files!

After finishing all above steps, you are very close to get the final processed files, the last step is to run `./Data_processing_tools/Rflytool_main.py` in your editor or excute it in the command-line terminal as:

```
cd ./Data_processing_tools
python Rflytool_main.py
```

The processed files will be restored in `--restore_path`, which you set in Data type selection by using `argparse`. And now, you could use processed files to take researches like fault detection and isolation (FDI) or health assessment (HA) as you want!

Attention:
The first time you run this toolkit may take a long time to generate processed files. For the GTD data files converted from XLSX to CSV format is time-consuming and the conversion of ULog to CSV also takes time. The converted CSV files will be restored in the original dataset path, they may take up some storage space of your device, but they will speed up the next run. So please consider whether to retain these intermediate files at your discretion.

Note:
The `ULog` and `BAG` data in real flight data have already been converted in original `.rar` files.

Requirements

It is strongly recommended to use the same Python version and related environment to ensure better and smoother usage of this code. The contents of `requirements.txt` is shown in the following.

```
python==3.9.7
matplotlib==3.4.3
numpy==1.20.3
openpyxl==3.0.9
pandas==1.3.4
scipy==1.7.1
```

Note

If you have any question or have new feature suggestions, please create an issue in Github to let us know. Or you can contact us by e-mail: lexiangli@buaa.edu.cn.