data-processor

July 3, 2023

[2]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1 Funtion and Constants Definition

[3]:
```python
COLS = ["type", "time (ms)"]

DATA_NEW_HILS_PREFIX = "./data/new-hils/"
DATA_NEW_HILS_1_PREFIX = f"{DATA_NEW_HILS_PREFIX}/2023-07-22_001/"
DATA_NEW_HILS_2_PREFIX = f"{DATA_NEW_HILS_PREFIX}/2023-07-22_002/"

DATA_OLD_HILS_PREFIX = "./data/old-hils/"

DATA_RKB_PREFIX = "./data/rkb/"
```

[4]:
```python
def remove_outlier_using_turkey_method(ser: pd.Series):
    q1 = ser.quantile(0.25)
    q3 = ser.quantile(0.75)
    iqr = q3 - q1
    print(q1, q3, iqr)
    lb = q1 - 1.5 * iqr
    ub = q3 + 1.5 * iqr
    return ser[(ser >= lb) & (ser <= ub)]

def get_real_rtt(df_raw_rtt: pd.DataFrame, df_process_latency: pd.DataFrame):
    time = COLS[1]
    ser = pd.Series()
    ser = df_raw_rtt[time] - df_process_latency[time]
    return ser
```

2 ZeroMQ RTT and Latency

This section calculates the RTT (round-trip time) and latency of ZeroMQ. Latency is approximated with the formula:

$$\text{latency} \approx \frac{\text{RTT}}{2}$$

The round-trip time is

$$\text{RTT} = T_e - T_s - t_p$$

Where - RTT: rount-trip time (in ms), - T_e: timestamp of when control is received, - T_s: timestamp of when the first camera chunk is sent, and - t_p: camera sensor data processing time.

For testing, the sensor that will be used is camera. This is because only 2 sensors are used: GNSS and camera and camera has the largest data size between the two. To get the worst case scenario, we are choosing the sensor with the biggest data.

Also a note, because the camera data is large it is split into chunks and each chunk is sent one by one as a multipart message. But, we will calculate the RTT from the first chunk. The reason for choosing the first chunk so we get the latency of sending the whole camera data. Fortunately, this shouldn't really affect the calculation of the whole RTT as seen in Camera Receive Time section.

[5]:
```
raw_rtt_1 = pd.read_csv(DATA_NEW_HILS_1_PREFIX + "log_delta_time_raw_rtt.csv",␣
↪names=COLS, header=None)
raw_rtt_2 = pd.read_csv(DATA_NEW_HILS_2_PREFIX + "log_delta_time_raw_rtt.csv",␣
↪names=COLS, header=None)
process_latency_1 = pd.read_csv(DATA_NEW_HILS_1_PREFIX +␣
↪"log_delta_time_process_latency.csv", names=COLS, header=None)
process_latency_2 = pd.read_csv(DATA_NEW_HILS_2_PREFIX +␣
↪"log_delta_time_process_latency.csv", names=COLS, header=None)
```

[6]:
```
raw_rtt_1.tail()
```

[6]:

	type	time (ms)
3121	ZEROMQ_RAW_RTT_3121	62
3122	ZEROMQ_RAW_RTT_3122	63
3123	ZEROMQ_RAW_RTT_3123	60
3124	ZEROMQ_RAW_RTT_3124	63
3125	ZEROMQ_RAW_RTT_3125	66

[7]:
```
real_rtt_1 = get_real_rtt(raw_rtt_1, process_latency_1)
real_rtt_1 = remove_outlier_using_turkey_method(real_rtt_1)

real_rtt_2 = get_real_rtt(raw_rtt_2, process_latency_2)
real_rtt_2 = remove_outlier_using_turkey_method(real_rtt_2)

real_rtt = np.append(real_rtt_1.to_numpy(), real_rtt_2.to_numpy())
real_rtt = pd.Series(real_rtt)
real_rtt.describe()
```

```
11.0 25.0 14.0
21.0 25.0 4.0
```

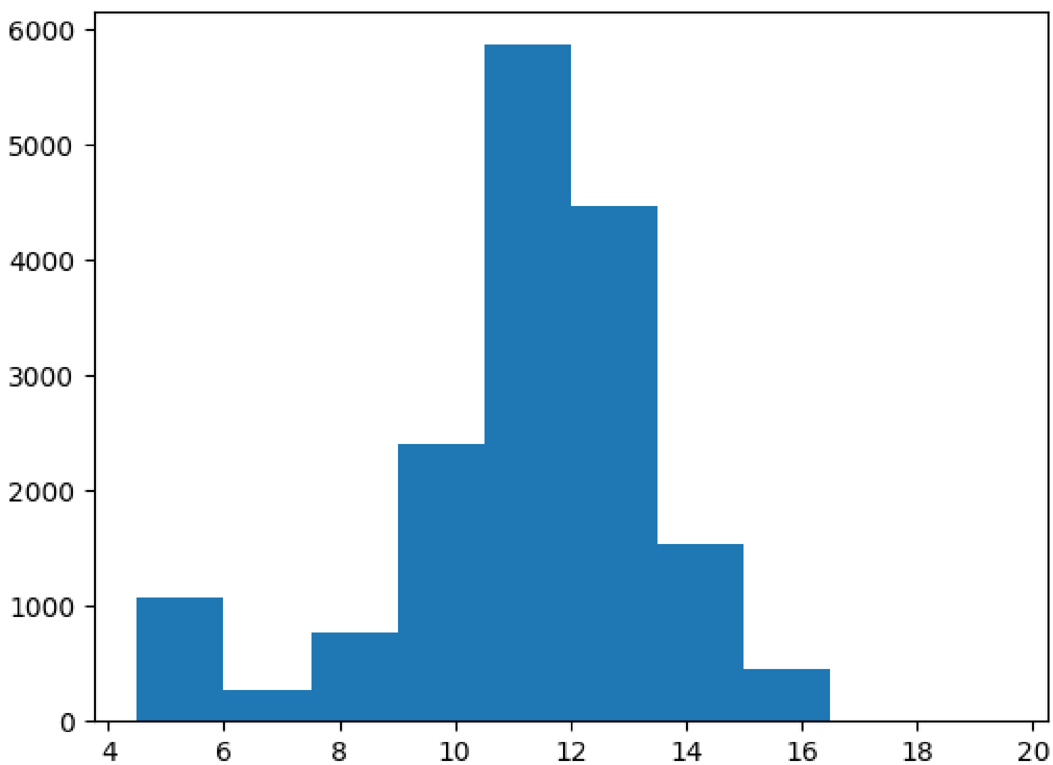
[7]:
```
count    16891.000000
mean        21.979930
std          4.497563
min          9.000000
```

```
25%          20.000000
50%          22.000000
75%          25.000000
max          39.000000
dtype: float64
```

[8]:
```
latencies = real_rtt / 2
latencies.describe()
```

[8]:
```
count    16891.000000
mean        10.989965
std          2.248782
min          4.500000
25%         10.000000
50%         11.000000
75%         12.500000
max         19.500000
dtype: float64
```

[9]:
```
plt.hist(latencies)
plt.show()
```



3 Camera Receive Time

This section shows the difference in time received for each chunk of camera data.

[47]:
```
rkb_log_time = pd.read_csv(DATA_RKB_PREFIX + "log_time.csv", names=COLS,␣
↪header=None)
valid_types = [f"ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_{i}" for i in range(100)]
```

[48]:
```
receive_time = pd.DataFrame(columns = COLS)
receive_time[COLS[0]] = rkb_log_time[COLS[0]]
receive_time[COLS[1]] = np.floor(rkb_log_time[COLS[1]] / 1_000_000).astype(np.
↪uint64)
receive_time = receive_time[receive_time[COLS[0]].isin(valid_types)]
```

[49]:
```
receive_time.head(11)
```

[49]:

	type	time (ms)
1	ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_0	1687430302976
2	ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_1	1687430302976
3	ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_2	1687430302976
4	ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_3	1687430302977
5	ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_4	1687430302977
6	ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_5	1687430302977
7	ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_6	1687430302977
8	ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_7	1687430302977
9	ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_8	1687430302977
10	ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_9	1687430302977
11	ZEROMQ_ZMQ_ENDPOINT_CAMERA_RECEIVE_10	1687430302977

[61]:
```
receive_time_chunks = np.array_split(receive_time, receive_time.shape[0] / 11)
```

[62]:
```
receive_time_diff_sums = np.ndarray((len(receive_time_chunks),), dtype=np.int8)
for i in range(len(receive_time_chunks)):
    chunk = receive_time_chunks[i]
    chunk[COLS[1]] = chunk[COLS[1]].diff()
    chunk.dropna(inplace=True)
    receive_time_diff_sums[i] = chunk[COLS[1]].sum()
# receive_time_chunks[0]
receive_time_diff_sums = pd.Series(receive_time_diff_sums)
receive_time_diff_sums
```

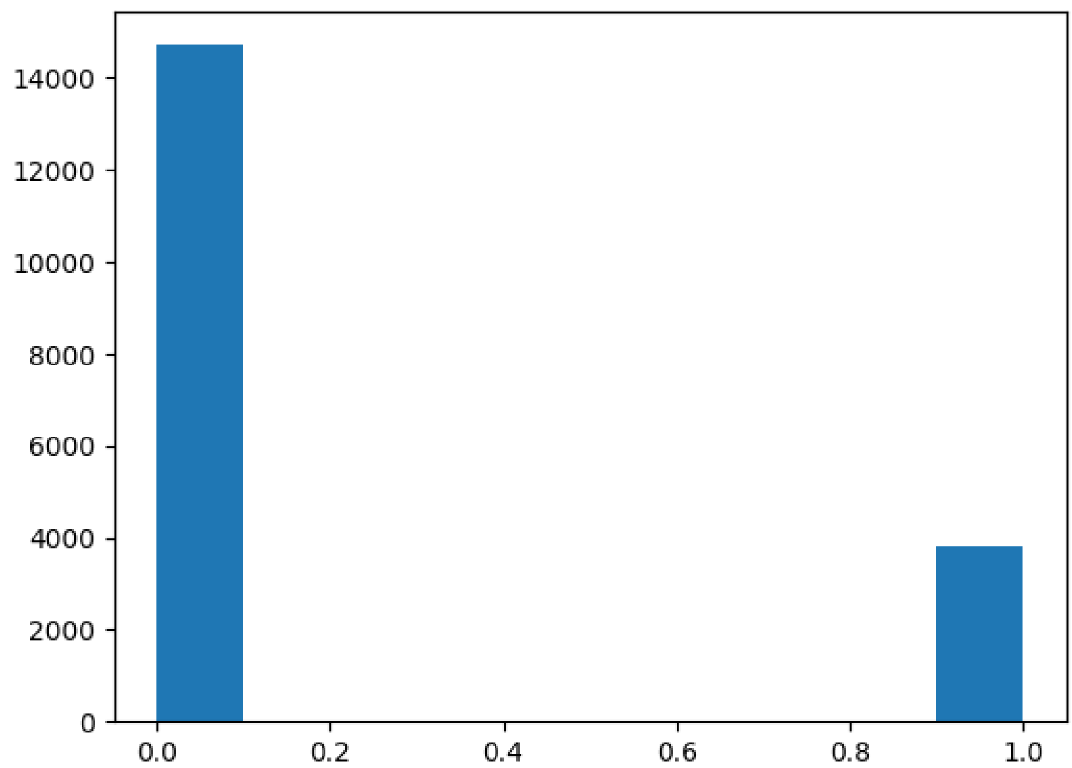
[62]:
```
0        1
1        1
2        0
3        0
4        0
        ..
18493    0
```

```
18494    0
18495    1
18496    0
18497    0
Length: 18498, dtype: int8
```

[77]:
```python
receive_time_diff_sums_df = pd.Series(sums)
receive_time_diff_sums_df.describe()
```

[77]:
```
count    18498.000000
mean         0.205157
std          0.403827
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
dtype: float64
```

[64]:
```python
plt.hist(receive_time_diff_sums)
plt.show()
```



As can be seen, the difference in latency caused by chunking the camera frame is negligible. As the difference of time between each chunk is either 0 or 1, with most of the data being 0.

4 Old HILS "Scientific" Latency

This section calculates the latency of the old HILS connector implementation. The details of the implementation can be seen in the paper or the book. This part only calculates latency during the process of sending data from a "sender" (produces sensor data) to a "receiver" (consumer of sensor data). The latency is only scientific and not real because the HTTP latency is not accounted for.

[71]:
```
old_hils_recv = pd.read_csv(DATA_OLD_HILS_PREFIX + "receive_time.csv",␣
  ↪names=COLS, header=None)
old_hils_send = pd.read_csv(DATA_OLD_HILS_PREFIX + "send_time.csv", names=COLS,␣
  ↪header=None)
```

[72]:
```
old_hils_recv.head()
```

[72]:
```
                          type  time (ms)
0  OLD_HILS_RECEIVE_PROCESS_1         15
1  OLD_HILS_RECEIVE_PROCESS_2          7
2  OLD_HILS_RECEIVE_PROCESS_3          6
3  OLD_HILS_RECEIVE_PROCESS_4          6
4  OLD_HILS_RECEIVE_PROCESS_5         13
```

[75]:
```
old_hils_rtt = old_hils_recv + old_hils_send
old_hils_rtt[COLS[0]] = "old HILS RTT"
old_hils_rtt.head()
```

[75]:
```
           type  time (ms)
0  old HILS RTT         58
1  old HILS RTT         51
2  old HILS RTT         50
3  old HILS RTT         61
4  old HILS RTT         52
```

[78]:
```
old_hils_estimated_combined_lat = old_hils_rtt[COLS[1]] / 2
old_hils_estimated_combined_lat.describe()
```

[78]:
```
count    1000.000000
mean       28.424500
std         4.680006
min        16.000000
25%        25.000000
50%        28.500000
75%        31.500000
max        45.500000
Name: time (ms), dtype: float64
```

[80]:
```
plt.hist(old_hils_estimated_combined_lat)
plt.show()
```