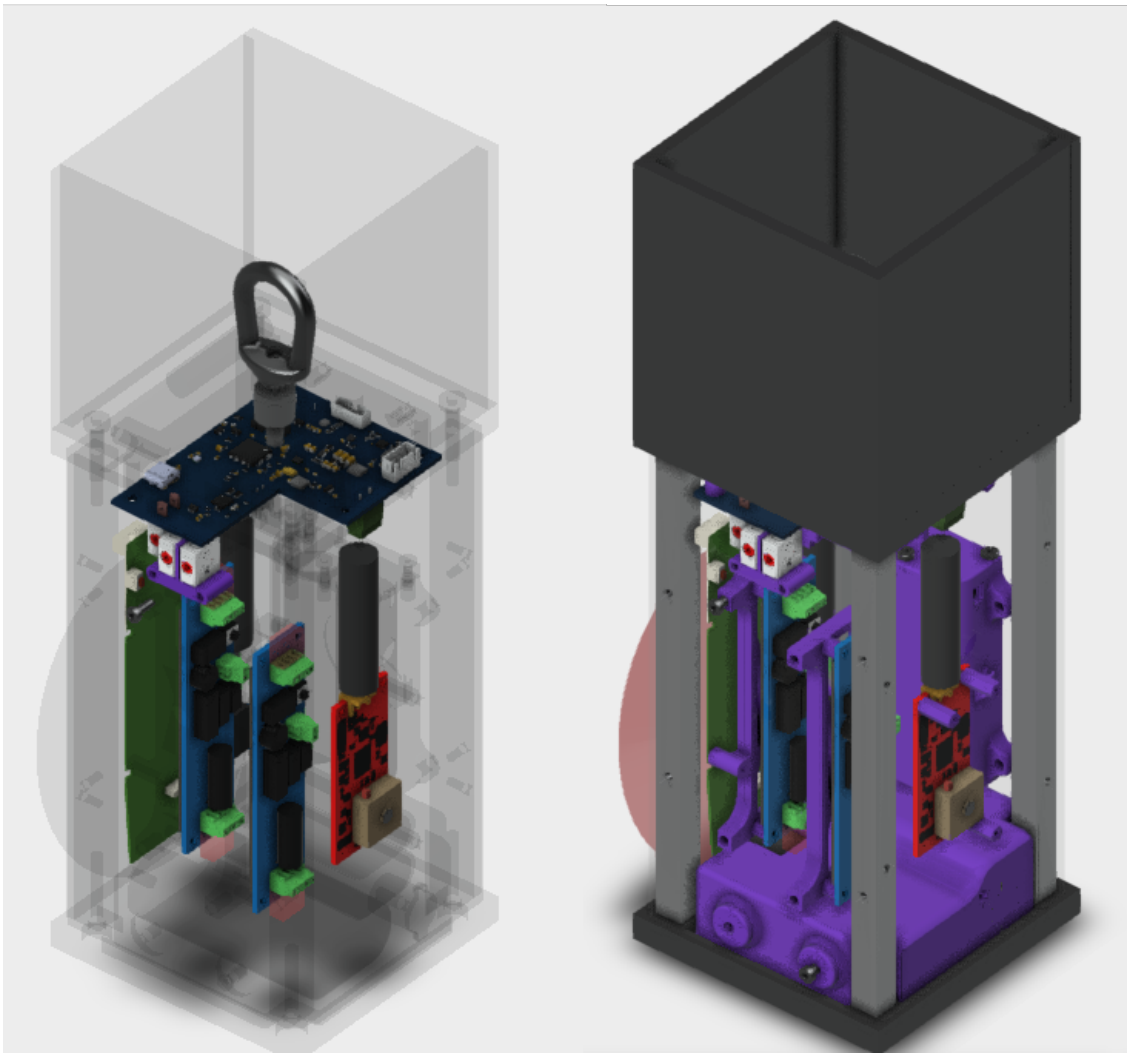
Payload Data Analysis

June 23, 2024

1 Preface

Heres a bunch of data recovered from the Grim Reefer, the payload cameras, and the payload RRC3s. All this data is sourced from the RIT Launch Initiative flight-data github repository

If you wanna skip all the boring stuff, go to Flight Analysis



I would like to formally apologive to Yev for not doing sig figs.

Contents

[1053]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from IPython.display import Markdown
from scipy import integrate
```

[844]:
```
def comma_ax(ax):
    ax.get_xaxis().set_major_formatter(
        matplotlib.ticker.FuncFormatter(lambda x, p: format(int(x), ',')))
```

2 Data Sources

2.1 Grim

- Boost Detect 6 Axis IMU Buffer
- Boost Detect Altimeter Buffer
- Flight Fast Data (6 Axis IMU, Pressure)
- Flight Slow Data (Temperature, Humidity, Voltages, and Currents)
- Flight ADC Data

[845]:
```
fast_i = pd.read_csv(
    'https://raw.githubusercontent.com/RIT-Launch-Initiative/flight-data/main/
 ↪2024/OMEN/GrimReefer/InitialOutputs/fast.csv')
slow_i = pd.read_csv(
    'https://raw.githubusercontent.com/RIT-Launch-Initiative/flight-data/main/
 ↪2024/OMEN/GrimReefer/InitialOutputs/slow.csv')
adc_i = pd.read_csv(
    'https://raw.githubusercontent.com/RIT-Launch-Initiative/flight-data/main/
 ↪2024/OMEN/GrimReefer/InitialOutputs/adc.csv')

pre_imu_i = pd.read_csv(
    'https://raw.githubusercontent.com/RIT-Launch-Initiative/flight-data/main/
 ↪2024/OMEN/GrimReefer/InitialOutputs/pre_imu.csv')
pre_alt_i = pd.read_csv(
    'https://raw.githubusercontent.com/RIT-Launch-Initiative/flight-data/main/
 ↪2024/OMEN/GrimReefer/InitialOutputs/pre_alt.csv')
```

2.2 RRC3s

- Altitude
- Pressure
- Velocity
- Temperature
- Voltage

[846]:
```
rrc3_1i = pd.read_csv(
    'https://raw.githubusercontent.com/RIT-Launch-Initiative/flight-data/main/
↪2024/OMEN/Richie_RRC3_Primary_SN3.csv')
rrc3_2i = pd.read_csv(
    'https://raw.githubusercontent.com/RIT-Launch-Initiative/flight-data/main/
↪2024/OMEN/Richie_RRC3_Secondary_SN4.csv')
```

2.3 Cameras

- 6 Axis IMU

[847]:
```
cam1i = pd.read_csv(
    'https://raw.githubusercontent.com/RIT-Launch-Initiative/flight-data/main/
↪2024/OMEN/GrimReefer/Camera%20Data/Split4g_0000.csv', delimiter='\t')
```

3 Data Cleanup

Here we cleanup data and merge it where applicable. Unless otherwise specified, linear interpolation is used to fill in gaps when joining.

3.1 Grim

3.1.1 Index by timestamp

[848]:
```
fast = fast_i.copy().set_index('timestamp (ms)')
slow = slow_i.copy().set_index('timestamp (ms)')
adc = adc_i.copy().set_index('timestamp (ms)')
pre_imu = pre_imu_i.copy().set_index('timestamp (ms)')
# dont read pressure here before launch
pre_imu.drop('pressure (kPa)', axis=1, inplace=True)
pre_alt = pre_alt_i.copy().set_index('timestamp (ms)')
```

3.1.2 Combine same data from different sensor files

Concatenate pre imu and flight imu, pre altitude and flight altitude

[849]:
```
all_imu = pd.concat([fast, pre_imu], sort=False).drop(
    ['pressure (kPa)'], axis=1).sort_index()
pressure = pd.concat([pre_alt['pressure (kPa)'], fast['pressure (kPa)']])
temp = pd.concat([pre_alt['temperature (degrees C)'],
                  slow['temperature (degrees C)']])
```

3.1.3 All the grim data

outer join on all grim data, linearly interpolating to fill holes

[850]:
```
grim = pd.DataFrame(pressure.copy()) \
.join(all_imu.copy(), how='outer') \
```

```
.join(temp.copy(), how='outer') \
.join(slow.copy().drop('temperature (degrees C)', axis=1), how='outer') \
.join(adc.copy(), how='outer') \
.interpolate('linear', limit_direction='both')
```

3.2 RRC3

Convert timestamps to milliseconds to match grim.

[851]:
```
rrc3_1 = rrc3_1i.copy()
rrc3_1['timestamp'] = rrc3_1['Time [s]'] * 1000
rrc3_1 = rrc3_1.set_index('timestamp')
rrc3_1_events = rrc3_1['Events'].dropna().drop_duplicates()[1:]
```

[852]:
```
rrc3_2 = rrc3_2i.copy()
rrc3_2['timestamp'] = rrc3_2['Time [s]'] * 1000
rrc3_2 = rrc3_2.set_index('timestamp')
rrc3_2_events = rrc3_2['Events'].dropna().drop_duplicates()[1:]
```

3.3 Camera

Drop raw columns so we can use unitted columns.

[853]:
```
cam1 = cam1i.copy()
cam1['timestamp (ms)'] = cam1['t'].rename('timestamp (ms)')
cam1.set_index('timestamp (ms)', inplace=True)
cam1.drop(['t', 'rx', 'ry', 'rz', 'ax', 'ay',
           'az', 't[s]'], axis=1, inplace=True)
```

4 Flight Events

These numbers are found from looking at the data and finding the point that looks right. They are optimized for showing the data not pure mathematical rigor. Additionally, these timestamps are ***not*** time aligned so while all grim data references a common timestamp, the RRC3s do not at this point.

[1056]:
```
# T+ according to grim. in ms
motor_light = 240
flame_out = 2_900

coast_start = 3_175
coast_end = 27_900


apogee = 27_039

charge1_span = (28_090, 29_100)
charge2_span = (30_160, 30_800)
```

```
parachute_catch_1 = 38_215
parachute_catch_2 = 39_066

flight_end = 280_000+3_000


events = pd.DataFrame([
    ["Motor Light", motor_light],
    ["Flame Out", flame_out],
    ["Coast Start", coast_start],
    ["Coast End", coast_end],
    ["Apogee", apogee],
    ["Charge 1 Spike Start", charge1_span[0]],
    ["Charge 1 Spike End", charge1_span[1]],
    ["Charge 2 Spike Start", charge2_span[0]],
    ["Charge 2 Spike End", charge2_span[1]],
    ["Parachte Snatch 1", parachute_catch_1],
    ["Parachte Snatch 2", parachute_catch_2],
    ["Flight End", flight_end]
], columns=["Event", "Time (ms)"]).set_index("Event").style.format("{:,.0f}");
events
```

[1056]: <pandas.io.formats.style.Styler at 0x7fc87caf6a20>

RRC3 1 Events (RRC3 1 Time)

[855]:
```
pd.DataFrame(rrc3_1_events)
```

[855]:

timestamp	Events
25700.0	Drogue
227810.0	Main

RRC3 2 Events (RRC3 2 Time)

[856]:
```
pd.DataFrame(rrc3_2_events)
```

[856]:

timestamp	Events
25600.0	Drogue
227410.0	Main

5 Calculated & Filtered Data

5.1 Magnitude of Acceleration

5.1.1 Grim

Calculate magnitude of acceleration for all grim accelerometer entries. $||a|| = \sqrt{x^2 + y^2 + z^2}$

[857]:
```
ax = grim['accx (m/s^2)']
ay = grim['accy (m/s^2)']
az = grim['accz (m/s^2)']
grim['acc (m/s^2)'] = np.sqrt(ax * ax + ay * ay + az * az)
```

[858]:
```
ax = pre_imu['accx (m/s^2)']
ay = pre_imu['accy (m/s^2)']
az = pre_imu['accz (m/s^2)']
pre_imu['acc (m/s^2)'] = np.sqrt(ax * ax + ay * ay + az * az)
```

[859]:
```
ax = fast['accx (m/s^2)']
ay = fast['accy (m/s^2)']
az = fast['accz (m/s^2)']
fast['acc (m/s^2)'] = np.sqrt(ax * ax + ay * ay + az * az)
```

5.1.2 Cameras

Calculate magnitude of acceleration for all camera accelerometer entries. $||a|| = \sqrt{x^2 + y^2 + z^2}$

[860]:
```
ax = cam1['ax[m/s2]']
ay = cam1['ay[m/s2]']
az = cam1['az[m/s2]']
cam1['a[m/s2]'] = np.sqrt(ax * ax + ay * ay + az * az)
```

5.2 Altitude

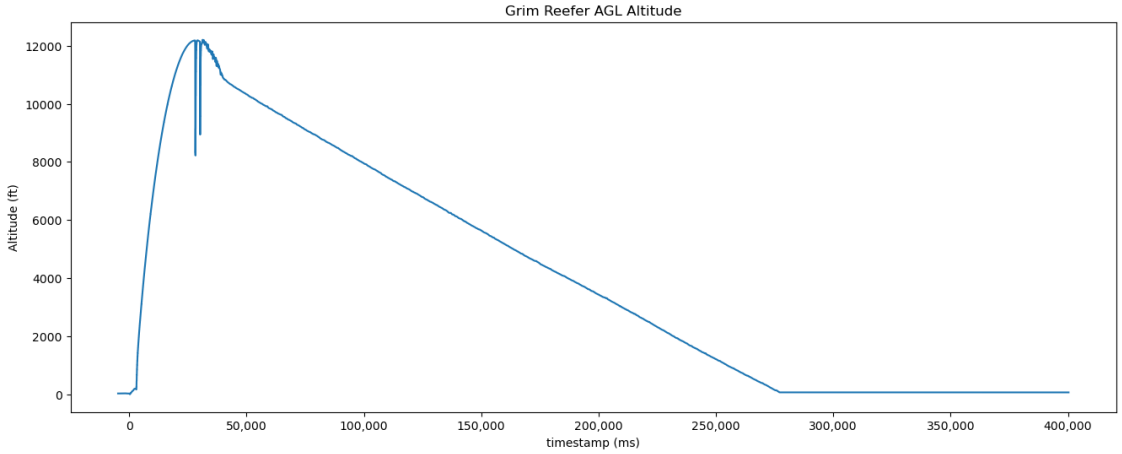
Get the altitude from pressure using the RRC3 Conversion function found here

$$h_{alt} = \left(1 - \left(\frac{P_{sta}}{1,013.25}^{0.190284}\right)\right) \times 134,366.34$$

[861]:
```
from math import pow


def rrc3_pressure_conversion_function_ft(press_kpa):
    pressure = press_kpa * 10
    altitude = (1 - pow(pressure / 1_013.25, 0.190284)) * 145_366.45
    return altitude


def ft_to_meters(ft): return ft * 0.3048
```

[862]:
```
grim['alt_asl_ft'] = grim['pressure (kPa)'].apply(
    rrc3_pressure_conversion_function_ft)
grim_lowest_point_ft = grim['alt_asl_ft'].min()
grim['alt_agl_ft'] = grim['alt_asl_ft'] - grim_lowest_point_ft
```

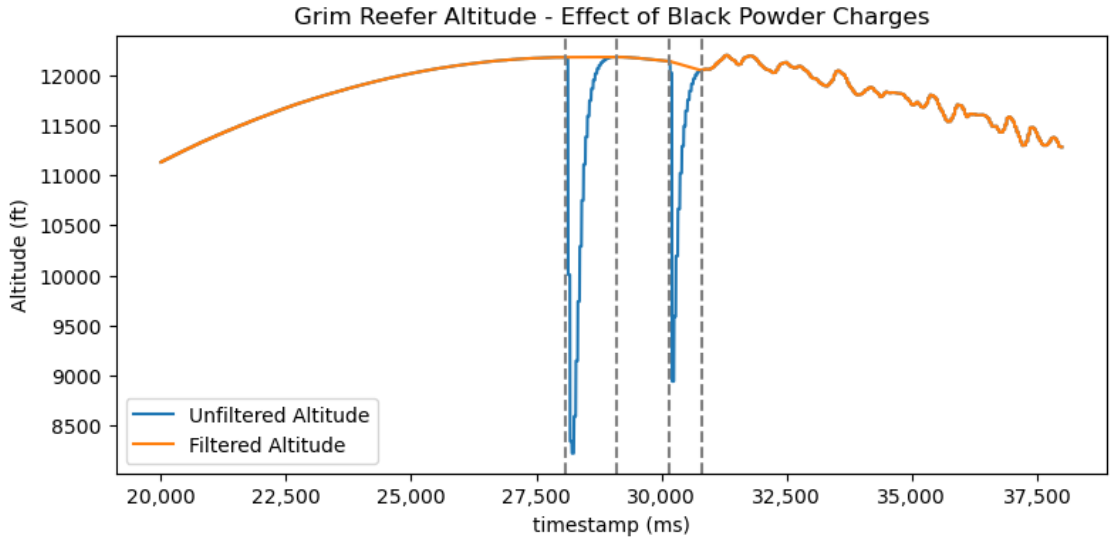
[863]:
```
p = grim['alt_agl_ft'].plot(figsize=(16, 6))
p.set_title("Grim Reefer AGL Altitude")
p.set_ylabel("Altitude (ft)")

comma_ax(p)
```



5.2.1 Fill Holes Caused by Black Powder Charges

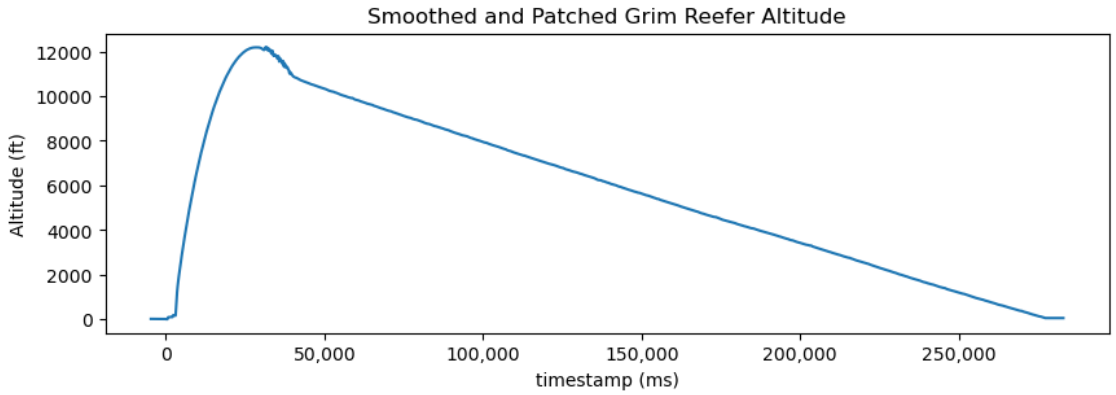
[864]:
```
grim['alt_agl_ft_wout_charges'] = grim['alt_agl_ft'].copy()
grim['alt_agl_ft_wout_charges'].loc[charge1_span[0]
    :charge1_span[1]] = float('nan')
grim['alt_agl_ft_wout_charges'].loc[charge2_span[0]
    :charge2_span[1]] = float('nan')

p = grim[['alt_agl_ft', 'alt_agl_ft_wout_charges']].
 ↪rename(columns={'alt_agl_ft':'Unfiltered Altitude','alt_agl_ft_wout_charges':
 ↪'Filtered Altitude'}).interpolate(
    'linear').loc[20000:38000].plot(figsize=(9, 4))
p.axvline(charge1_span[0], color='gray', linestyle='--')
p.axvline(charge1_span[1], color='gray', linestyle='--')

p.axvline(charge2_span[0], color='gray', linestyle='--')
p.axvline(charge2_span[1], color='gray', linestyle='--')
p.set_title("Grim Reefer Altitude - Effect of Black Powder Charges")
p.set_ylabel("Altitude (ft)")
```

```
comma_ax(p)
```



[865]:
```
grim['alt_agl_ft_wout_charges_smoothed'] = grim['alt_agl_ft_wout_charges'].
↪rolling(
    600, center=True).mean().interpolate(limit_direction='both')
```

[866]:
```
p = grim['alt_agl_ft_wout_charges_smoothed'][:flight_end].plot(figsize=(10, 3))
p.set_ylabel("Altitude (ft)")
p.set_title("Smoothed and Patched Grim Reefer Altitude")
comma_ax(p)
```



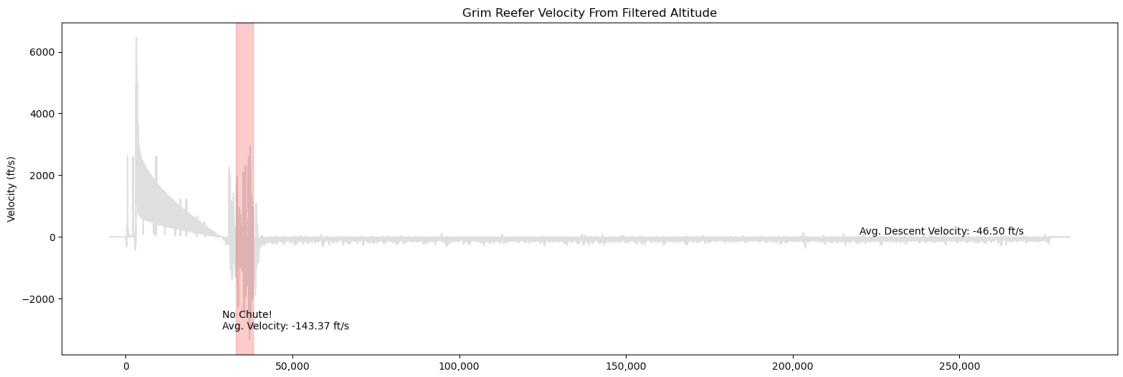
[867]:
```
grim['alt_vel_est'] = (grim['alt_agl_ft_wout_charges_smoothed'].diff(
) / (grim.index.to_series().diff() / 1000)).dropna()
```

```
grim['alt_vel_est'].interpolate(limit_direction='both', inplace=True)
```

5.2.2 Try to get Velocity from Altitude

with limited success

[868]:
```
vel = grim['alt_vel_est'][:flight_end]
plt.figure(figsize=(19, 6))
p = plt.axes()
p.plot(vel.index, vel, color='#00000020')
# p.plot(grim['alt_agl_ft_wout_charges'])

p.set_title("Grim Reefer Velocity From Filtered Altitude")
p.set_ylabel("Velocity (ft/s)")


freefall_range = (33_000, parachute_catch_1)
nochute_avg_vel = vel[freefall_range[0]:freefall_range[1]].mean()

p.axvspan(freefall_range[0], freefall_range[1], color='red', alpha=0.2)
p.annotate(f"No Chute!\nAvg. Velocity: {nochute_avg_vel:.2f} ft/s", (29_000,␣
 ↪-3000))
# min_vel = smooth_vel.min()
# p.axhline(min_vel, color='gray', linestyle='--')
# p.annotate(f"Estimated Velocity at parachute open: {
#             min_vel:.2f} ft/s", (48000, -280))



descent_vel = vel.loc[200_000:250_000].mean()
p.annotate(f"Avg. Descent Velocity: {descent_vel:.2f} ft/s", (220000, 80))

comma_ax(p)
```



6 Aligning Times

grim is the timestamp to follow. Align to that

NOTE: The Camera timescale is out of wack with RRC3s and Grim. It is some factor of the others rather than just an offset. Here it is aligned at snatch force and gets worse as you move away from that point

6.1 Cameras

line up the ground hit (highest acceleration (easy to find)) and add manual adjustment from there

[869]:
```
grim_ground_hit = grim['acc (m/s^2)'].idxmax()
cam1_ground_hit = cam1['a[m/s2]'].idxmax()
cam1_ground_hit_difference = grim_ground_hit - cam1_ground_hit
cam1_manual_difference = -1105+50
```

[870]:
```
# save the old index
cam1['timestamp'] = cam1.index
```

[871]:
```
print(f"Grim hit the ground {grim_ground_hit} ms grim time")
print(f"Cam1 hit the ground {cam1_ground_hit} ms cam1 time")
print(f"We need to shit by {cam1_ground_hit_difference} ms")
print(f"Manual Adjustement by {cam1_manual_difference} ms")
```

```
Grim hit the ground 276836.0 ms grim time
Cam1 hit the ground 272584 ms cam1 time
We need to shit by 4252.0 ms
Manual Adjustement by -1055 ms
```

[872]:
```
cam1.index = (cam1['timestamp'] +
              cam1_ground_hit_difference + cam1_manual_difference)
```

6.2 RRC3s

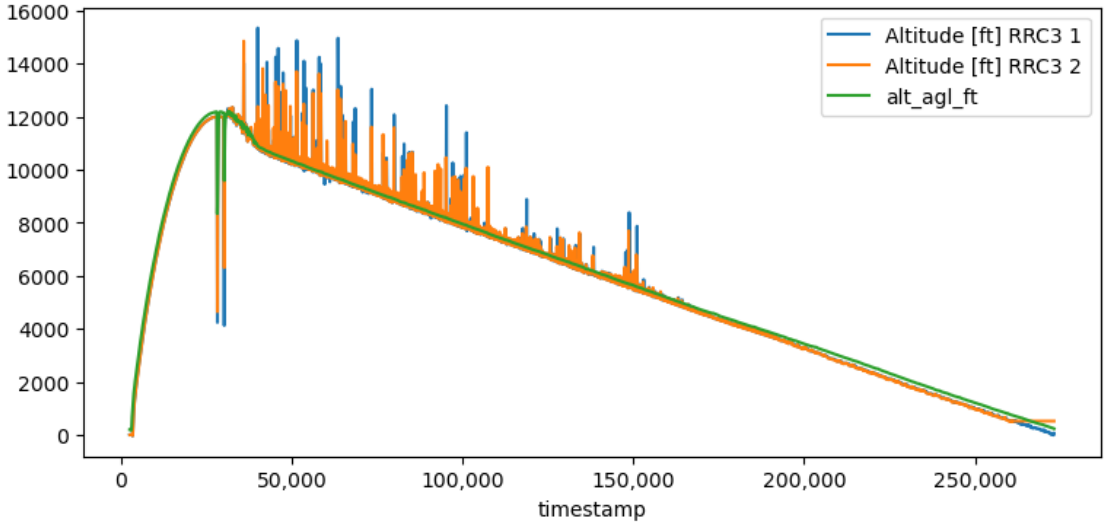
Aligned by charge spikes. Based on the Grim's IMU and barometer, barometer responds faster to the charges than it does a change in ambient pressure

[873]:
```
rrc3_1['timestamp'] = rrc3_1.index.to_series()
rrc3_2['timestamp'] = rrc3_2.index.to_series()
```

[874]:
```
rrc3_1_manual = 2550
rrc3_1.index = rrc3_1['timestamp'] + rrc3_1_manual
rrc3_2.index = rrc3_2['timestamp'] + 2600
```

[875]:
```
combo = rrc3_1.join(rrc3_2, how='outer', lsuffix=' RRC3 1', rsuffix=' RRC3 2').
 ↪join(
   grim)[['Altitude [ft] RRC3 1', 'Altitude [ft] RRC3 2', 'alt_agl_ft']].
 ↪interpolate()
p = combo.plot(figsize=(9, 4))
```

```
comma_ax(p)
```



7 Data by Type

7.1 Acceleration

Create a table of Grim Acceleration data and Camera Acceleration data

[876]:
```
grim_cols = ['acc (m/s^2)', 'accx (m/s^2)', 'accy (m/s^2)', 'accz (m/s^2)']
cam1_cols = ['a[m/s2]', 'ax[m/s2]', 'ay[m/s2]', 'az[m/s2]']
accels = grim[grim_cols
              ].join(cam1[cam1_cols]).interpolate(limit_direction='forward')
accels.rename(columns={
    'acc (m/s^2)': 'grim acc',
    'accx (m/s^2)': 'grim accx',
    'accy (m/s^2)': 'grim accy',
    'accz (m/s^2)': 'grim accz',
    'a[m/s2]': 'cam acc',
    'ax[m/s2]': 'cam accx',
    'ay[m/s2]': 'cam accy',
    'az[m/s2]': 'cam accz',
}, inplace=True)

accels['cam acc'] *= 9.81
accels['cam accx'] *= 9.81
accels['cam accy'] *= 9.81
accels['cam accz'] *= 9.81

accels_trimmed = accels.loc[0:flight_end]
```

```
accels_trimmed
```

[876]:
```
                 grim acc  grim accx  grim accy  grim accz   cam acc  \
timestamp (ms)
0.0             10.423808   -0.34400    0.94700   10.37500       NaN
2.0             10.460771   -0.33700    0.94000   10.41300       NaN
4.0             10.497752   -0.33000    0.93300   10.45100       NaN
6.0             10.450612   -0.33900    0.93700   10.40300       NaN
8.0             10.481617   -0.30100    0.91800   10.43700       NaN
…                     …          …          …          …         …
282999.2         9.968669   -0.52245    9.92595    0.75955  9.827633
282999.4         9.969657   -0.52390    9.92690    0.75910  9.826836
282999.6         9.970645   -0.52535    9.92785    0.75865  9.826039
282999.8         9.971633   -0.52680    9.92880    0.75820  9.825243
283000.0         9.972621   -0.52825    9.92975    0.75775  9.824446

                cam accx  cam accy  cam accz
timestamp (ms)
0.0                  NaN       NaN       NaN
2.0                  NaN       NaN       NaN
4.0                  NaN       NaN       NaN
6.0                  NaN       NaN       NaN
8.0                  NaN       NaN       NaN
…                      …         …         …
282999.2        9.799462 -0.571931 -0.475172
282999.4        9.798504 -0.573847 -0.476130
282999.6        9.797546 -0.575763 -0.477088
282999.8        9.796588 -0.577679 -0.478046
283000.0        9.795630 -0.579595 -0.479004

[1303919 rows x 8 columns]
```

7.2 Gyroscope

Create a table of Grim Gyroscope data and Camera Gyroscope data

[877]:
```
grim_cols = ['gyrox (rad/s)', 'gyroy (rad/s)', 'gyroz (rad/s)']
cam1_cols = ['rx[rad/s2]', 'ry[rad/s2]', 'rz[rad/s2]']

gyros = grim[grim_cols].join(cam1[cam1_cols], how='outer').
 ↪interpolate(limit_direction='forward')
gyros.rename(columns={
    'gyrox (rad/s)': 'grim x',
    'gyroy (rad/s)': 'grim y',
    'gyroz (rad/s)': 'grim z',

    'rx[rad/s2]': 'cam1 x',
```

```
    'ry[rad/s2]': 'cam1 y',
    'rz[rad/s2]': 'cam1 z',
}, inplace=True)

gyros_trimmed = gyros.loc[0:flight_end]
gyros_trimmed
```

[877]:

	grim x	grim y	grim z	cam1 x	cam1 y	cam1 z
0.0	0.00700	-0.02300	0.02100	NaN	NaN	NaN
2.0	0.00700	-0.02400	0.02500	NaN	NaN	NaN
4.0	0.00700	-0.02500	0.02900	NaN	NaN	NaN
6.0	0.01300	-0.02300	0.02900	NaN	NaN	NaN
8.0	0.01700	-0.03400	0.02900	NaN	NaN	NaN
…	…	…	…	…	…	
282999.2	0.00715	-0.02495	0.00815	0.007457	0.018642	8.53117
282999.4	0.00730	-0.02490	0.00930	0.007457	0.018642	8.53117
282999.6	0.00745	-0.02485	0.01045	0.007457	0.018642	8.53117
282999.8	0.00760	-0.02480	0.01160	0.007457	0.018642	8.53117
283000.0	0.00775	-0.02475	0.01275	0.007457	0.018642	8.53117

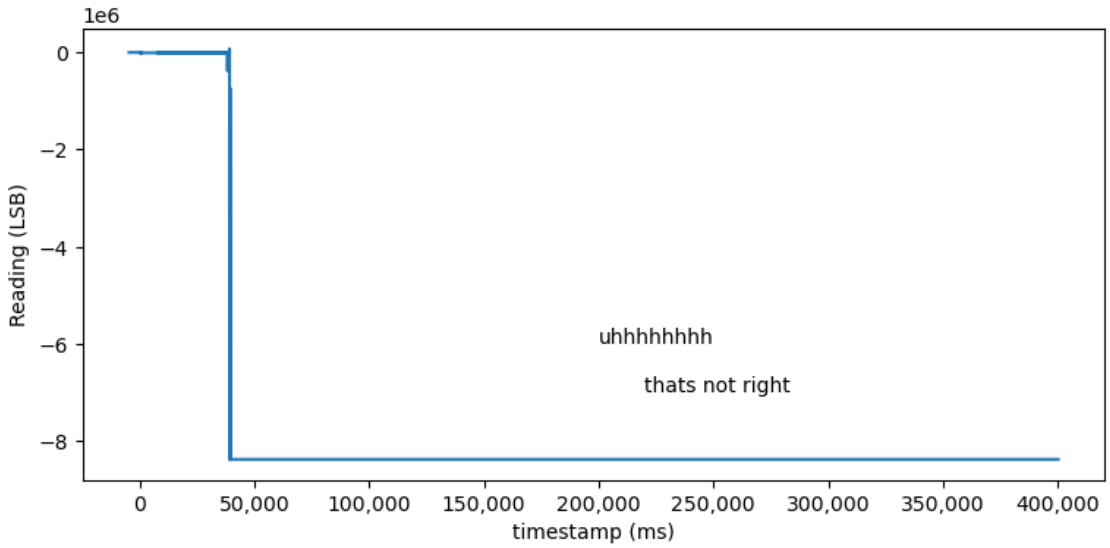
[1326584 rows x 6 columns]

7.3 Load Cell

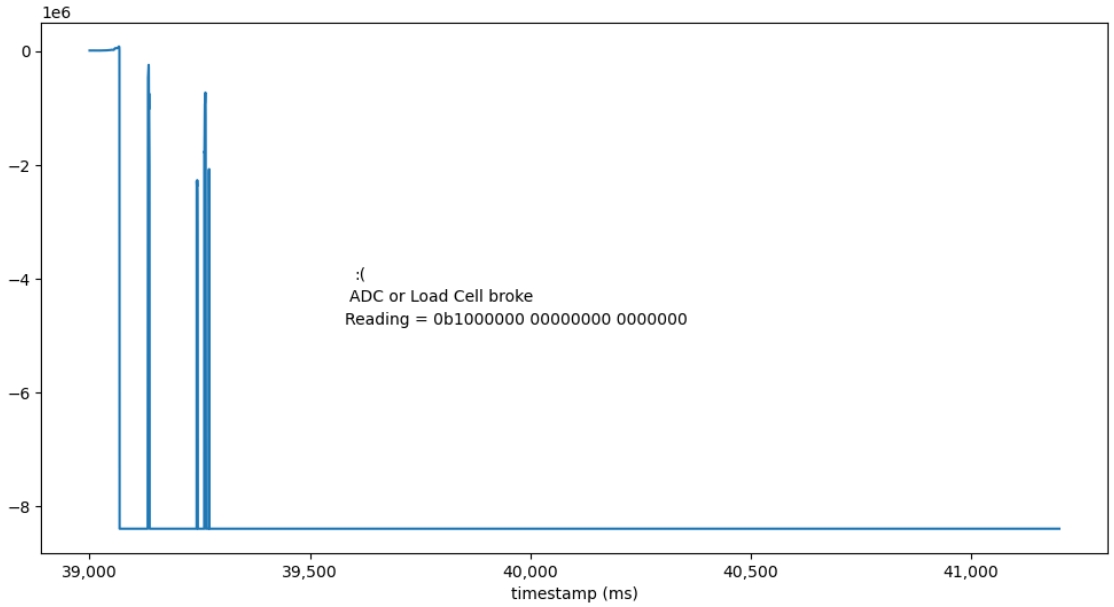
Convert ADC reading into a usable force value. Conversions from here. Calibrated by adding a bunch of lathe tools and reading out the voltage.

[878]:
```
def voltage_to_force(x):
    return 36918*x + 22


def to_volts(val):
    return 2.4 * (val) / (float(0x7fffff))


def reading_to_force(val):
    return voltage_to_force(to_volts(val))


load_cell_max_rating_kg = 50
```

[879]:
```
p = grim['reading (LSB)'].plot(figsize=(9, 4))
p.annotate("uhhhhhhhh", (200_000, -6_000_000))
p.annotate("thats not right", (220_000, -7_000_000))

p.set_ylabel("Reading (LSB)")

comma_ax(p)
```



[1057]:
```
p = grim['reading (LSB)'][39_000:41_200].plot(figsize=(12, 6))
p.annotate(':(', (39600, -4e6))
p.annotate('ADC or Load Cell broke', (39590, -4.4e6))
p.annotate('Reading = 0b1000000 00000000 0000000', (39578, -4.8e6))

comma_ax(p)
```



Negative max occurs when input signal moves out of range of what the ADC can read with those

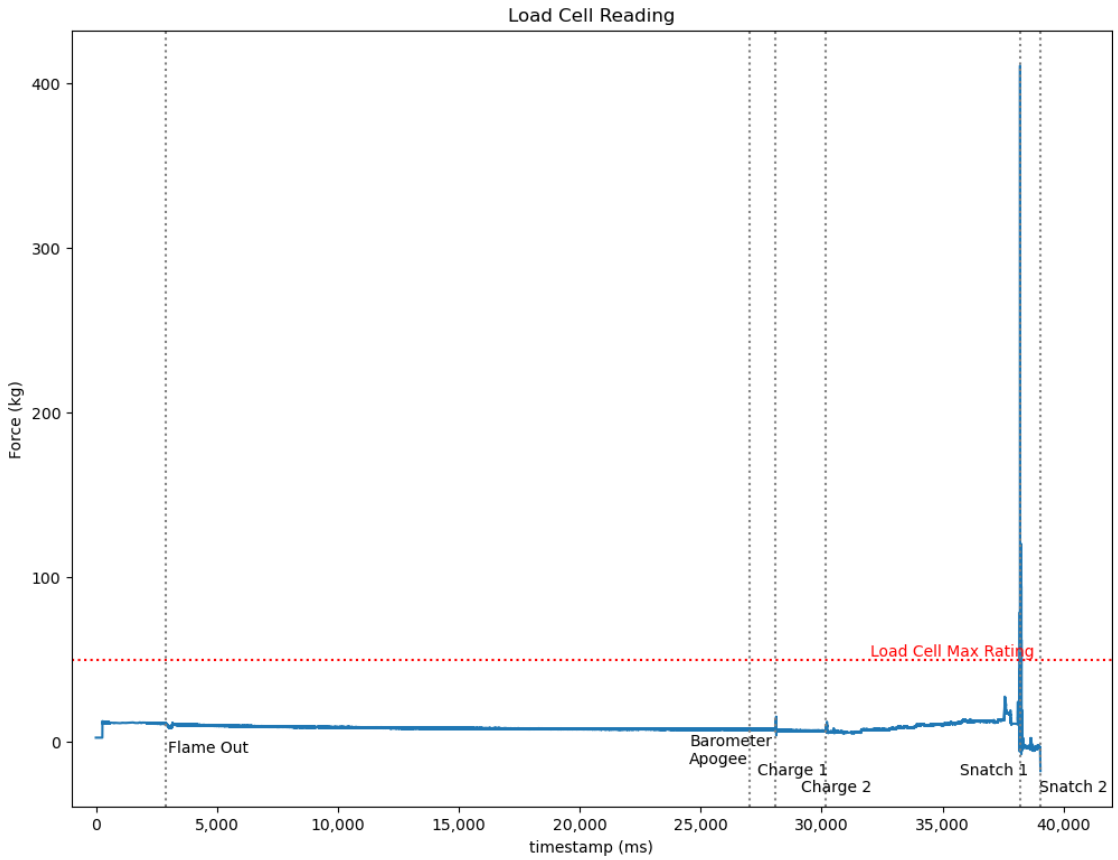
gain settings. unsure about the others

[881]:
```
weird_adc = pd.DataFrame(adc['reading (LSB)'].loc[39200:].copy().
 ↪drop_duplicates())
pd.DataFrame(adc['reading (LSB)'].loc[39200:39500].drop_duplicates().
 ↪apply(lambda x : "{0:b}".format(int(x))))
weird_adc['reading (binary)'] = weird_adc['reading (LSB)'].apply(lambda x :␣
 ↪f"{int(x):b}")
weird_adc['volts'] = weird_adc['reading (LSB)'].apply(to_volts)
weird_adc['force (N) (doubtful)'] =weird_adc['reading (LSB)'].
 ↪apply(reading_to_force)
```

We can still look at the ADC before it exploded

[882]:
```
# negated so that more force is more positive. sign is arbitrary
grim['good readings'] = grim['reading (LSB)'][0:39050] * -1
grim['good force(n)'] = grim['good readings'].apply(reading_to_force)
grim['good force(kg)'] = grim['good force(n)'] / 9.81
```

[1071]:
```
p = grim['good force(kg)'][:40000].plot(figsize=(12, 9))
plt.figure().set_dpi(80)
p.set_xbound(-1000, 42_000)
p.set_ylabel("Force (kg)")

p.annotate("Charge 1", (charge1_span[0] - 750, -20))
p.axvline(charge1_span[0], linestyle=':', color='gray')

p.annotate("Charge 2", (charge2_span[0] - 1000, -30))
p.axvline(charge2_span[0], linestyle=':', color='gray')

p.annotate("Load Cell Max Rating", (32_000, 52), color='red')
p.axhline(load_cell_max_rating_kg, color='red', linestyle=':')

p.annotate("Barometer\nApogee", (apogee-2500, -13))
p.axvline(apogee, color='gray', linestyle=':')

p.annotate("Snatch 1", (35_700, -20))
p.axvline(parachute_catch_1, color='gray', linestyle=':')

p.annotate("Snatch 2", (39_000, -30))
p.axvline(parachute_catch_2, color='gray', linestyle=':')

p.annotate("Flame Out", (flame_out+100, -6))
p.axvline(flame_out, color='gray', linestyle=':')

p.set_title("Load Cell Reading")
comma_ax(p)
```



<Figure size 512x384 with 0 Axes>

8 Comparisons

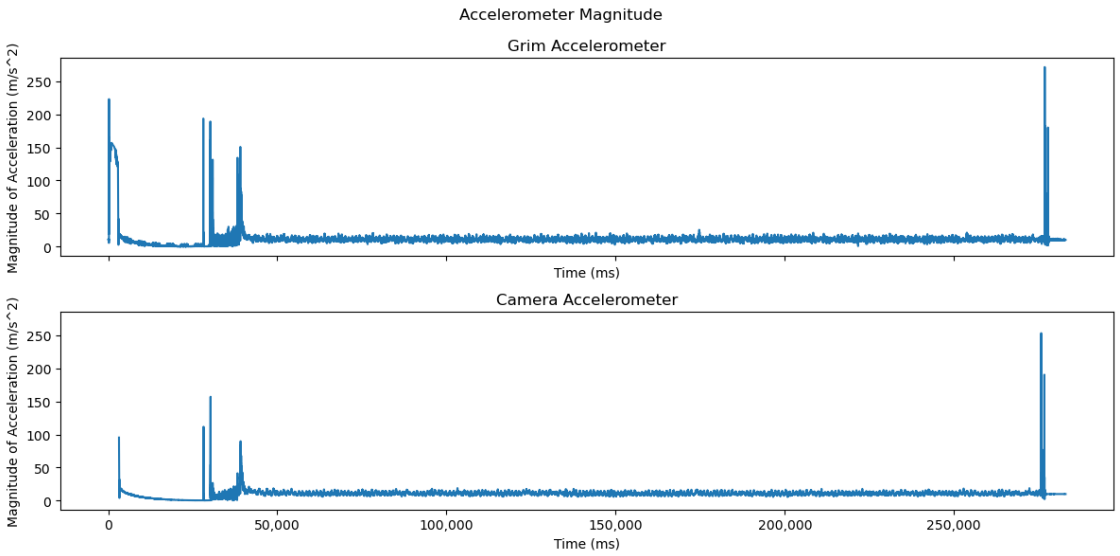
[1072]:
```
def compareSeries(a: pd.Series, b: pd.Series, atitle: str = "", btitle: str =
↪"", xlabel: str = "", ylabel: str ="", title: str="", useTex: bool = False,
↪figsize=(10, 6), shareY=False):
    plt.rcParams['text.usetex'] = useTex

    fig, axs = plt.subplots(2, sharex=True, figsize=figsize, sharey=shareY)
    fig.suptitle(title)

    axs[0].set_title(atitle)
    axs[0].plot(a.index, a)
    axs[0].set_xlabel(xlabel)
    axs[0].set_ylabel(ylabel)

    axs[1].set_title(btitle)
    axs[1].plot(b.index, b)
```

```
    axs[1].set_xlabel(xlabel)
    axs[1].set_ylabel(ylabel)

    fig.tight_layout()
    comma_ax(axs[0])
    return (fig, axs)
```

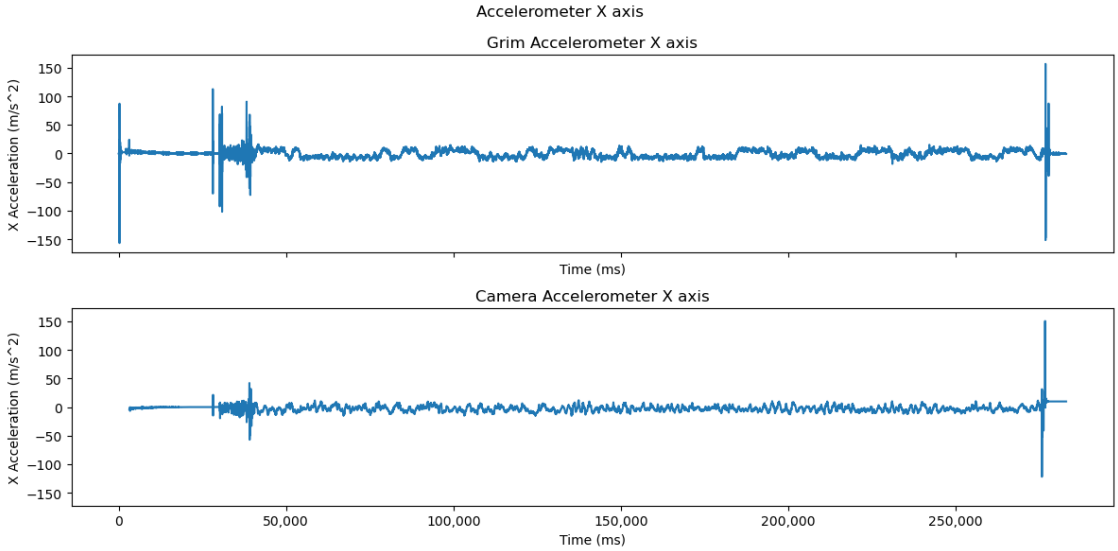
[1093]:
```
def compareNSeries(title: str, sers, figsize=(10, 6), sharey=False):
    '''
    sers is (title, xlabel, ylabel, data)
    '''
    fig, axs = plt.subplots(len(sers), sharex=True, figsize=figsize,␣
 ↪sharey=sharey)
    if len(sers)==1:
        axs = [axs]
    fig.suptitle(title)
    for i, ser in enumerate(sers):
        axs[i].set_title(ser[0])
        axs[i].set_xlabel(ser[1])
        axs[i].set_ylabel(ser[2])
        axs[i].plot(ser[3].index, ser[3])
    comma_ax(axs[0])
    fig.tight_layout()
    return fig, axs
```

8.1 Camera vs Grim Accelerometer

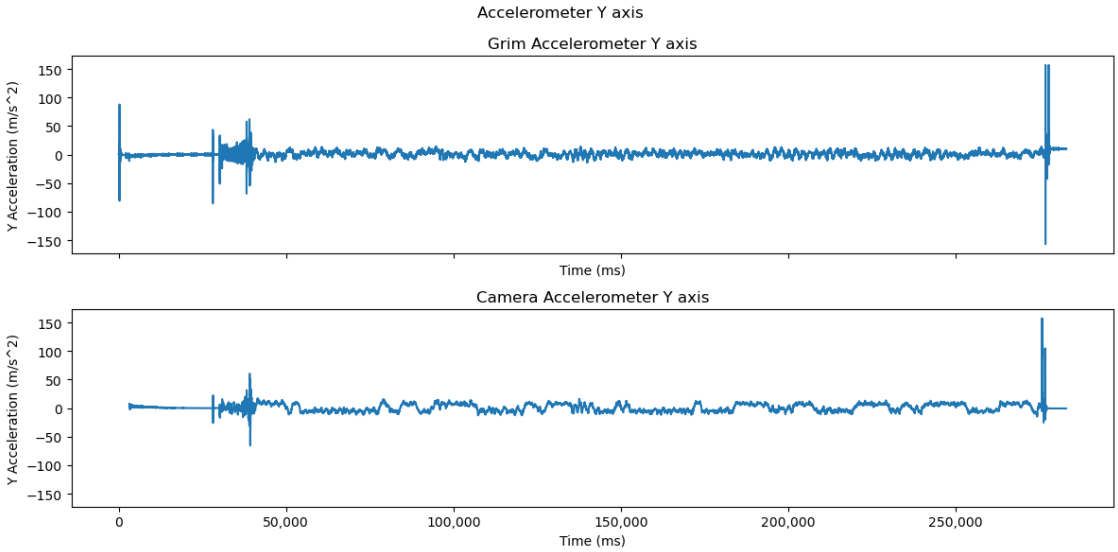
[886]:
```
compareSeries(accels_trimmed['grim acc'], accels_trimmed['cam acc'], 'Grim␣
 ↪Accelerometer', 'Camera Accelerometer', 'Time (ms)', 'Magnitude of␣
 ↪Acceleration (m/s^2)', 'Accelerometer Magnitude', shareY=True);
```



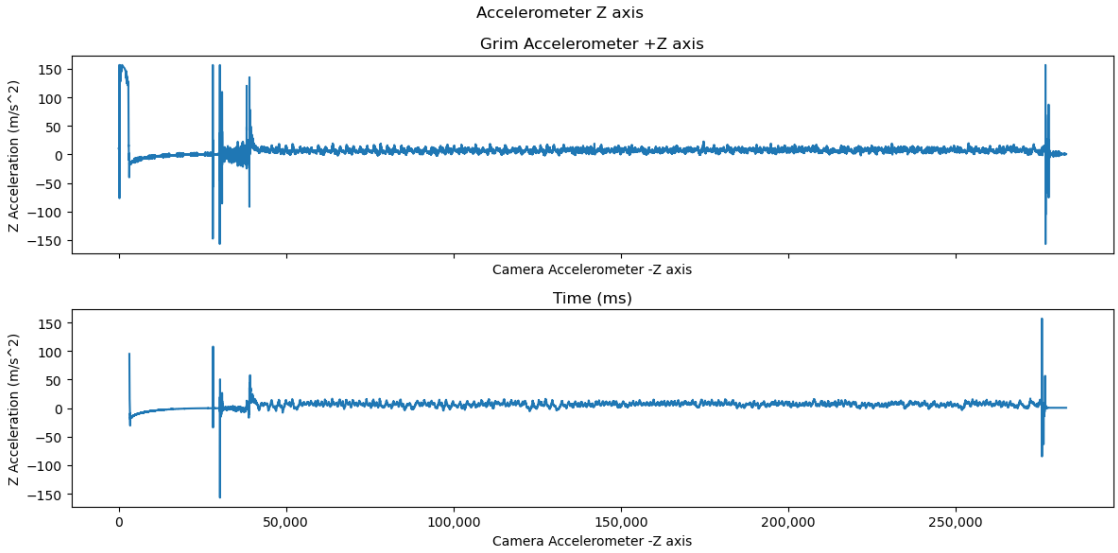
[887]:
```
compareSeries(accels_trimmed['grim accx'], accels_trimmed['cam accx'], 'Grim␣
↪Accelerometer X axis', 'Camera Accelerometer X axis', 'Time (ms)', 'X␣
↪Acceleration (m/s^2)', 'Accelerometer X axis', shareY=True);
```



[888]:
```
compareSeries(accels_trimmed['grim accy'], accels_trimmed['cam accy'], 'Grim␣
↪Accelerometer Y axis', 'Camera Accelerometer Y axis', 'Time (ms)', 'Y␣
↪Acceleration (m/s^2)', 'Accelerometer Y axis', shareY=True);
```

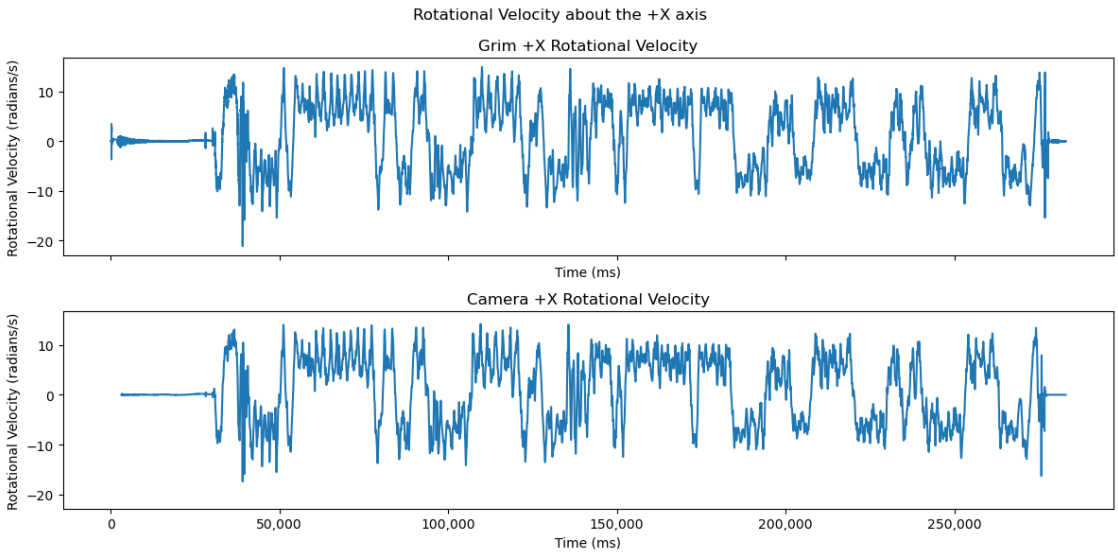


[889]:
```
compareSeries(accels_trimmed['grim accz'], -accels_trimmed['cam accz'], 'Grim␣
↪Accelerometer +Z axis', 'Time (ms)', 'Camera Accelerometer -Z axis', 'Z␣
↪Acceleration (m/s^2)', 'Accelerometer Z axis', shareY=True);
```

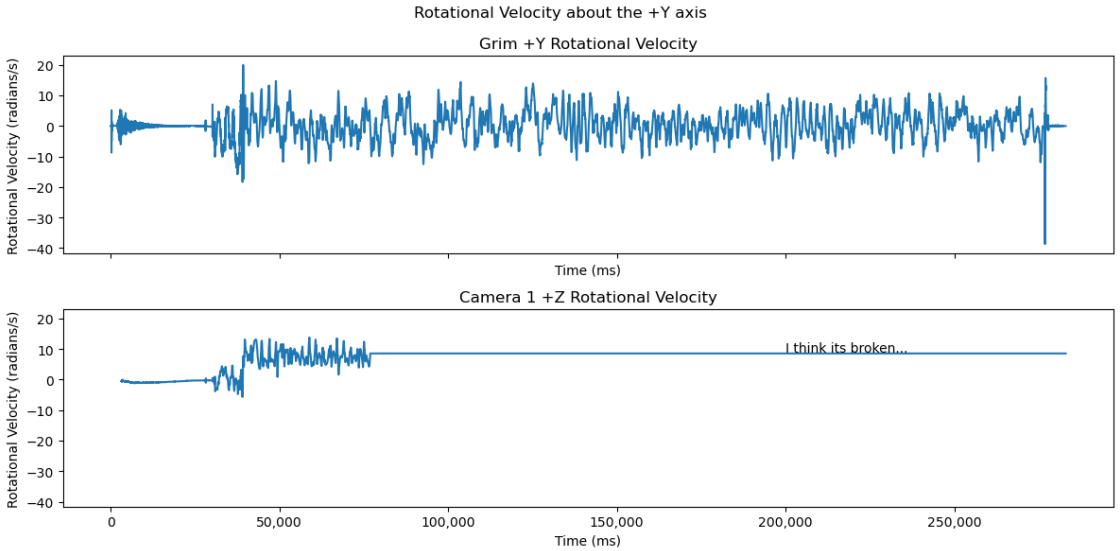


8.2 Camera vs Grim Gyroscope

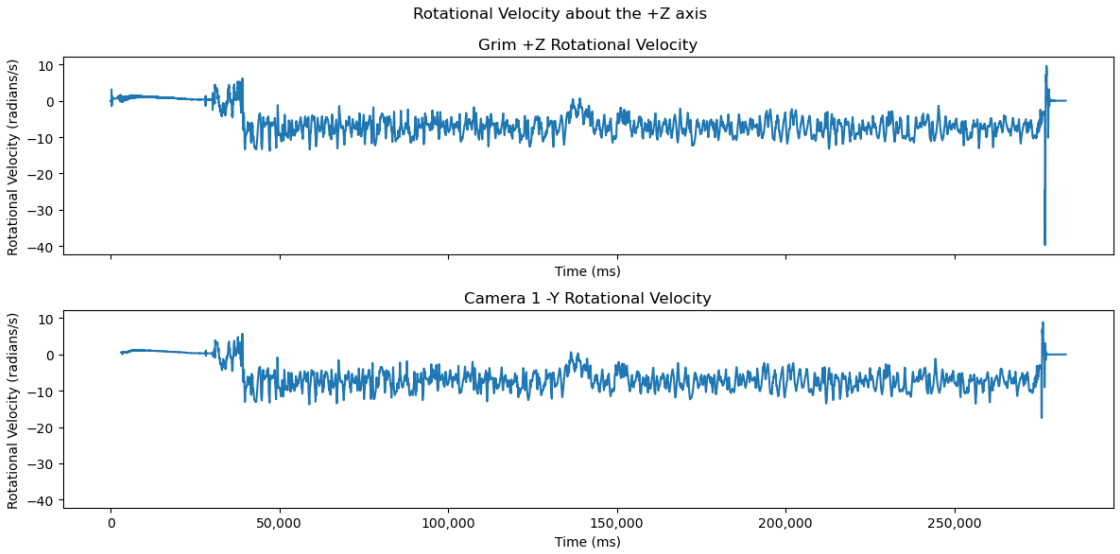
[890]:
```
compareSeries(gyros_trimmed['grim x'], gyros_trimmed['cam1 x'], 'Grim +X␣
↪Rotational Velocity', 'Camera +X Rotational Velocity', 'Time (ms)',␣
↪'Rotational Velocity (radians/s)', 'Rotational Velocity about the +X axis',␣
↪shareY=True);
```



[891]:
```
fig, axs = compareSeries(gyros_trimmed['grim y'], gyros_trimmed['cam1 z'],␣
↪'Grim +Y Rotational Velocity', 'Camera 1 +Z Rotational Velocity', 'Time␣
↪(ms)', 'Rotational Velocity (radians/s)', 'Rotational Velocity about the +Y␣
↪axis', shareY=True);
axs[1].annotate('I think its broken...', (200_000, 9));
```



[892]:
```
compareSeries(gyros_trimmed['grim z'], gyros_trimmed['cam1 y']*-1, 'Grim +Z␣
↪Rotational Velocity', 'Camera 1 -Y Rotational Velocity', 'Time (ms)',␣
↪'Rotational Velocity (radians/s)', 'Rotational Velocity about the +Z axis',␣
↪shareY = True);
```



9 Flight Analysis

9.1 Overall

9.1.1 Pad Time

The system was on for 3 hours, 44 minutes, and 13 seconds before detecting boost.

9.1.2 Flight Time

[893]:
```
print(f"Flight Time: {flight_end // 1000 // 60} minutes {flight_end // 1000 %␣
↪60} seconds")
```

```
Flight Time: 4 minutes 43 seconds
```

9.1.3 How fast did the sensors actually collect

i2c lockup issue, flash speeds, and more all contribute to not being able to collect as fast as we want. What did we actually get

[894]:
```
fast_period = 1 / (len(fast) /  (fast.index.max() - fast.index.min()))
slow_period = 1 / (len(slow) /  (slow.index.max() - slow.index.min()))
adc_period = 1 / (len(adc)  /  (adc.index.max() - adc.index.min()))
```

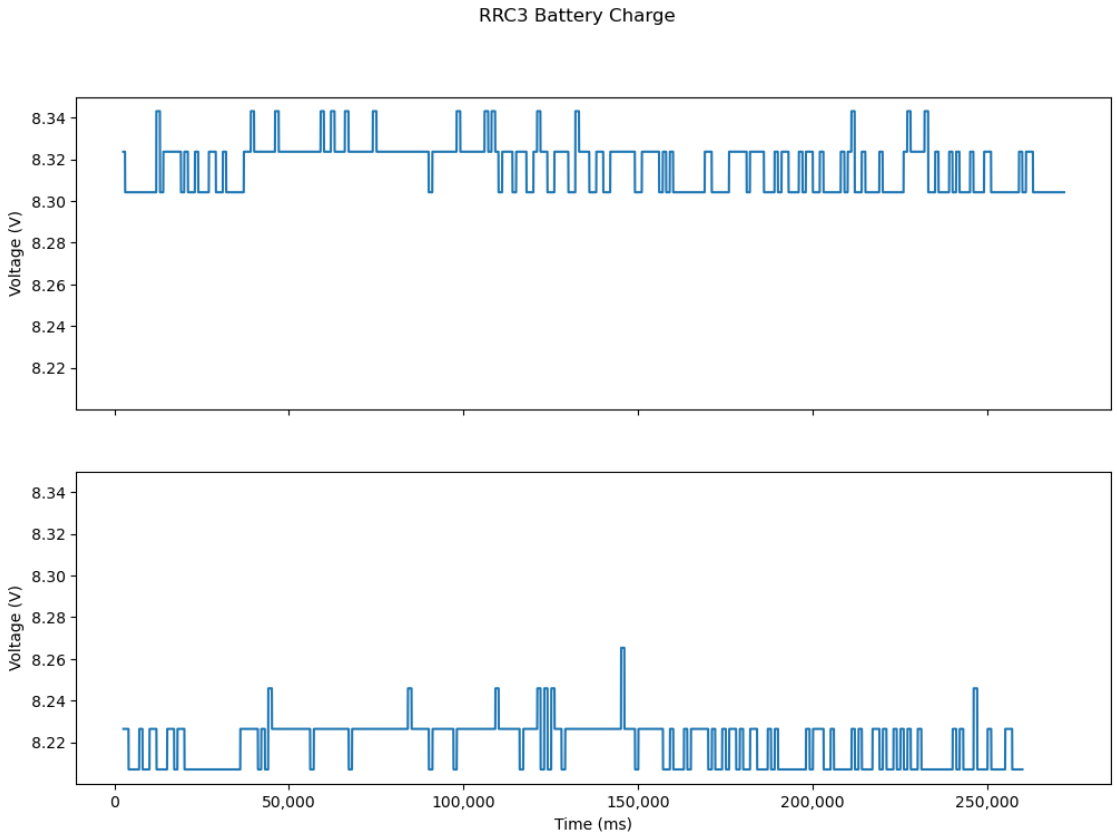
```
pre_imu_period = 1 / (len(pre_imu)  /  (pre_imu.index.max() - pre_imu.index.
 ↪min()))
pre_alt_period = 1 / (len(pre_alt)  /  (pre_alt.index.max() - pre_alt.index.
 ↪min()))

print(f"IMU, pressure period:                        {fast_period: 3.2f} ms")
print(f"Battery, temperature, humidity  period:     {slow_period: 3.2f} ms")
print(f"ADC period:                                  {adc_period: 3.2f} ms")
print()
print(f"IMU boost Detect period:                     {pre_imu_period: 3.2f} ms")
print(f"Altitude boost Detect period:                {pre_alt_period: 3.2f} ms")
```

```
IMU, pressure period:                        3.55 ms
Battery, temperature, humidity  period:      997.50 ms
ADC period:                                  0.22 ms

IMU boost Detect period:                     2.77 ms
Altitude boost Detect period:                9.98 ms
```

9.1.4 Batteries

RRC3s

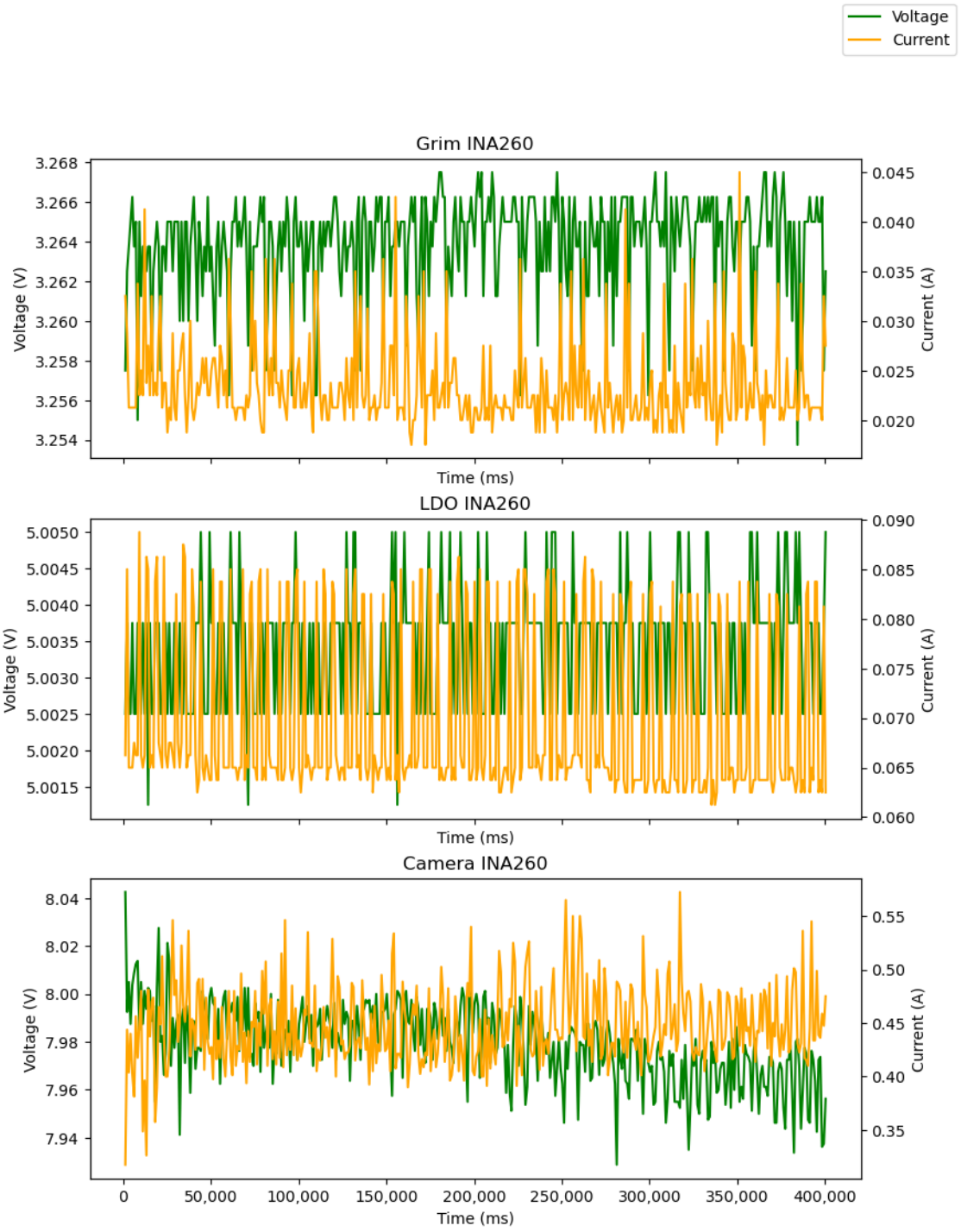
[1074]:
```
fig, axs = plt.subplots(2, 1, sharex=True, sharey=True, figsize=(12, 8))
fig.suptitle("RRC3 Battery Charge")
axs[0].plot(rrc3_1.index, rrc3_1['Voltages [V]'])
axs[1].plot(rrc3_2.index, rrc3_2['Voltages [V]'])
axs[0].set_ylabel("Voltage (V)")
axs[1].set_ylabel("Voltage (V)")
axs[1].set_xlabel("Time (ms)");

comma_ax(axs[0])
```



Grim

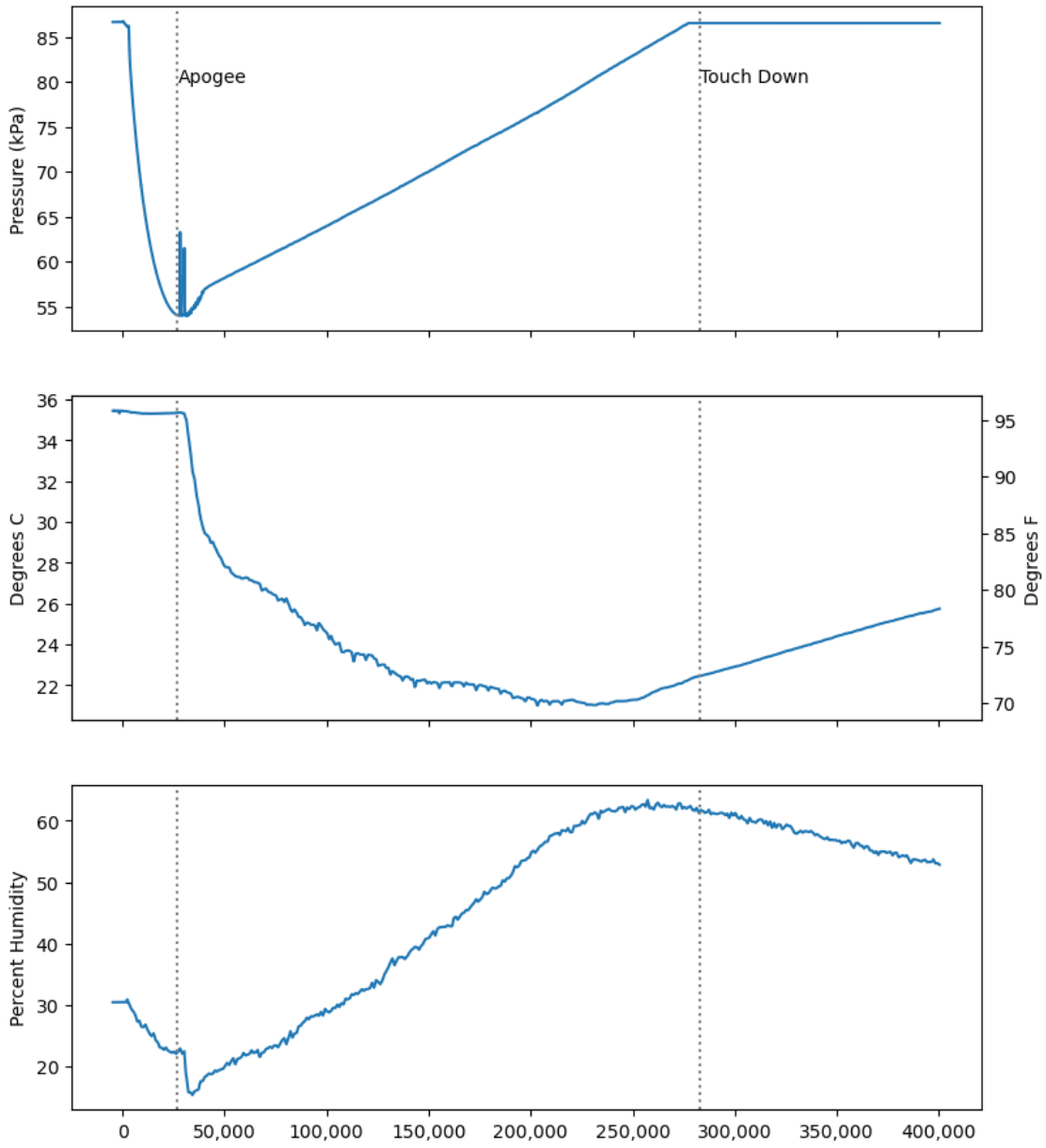
[1075]:
```
readings = [
    ('grim_voltage (mV)','grim_current (mA)', 3.3, "Grim INA260"),
    ('load_cell_voltage (mV)', 'load_cell_current (mA)', 5, "LDO INA260"),
    ('bat_voltage (mV)', 'bat_current (mA)', 8, "Camera INA260")
]
fig, axs = plt.subplots(len(readings),1,sharex=True, figsize=(9, 12))
vl = None
cl = None
for i, (ax, reading) in enumerate(zip(axs, readings)):
    left_axis = reading[0].split(' ')[0]
    right_axis = reading[1].split(' ')[0]
    voltage_v = slow[reading[0]] / 1000.0
    current_a = slow[reading[1]] / 1000.0
    target_voltage = reading[2]
    ax.set_title(reading[3])
    ax.plot(voltage_v.index, voltage_v, color = 'green', label = "Voltage" if i␣
 ↪== 0 else None)
    ax.set_ylabel("Voltage (V)")
```

```
    ax2 = ax.twinx()
    ax2.plot(current_a.index, current_a, color = 'orange', label = "Current" if␣
 ↪i == 0 else None)
    ax2.set_ylabel("Current (A)")

    ax.set_xlabel("Time (ms)")
comma_ax(axs[0])
plt.figlegend();
```


Voltage
Current
Grim INA260
Voltage (V)
Current (A)
Time (ms)
LDO INA260
Voltage (V)
Current (A)
Time (ms)
Camera INA260
Voltage (V)
Current (A)
Time (ms)

9.1.5 Atmospheric Conditions

[897]:
```
flight = grim[:]
fig, axs = plt.subplots(3, 1, sharex=True, figsize = (9, 11))

axs[0].plot(flight.index, flight['pressure (kPa)'])
axs[0].set_ylabel("Pressure (kPa)")
axs[0].annotate("Apogee", (apogee, 80))
axs[0].annotate("Touch Down", (flight_end, 80))

axs[1].plot(flight.index, flight['temperature (degrees C)'])
axs[1].set_ylabel("Degrees C")

y2_lim = [x*9/5 + 32 for x in axs[1].get_ylim()]

ax2 = axs[1].twinx()
ax2.set_ylim(y2_lim)
ax2.set_ylabel("Degrees F")

axs[2].plot(flight.index, flight['humidity (% humidity)'])
axs[2].set_ylabel("Percent Humidity");

axs[0].axvline(apogee, color = 'gray', linestyle = ":")
axs[1].axvline(apogee, color = 'gray', linestyle = ":")
axs[2].axvline(apogee, color = 'gray', linestyle = ":")

axs[0].axvline(flight_end, color = 'gray', linestyle = ":")
axs[1].axvline(flight_end, color = 'gray', linestyle = ":")
axs[2].axvline(flight_end, color = 'gray', linestyle = ":")


comma_ax(axs[0])
```



9.2 Boost

9.2.1 Detection

Boost was detected by the IMU, the Altimeter did not see any change until after the flight started.

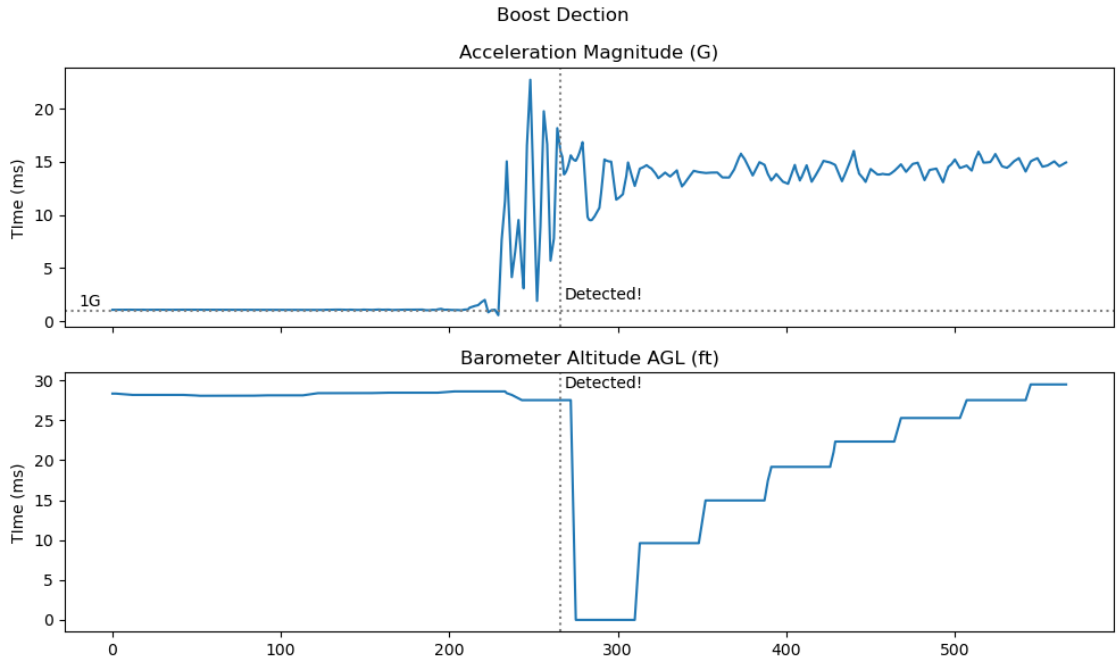
Grim waited until it felt an average acceleration of 5G over 250ms. It then considered the start of that 250ms window the start of the flight. Since the acceleration was substantially higher than 5G, the average was higher and as such the first couple hundred ms are considered part of the flight despite not actually being under power. This can be accounted after the fact, however, I didnt do that.

In reality, the software can not always keep up to match the 250ms window and its buffer is actually slightly longer.

[898]:
```
pre_imu_window_end = pre_imu.index.max()
pre_alt_window_end = pre_alt.index.max()
print(f"Actual IMU boost detect window length: {pre_imu_window_end} ms")
```

```
Actual IMU boost detect window length: 266 ms
```

[1077]:
```
detection_t = int(pre_imu.index.max())
window_end = detection_t+300
fig, axs = compareSeries((grim['acc (m/s^2)'] / 9.81)[0:window_end],␣
 ↪grim['alt_agl_ft'][0:window_end], "Acceleration Magnitude (G)", "Barometer␣
 ↪Altitude AGL (ft)", '', 'TIme (ms)', 'Boost Dection', figsize=(10, 6))

axs[0].axhline(1, color = 'gray', linestyle=':')
axs[0].annotate("1G", (-20, 1.4))
axs[0].annotate("Detected!", (detection_t+2, 2))

axs[1].annotate("Detected!", (detection_t+2, 29))

axs[0].axvline(detection_t, color='gray', linestyle=':')
axs[1].axvline(detection_t, color='gray', linestyle=':');
```

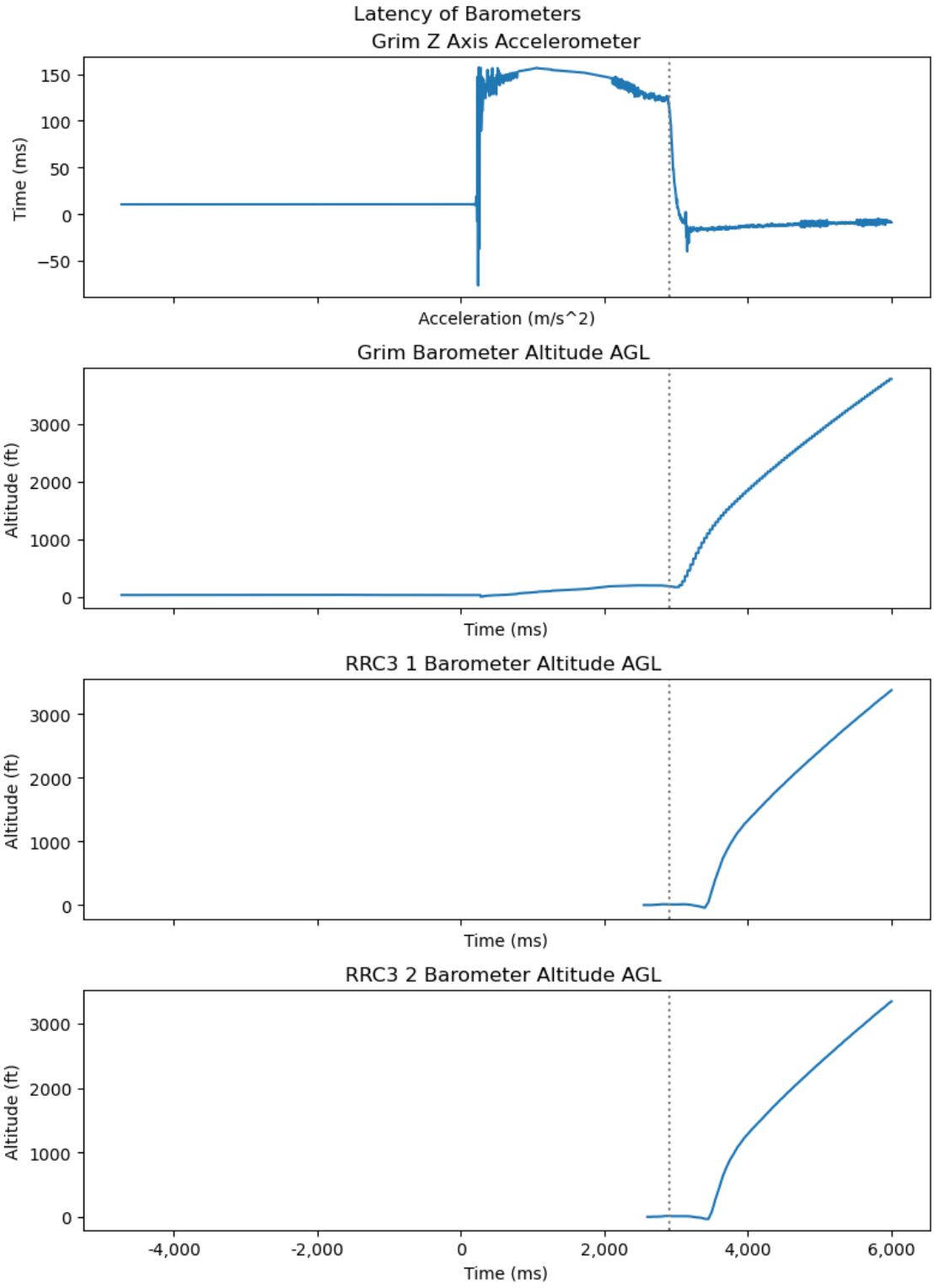


Interestingly, the barometer altitude drops when the motor first fires. Additionally, the altimeter lags substantially behind, not really responding until the entire boost is done.

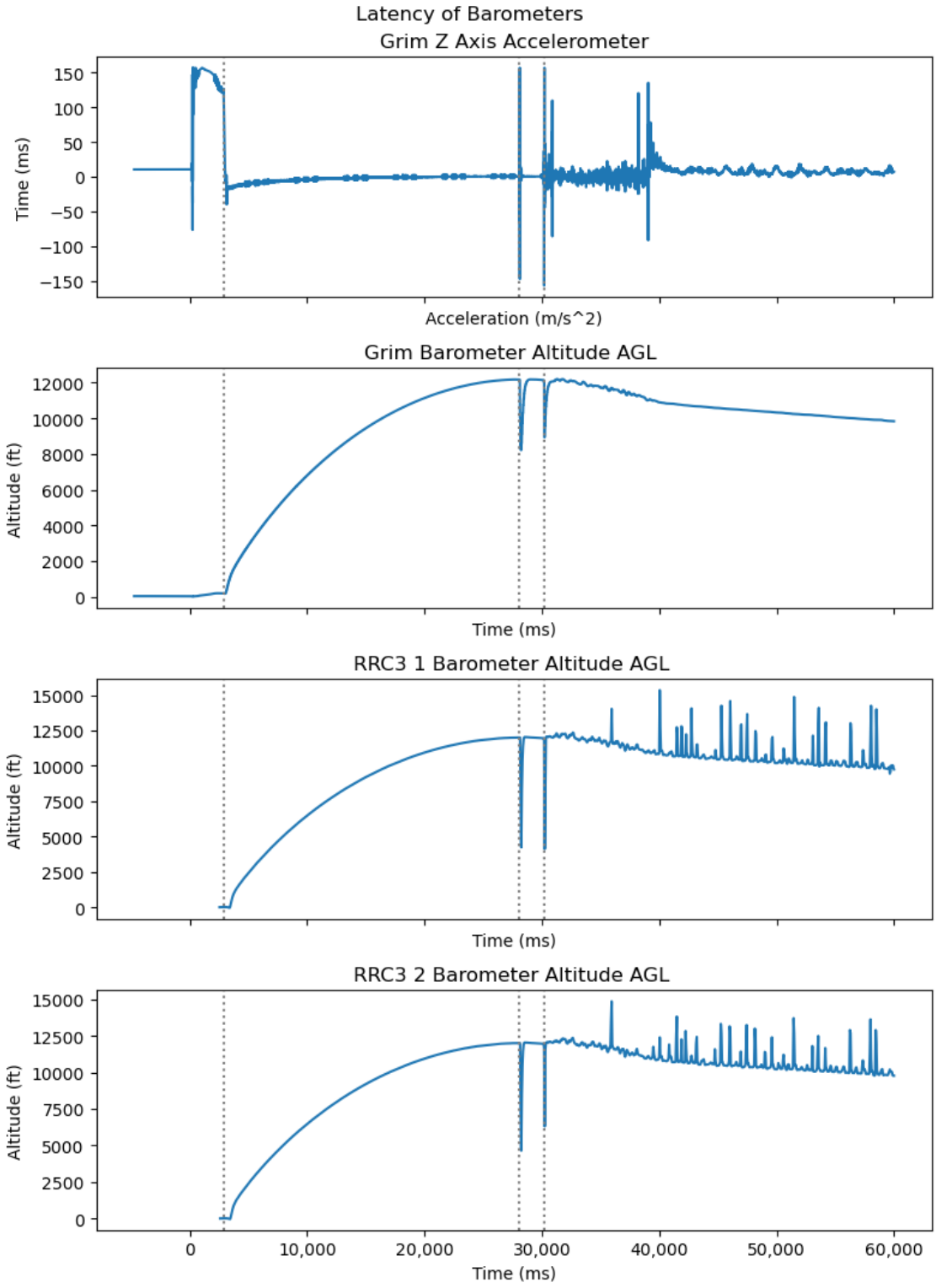
The lag also occurs in the RRC3s. When aligned by the time of first charge detonation, they lag behind similarly to the Grim Barometer.

9.2.2 Barometer Latency

[900]:
```
window_start = -5000
window_end = 6000
fig, axs = compareNSeries("Latency of Barometers", [
    ("Grim Z Axis Accelerometer", "Acceleration (m/s^2)", "Time (ms)",␣
 ↪accels['grim accz'][window_start:window_end]),
    ("Grim Barometer Altitude AGL ",  "Time (ms)", "Altitude (ft)",␣
 ↪grim['alt_agl_ft'][window_start:window_end]),
    ("RRC3 1 Barometer Altitude AGL ", "Time (ms)", "Altitude (ft)",␣
 ↪rrc3_1['Altitude [ft]'][window_start:window_end]),
    ("RRC3 2 Barometer Altitude AGL ", "Time (ms)", "Altitude (ft)",␣
 ↪rrc3_2['Altitude [ft]'][window_start:window_end]),
    ], figsize = (8, 11))
fig.tight_layout()
for ax in axs:
    ax.axvline(flame_out, color='gray', linestyle=':')
```


Latency of Barometers
Grim Z Axis Accelerometer
Time (ms)
150
100
50
0
−50
Acceleration (m/s^2)
Grim Barometer Altitude AGL
Altitude (ft)
3000
2000
1000
0
Time (ms)
RRC3 1 Barometer Altitude AGL
Altitude (ft)
3000
2000
1000
0
Time (ms)
RRC3 2 Barometer Altitude AGL
Altitude (ft)
3000
2000
1000
0
-4,000
-2,000
0
2,000
4,000
6,000
Time (ms)

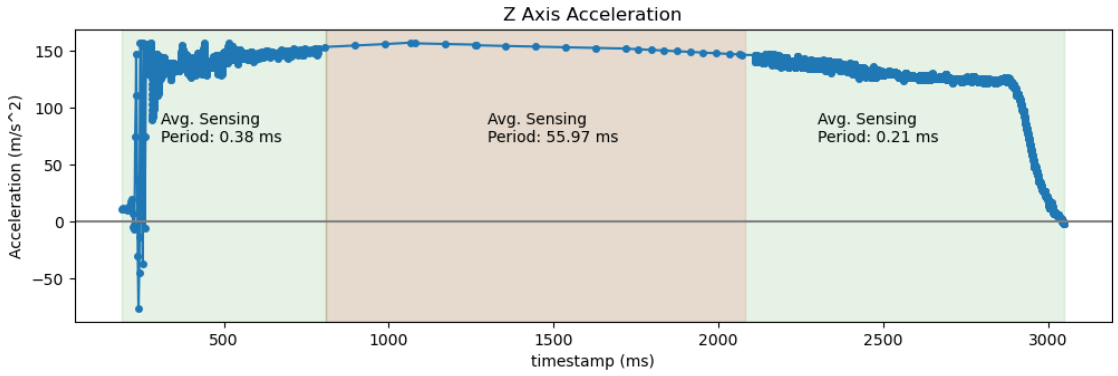
[901]:
```
window_start = -5000
window_end = 60_000
fig, axs = compareNSeries("Latency of Barometers", [
    ("Grim Z Axis Accelerometer", "Acceleration (m/s^2)", "Time (ms)",␣
 ↪accels['grim accz'][window_start:window_end]),
    ("Grim Barometer Altitude AGL ",  "Time (ms)", "Altitude (ft)",␣
 ↪grim['alt_agl_ft'][window_start:window_end]),
    ("RRC3 1 Barometer Altitude AGL ", "Time (ms)", "Altitude (ft)",␣
 ↪rrc3_1['Altitude [ft]'][window_start:window_end]),
    ("RRC3 2 Barometer Altitude AGL ", "Time (ms)", "Altitude (ft)",␣
 ↪rrc3_2['Altitude [ft]'][window_start:window_end]),
    ], figsize = (8, 11))
fig.tight_layout()
for ax in axs:
    ax.axvline(flame_out, color='gray', linestyle=':')
    ax.axvline(charge1_span[0], color='gray', linestyle=':')
    ax.axvline(charge2_span[0], color='gray', linestyle=':')
```


Latency of Barometers
Grim Z Axis Accelerometer
Time (ms)
Acceleration (m/s^2)
Grim Barometer Altitude AGL
Altitude (ft)
Time (ms)
RRC3 1 Barometer Altitude AGL
Altitude (ft)
Time (ms)
RRC3 2 Barometer Altitude AGL
Altitude (ft)
Time (ms)

9.2.3 Computer Performance

The Grim Reefer failed miserably to hit its targets between T+850 ms and T+2080 ms. The most likely reason for this is that this is when pre boost data is being flushed to flash such that in the case of any failure during flight, we still retrieve some data. Unfortunately, the cameras have not started recording at this point so we do not have that data to fill in the gaps. Nonetheless, we still have data albeit at a slower rate

[902]:
```
burn_range = (190, flame_out+150)
bad_range = (810, 2080)

sensing_period1 = grim[burn_range[0]:bad_range[0]].index.to_series().diff().
 ↪dropna().mean()

sensing_period2 = grim[bad_range[0]:bad_range[1]].index.to_series().diff().
 ↪dropna().mean()
sensing_period3 = grim[bad_range[1]:burn_range[1]].index.to_series().diff().
 ↪dropna().mean()


data = grim[burn_range[0]:burn_range[1]]
p = data['accz (m/s^2)'].plot(figsize=(12, 3.4), marker='o', markersize=4)
p.set_title("Z Axis Acceleration")
p.set_ylabel("Acceleration (m/s^2)")

p.annotate(f"Avg. Sensing\nPeriod: {sensing_period1:.2f} ms", (310, 70))
p.axvspan(burn_range[0], bad_range[0]-2, color='green', alpha=0.1)

p.annotate(f"Avg. Sensing\nPeriod: {sensing_period2:.2f} ms", (1300, 70))
p.axvspan(bad_range[0], bad_range[1], color='red', alpha=0.1)

p.annotate(f"Avg. Sensing\nPeriod: {sensing_period3:.2f} ms", (2300, 70))
p.axvspan(bad_range[0]-2, burn_range[1],  color='green', alpha=0.1)


p.axhline(0, color='gray');
```



9.2.4 Motor Performance

[903]:
```
really_burning = (0, flame_out+145)
wider_window = (0, flame_out + 2000)
data = grim['accz (m/s^2)'][wider_window[0]:wider_window[1]]
burn_data = grim['accz (m/s^2)'][really_burning[0]:really_burning[1]]


avg_acc = burn_data.mean()

grim_g = data[0:200].mean()
accz_scale = 9.81 / grim_g

# Acceleration due to gravity does not affect velocity
motor_acc = data - grim_g
vel = integrate.cumulative_trapezoid(
    motor_acc, motor_acc.index) / 1000.0 * 3.281

pos = integrate.cumulative_trapezoid(vel, motor_acc.index[1:]) / 1000.0

print(f"The motor fired for {really_burning[1]-really_burning[0]}ms causing an␣
 ↪average of {
      avg_acc / 9.81:.2f} G of measured vertical acceleration")
print(f"By integrating vertical acceleration wrt. time, we calculate a vertical␣
 ↪velocity of {
      vel.max(): .2f} ft/s when the motor flames out")
print(
    f"This is however to be taken with a grain of salt as the accelerometer is␣
 ↪not perfectly calibrated and has some bias (sitting still is not 9.81 m/s␣
 ↪but rather {grim_g:.2f} m/s)")

print(f"By simply scaling by (Expected 1G) / (Observed 1G) we get an average␣
 ↪acceleration of {
      avg_acc * accz_scale / 9.81:.2f} G and velocity of {vel.max() *␣
 ↪accz_scale:.2f} m/s")

rrc31_maxvel = rrc3_1['Velocity [ft/s]'].loc[:coast_end].max()
rrc32_maxvel = rrc3_2['Velocity [ft/s]'].loc[:coast_end].max()
print()
print(f"RRC3 1 Measured a max velocity {rrc31_maxvel}")
print(f"RRC3 2 Measured a max velocity {rrc32_maxvel}")

print()
print(f"OpenRocket Simulations predicted ~1050 ft/s")
```

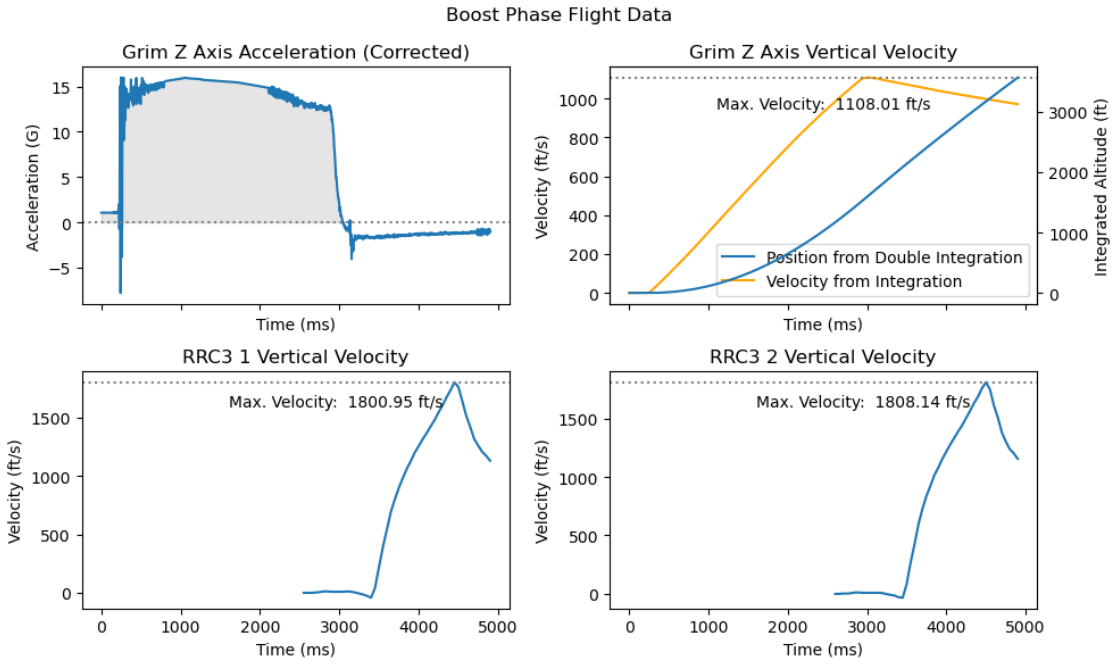
```
altitude_at_end_of_boost_grim_calc = pos.max()
print()
altitude_at_end_of_boost_grim = grim['alt_agl_ft_wout_charges_smoothed'][:
 ↪coast_start].max()
altitude_at_end_of_boost_rrc31 = rrc3_1['Altitude [ft]'][:coast_start].max()
altitude_at_end_of_boost_rrc32 = rrc3_2['Altitude [ft]'][:coast_start].max()
print('The height change during boost is highly contested. See Barometer␣
 ↪Latency section for why.')

display(pd.DataFrame([
    ['grim integrate', altitude_at_end_of_boost_grim_calc],
    ['grim barometer', altitude_at_end_of_boost_grim],
    ['RRC3 1', altitude_at_end_of_boost_rrc31],
    ['RRC3 2', altitude_at_end_of_boost_rrc32],
              ], columns = ["Method", "Altitude Gain (ft)"]).
 ↪set_index("Method"))
```

```
The motor fired for 3045ms causing an average of 12.40 G of measured vertical
acceleration
By integrating vertical acceleration wrt. time, we calculate a vertical velocity
of  1178.67 ft/s when the motor flames out
This is however to be taken with a grain of salt as the accelerometer is not
perfectly calibrated and has some bias (sitting still is not 9.81 m/s but rather
10.44 m/s)
By simply scaling by (Expected 1G) / (Observed 1G) we get an average
acceleration of 11.65 G and velocity of 1108.01 m/s

RRC3 1 Measured a max velocity 1800.952
RRC3 2 Measured a max velocity 1808.143

OpenRocket Simulations predicted ~1050 ft/s

The height change during boost is highly contested. See Barometer Latency
section for why.
```

Method	Altitude Gain (ft)
grim integrate	3792.314999
grim barometer	448.457232
RRC3 1	12.456400
RRC3 2	12.458710

[1083]:
```
fig, axs = plt.subplots(2, 2, figsize=(10, 6), sharex=True)
fig.suptitle("Boost Phase Flight Data")

axs[0][0].plot(data / 9.81)
axs[0][0].set_title("Grim Z Axis Acceleration (Corrected)")
axs[0][0].set_ylabel("Acceleration (G)")
```

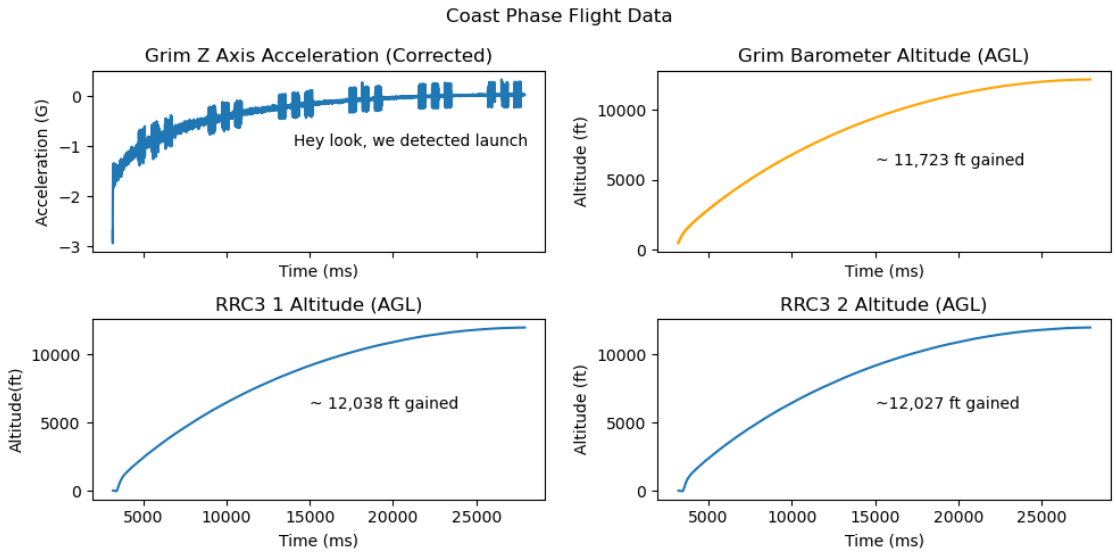
```
axs[0][0].set_xlabel("Time (ms)")

axs[0][0].fill_between(burn_data.index, burn_data / 9.81, color='gray', alpha=0.
 ↪2)
axs[0][0].axhline(0, color='gray', linestyle=":")

velL = axs[0][1].plot(motor_acc.iloc[1:].index, vel * accz_scale,␣
 ↪color='orange', label = "Velocity from Integration")
axs[0][1].set_title("Grim Z Axis Vertical Velocity")
axs[0][1].set_ylabel("Velocity (ft/s)")
axs[0][1].set_xlabel("Time (ms)")

axs[0][1].axhline(vel.max() * accz_scale, color='gray', linestyle = ':')
axs[0][1].annotate(f"Max. Velocity: {vel.max() * accz_scale: .2f} ft/s",␣
 ↪(1100,950))

ax2 = axs[0][1].twinx()
ax2.set_ylabel("Integrated Altitude (ft)")
posL = ax2.plot(motor_acc.iloc[2:].index, pos * accz_scale, label = "Position␣
 ↪from Double Integration")

lines = posL+velL
axs[0][1].legend(lines, [l.get_label() for l in lines])
# ax2.legend()

rrc3_1_window = rrc3_1[:wider_window[1]]
axs[1][0].plot(rrc3_1_window['Velocity [ft/s]'].index, rrc3_1_window['Velocity␣
 ↪[ft/s]'])
axs[1][0].set_title("RRC3 1 Vertical Velocity")
axs[1][0].set_ylabel("Velocity (ft/s)")
axs[1][0].set_xlabel("Time (ms)")
axs[1][0].axhline(rrc31_maxvel, color='gray', linestyle = ':')
axs[1][0].annotate(f"Max. Velocity: {rrc31_maxvel: .2f} ft/s", (1600,1600))


rrc3_2_window = rrc3_2[:wider_window[1]]
axs[1][1].plot(rrc3_2_window['Velocity [ft/s]'].index, rrc3_2_window['Velocity␣
 ↪[ft/s]'])
axs[1][1].set_title("RRC3 2 Vertical Velocity")
axs[1][1].set_ylabel("Velocity (ft/s)")
axs[1][1].set_xlabel("Time (ms)")
axs[1][1].axhline(rrc32_maxvel, color='gray', linestyle = ':')
axs[1][1].annotate(f"Max. Velocity: {rrc32_maxvel: .2f} ft/s", (1600,1600));

fig.tight_layout();
```



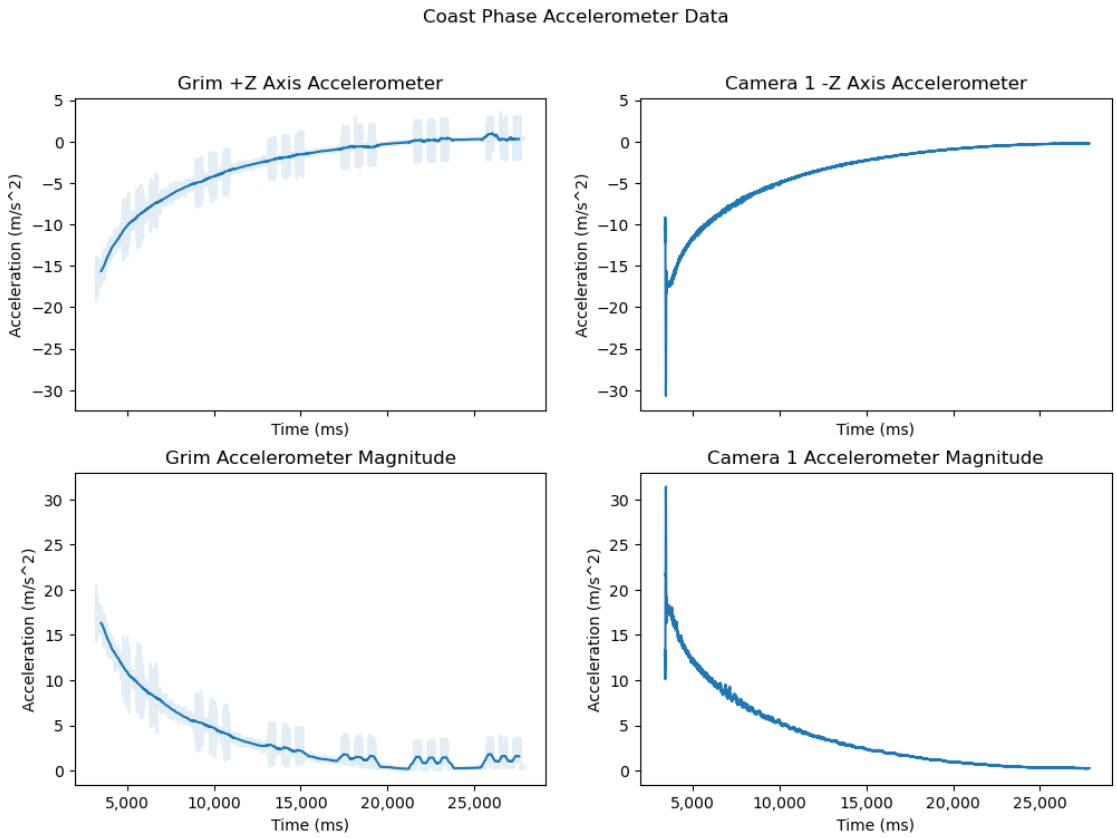
9.3 Coast

[1086]:
```
coast_window = (coast_start, coast_end)
rrc3_1_window = rrc3_1[coast_window[0]:coast_window[1]]
rrc3_2_window = rrc3_2[coast_window[0]:coast_window[1]]
grim_window = grim[coast_window[0]:coast_window[1]]

def get_range(series):
 return series.max() - series.min()


fig, axs = plt.subplots(2, 2, figsize=(10, 5), sharex=True)

fig.suptitle("Coast Phase Flight Data")

axs[0][0].plot(grim_window['accz (m/s^2)'] / 9.81 * accz_scale)
axs[0][0].set_title("Grim Z Axis Acceleration (Corrected)")
axs[0][0].set_ylabel("Acceleration (G)")
axs[0][0].set_xlabel("Time (ms)")

axs[0][0].annotate("Hey look, we detected launch", (14_000, -1))

axs[0][1].plot(grim_window['alt_agl_ft'], color='orange')
axs[0][1].set_title("Grim Barometer Altitude (AGL)")
axs[0][1].set_ylabel("Altitude (ft)")
```

```
axs[0][1].set_xlabel("Time (ms)")

axs[0][1].annotate(f"~{get_range(grim_window['alt_agl_ft']): ,.0f} ft gained",␣
 ↪(15_000, 6_000))

axs[1][0].plot(rrc3_1_window['Altitude [ft]'].index, rrc3_1_window['Altitude␣
 ↪[ft]'])
axs[1][0].set_title("RRC3 1 Altitude (AGL)")
axs[1][0].set_ylabel("Altitude(ft)")
axs[1][0].set_xlabel("Time (ms)")

axs[1][0].annotate(f"~{get_range(rrc3_1_window['Altitude [ft]']): ,.0f} ft␣
 ↪gained", (15_000, 6_000))



axs[1][1].plot(rrc3_2_window['Altitude [ft]'].index, rrc3_2_window['Altitude␣
 ↪[ft]'])
axs[1][1].set_title("RRC3 2 Altitude (AGL)")
axs[1][1].set_ylabel("Altitude (ft)")
axs[1][1].set_xlabel("Time (ms)")

axs[1][1].annotate(f"~{get_range(rrc3_2_window['Altitude [ft]']):,.0f} ft␣
 ↪gained", (15_000, 6_000))

fig.tight_layout();
```



[1089]:
```
cam1_window = cam1[coast_window[0]+200:coast_window[1]-50]
fig, axs = plt.subplots(2, 2, figsize = (12, 8), sharex=True)
fig.suptitle("Coast Phase Accelerometer Data")
axs[0][0].plot(grim_window['accz (m/s^2)'][3_200:], alpha = 0.125)
axs[0][0].plot(grim_window['accz (m/s^2)'][3_200:].rolling(2800, center=True).
 ↪mean().dropna(), c= 'C0')
axs[0][0].set_title("Grim +Z Axis Accelerometer")
axs[0][0].set_xlabel("Time (ms)")
axs[0][0].set_ylabel("Acceleration (m/s^2)")

axs[0][1].plot(cam1_window['az[m/s2]']*-9.81)
axs[0][1].set_title("Camera 1 -Z Axis Accelerometer")
axs[0][1].set_xlabel("Time (ms)")
axs[0][1].set_ylabel("Acceleration (m/s^2)")
axs[0][1].sharey(axs[0][0])
comma_ax(axs[0][0])
comma_ax(axs[0][1])

axs[1][0].plot(grim_window['acc (m/s^2)'][3_200:], alpha = 0.125)
axs[1][0].plot(grim_window['acc (m/s^2)'][3_200:].rolling(2800, center=True).
 ↪mean().dropna(), c= 'C0')
axs[1][0].set_title("Grim Accelerometer Magnitude")
axs[1][0].set_xlabel("Time (ms)")
axs[1][0].set_ylabel("Acceleration (m/s^2)")

axs[1][1].plot(cam1_window['a[m/s2]']*9.81)
axs[1][1].set_title("Camera 1 Accelerometer Magnitude")
axs[1][1].set_xlabel("Time (ms)")
axs[1][1].set_ylabel("Acceleration (m/s^2)")
axs[1][1].sharey(axs[1][0])

comma_ax(axs[1][0])
comma_ax(axs[1][1])
```

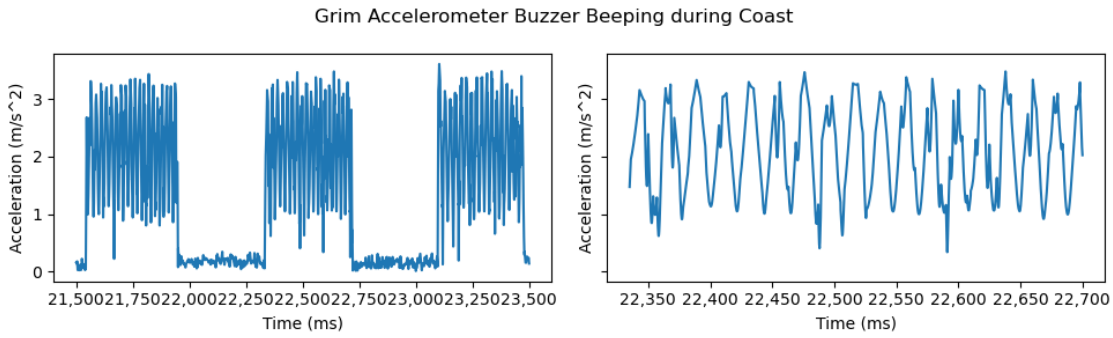


9.3.1 Sooooo yea we measured the buzzer

The accelerometer is precise enough and the buzzer strong enough that you can identify the beep-code from the IMU

[1091]:
```
section = accels.loc[21500:23500]
beep = accels.loc[22335:22700]

fig, axs = plt.subplots(1, 2, sharey=True, figsize=(10, 3))

fig.suptitle("Grim Accelerometer Buzzer Beeping during Coast")
axs[0].set_ylabel('Acceleration (m/s^2)')
axs[0].set_xlabel('Time (ms)')
comma_ax(axs[0])
axs[0].plot(section.index, section['grim acc'])


axs[1].set_ylabel('Acceleration (m/s^2)')
axs[1].set_xlabel('Time (ms)')
comma_ax(axs[1])
axs[1].plot(beep.index, beep['grim acc'])
```

```
fig.tight_layout()
```



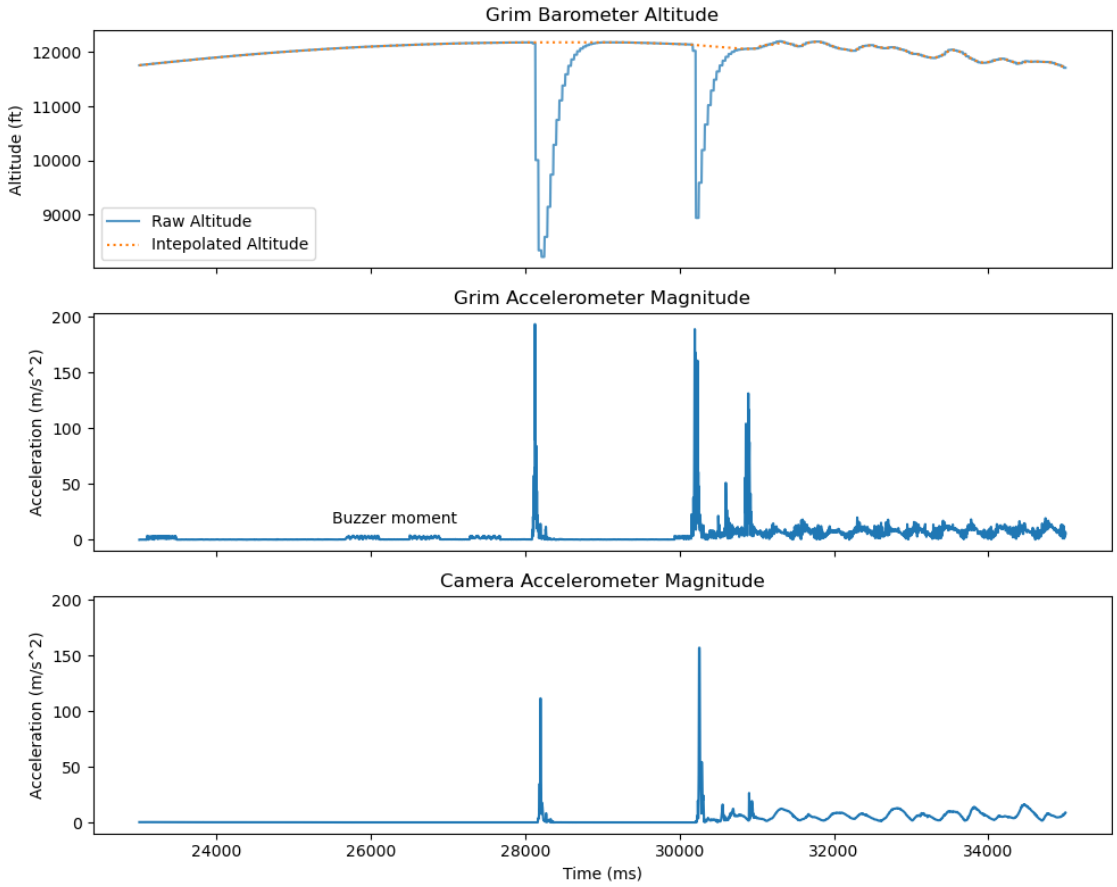
Figuring out the note the buzzer plays is left as an excercise to the reader

9.4 Apogee

Apogee is cool and also when things got a little silly

9.4.1 Charges Go Off

[1092]:
```
window = (23_000, 35_000)
grim_window = grim[window[0]:window[1]]
cam1_window = cam1[window[0]:window[1]]

fig, axs = plt.subplots(3, 1, figsize=(10, 8), sharex=True)

axs[0].plot(grim_window['alt_agl_ft'], label = "Raw Altitude", alpha = 0.75)
axs[0].plot(grim_window['alt_agl_ft_wout_charges_smoothed'], linestyle = ':',␣
 ↪label = "Intepolated Altitude")
axs[0].legend()
axs[0].set_title("Grim Barometer Altitude")
axs[0].set_ylabel("Altitude (ft)")

axs[1].plot(grim_window['acc (m/s^2)'])
axs[1].annotate("Buzzer moment", (25_500, 15))
axs[1].set_ylabel("Acceleration (m/s^2)")
axs[1].set_title("Grim Accelerometer Magnitude")

axs[2].plot(cam1_window['a[m/s2]']*9.81)
axs[2].sharey(axs[1])
axs[2].set_ylabel("Acceleration (m/s^2)")
axs[2].set_title("Camera Accelerometer Magnitude")
axs[2].set_xlabel("Time (ms)")
fig.tight_layout()
```

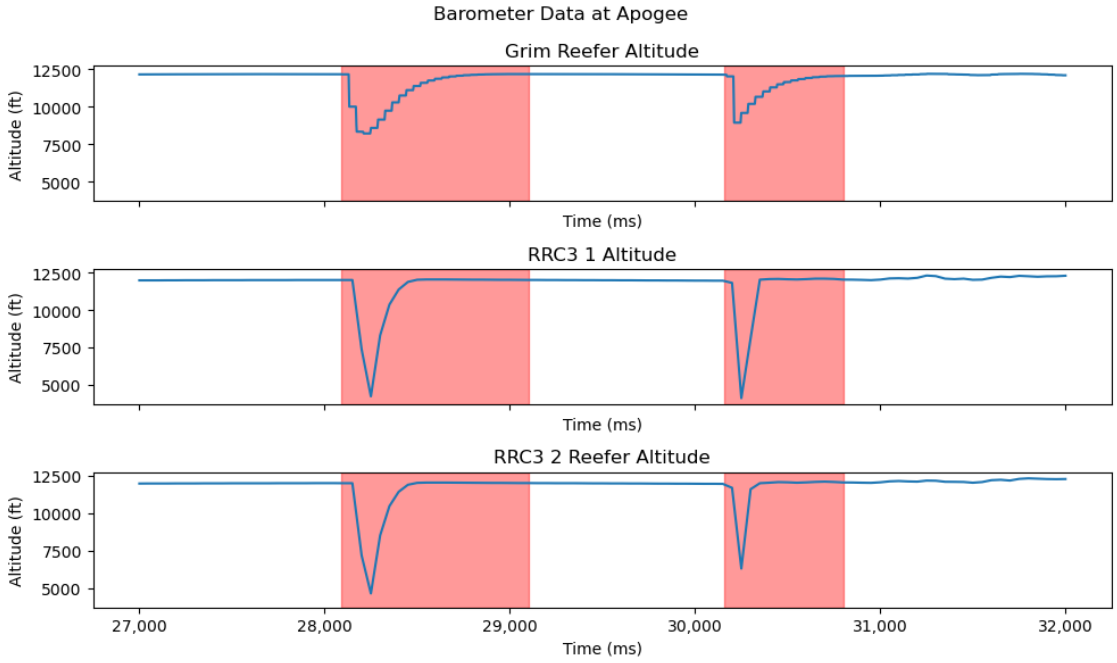


9.4.2 Barometers Have a Bad Time

Since the black powder charges cause a spike in pressure, they interfere with using the barometers to measure pressure and thus altitude.

[1094]:
```
window = (27_000, 32_000)
grim_window = grim[window[0]:window[1]]
rrc31_window = rrc3_1[window[0]:window[1]]
rrc32_window = rrc3_2[window[0]:window[1]]


fig, axs = compareNSeries("Barometer Data at Apogee",
                          [["Grim Reefer Altitude", "Time (ms)", "Altitude␣
 ↪(ft)", grim_window['alt_agl_ft']],
                           ["RRC3 1 Altitude", "Time (ms)", "Altitude (ft)",␣
 ↪rrc31_window['Altitude [ft]']],
                           ["RRC3 2 Reefer Altitude", "Time (ms)", "Altitude␣
 ↪(ft)", rrc32_window['Altitude [ft]']],
                          ], sharey=True)
```

```
for ax in axs:
    ax.axvspan(charge1_span[0], charge1_span[1], color = 'red', alpha = 0.4)
    ax.axvspan(charge2_span[0], charge2_span[1], color = 'red', alpha = 0.4)
```



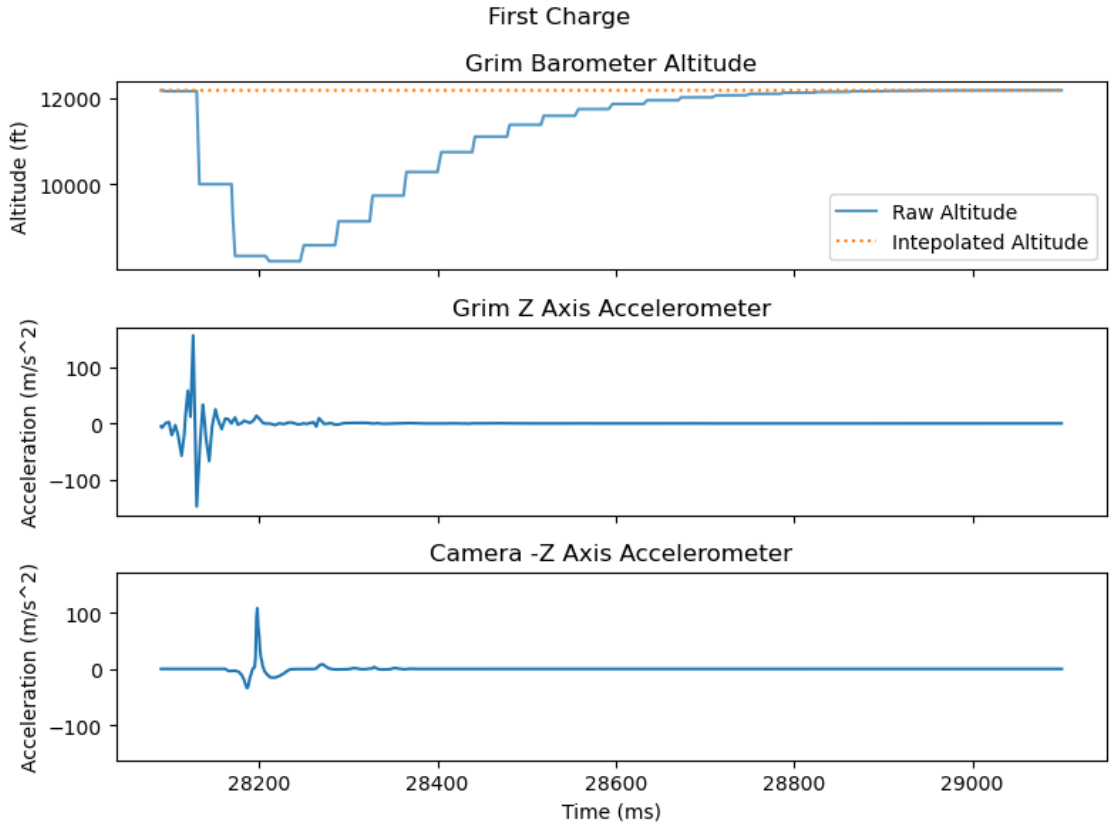
[910]:
```
print(f"First spike began at  T+ {charge1_span[0]:,} ms and lasted␣
  ↪{charge1_span[1] - charge1_span[0]} ms")
print(f"Second spike began at T+ {charge2_span[0]:,} ms and lasted␣
  ↪{charge2_span[1] - charge2_span[0]} ms")
```

```
First spike began at  T+ 28,090 ms and lasted 1010 ms
Second spike began at T+ 30,160 ms and lasted 640 ms
```

9.4.3 Charge 1

[1096]:
```
window = charge1_span
grim_window = grim[window[0]:window[1]]
cam1_window = cam1[window[0]:window[1]]

fig, axs = plt.subplots(3, 1, figsize=(8, 6), sharex=True)
fig.suptitle("First Charge")
axs[0].plot(grim_window['alt_agl_ft'], label = "Raw Altitude", alpha = 0.75)
axs[0].plot(grim_window['alt_agl_ft_wout_charges_smoothed'], linestyle = ':',␣
  ↪label = "Intepolated Altitude")
axs[0].legend()
```

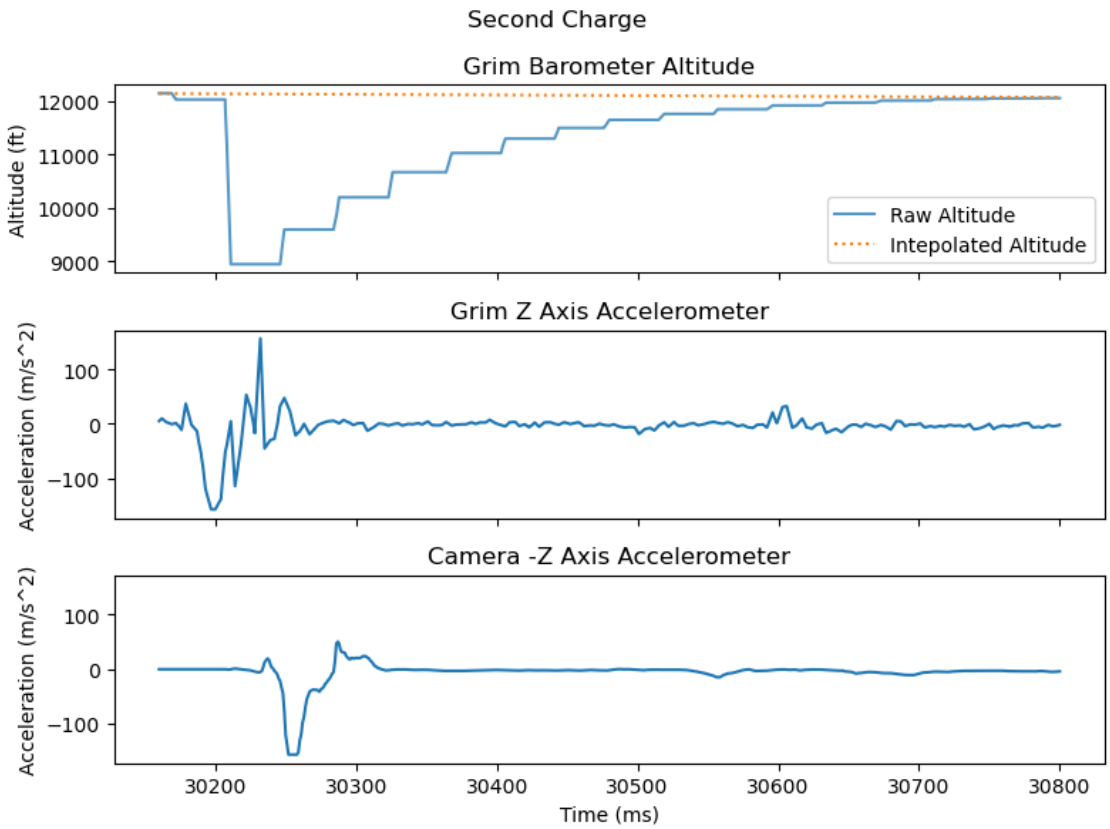
```
axs[0].set_title("Grim Barometer Altitude")
axs[0].set_ylabel("Altitude (ft)")

axs[1].plot(grim_window['accz (m/s^2)'])
axs[1].annotate("Buzzer moment", (25_500, 15))
axs[1].set_ylabel("Acceleration (m/s^2)")
axs[1].set_title("Grim Z Axis Accelerometer")

axs[2].plot(cam1_window['az[m/s2]']*-9.81)
axs[2].sharey(axs[1])
axs[2].set_ylabel("Acceleration (m/s^2)")
axs[2].set_title("Camera -Z Axis Accelerometer")
axs[2].set_xlabel("Time (ms)")
fig.tight_layout()
```



9.4.4 Charge 2

[1097]:
```
window = charge2_span
grim_window = grim[window[0]:window[1]]
cam1_window = cam1[window[0]:window[1]]
```

```
fig, axs = plt.subplots(3, 1, figsize=(8, 6), sharex=True)
fig.suptitle("Second Charge")
axs[0].plot(grim_window['alt_agl_ft'], label = "Raw Altitude", alpha = 0.75)
axs[0].plot(grim_window['alt_agl_ft_wout_charges_smoothed'], linestyle = ':',␣
 ↪label = "Intepolated Altitude")
axs[0].legend()
axs[0].set_title("Grim Barometer Altitude")
axs[0].set_ylabel("Altitude (ft)")

axs[1].plot(grim_window['accz (m/s^2)'])
axs[1].annotate("Buzzer moment", (25_500, 15))
axs[1].set_ylabel("Acceleration (m/s^2)")
axs[1].set_title("Grim Z Axis Accelerometer")

axs[2].plot(cam1_window['az[m/s2]']*-9.81)
axs[2].sharey(axs[1])
axs[2].set_ylabel("Acceleration (m/s^2)")
axs[2].set_title("Camera -Z Axis Accelerometer")
axs[2].set_xlabel("Time (ms)")
fig.tight_layout()
```

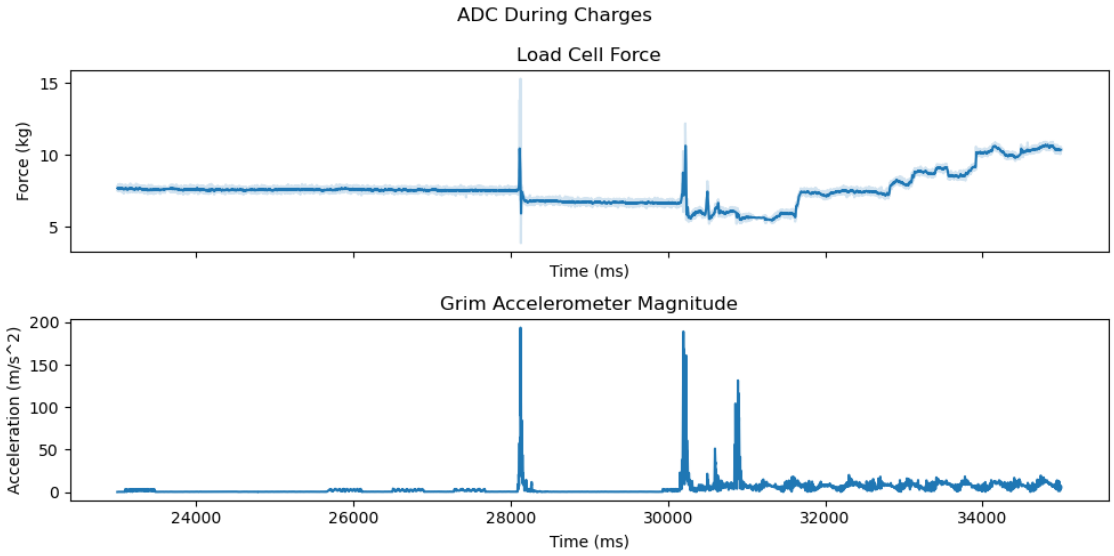


By some mechanism, the grim reefer was less affected in magnitude by these pressure spikes but took longer to recover.

9.4.5 Load Cell

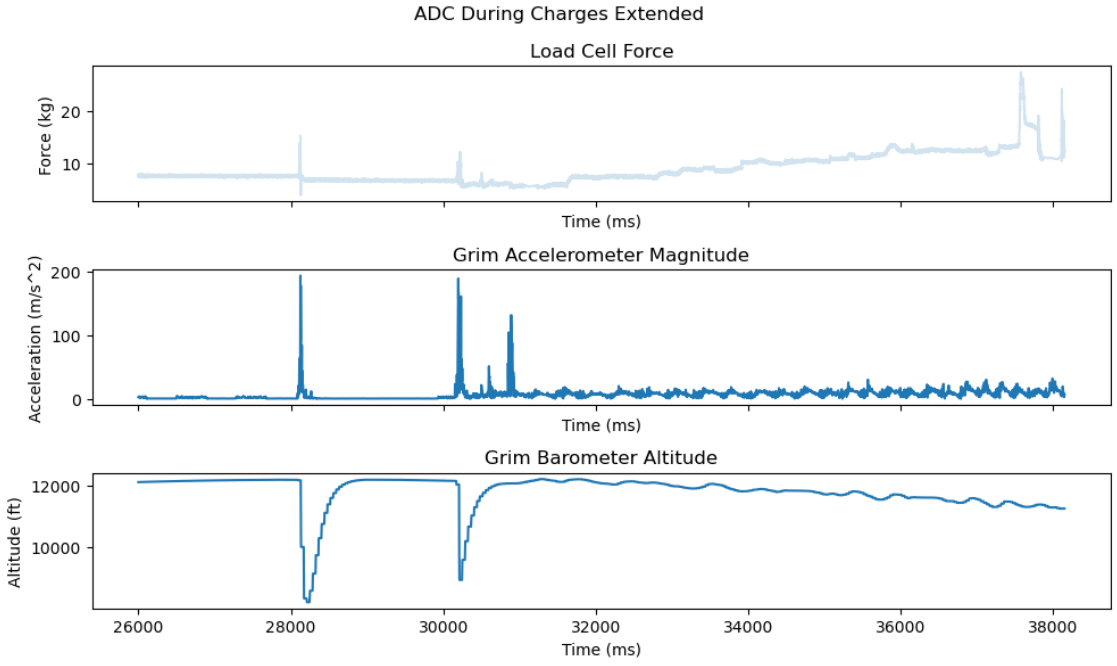
Load Cell During Charges

[1098]:
```
window = (23_000, 35_000)
grim_window = grim[window[0]:window[1]]
cam1_window = cam1[window[0]:window[1]]

fig, axs = plt.subplots(2,1, sharex=True, figsize=(10, 5))
fig.suptitle("ADC During Charges")
axs[1].plot(grim_window['acc (m/s^2)'])
axs[1].set_xlabel("Time (ms)")
axs[1].set_ylabel("Acceleration (m/s^2)")
axs[1].set_title("Grim Accelerometer Magnitude")


axs[0].plot(grim_window['good force(kg)'], alpha = 0.2)
axs[0].plot(grim_window['good force(kg)'].rolling(50, center=True).mean(),␣
↪color = 'C0')
axs[0].set_xlabel("Time (ms)")
axs[0].set_ylabel("Force (kg)")
axs[0].set_title("Load Cell Force")
fig.tight_layout()
```



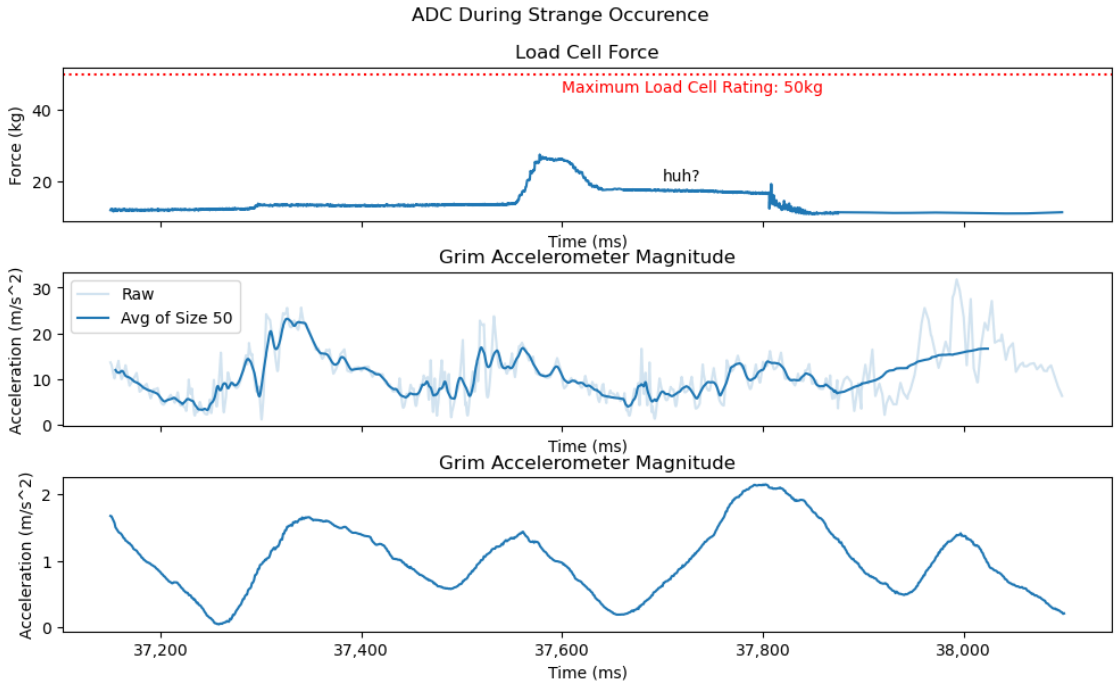
Load Cell During Charges and Later Window from before charges go off to a little before the parachute catches.

[1099]:
```
window = (26_000, 38_150)
grim_window = grim[window[0]:window[1]]
cam1_window = cam1[window[0]:window[1]]

fig, axs = plt.subplots(3,1, sharex=True, figsize=(10, 6))
fig.suptitle("ADC During Charges Extended")
axs[2].plot(grim_window['alt_agl_ft'])
axs[2].set_xlabel("Time (ms)")
axs[2].set_ylabel("Altitude (ft)")
axs[2].set_title("Grim Barometer Altitude")

axs[1].plot(grim_window['acc (m/s^2)'])
axs[1].set_xlabel("Time (ms)")
axs[1].set_ylabel("Acceleration (m/s^2)")
axs[1].set_title("Grim Accelerometer Magnitude")


axs[0].plot(grim_window['good force(kg)'], alpha = 0.2)
# axs[0].plot(grim_window['good force(kg)'].rolling(50, center=True).mean(),␣
 ↪color = 'C0')
axs[0].set_xlabel("Time (ms)")
axs[0].set_ylabel("Force (kg)")
axs[0].set_title("Load Cell Force")
fig.tight_layout()
```



Strange ADC Bump not really sure what this is

[1102]:
```
window = (37_150, 38_100)
grim_window = grim[window[0]:window[1]]
cam1_window = cam1[window[0]:window[1]]

imu_avg_size = 50
fig, axs = plt.subplots(3,1, sharex=True, figsize=(10, 6))
fig.suptitle("ADC During Strange Occurence")



axs[0].axhline(50, color='red', linestyle = ':')
axs[0].annotate("Maximum Load Cell Rating: 50kg", (37600 , 45), color = 'red')

axs[0].plot(grim_window['good force(kg)'])
axs[0].set_xlabel("Time (ms)")
axs[0].set_ylabel("Force (kg)")
axs[0].set_title("Load Cell Force")
fig.tight_layout()


axs[1].plot(grim_window['acc (m/s^2)'], alpha = 0.2, label = 'Raw')
axs[1].plot(grim_window['acc (m/s^2)'].rolling(imu_avg_size, center=True).
 ↪mean(), c = "C0", label = f'Avg of Size {imu_avg_size}')
axs[1].set_xlabel("Time (ms)")
axs[1].set_ylabel("Acceleration (m/s^2)")
axs[1].set_title("Grim Accelerometer Magnitude")
axs[1].legend()

axs[2].plot(cam1_window['a[m/s2]'], label = 'Raw (tho they probably filter␣
 ↪inside camera)')
axs[2].set_xlabel("Time (ms)")
axs[2].set_ylabel("Acceleration (m/s^2)")
axs[2].set_title("Grim Accelerometer Magnitude")

axs[0].annotate("huh?", (37_700, 20))

comma_ax(axs[0])
comma_ax(axs[1])
comma_ax(axs[2])
```



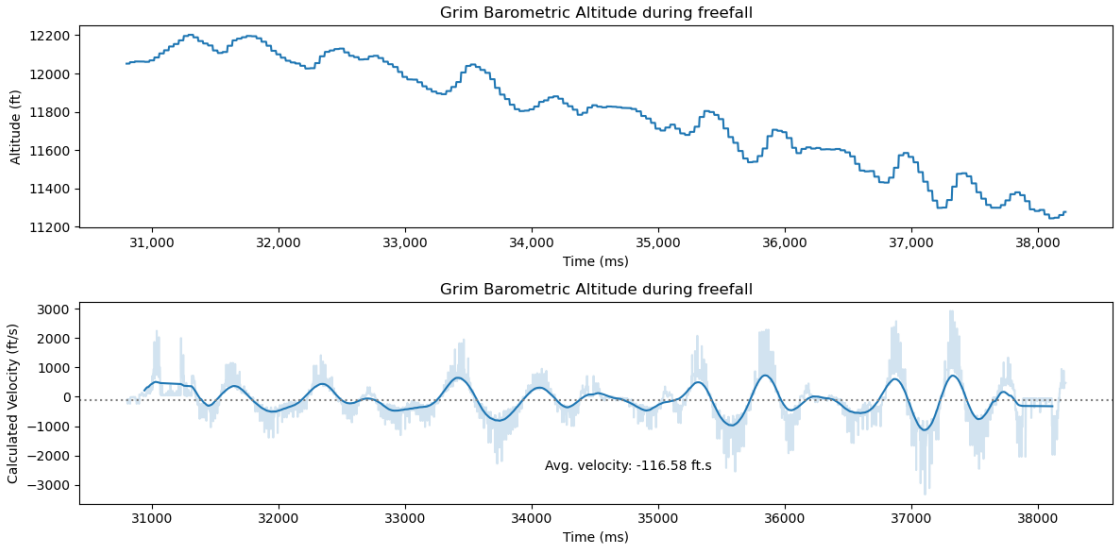
9.5 Freefall

[1054]:
```
display(Markdown(f"After the payload is ejected, there is a {parachute_catch_1␣
↪- charge2_span[1]} ms delay until the parachute catches where the payload is␣
↪in freefall"))
```

After the payload is ejected, there is a 7415 ms delay until the parachute catches where the payload is in freefall

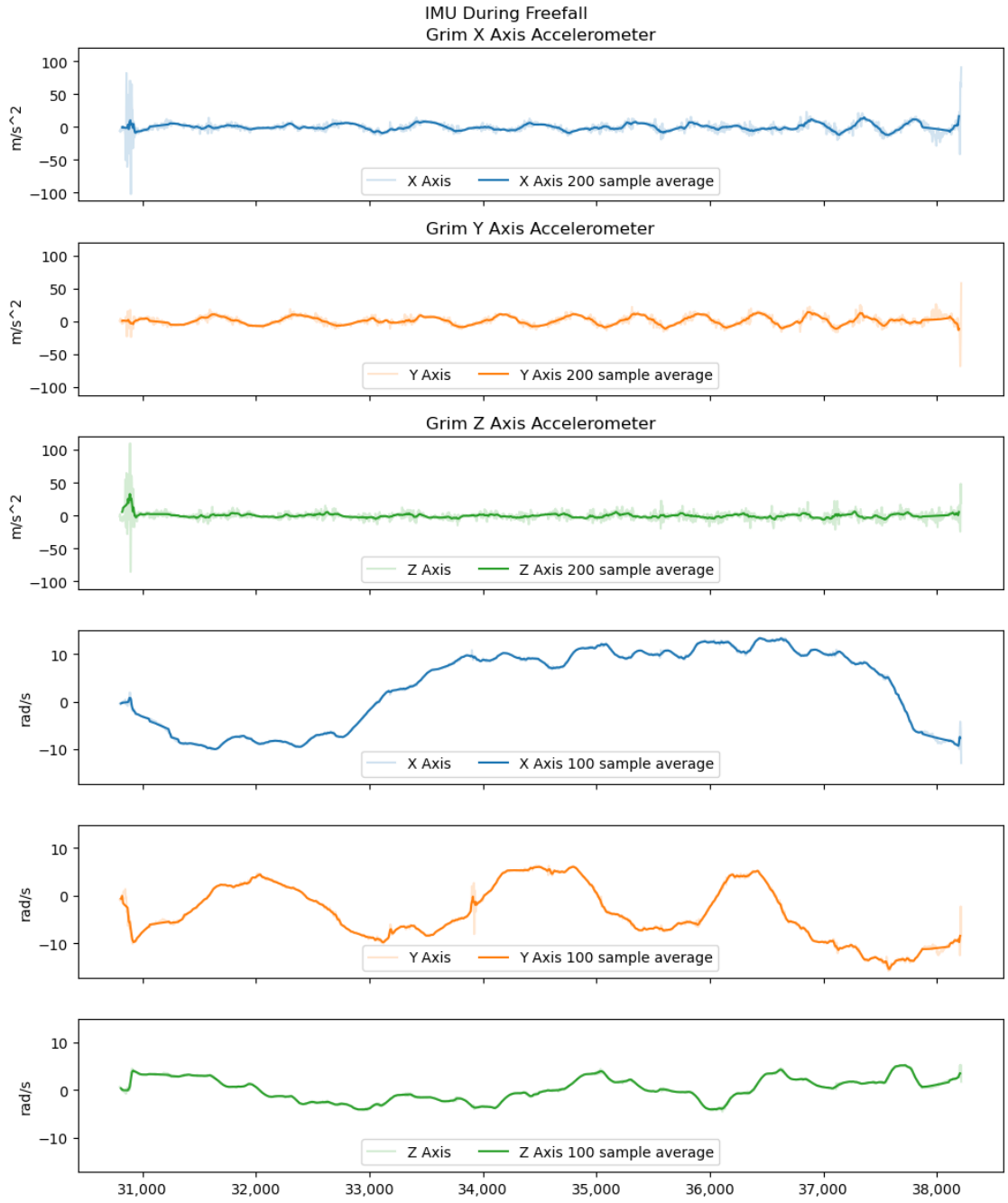
[1105]:
```
window = (charge2_span[1], parachute_catch_1)
grim_window = grim[window[0]:window[1]]
cam1_window = cam1[window[0]:window[1]]

fig, axs = plt.subplots(2, 1,figsize=(12,6))
# axs[0] = grim_window['alt_agl_ft_wout_charges'].plot(figsize=(12, 6))
axs[0].plot(grim_window.index, grim_window['alt_agl_ft_wout_charges'])
axs[0].set_title("Grim Barometric Altitude during freefall")
axs[0].set_xlabel("Time (ms)")
axs[0].set_ylabel("Altitude (ft)")

window_size = 1000
axs[1].plot(grim_window.index, grim_window['alt_vel_est'], c = 'C0', alpha = 0.
↪2, label = "Raw")
axs[1].plot(grim_window.index, grim_window['alt_vel_est'].rolling(window_size,␣
↪center=True).mean(), c = 'C0', label = f"{window_size} Avg")
```

```
axs[1].set_title("Grim Barometric Altitude during freefall")
axs[1].set_xlabel("Time (ms)")
axs[1].set_ylabel("Calculated Velocity (ft/s)")
freefall_vel_avg = grim_window['alt_vel_est'].mean()
axs[1].axhline(freefall_vel_avg, c='gray', ls=":")
axs[1].annotate(f"Avg. velocity: {freefall_vel_avg:,.2f} ft.s", (34_100, -2500))

fig.tight_layout()
comma_ax(axs[0])
```



[1107]:
```
window = (charge2_span[1], parachute_catch_1)
grim_window = grim[window[0]:window[1]]
cam1_window = cam1[window[0]:window[1]]

rolling_size = 200
fig, axs = plt.subplots(6, 1, sharex=True, figsize=(10, 12))
fig.suptitle("IMU During Freefall")
axs[0].plot(grim_window['accx (m/s^2)'], alpha = 0.2, c = 'C0', label = 'X␣
 ↪Axis')
axs[0].plot(grim_window['accx (m/s^2)'].rolling(rolling_size, center=True).
 ↪mean(), c= 'C0', label = f"X Axis {rolling_size} sample average")
axs[0].legend(loc = 8, ncol = 2)
axs[0].set_title("Grim X Axis Accelerometer")
axs[0].set_ylabel("m/s^2")

axs[1].plot(grim_window['accy (m/s^2)'], alpha = 0.2, c = 'C1', label = 'Y␣
 ↪Axis')
```

```
axs[1].plot(grim_window['accy (m/s^2)'].rolling(rolling_size, center=True).
 ↪mean(), c= 'C1', label = f"Y Axis {rolling_size} sample average")
axs[1].legend(loc = 8, ncol = 2)
axs[1].set_title("Grim Y Axis Accelerometer")
axs[1].sharey(axs[0])
axs[1].set_ylabel("m/s^2")


axs[2].plot(grim_window['accz (m/s^2)'], alpha = 0.2, c = 'C2', label = 'Z␣
 ↪Axis')
axs[2].plot(grim_window['accz (m/s^2)'].rolling(rolling_size, center=True).
 ↪mean(), c= 'C2', label = f"Z Axis {rolling_size} sample average")
axs[2].legend(loc = 8, ncol = 2)
axs[2].set_title("Grim Z Axis Accelerometer")
axs[2].sharey(axs[0])
axs[2].set_ylabel("m/s^2")


rolling_size = 100
axs[3].plot(grim_window['gyrox (rad/s)'], alpha = 0.2, c = 'C0', label = 'X␣
 ↪Axis')
axs[3].plot(grim_window['gyrox (rad/s)'].rolling(rolling_size, center=True).
 ↪mean(), c= 'C0', label = f"X Axis {rolling_size} sample average")
axs[3].legend(loc = 8, ncol = 2)
axs[3].set_ylabel("rad/s")

axs[4].plot(grim_window['gyroy (rad/s)'], alpha = 0.2, c = 'C1', label = 'Y␣
 ↪Axis')
axs[4].plot(grim_window['gyroy (rad/s)'].rolling(rolling_size, center=True).
 ↪mean(), c= 'C1', label = f"Y Axis {rolling_size} sample average")
axs[4].legend(loc = 8, ncol = 2)
axs[4].sharey(axs[3])
axs[4].set_ylabel("rad/s")

axs[5].plot(grim_window['gyroz (rad/s)'], alpha = 0.2, c = 'C2', label = 'Z␣
 ↪Axis')
axs[5].plot(grim_window['gyroz (rad/s)'].rolling(rolling_size, center=True).
 ↪mean(), c= 'C2', label = f"Z Axis {rolling_size} sample average")
axs[5].legend(loc = 8, ncol = 2)
axs[5].sharey(axs[3])
axs[5].set_ylabel("rad/s")



fig.tight_layout()
comma_ax(axs[0])
```

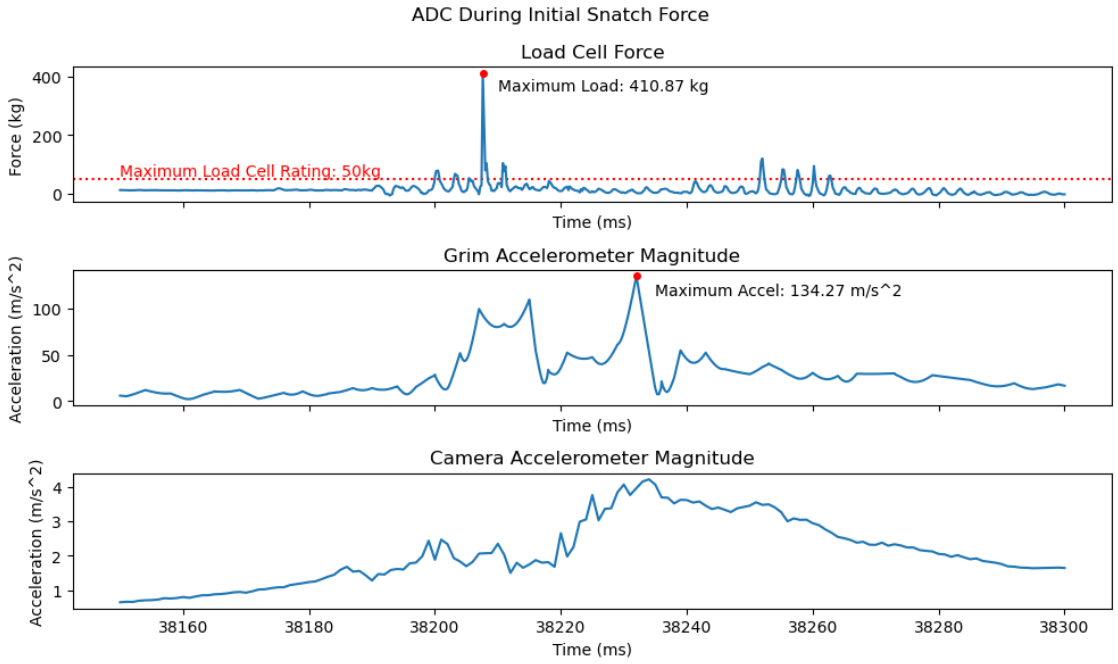


9.6 Parachute Deploy

Load Cell During Initial Snatch Force

[1108]:
```
window = (38_150, 38_300)
grim_window = grim[window[0]:window[1]]
cam1_window = cam1[window[0]:window[1]]
```

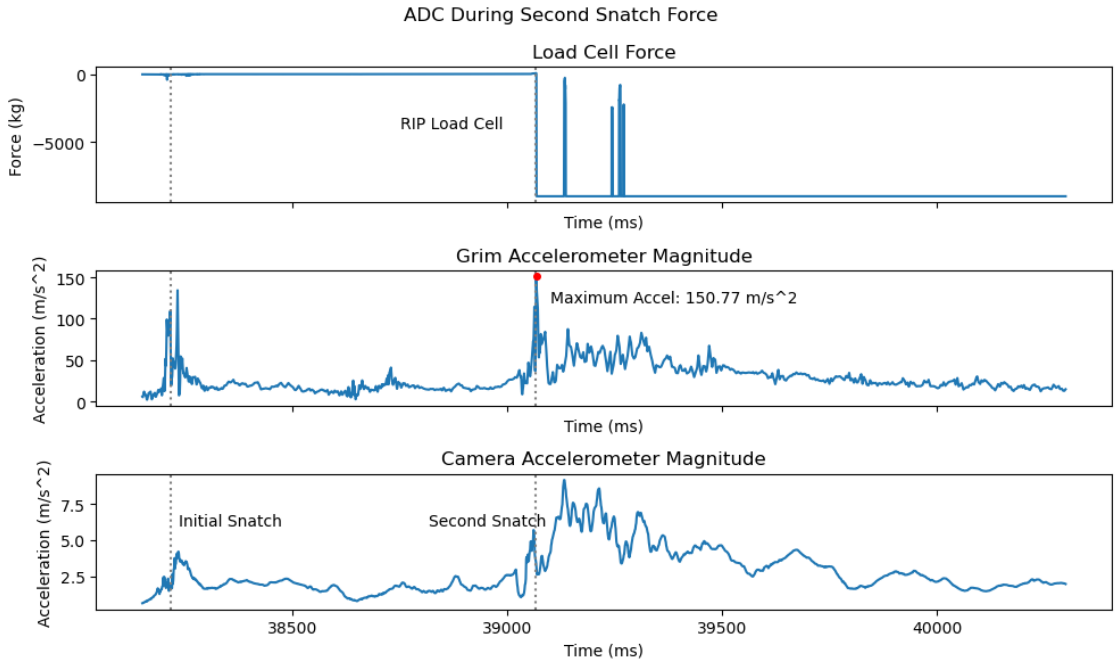
```
fig, axs = plt.subplots(3,1, sharex=True, figsize=(10, 6))
fig.suptitle("ADC During Initial Snatch Force")

axs[0].axhline(50, color='red', linestyle = ':')
axs[0].annotate("Maximum Load Cell Rating: 50kg", (38150 , 60), color = 'red')
axs[0].plot(grim_window['good force(kg)'])
axs[0].set_xlabel("Time (ms)")
axs[0].set_ylabel("Force (kg)")
axs[0].set_title("Load Cell Force")

axs[1].plot(grim_window['acc (m/s^2)'])
axs[1].set_xlabel("Time (ms)")
axs[1].set_ylabel("Acceleration (m/s^2)")
axs[1].set_title("Grim Accelerometer Magnitude")

axs[2].plot(cam1_window['a[m/s2]'])
axs[2].set_xlabel("Time (ms)")
axs[2].set_ylabel("Acceleration (m/s^2)")
axs[2].set_title("Camera Accelerometer Magnitude")

max_load1, max_load1_loc = grim_window['good force(kg)'].max(),␣
 ↪grim_window['good force(kg)'].idxmax()
max_acc1, max_acc1_loc = grim_window['acc (m/s^2)'].max(), grim_window['acc (m/
 ↪s^2)'].idxmax()

max_ratio = max_load1 / max_acc1

axs[0].annotate(f"Maximum Load: {max_load1:.2f} kg", (38_210, 350))
axs[1].annotate(f"Maximum Accel: {max_acc1:.2f} m/s^2", (38_235, 115))

axs[0].plot(max_load1_loc, max_load1, marker ='o', markersize=4, c = 'red')
axs[1].plot(max_acc1_loc, max_acc1, marker ='o', markersize=4, c = 'red')

fig.tight_layout()
```



Load Cell During Second Snatch Force

[1112]:
```
window = (38_150, 40_300)
grim_window = grim[window[0]:window[1]]
cam1_window = cam1[window[0]:window[1]]

fig, axs = plt.subplots(3,1, sharex=True, figsize=(10, 6))
fig.suptitle("ADC During Second Snatch Force")

axs[0].plot(grim_window['reading (LSB)'].map(reading_to_force) / 9.81)
axs[0].set_xlabel("Time (ms)")
axs[0].set_ylabel("Force (kg)")
axs[0].set_title("Load Cell Force")

axs[1].plot(grim_window['acc (m/s^2)'])
axs[1].set_xlabel("Time (ms)")
axs[1].set_ylabel("Acceleration (m/s^2)")
axs[1].set_title("Grim Accelerometer Magnitude")

axs[2].plot(cam1_window['a[m/s2]'])
axs[2].set_xlabel("Time (ms)")
axs[2].set_ylabel("Acceleration (m/s^2)")
axs[2].set_title("Camera Accelerometer Magnitude")

axs[0].axvline(parachute_catch_2, c = "gray", ls = ':')
axs[1].axvline(parachute_catch_2, c = "gray", ls = ':')
```

```
axs[2].axvline(parachute_catch_2, c = "gray", ls = ':')

axs[0].axvline(parachute_catch_1, c = "gray", ls = ':')
axs[1].axvline(parachute_catch_1, c = "gray", ls = ':')
axs[2].axvline(parachute_catch_1, c = "gray", ls = ':')

axs[0].annotate("RIP Load Cell", (38750, -4000))

axs[2].annotate("Initial Snatch", (parachute_catch_1+20, 6))
axs[2].annotate("Second Snatch", (parachute_catch_2-250, 6))

max_acc2, max_acc2_loc = grim_window['acc (m/s^2)'].max(), grim_window['acc (m/
↪s^2)'].idxmax()
axs[1].plot(max_acc2_loc, max_acc2, marker ='o', markersize=4, c = 'red')
axs[1].annotate(f"Maximum Accel: {max_acc2:.2f} m/s^2", (39100, 120))

fig.tight_layout()
```



9.6.1 Flawed Comparisons

If we assume, that the max load felt by the load cell and the max acceleration of the payload measured by the IMU during a snatch event (Big If), we can make a guess at what the actual load that would have been felt had the load cell not died.

$$\frac{\text{load 1}}{\text{accel 1}} = \frac{\text{load 2}}{\text{accel 2}}$$

$$\text{load 1} \times \frac{\text{accel 2}}{\text{accel 1}} = \text{load 2}$$

[1113]:
```
display(Markdown(f"Second Snatch Load Estimate = {max_load1 * max_acc2 /␣
↪max_acc1:.2f} kg"))
```

Second Snatch Load Estimate = 461.37 kg

9.7 Descent

[1116]:
```
window = (50_000, 250_000)
grim_window=grim[window[0]:window[1]]
rrc3_1_window = rrc3_1[window[0]:window[1]]
rrc3_2_window = rrc3_2[window[0]:window[1]]
```

[1117]:
```
def get_slope(data):
    rise = data.iloc[0] - data.iloc[-1]
    run = data.iloc[0].name - data.iloc[-1].name
    return rise / run
```

[1118]:
```
grim_slope_ft_s =␣
↪get_slope(grim_window[['alt_agl_ft_wout_charges']])['alt_agl_ft_wout_charges']␣
↪* 1000.0
rrc31_slope_ft_s = get_slope(rrc3_1_window[['Altitude [ft]']])['Altitude [ft]']␣
↪* 1000.0
rrc32_slope_ft_s = get_slope(rrc3_2_window[['Altitude [ft]']])['Altitude [ft]']␣
↪* 1000.0
```

The RRC3 Pressure is very noticably weird here.

[1119]:
```
fig, axs = plt.subplots(3, 1, figsize = (10, 8), sharex=True)

fig.suptitle("Descent")

axs[0].plot(grim_window.index, grim_window['alt_agl_ft_wout_charges'])
axs[0].set_title("Grim Barometric Altitude")
axs[1].plot(rrc3_1_window.index, rrc3_1_window['Altitude [ft]'])
axs[1].set_title("RRC3 1 Barometric Altitude")
axs[2].plot(rrc3_2_window.index, rrc3_2_window['Altitude [ft]'])
axs[2].set_title("RRC3 2 Barometric Altitude")

axs[0].annotate(f"Average Descent Rate: {grim_slope_ft_s:.2f} ft/s", (170_000,␣
↪8000))
axs[1].annotate(f"Average Descent Rate: {rrc31_slope_ft_s:.2f} ft/s", (170_000,␣
↪8000))
axs[2].annotate(f"Average Descent Rate: {rrc32_slope_ft_s:.2f} ft/s", (170_000,␣
↪8000))
```

```
for ax in axs:
    ax.set_xlabel("Time (ms)")
    ax.set_ylabel("Altitude (ft)")

fig.tight_layout()
```