Imports

In [2]:
```
%load_ext autoreload
%autoreload 2

#import viewdat_cno_lib as vdl
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from collections import namedtuple
import geopy.distance
import datetime
import time
import os
import subprocess
import math
import glob
import shutil
```

Functions

In [3]:
```
def combine_T04(hourly_dir, out_name):
    if os.path.isfile(out_name):
        print(f'{out_name} already exists.. deleting it')
        os.remove(out_name)

    #print(f"   combine({fname_base}, {out_name})")
    # List of files to concat
    flist = sorted(glob.glob('*T04', root_dir=os.path.join(hourly_dir)))

    with open(out_name, 'wb') as wfd:
        for f in flist:
            f_path = os.path.join(hourly_dir, f)
            print(f"    appending {f} to {out_name}")
            with open(f_path, 'rb') as fd:
                shutil.copyfileobj(fd, wfd)
```

In [4]:
```
def pos_read(fname_tsv, header_n_rows=159):
    if not os.path.isfile(fname_tsv):
        print(f"Unable to find {fname_tsv}")
        return None
    # Import the data into a Pandas DataFrame, and do some cleanup
    df = pd.read_csv(fname_tsv, delimiter='\t', header=header_n_rows, na_
    df.rename(columns=lambda x: x.strip(), inplace=True)  # drop whitespa
    df = df.replace({'Nan': np.nan})
    #print(list(df.columns))
    cols_rename = {
        '%Week': 'Week',
        # 'TRACK': 'Track',
    }
    df.rename(columns=cols_rename, inplace=True)
    #df = week_rollover_unwrap(df)
    return df
```

In [5]:
```
# This class was brazenly stolen from mutils.py in the Sunnyvale CVS repo
class OrbitConst(object):
    """
    Orbital constants
    """
    PI = 3.14159265358979323846
    A_WGS84 = 6378137.0
    B_WGS84 = 6356752.314245179
    E2_WGS84 = 6.69437999013e-3
    ONE_MIN_E2 = 0.99330562000987
    SQRT_ONE_MIN_E2 = 9.96647189335258e-1


# This function was brazenly stolen from mutils.py in the Sunnyvale CVS r
def llh2enu(llh, ref_llh, is_rad=False, is_ref_rad=False):
    """
    Convert lat/lon/height to delta east/north/up.
    llh = array of lat/lon/height [lat/lon in degrees by default]
    ref_llh = point or array of lat/lon/height [lat/lon in degrees by def
    is_rad = True -> "llh" lat/lon is radians, else in degrees
    is_ref_rad = True -> "ref_llh" lat/lon is radians, else in degrees
    """
    # Make sure inputs are arrays
    if len(np.shape(llh)) == 1:
        llh = llh.reshape((1, len(llh)))
    if len(np.shape(ref_llh)) == 1:
        ref_llh = ref_llh.reshape((1, len(ref_llh)))

    # Convert to radians?
    scale1 = np.ones(np.shape(llh))
    scale2 = np.ones(np.shape(ref_llh))
    ref_lat = np.copy(ref_llh[:, 0])
    if not is_rad:
        scale1[:, 0:2] *= OrbitConst.PI/180
    if not is_ref_rad:
        scale2[:, 0:2] *= OrbitConst.PI/180
        ref_lat *= OrbitConst.PI/180

    # Compute the residuals
    dllh = llh*scale1 - ref_llh*scale2

    # Compute Radii of Curvature
    W = np.sqrt(1 - OrbitConst.E2_WGS84 * np.sin(ref_lat)**2)
    N = OrbitConst.A_WGS84 / W
    M = OrbitConst.A_WGS84 * (1 - OrbitConst.E2_WGS84) / W**3

    # Compute Metric Components
    dE = dllh[:, 1] * N * np.cos(ref_lat)
    dN = dllh[:, 0] * M
    dU = dllh[:, 2]

    return (dE, dN, dU)
```

In [6]:
```
def calc_heading(lat1, long1, lat2, long2):
    rlat1 = math.radians(lat1)
    rlat2 = math.radians(lat2)
    rlon1 = math.radians(long1)
    rlon2 = math.radians(long2)
    dLon = rlon2 - rlon1
```

```
    x = math.cos(rlat2) * math.sin(dLon)
    y = math.cos(rlat1) * math.sin(rlat2)
    y -= math.sin(rlat1) * math.cos(rlat2) * math.cos(dLon)
    heading = np.arctan2(x, y)
    heading = math.degrees(heading)
    heading = (heading + 180.0) % 360.0   # convert from -180:+180 to 0:3
    return heading
```

In [7]:
```
from datetime import datetime, timedelta
import pytz

def gps_datetime(time_week, time_ms, leap_seconds=37):
    # 37 leap seconds seems to be correct for 2016-12-31 through at least
    gps_epoch = datetime(1980, 1, 6, tzinfo=pytz.utc)
    # gps_time - utc_time = leap_seconds
    return gps_epoch + timedelta(weeks=time_week, milliseconds=time_ms, s

print(gps_datetime(2292, 266792379))
print(gps_datetime(2319, 500177.0*1000))
```

```
2023-12-13 02:05:55.379000+00:00
2024-06-21 18:55:40+00:00
```

In [8]:
```
def delta_enu_start(df):
    df[['dN', 'dE', 'dH']] = df[['N', 'E', 'Ele']] - df.iloc[0][['N', 'E'
    return df
```

In [9]:
```
def delta_pos_start(df):
    start = df.iloc[0]
    start_pt = np.array([start.MLat, start.MLon, start.MHgt])
    #print(f'start_pt: {start_pt}')
    llh = np.array(df[['MLat', 'MLon', 'MHgt']])
    #print('llh:')
    #print(llh)
    a = np.array(llh2enu(llh, start_pt))
    #print('a:')
    #display(a)
    #print(a.shape)
    #print(df.shape)
    df[['dE', 'dN', 'dU']] = a.transpose()
    return df
```

In [10]:
```
def dms_to_dd(d:float, m:float, s:float):
    dd = d + float(m)/60 + float(s)/3600
    if d<0:
        dd = float(d) - float(m)/60 - float(s)/3600
    else:
        dd = float(d) + float(m)/60 + float(s)/3600
    return dd
```

In [11]:
```
def import_t04_dir(dir_name, fname_base):
    fname_T04 = f'{fname_base}.T04'
    fname_tsv = f'{fname_base}_pos.tsv'

    combine_T04(dir_name, fname_T04)
    cmd = f'viewdat -d35:2 --t04_vector_position -mb -h {fname_T04} -o{fn
    print(f'{cmd=}')
    subprocess.run(cmd)
    df = pos_read(fname_tsv, 159) #102)
```

```
    df = df[['Week', 'Time', 'RefSys',
             'MFixMode', 'MFixType', 'MFixInfo',
             'MLat', 'MLon', 'MHgt', 'MSigN', 'MSigE', 'MSigU', 'MSigEN',
             'RefLat', 'RefLon', 'RefHgt',
             'VLat', 'VLon', 'VHgt', 'VSigN', 'VSigE', 'VSigU', 'VSigEN',
             'corrAge']]
    df = delta_pos_start(df)
    df['time_utc'] = df.apply(lambda x: gps_datetime(time_week = x['Week'
    return df
```

Viewdat (Judo, R780)

In [12]:
```
df_judo = import_t04_dir('Judo_EB93', 'judo')
#display(df_judo)
```

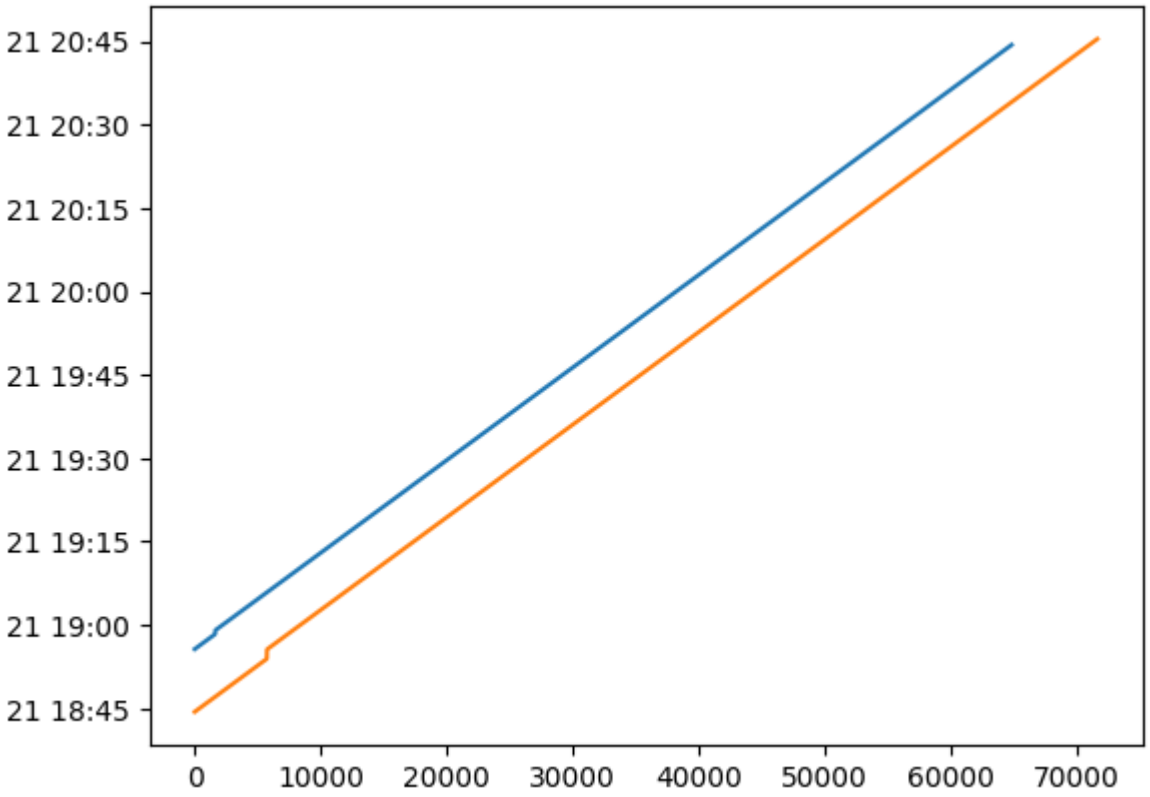
```
judo.T04 already exists.. deleting it
    appending EB1-5093__202406211845.T04 to judo.T04
    appending EB1-5093__202406211900.T04 to judo.T04
    appending EB1-5093__202406211915.T04 to judo.T04
    appending EB1-5093__202406211930.T04 to judo.T04
    appending EB1-5093__202406211945.T04 to judo.T04
    appending EB1-5093__202406212000.T04 to judo.T04
    appending EB1-5093__202406212015.T04 to judo.T04
    appending EB1-5093__202406212030.T04 to judo.T04
cmd=['viewdat', '-d35:2', '--t04_vector_position', '-mb', '-h', 'judo.T04'
, '-ojudo_pos.tsv']
```

In [13]:
```
df_r780= import_t04_dir('R780', 'r780')
#display(df_r780)
```

```
r780.T04 already exists.. deleting it
    appending Trimble___202406211845.T04 to r780.T04
    appending Trimble___202406211900.T04 to r780.T04
    appending Trimble___202406211915.T04 to r780.T04
    appending Trimble___202406211930.T04 to r780.T04
    appending Trimble___202406211945.T04 to r780.T04
    appending Trimble___202406212000.T04 to r780.T04
    appending Trimble___202406212015.T04 to r780.T04
    appending Trimble___202406212030.T04 to r780.T04
    appending Trimble___202406212045.T04 to r780.T04
cmd=['viewdat', '-d35:2', '--t04_vector_position', '-mb', '-h', 'r780.T04'
, '-or780_pos.tsv']
```

In [14]:
```
judo_N = df_judo.shape[0] -1
print('Judo UTC:', df_judo.time_utc[0],  df_judo.time_utc[judo_N]) #, df_
r780_N = df_r780.shape[0] -1
print('R780 UTC:', df_r780.time_utc[0],  df_r780.time_utc[r780_N]) #, df_
plt.plot(df_judo.time_utc)
plt.plot(df_r780.time_utc)
print(df_judo.shape[0])
#print(df_judo.time_utc[1800:1805])
#print(df_72310017.t[0:5])
```

```
Judo UTC: 2024-06-21 18:55:40+00:00 2024-06-21 20:44:22.500000+00:00
R780 UTC: 2024-06-21 18:44:23.100000+00:00 2024-06-21 20:45:25.800000+00:0
0
64829
```



ATS

In [15]:
```
count = 0
header = None
mylist =[]
with open("ATSTest_LogFile JUDO IN LINE and PERP.txt") as file:
    for line in file:
        if line.startswith('ATSDataEvent'):
            line = line.strip()
            #print(line)
            if header is None:
                header = line.split("\t")
                #print(header)
                #print("")
            else:
                d = dict(zip(header, line.split("\t")))
                mylist.append(d)
                #if count < 3:
                #    print(line.split("\t"))
                #    print("")

            count += 1
            #if count > 10:
            #    break

df_ats = pd.DataFrame(mylist)
df_ats = df_ats.drop(columns=['ATSDataEvent'])

df_ats = df_ats.astype({
    'SerialNumber': 'int',
    'N': 'float',
    'E': 'float',
```

```
    'Ele': 'float',
})
df_ats['t'] = pd.to_datetime(df_ats['Date'] + ' ' + df_ats['PCClock'], ut
df_ats.t = df_ats.t + pd.Timedelta('06:00:00')
display(df_ats[['Date', 'Time', 'PCClock', 't']])
#display(df_ats.info())

#
# Split out the two MT Targets
#
df_72613087 = df_ats.query("SerialNumber==72613087")[['t', 'Date', 'Time'
df_72613087 = df_72613087[df_72613087.N != 0]

df_72310017 = df_ats.query("SerialNumber==72310017")[['t', 'Date', 'Time'
df_72310017 = df_72310017[df_72310017.N != 0]

df_72310017 = delta_enu_start(df_72310017)
df_72613087 = delta_enu_start(df_72613087)

display(df_72310017)
```

	Date	Time	PCClock	t
0	6/21/2024	13:04:03.638	13:04:03.545	2024-06-21 19:04:03.545000+00:00
1	6/21/2024	13:04:03.639	13:04:03.550	2024-06-21 19:04:03.550000+00:00
2	6/21/2024	13:04:03.640	13:04:03.591	2024-06-21 19:04:03.591000+00:00
3	6/21/2024	13:04:03.641	13:04:03.598	2024-06-21 19:04:03.598000+00:00
4	6/21/2024	13:04:03.722	13:04:03.644	2024-06-21 19:04:03.644000+00:00
...	...	...	...	...
231136	6/21/2024	14:42:22.431	14:42:22.402	2024-06-21 20:42:22.402000+00:00
231137	6/21/2024	14:42:22.516	14:42:22.451	2024-06-21 20:42:22.451000+00:00
231138	6/21/2024	14:42:22.517	14:42:22.503	2024-06-21 20:42:22.503000+00:00
231139	6/21/2024	14:42:22.593	14:42:22.553	2024-06-21 20:42:22.553000+00:00
231140	6/21/2024	14:42:24.442	14:42:22.602	2024-06-21 20:42:22.602000+00:00

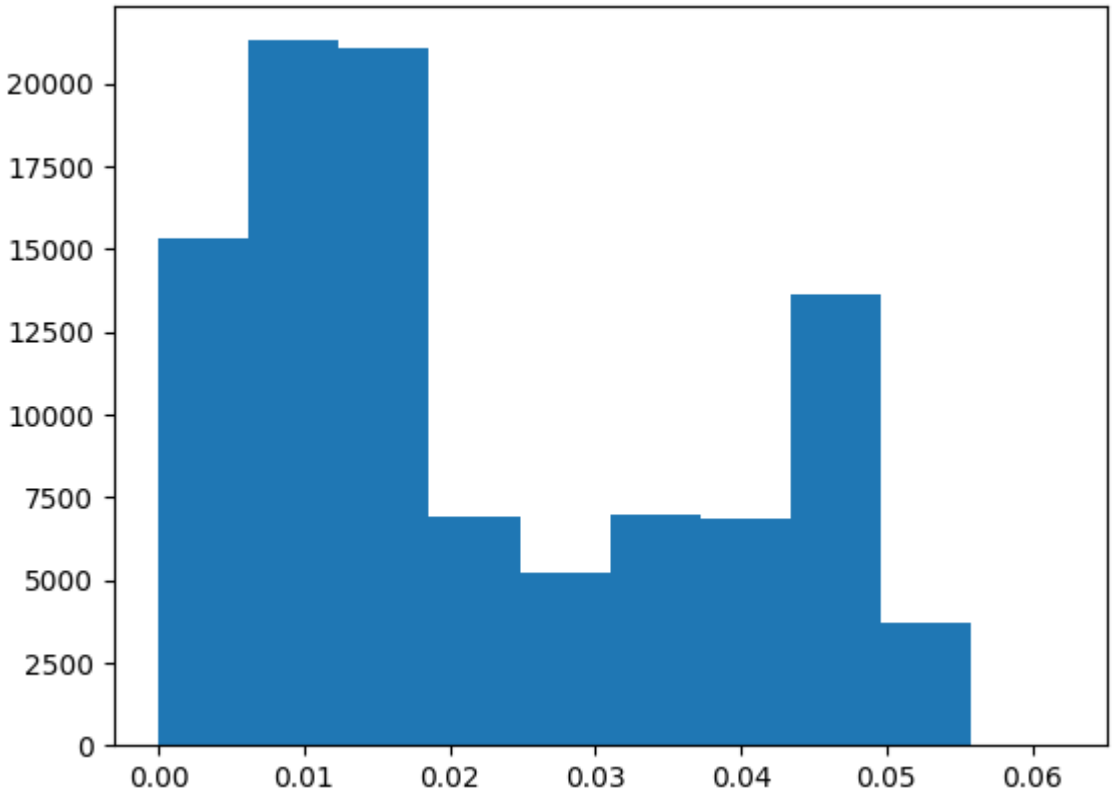
231141 rows × 4 columns

	t	Date	Time	SerialNumber	PCTics	P
1	2024-06-21 19:04:03.550000+00:00	6/21/2024	13:04:03.639	72310017	607240031	13:04:
3	2024-06-21 19:04:03.598000+00:00	6/21/2024	13:04:03.641	72310017	607240078	13:04:
5	2024-06-21 19:04:03.650000+00:00	6/21/2024	13:04:03.723	72310017	607240125	13:04:
7	2024-06-21 19:04:03.698000+00:00	6/21/2024	13:04:03.724	72310017	607240171	13:04:
9	2024-06-21 19:04:03.749000+00:00	6/21/2024	13:04:03.801	72310017	607240218	13:04:
...	...	...	...	...	...	
231083	2024-06-21 20:42:19.985000+00:00	6/21/2024	14:42:20.015	72310017	613136421	14:42:
231085	2024-06-21 20:42:20.037000+00:00	6/21/2024	14:42:20.090	72310017	613136484	14:42:
231087	2024-06-21 20:42:20.089000+00:00	6/21/2024	14:42:20.094	72310017	613136531	14:42:
231089	2024-06-21 20:42:20.136000+00:00	6/21/2024	14:42:22.031	72310017	613136578	14:42:
231091	2024-06-21 20:42:20.186000+00:00	6/21/2024	14:42:22.033	72310017	613136625	14:42:

97479 rows × 13 columns

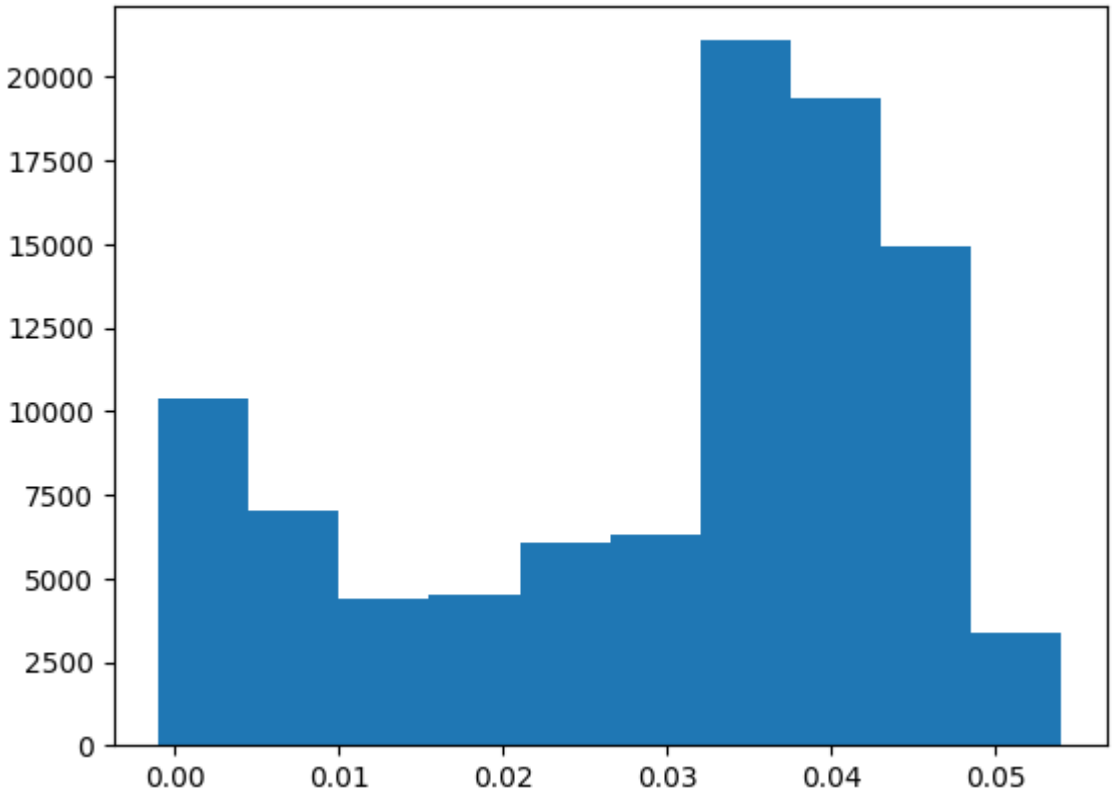
In [16]:
```
df_72613087['delta'] = df_ats.t.diff().dt.total_seconds()
#display(df_72613087)
#print("")
#print(df_72613087.delta.mean())
plt.hist(df_72613087.delta)
```

Out[16]:
```
(array([1.5303e+04, 2.1294e+04, 2.1080e+04, 6.9290e+03, 5.2010e+03,
        6.9640e+03, 6.8390e+03, 1.3614e+04, 3.7250e+03, 6.0000e+00]),
 array([0.    , 0.0062, 0.0124, 0.0186, 0.0248, 0.031 , 0.0372, 0.0434,
        0.0496, 0.0558, 0.062 ]),
 <BarContainer object of 10 artists>)
```



In [17]:
```
df_72310017['delta'] = df_ats.t.diff().dt.total_seconds()
#df_72310017.info()
#print(df_72310017.N.min())
#print(df_72310017.delta.mean())
plt.hist(df_72310017.delta)
#display(df_72310017)
```

Out[17]:
```
(array([10403.,  7014.,  4372.,  4523.,  6069.,  6294., 21091., 19389.,
        14949.,  3375.]),
 array([-0.001 ,  0.0045,  0.01  ,  0.0155,  0.021 ,  0.0265,  0.032 ,
         0.0375,  0.043 ,  0.0485,  0.054 ]),
 <BarContainer object of 10 artists>)
```



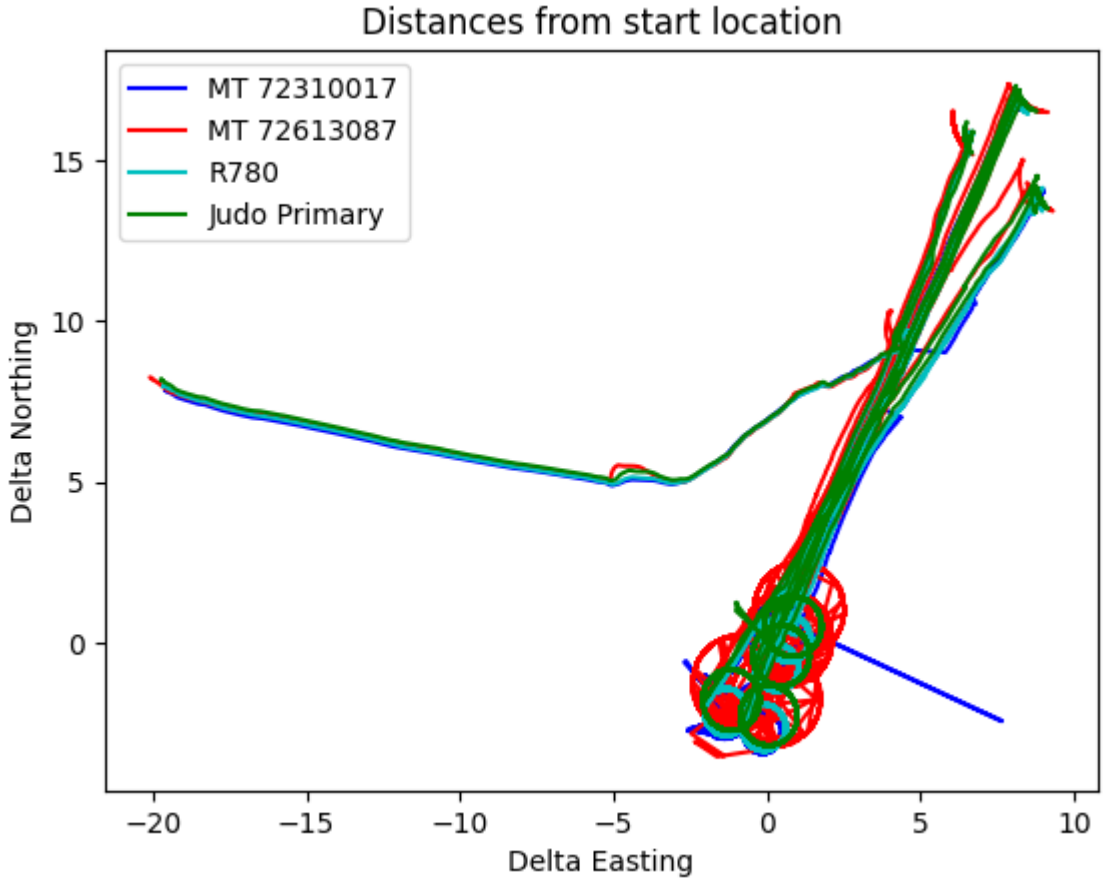
sn_list = df_ats.SerialNumber.unique() dfs = {} for sn in sn_list: q = f"SerialNumber=={sn}" print(q) dfq = df_ats.query(q) dfq['delta'] = dfq.t.diff().dt.total_seconds() dfs[sn] = dfq #plt.plot(df.PCClock, df.N) print(sn, dfq.size, dfq.delta.mean())

In [ ]:

Combined

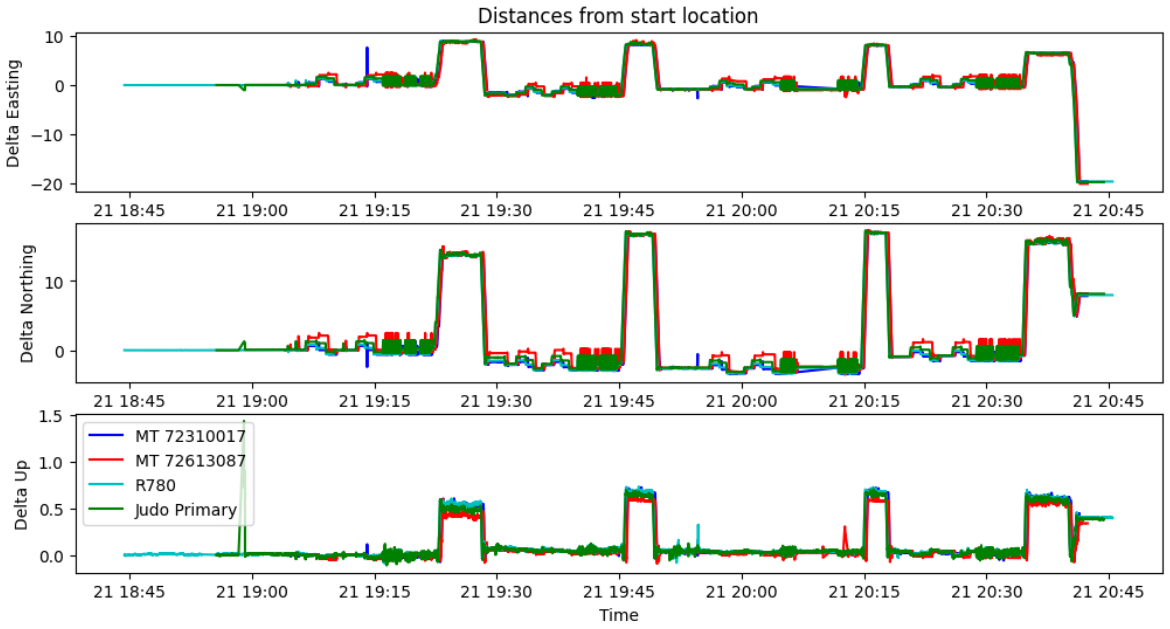
In [18]:
```
plt.plot(df_72310017.dE, df_72310017.dN, 'b-')
plt.plot(df_72613087.dE, df_72613087.dN, 'r-')
plt.plot(df_r780.dE, df_r780.dN, 'c-')
plt.plot(df_judo.dE, df_judo.dN, 'g-')

plt.xlabel('Delta Easting')
plt.ylabel('Delta Northing')
plt.title('Distances from start location')
plt.legend(['MT 72310017', 'MT 72613087', 'R780', 'Judo Primary'])
```

Out[18]: <matplotlib.legend.Legend at 0x7d103762a470>



In [23]:
```
fig, (ax0, ax1, ax2) = plt.subplots(3, 1, figsize=(12, 6))

ax0.plot(df_72310017.t, df_72310017.dE, 'b-')
ax0.plot(df_72613087.t, df_72613087.dE, 'r-')
ax0.plot(df_r780.time_utc, df_r780.dE, 'c-')
ax0.plot(df_judo.time_utc, df_judo.dE, 'g-')
ax0.set_title('Distances from start location')
ax0.set_ylabel('Delta Easting')

ax1.plot(df_72310017.t, df_72310017.dN, 'b-')
ax1.plot(df_72613087.t, df_72613087.dN, 'r-')
ax1.plot(df_r780.time_utc, df_r780.dN, 'c-')
ax1.plot(df_judo.time_utc, df_judo.dN, 'g-')
ax1.set_ylabel('Delta Northing')

ax2.plot(df_72310017.t, df_72310017.dH, 'b-')
ax2.plot(df_72613087.t, df_72613087.dH, 'r-')
ax2.plot(df_r780.time_utc, df_r780.dU, 'c-')
ax2.plot(df_judo.time_utc, df_judo.dU, 'g-')
ax2.set_ylabel('Delta Up')

plt.xlabel('Time')

plt.legend(['MT 72310017', 'MT 72613087', 'R780', 'Judo Primary'])
```

Out[23]: `<matplotlib.legend.Legend at 0x7d10376a6710>`


Distances from start location
Delta Easting
Delta Northing
Delta Up
MT 72310017
MT 72613087
R780
Judo Primary
21 18:45
21 19:00
21 19:15
21 19:30
21 19:45
21 20:00
21 20:15
21 20:30
21 20:45
Time

In [ ]: