In [ ]:

```python
station_name = "Barbados"
test_year = 2020
```

In [ ]:

```python
from climatereconstructionai import evaluate

evaluate(f"test_args_{station_name.lower()}.txt")
```

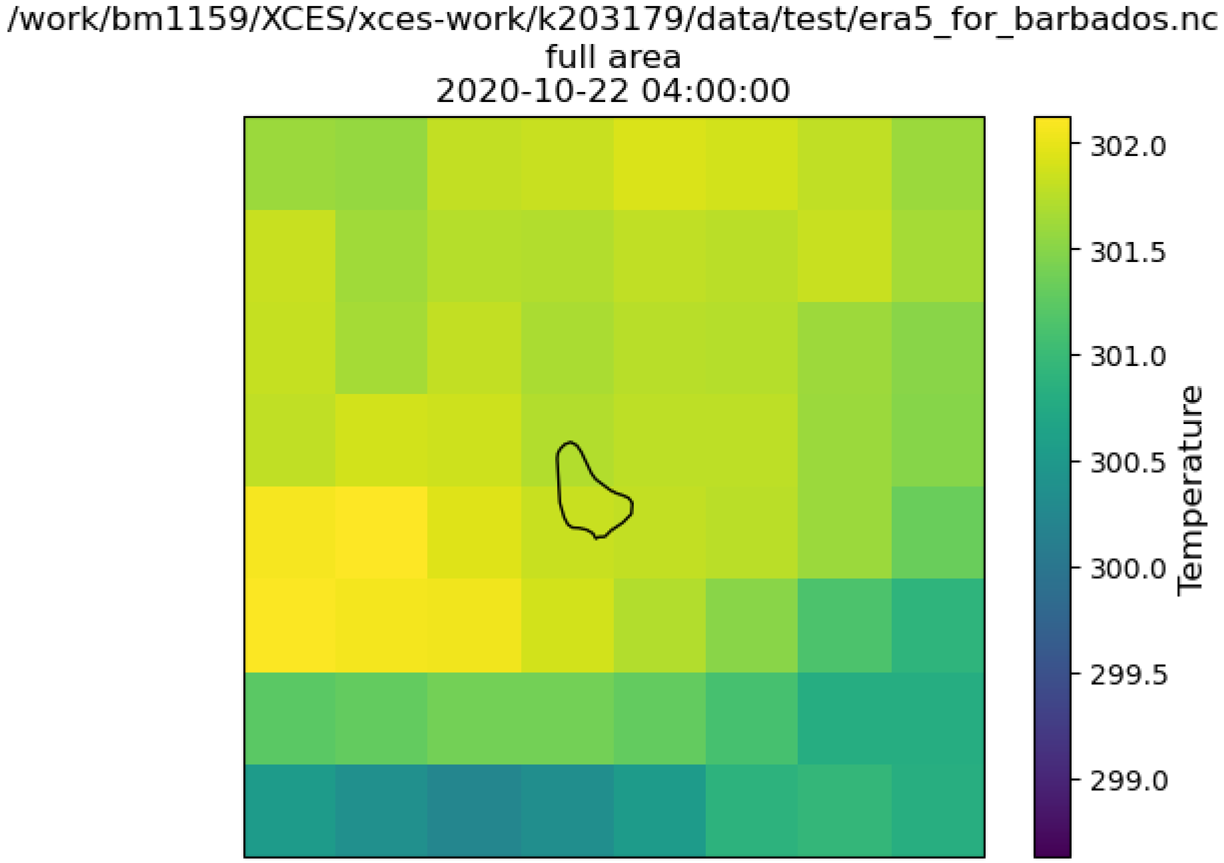
```
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/climatereconstructionai/utils/normalizer.py:10: RuntimeWarning: Mean of empty slice
  img_mean.append(np.nanmean(np.array(img_data[i])))
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/numpy/lib/nanfunctions.py:1879: RuntimeWarning: Degrees of freedom <= 0 for slice.
  var = nanvar(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
100%|██████████| 1/1 [00:01<00:00,  1.59s/it]
```

In [ ]:

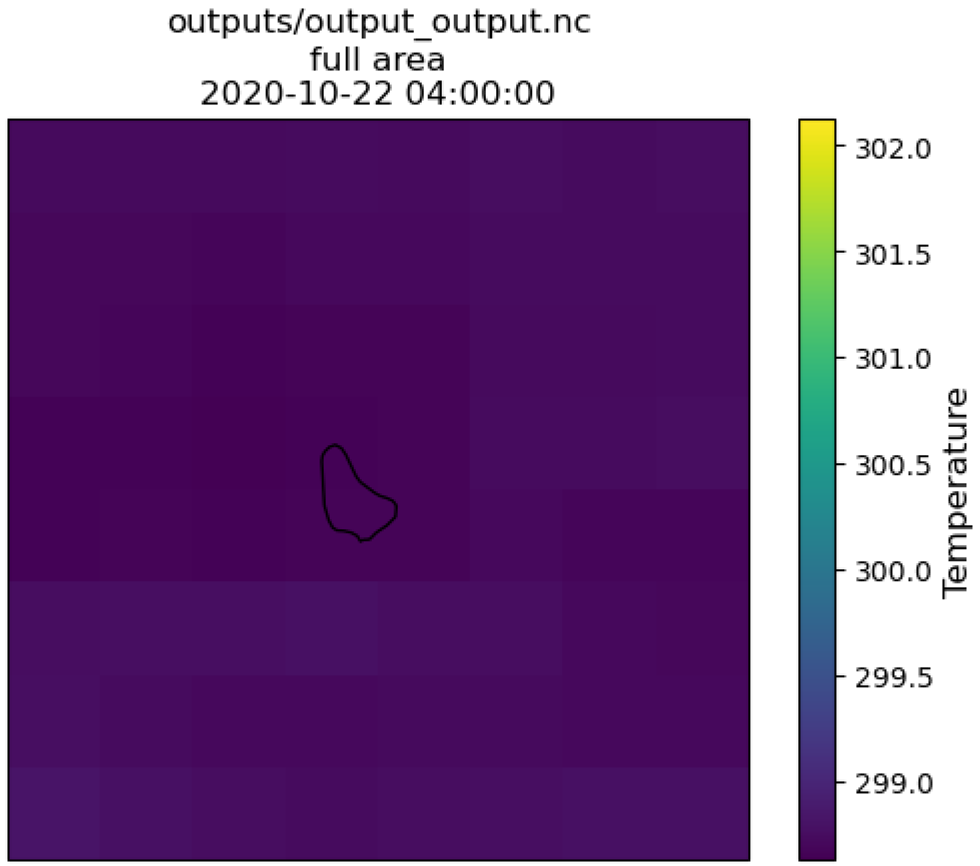
```python
import xarray as xr
from utils import DataSet, DatasetPlotter
import numpy as np
import os

test_folder_path = "/work/bm1159/XCES/xces-work/k203179/data/test"
reconstructed_folder_path = "outputs/output_output.nc"
era5_file = f"{test_folder_path}/era5_for_{station_name.lower()}.nc"

# get measurements values

measurements_data = xr.open_dataset(test_folder_path + f"/reality_{station_name.lower()}.nc")

# plot era5 and output at timesteps [x, ...]
plot_timestep = 2000

era5_ds = DataSet(era5_file)
output_ds = DataSet(reconstructed_folder_path)

vmin = min(
    np.min(era5_ds.dataset.variables['tas'][plot_timestep, :, :]),
    np.min(output_ds.dataset.variables['tas'][plot_timestep, :, :]),
)

vmax = max(
    np.max(era5_ds.dataset.variables['tas'][plot_timestep, :, :]),
    np.max(output_ds.dataset.variables['tas'][plot_timestep, :, :]),
)
```

In [ ]:

```python
plotter = DatasetPlotter(era5_ds)
plotter.time_index_list = [plot_timestep]
plotter.vmin = vmin
plotter.vmax = vmax
plotter.plot()
```



In [ ]:
```
plotter = DatasetPlotter(output_ds)
plotter.time_index_list = [plot_timestep]
plotter.vmin = vmin
plotter.vmax = vmax
plotter.plot()
```



In [ ]:
```python
# get coordinates from measurements nc file
import numpy as np

station_lon, station_lat = measurements_data.lon.values[0], measurements_data.lat.values[0]
print(f"station is at {station_lon}, {station_lat}")

# get nearest coordinates in era5
def get_left_right_nearest_elem_in_sorted_array(array, value):
    length = len(array)
    left = len(list(filter(lambda x: x <= value, array))) - 1
    right = length - len(list(filter(lambda x: x >= value, array)))
    nearest = min(left, right, key=lambda x: abs(array[x] - value))
    return left, right, nearest

test_array = [1, 2, 3, 4, 5, 6, 7, 8]
test_search = 5.51

print(f"searching for {test_search} in {test_array}")
left_idx, right_idx, nearest_idx =  get_left_right_nearest_elem_in_sorted_array(test_array, test_search)
print(f"idx left to {test_search} is {left_idx}, idx right to {test_search} is {right_idx}, nearest idx is {nearest_idx}")
print(f"mid crop: {test_array[left_idx:right_idx+1]}")
```

```
station is at -59.54316, 13.16443
searching for 5.51 in [1, 2, 3, 4, 5, 6, 7, 8]
idx left to 5.51 is 4, idx right to 5.51 is 5, nearest idx is 5
mid crop: [5, 6]
```

In [ ]:
```python
def era_vs_reconstructed_comparision_to_df():
    era5_data = xr.open_dataset(era5_file)
    reconstructed_data = xr.open_dataset(reconstructed_folder_path)


    lon_left_idx, lon_right_idx, lon_nearest_idx = get_left_right_nearest_elem_in_sorted_array(era5_data.lon.values, station_lon % 360)
    lat_left_idx, lat_right_idx, lat_nearest_idx = get_left_right_nearest_elem_in_sorted_array(era5_data.lat.values, station_lat)

    era5_mid_values = era5_data.variables["tas"][:, lon_left_idx:lon_right_idx+1, lat_left_idx:lat_right_idx+1].mean(axis=(1,2))
    era5_nearest_values = era5_data.variables["tas"][:, lon_nearest_idx, lat_nearest_idx]

    reconstructed_data_values = reconstructed_data.variables["tas"].stack(grid=['lat', 'lon']).values
    measurements_data_values = measurements_data.variables["tas"][...].mean(axis=(1,2))


    # timeaxis
    time = measurements_data.variables["time"][:]

    import pandas as pd

    # create dataframe with all values
    df = pd.DataFrame()

    df["time"] = time

    # index should be time
    df.set_index("time", inplace=True)

    df["era5_mid"] = era5_mid_values
    df["era5_nearest"] = era5_nearest_values
    df["reconstructed_median"] = [np.median(x) for x in reconstructed_data_values]
    df["reconstructed_mean"] = [np.mean(x) for x in reconstructed_data_values]
    df["reconstructed_min"] = [np.min(x) for x in reconstructed_data_values]
```

```
    df["reconstructed_max"] = [np.max(x) for x in reconstructed_data_values]

    df["measurements"] = measurements_data_values

    return df
```

Generate Dataframe

- makes resampling easier

In [ ]:
```
hourly_df = era_vs_reconstructed_comparision_to_df()

# print a section of the df

start_print_date = "2020-10-22"
end_print_date = "2020-10-22"

hourly_df[start_print_date:end_print_date]
```

Out[ ]:

time	era5_mid	era5_nearest	reconstructed_median	reconstructed_mean	reconstructed_min	reconstructed_max	measurements
2020-10-22 00:00:00	301.842285	301.938965	298.618591	298.617065	298.538147	298.688232	298.495000
2020-10-22 01:00:00	301.855133	301.952301	298.612671	298.611572	298.487640	298.733185	298.513333
2020-10-22 02:00:00	301.840210	301.920776	298.892822	298.887695	298.810944	298.969757	298.846610
2020-10-22 03:00:00	301.814880	301.880310	298.604858	298.588043	298.424316	298.698181	298.120000
2020-10-22 04:00:00	301.781555	301.831848	298.714355	298.710876	298.628601	298.809692	298.041667
2020-10-22 05:00:00	301.655334	301.665588	298.116455	298.103546	297.936249	298.213715	298.235000
2020-10-22 06:00:00	301.565308	301.586792	298.182312	298.145599	297.883881	298.262787	298.238333
2020-10-22 07:00:00	301.387207	301.370605	298.428619	298.428955	298.326782	298.502228	298.413333
2020-10-22 08:00:00	301.302887	301.237457	298.447815	298.456421	298.380554	298.579559	298.399153
2020-10-22 09:00:00	301.291931	301.283142	299.104736	299.106384	299.033234	299.172729	298.431667
2020-10-22 10:00:00	301.197083	301.142883	298.802307	298.801208	298.744476	298.858521	298.778333
2020-10-22 11:00:00	301.339661	301.291321	299.956604	299.952026	299.851654	300.023438	300.001667
2020-10-22 12:00:00	301.468658	301.435944	300.333374	300.341187	300.207977	300.463440	300.600000
2020-10-22 13:00:00	301.510529	301.575470	301.900085	301.897247	301.805023	301.982666	301.250000
2020-10-22 14:00:00	301.483215	301.576965	301.512177	301.519165	301.411469	301.690918	301.851695
2020-10-22 15:00:00	301.440094	301.473297	301.512360	301.520569	301.453247	301.601990	301.955000
2020-10-22 16:00:00	301.441528	301.454224	302.387756	302.389954	302.290710	302.457947	301.935000
2020-10-22 17:00:00	301.448853	301.433228	302.659973	302.661011	302.566956	302.747620	302.301667
2020-10-22 18:00:00	301.450806	301.415161	301.482849	301.484833	301.395874	301.588409	302.178333
2020-10-22 19:00:00	301.521118	301.498657	301.970947	301.980347	301.842194	302.175659	301.411667
2020-10-22 20:00:00	300.810852	301.312317	301.279663	301.278412	301.148560	301.383301	300.614407
2020-10-22 21:00:00	300.350952	300.648804	299.939575	299.951721	299.760773	300.067627	299.291667
2020-10-22 22:00:00	300.710175	300.861053	297.548553	297.614380	297.400116	297.894684	298.381667
2020-10-22 23:00:00	300.906555	300.906555	298.199158	298.203430	298.129242	298.304199	298.323333

Implement plotting method of dataframe

In [ ]:

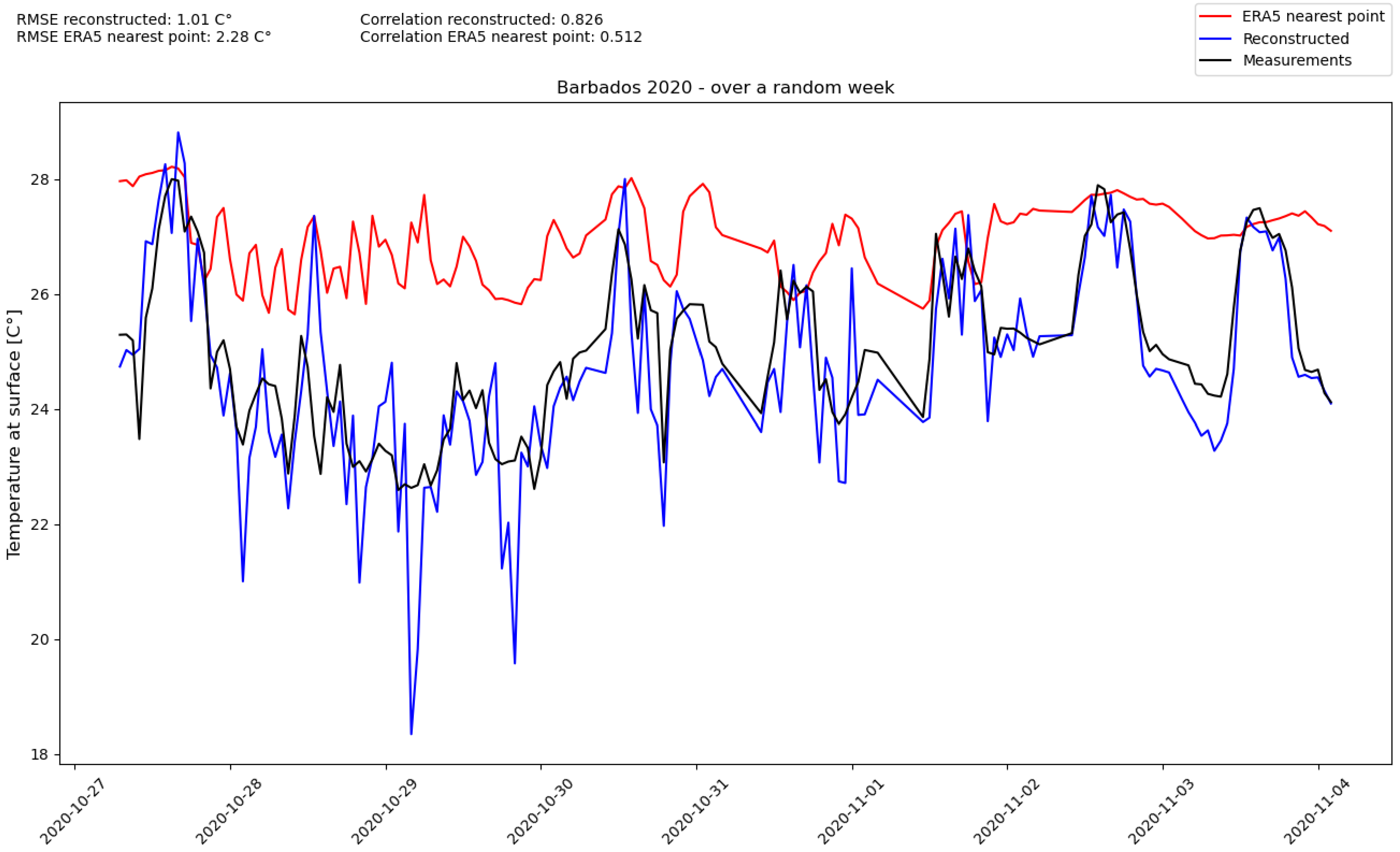
```
def plot_n_steps_of_df(df, as_delta, n=None, title=None, boxplot=False):

    from matplotlib import pyplot as plt

    time = df.index.values
    if n is None:
        n = len(df)

    # random slice of n consecutive datapoints
    import random
```

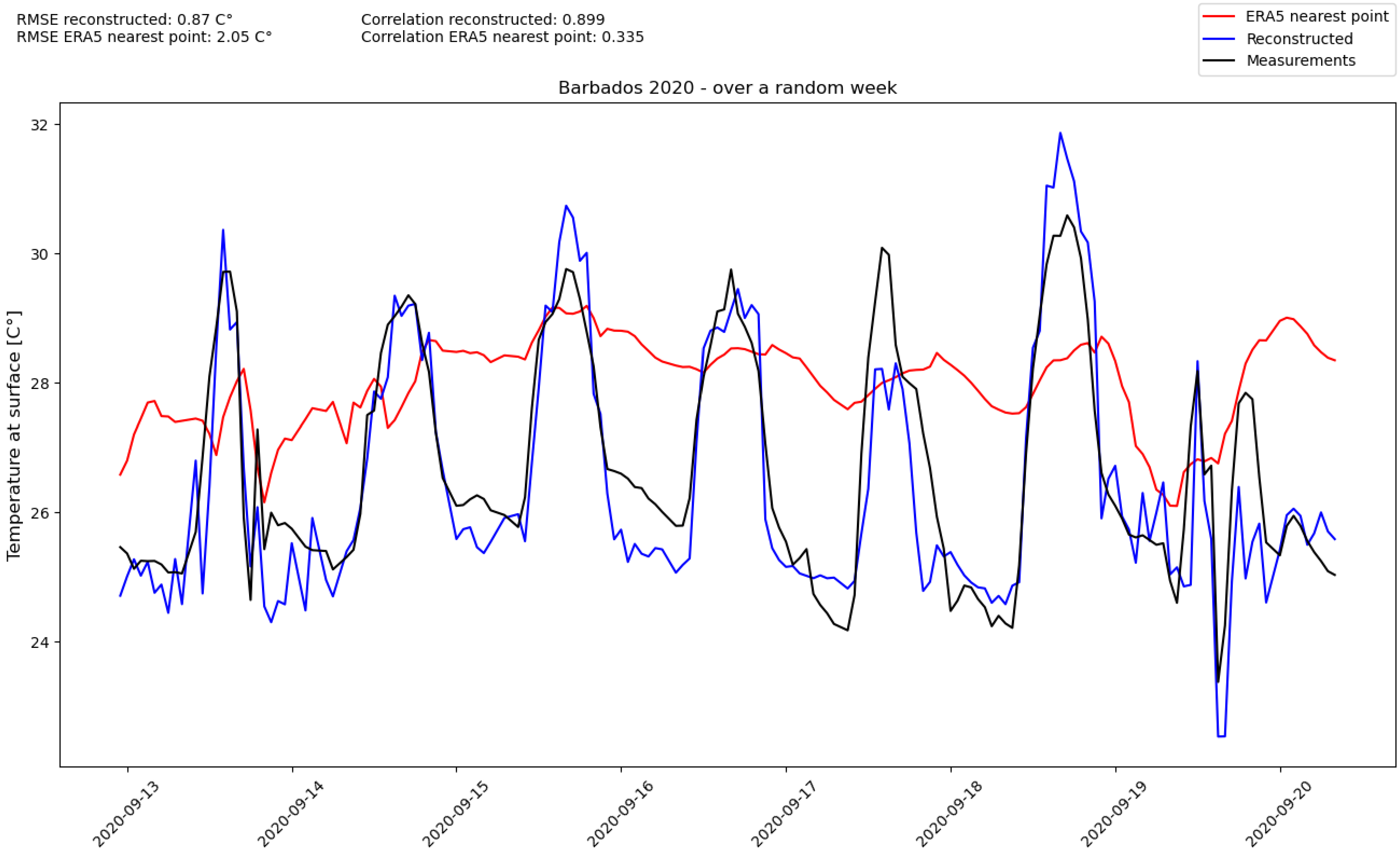
```python
    slice_start = random.randint(0, len(time) - n)
    time_slice = slice(slice_start, slice_start + n)

    time = time[time_slice]

   # era5_mid_values = df["era5_mid"].values -273.15
    era5_nearest_values = df["era5_nearest"].values - 273.15
    reconstructed_mean_values = df["reconstructed_mean"].values - 273.15
    reconstructed_median_values = df["reconstructed_median"].values - 273.15
    reconstructed_min_values = df["reconstructed_min"].values - 273.15
    reconstructed_max_values = df["reconstructed_max"].values - 273.15

    measurements_values = df["measurements"].values - 273.15

    rmse_reconstructed = np.sqrt(np.sum((reconstructed_median_values[time_slice] - measurements_values[time_slice])**2) / len(time))
   # rmse_era5_mid = np.sqrt(np.sum((era5_mid_values[time_slice] - measurements_values[time_slice])**2) / len(time))
    rmse_era5_nearest = np.sqrt(np.sum((era5_nearest_values[time_slice] - measurements_values[time_slice])**2) / len(time))

    correlation_reconstructed = np.corrcoef(reconstructed_median_values[time_slice], measurements_values[time_slice])[0,1]
   # correlation_era5_mid = np.corrcoef(era5_mid_values[time_slice], measurements_values[time_slice])[0,1]
    correlation_era5_nearest = np.corrcoef(era5_nearest_values[time_slice], measurements_values[time_slice])[0,1]

    if as_delta:
     #   era5_mid_values = era5_mid_values - measurements_values
        era5_nearest_values = era5_nearest_values - measurements_values
        reconstructed_mean_values = reconstructed_mean_values - measurements_values
        reconstructed_median_values = reconstructed_median_values - measurements_values
        reconstructed_min_values = reconstructed_min_values - measurements_values
        reconstructed_max_values = reconstructed_max_values - measurements_values
        measurements_values = measurements_values - measurements_values

        # y-axis title, temperature difference
        plt.ylabel("Delta calculated by subtracting measurement data [C°]")

    else:
        plt.ylabel("Temperature at surface [C°]")


    plt.plot(time, era5_nearest_values[time_slice], label="ERA5 nearest point", color="red")
    # plt.plot(time, era5_mid_values[time_slice], label="ERA5 nearest 4 points")

    if boxplot:
        for i in range(len(time)):
            plt.vlines(time[i], reconstructed_min_values[time_slice][i], reconstructed_max_values[time_slice][i], color="black", linewidth=1)
        plt.scatter(time, reconstructed_median_values[time_slice], label="Reconstructed", color="blue", s=8)
    else:
        plt.plot(time, reconstructed_median_values[time_slice], label="Reconstructed", color="blue")

    plt.plot(time, measurements_values[time_slice], label="Measurements", color="black")

    # x-axis labels 90 degrees
    plt.xticks(rotation=45)

    # title
    if title is not None:
        plt.title(title)


    # font size of legend
    plt.rcParams.update({'font.size': 10})
```

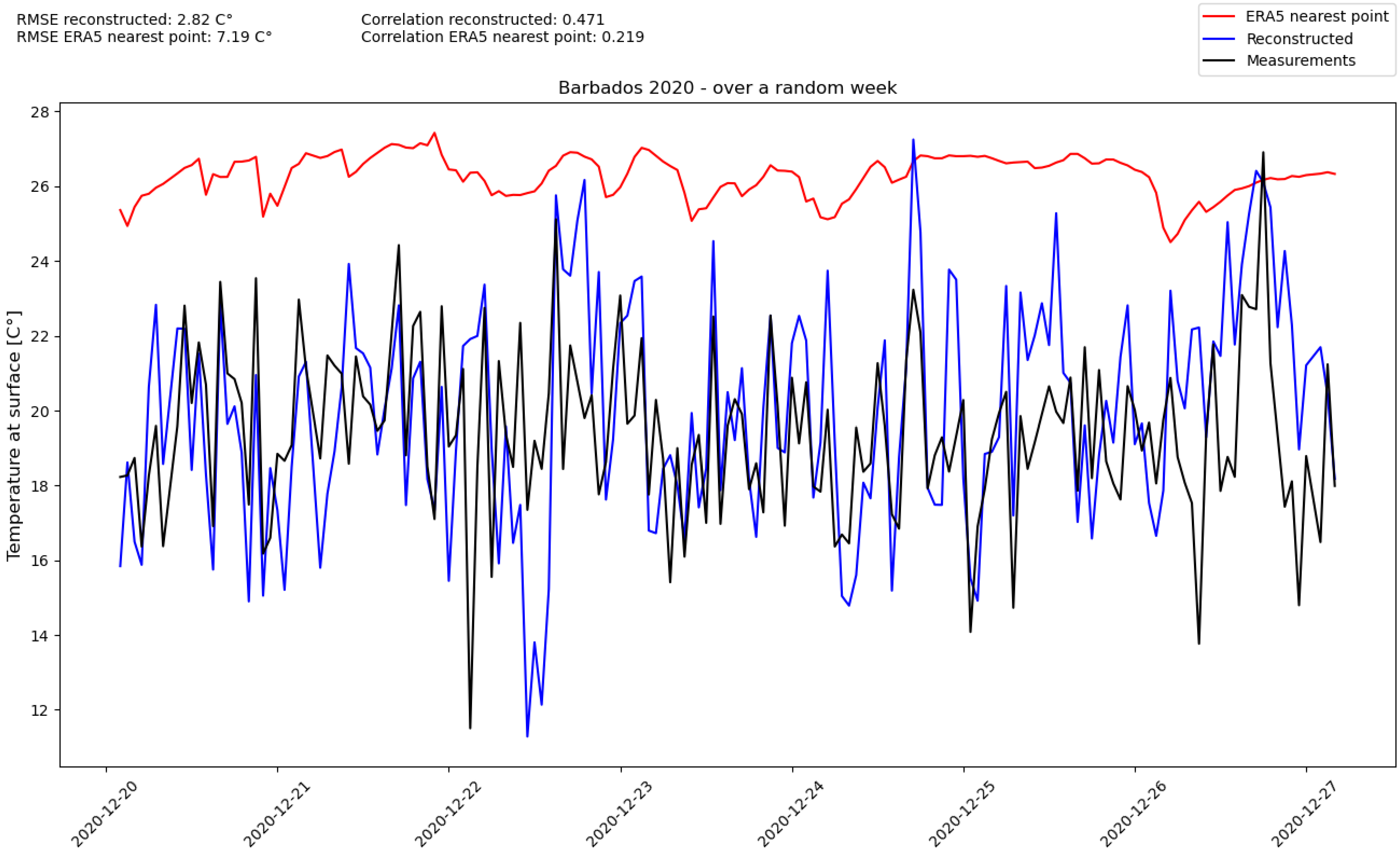
```
    # font size of axis labels
    plt.rcParams.update({'axes.labelsize': 12})

    plt.legend()
    # position legend below chart to the right
    plt.legend(bbox_to_anchor=(1, 1.15), loc='upper right', borderaxespad=0.)


    # text below diagram with RMSE and Correlation in fontsize 10
    plt.text(0.1,0.95, f"RMSE reconstructed: {rmse_reconstructed:.2f} C°\n" +
             f"RMSE ERA5 nearest point: {rmse_era5_nearest:.2f} C°",

             fontsize=10, transform=plt.gcf().transFigure)

    plt.text(0.3, 0.95, f"Correlation reconstructed: {correlation_reconstructed:.3f}\n" +
             f"Correlation ERA5 nearest point: {correlation_era5_nearest:.3f}",

             fontsize = 10, transform=plt.gcf().transFigure)

    # figure size A4 landscape
    plt.gcf().set_size_inches(16, 8)

    plt.show()
```

Plot Hourly (deltas), so errors against real measurements

In [ ]:
```
n = 168
if n == 168:
    title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - over a random week"
else:
    title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - {n} random consecutive hourly steps"
plot_n_steps_of_df(hourly_df, as_delta=False, n=n, title=title)
plot_n_steps_of_df(hourly_df, as_delta=False, n=n, title=title)
plot_n_steps_of_df(hourly_df, as_delta=False, n=n, title=title)
```

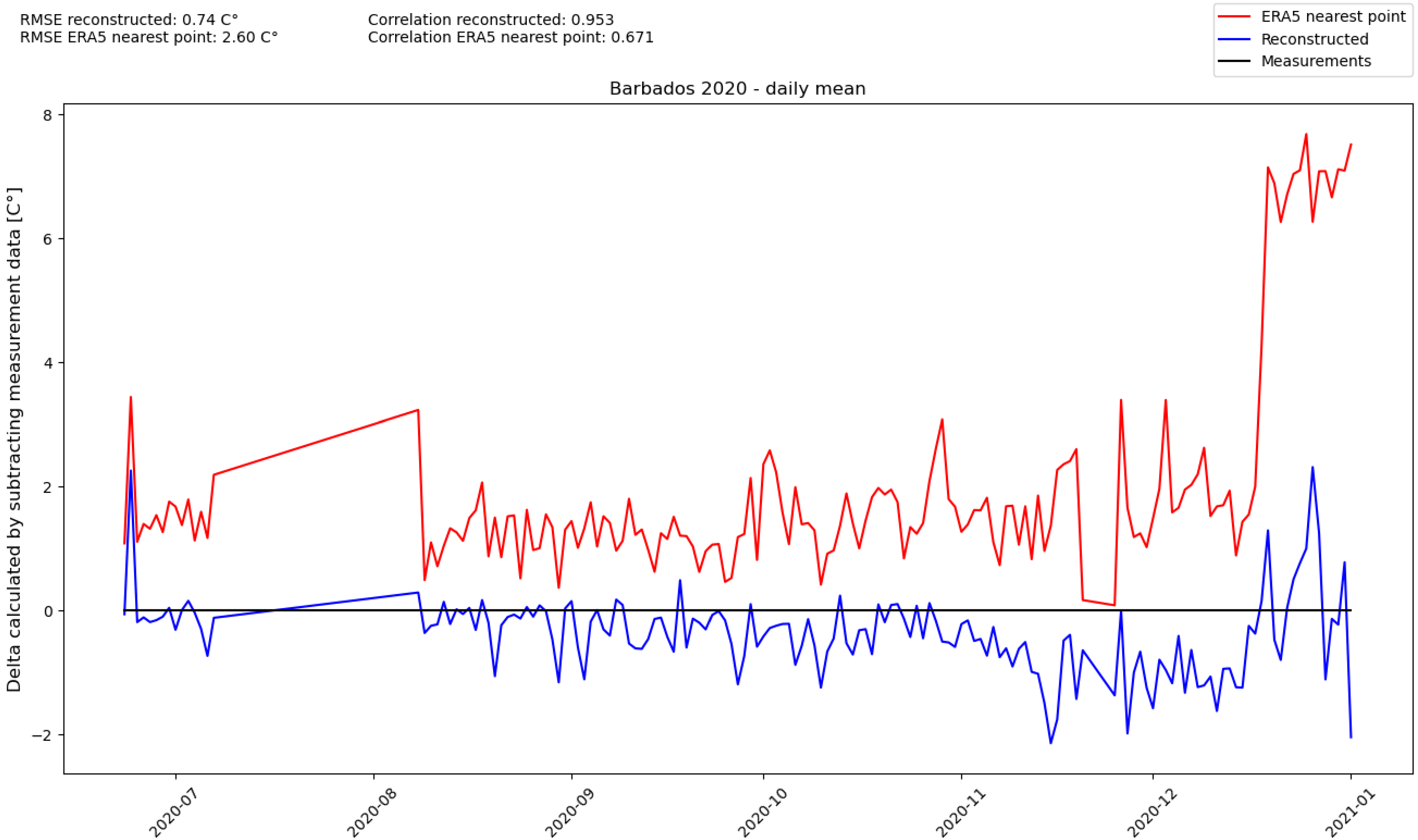

RMSE reconstructed: 1.01 C°
RMSE ERA5 nearest point: 2.28 C°
Correlation reconstructed: 0.826
Correlation ERA5 nearest point: 0.512
ERA5 nearest point
Reconstructed
Measurements
Barbados 2020 - over a random week
Temperature at surface [C°]
28
26
24
22
20
18
2020-10-27
2020-10-28
2020-10-29
2020-10-30
2020-10-31
2020-11-01
2020-11-02
2020-11-03
2020-11-04

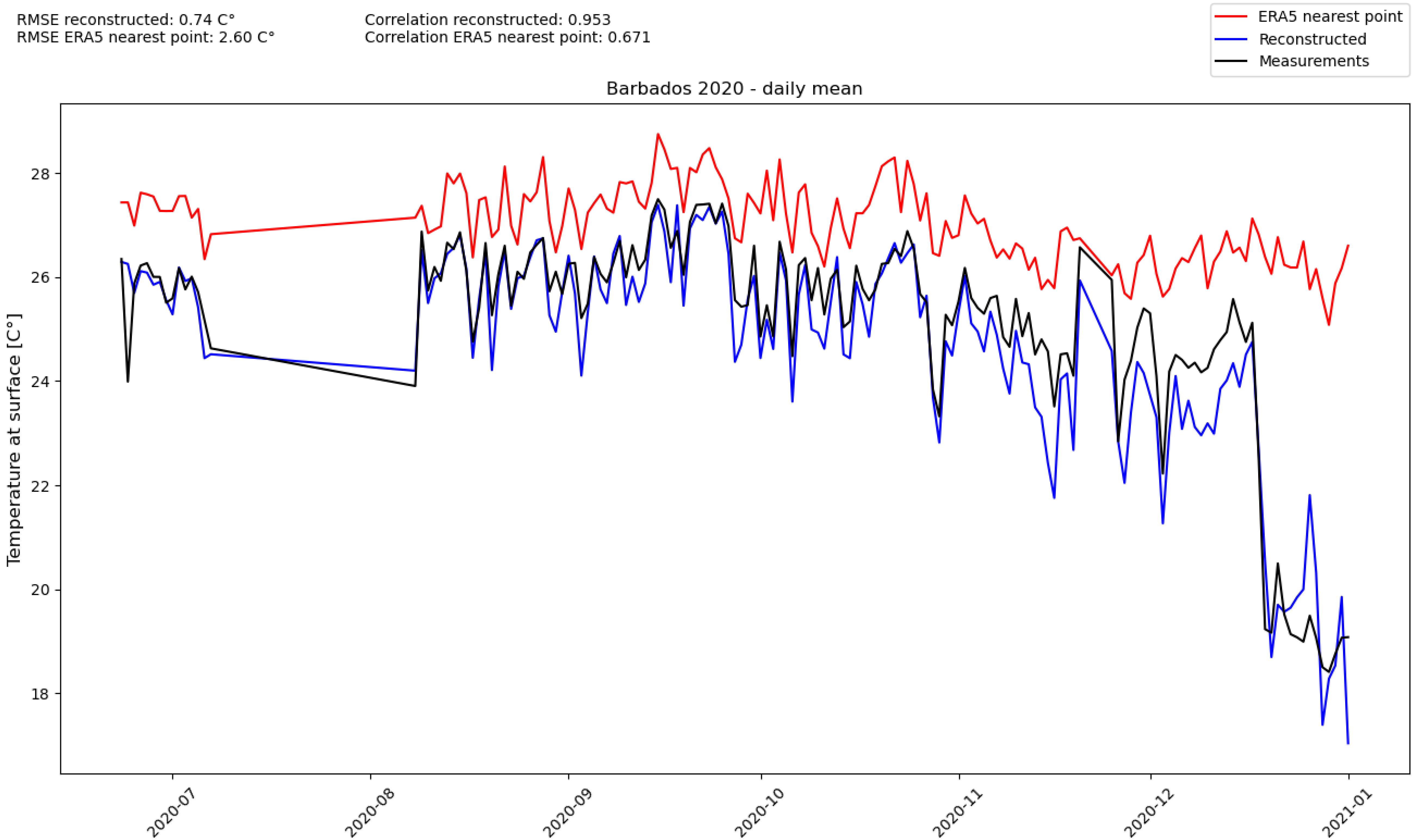

RMSE reconstructed: 0.87 C°
RMSE ERA5 nearest point: 2.05 C°
Correlation reconstructed: 0.899
Correlation ERA5 nearest point: 0.335
ERA5 nearest point
Reconstructed
Measurements
Barbados 2020 - over a random week
Temperature at surface [C°]
32
30
28
26
24
2020-09-13
2020-09-14
2020-09-15
2020-09-16
2020-09-17
2020-09-18
2020-09-19
2020-09-20



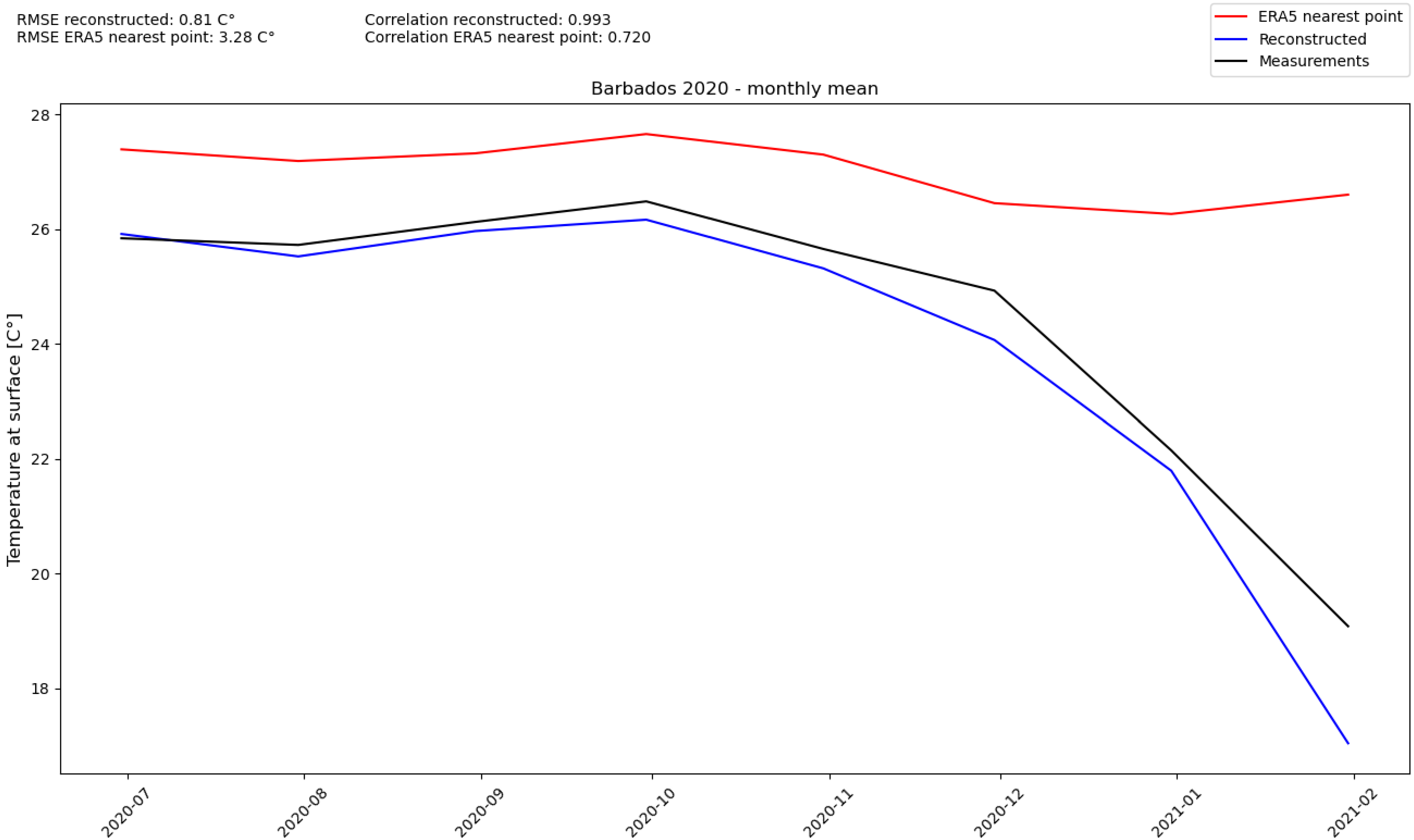
Resample Data from hourly to daily or monthly

In [ ]:
```
# drop reconstructed column
daily_df = hourly_df.resample("D").mean()
# drop nans
daily_df = daily_df.dropna()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - daily mean"
plot_n_steps_of_df(daily_df, as_delta=True, title=title)
plot_n_steps_of_df(daily_df, as_delta=False, title=title)
```


RMSE reconstructed: 0.74 C°
RMSE ERA5 nearest point: 2.60 C°
Correlation reconstructed: 0.953
Correlation ERA5 nearest point: 0.671
ERA5 nearest point
Reconstructed
Measurements
Barbados 2020 - daily mean
Delta calculated by subtracting measurement data [C°]
8
6
4
2
0
−2
2020-07
2020-08
2020-09
2020-10
2020-11
2020-12
2021-01

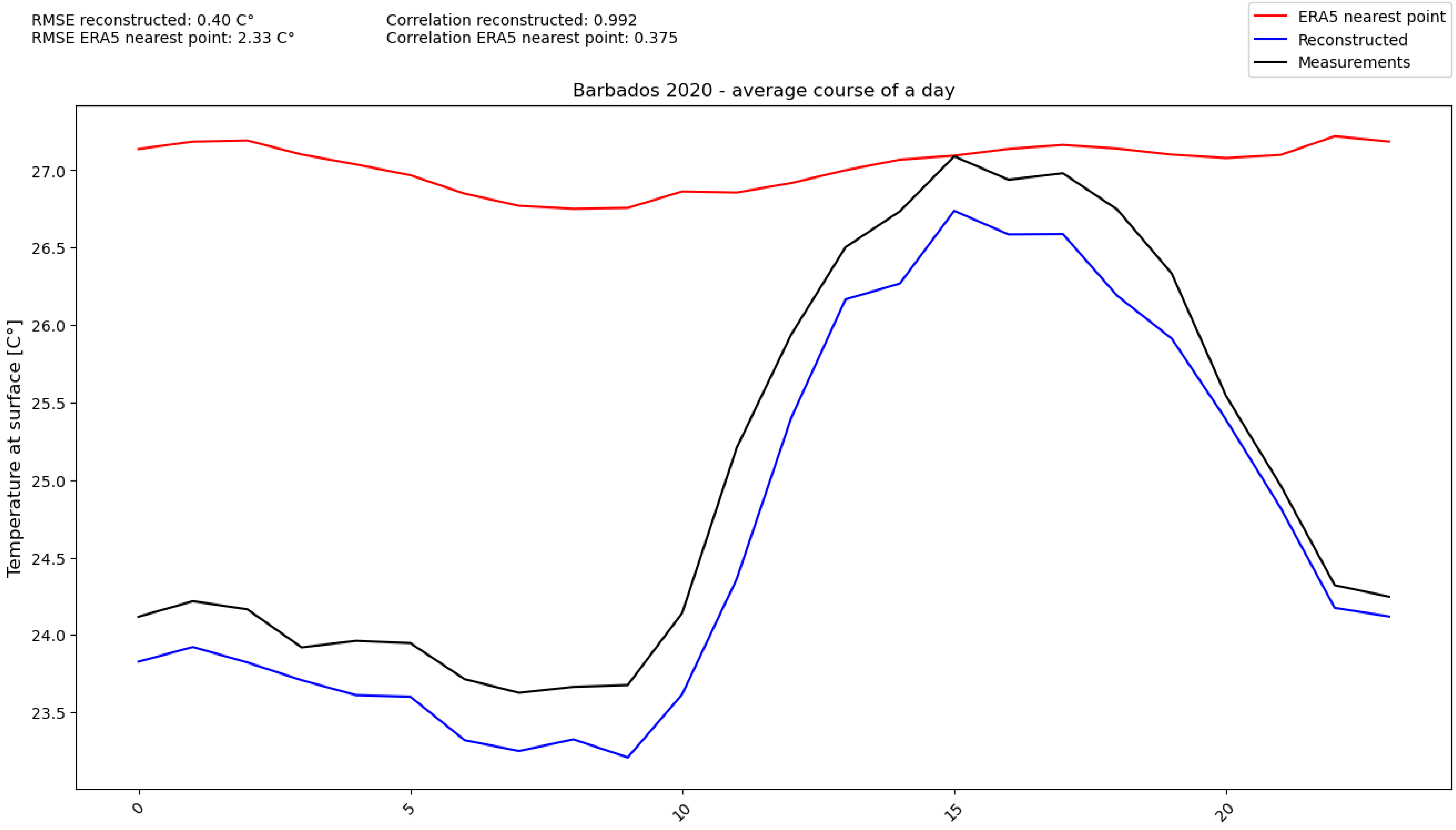


In [ ]:
```
# resample rows to monthly mean
df = hourly_df.resample("M").mean()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - monthly mean"
plot_n_steps_of_df(df, as_delta = False, title=title)
```



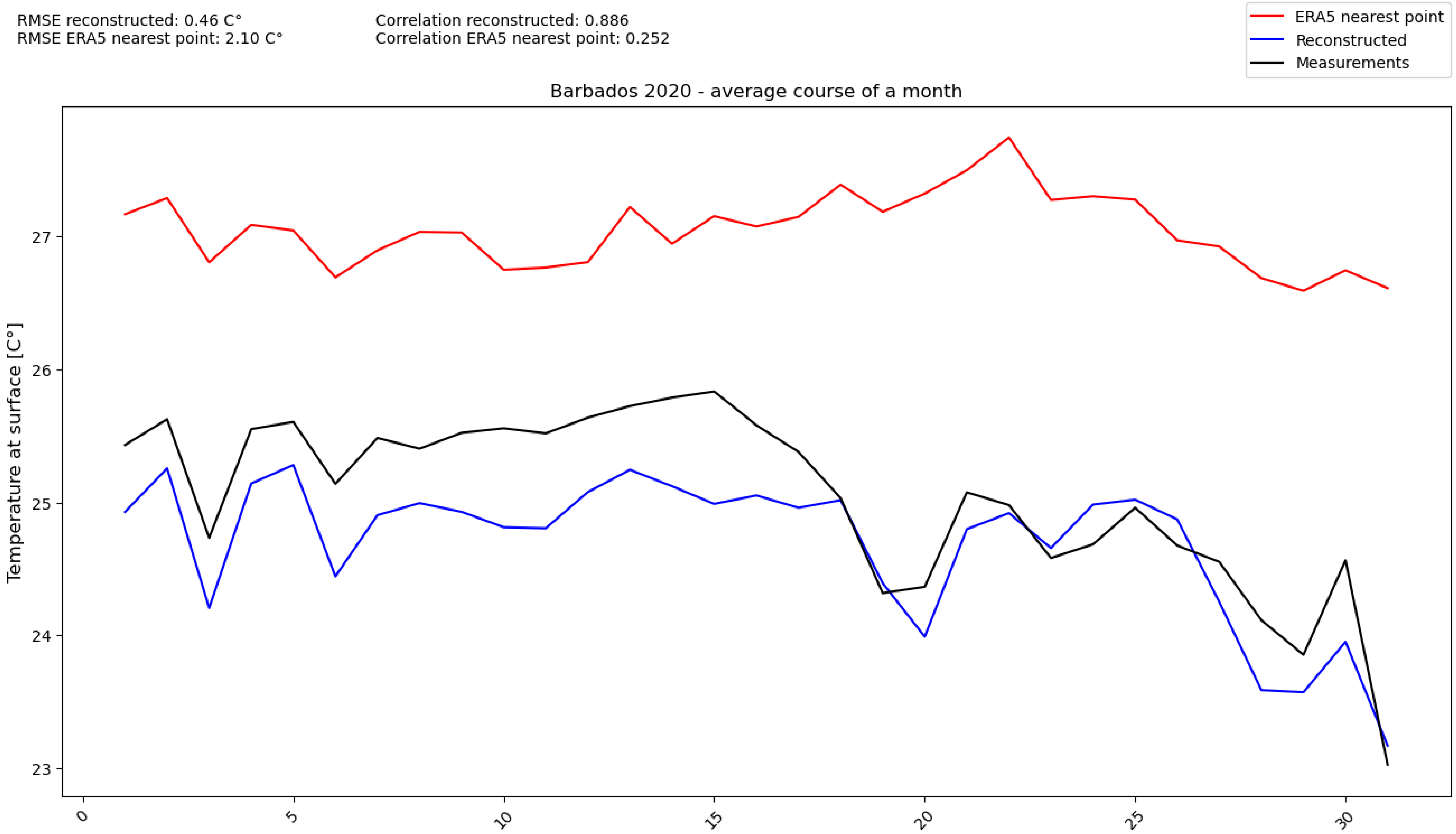
Average Course of the day

In [ ]:
```
# calculate the mean of each 24 hours over the whole year

day_course_df = hourly_df.groupby(hourly_df.index.hour).mean()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - average course of a day"
plot_n_steps_of_df(day_course_df, as_delta = False, title=title)
```



Average Course of the month

In [ ]:
```
# calculate the mean of each day of a month over the whole year

month_course_df = hourly_df.groupby(hourly_df.index.day).mean()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - average course of a month"
plot_n_steps_of_df(month_course_df, as_delta = False, title=title)
```



RMSE reconstructed: 0.46 C°
RMSE ERA5 nearest point: 2.10 C°
Correlation reconstructed: 0.886
Correlation ERA5 nearest point: 0.252
ERA5 nearest point
Reconstructed
Measurements
Barbados 2020 - average course of a month
Temperature at surface [C°]
27
26
25
24
23
0
5
10
15
20
25
30