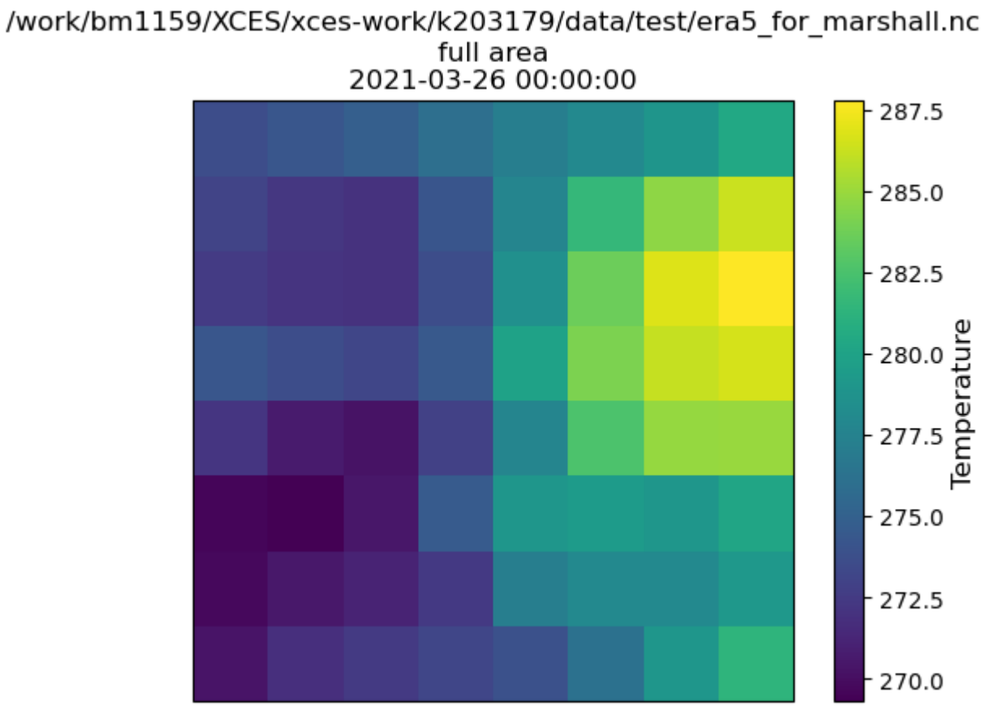
In [ ]:
```
station_name = "Marshall"
test_year = 2021
```

In [ ]:
```
from climatereconstructionai import evaluate

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

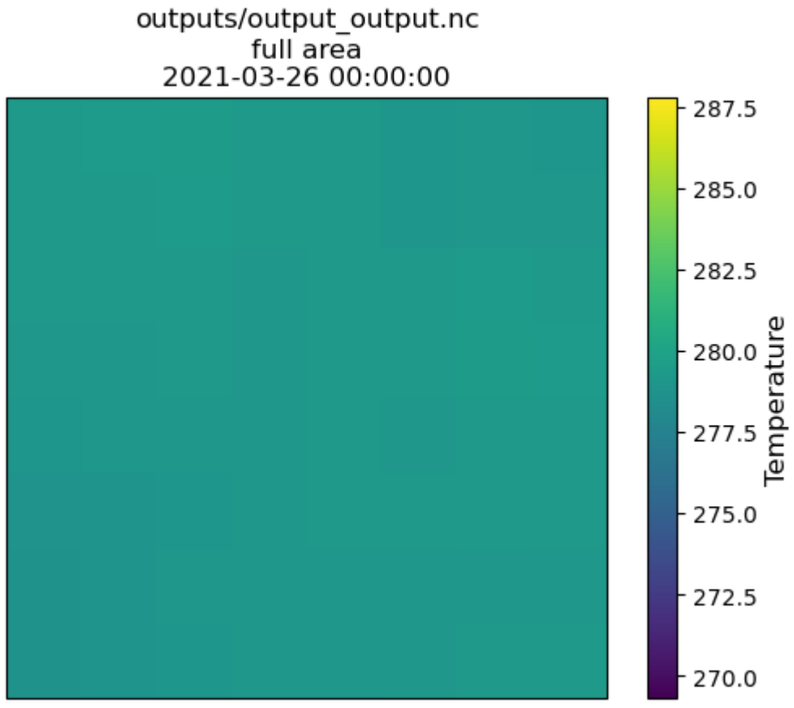
```
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/climatereconstructionai/utils/normalizer.py:10: RuntimeWarning: Mean of empty slice
  img_mean.append(np.nanmean(np.array(img_data[i])))
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/numpy/lib/nanfunctions.py:1879: RuntimeWarning: Degrees of freedom <= 0 for slice.
  var = nanvar(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
100%|██████████| 1/1 [00:04<00:00,  4.38s/it]
```

In [ ]:
```
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 [ ]:
```
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 -105.196, 39.9496
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 = "2021-03-26"
end_print_date = "2021-03-26"

hourly_df[start_print_date:end_print_date]
```

Out[ ]:

time	era5_mid	era5_nearest	reconstructed_median	reconstructed_mean	reconstructed_min	reconstructed_max	measurements
2021-03-26 00:00:00	276.218475	279.959747	279.178772	279.158234	278.714020	279.455566	280.486667
2021-03-26 01:00:00	278.066620	281.684601	279.205627	279.181335	278.817993	279.578888	279.163333
2021-03-26 02:00:00	274.503265	276.912476	276.742859	276.768219	276.366638	277.102783	277.186667
2021-03-26 03:00:00	274.186188	275.925781	277.356476	277.340515	276.996490	277.625366	276.965000
2021-03-26 04:00:00	272.871033	274.433105	276.508301	276.506744	276.265228	276.752747	277.176667
2021-03-26 05:00:00	272.874756	274.691833	277.030487	277.011169	276.662933	277.225342	276.503333
2021-03-26 06:00:00	272.583435	274.613647	276.152588	276.156799	275.923737	276.435425	275.681667
2021-03-26 07:00:00	271.781006	273.942200	275.585663	275.588287	275.402893	275.738892	275.941667
2021-03-26 08:00:00	271.247620	273.519531	275.900879	275.886658	275.734680	276.002686	275.460000
2021-03-26 09:00:00	269.865479	272.335754	274.884827	274.884705	274.717438	275.078064	274.700000
2021-03-26 10:00:00	269.352875	271.713318	275.078156	275.081207	274.924561	275.275421	273.968333
2021-03-26 11:00:00	268.684631	270.972015	274.900665	274.899170	274.755615	275.048309	273.260000
2021-03-26 12:00:00	268.646637	270.995270	274.769287	274.761780	274.600647	274.919128	273.605000
2021-03-26 13:00:00	268.101013	270.714355	274.950073	274.959351	274.769989	275.132111	274.563333
2021-03-26 14:00:00	269.453491	271.903687	276.358643	276.375580	276.214050	276.585480	276.630000
2021-03-26 15:00:00	272.864746	275.439453	277.202301	277.208374	276.966095	277.523865	279.180000
2021-03-26 16:00:00	274.429932	276.560181	278.336121	278.331360	278.133545	278.493500	278.458333
2021-03-26 17:00:00	275.660553	277.774506	278.691467	278.688477	278.538391	278.934723	279.718333
2021-03-26 18:00:00	275.945374	278.055328	278.845123	278.858032	278.686340	279.170319	279.113333
2021-03-26 19:00:00	275.928833	278.253479	278.906677	278.923431	278.715912	279.253510	278.310000
2021-03-26 20:00:00	275.580383	278.616425	278.884796	278.904907	278.536102	279.246216	278.826667
2021-03-26 21:00:00	275.214050	278.085571	278.447021	278.465118	278.020416	278.862061	279.731667
2021-03-26 22:00:00	275.176056	278.777924	278.755676	278.768036	278.536896	279.198059	278.070000
2021-03-26 23:00:00	274.879456	278.958954	277.427673	277.421295	277.165558	277.808990	276.098333

Implement plotting method of dataframe

In [ ]:
```python
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
```

```
 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()
```

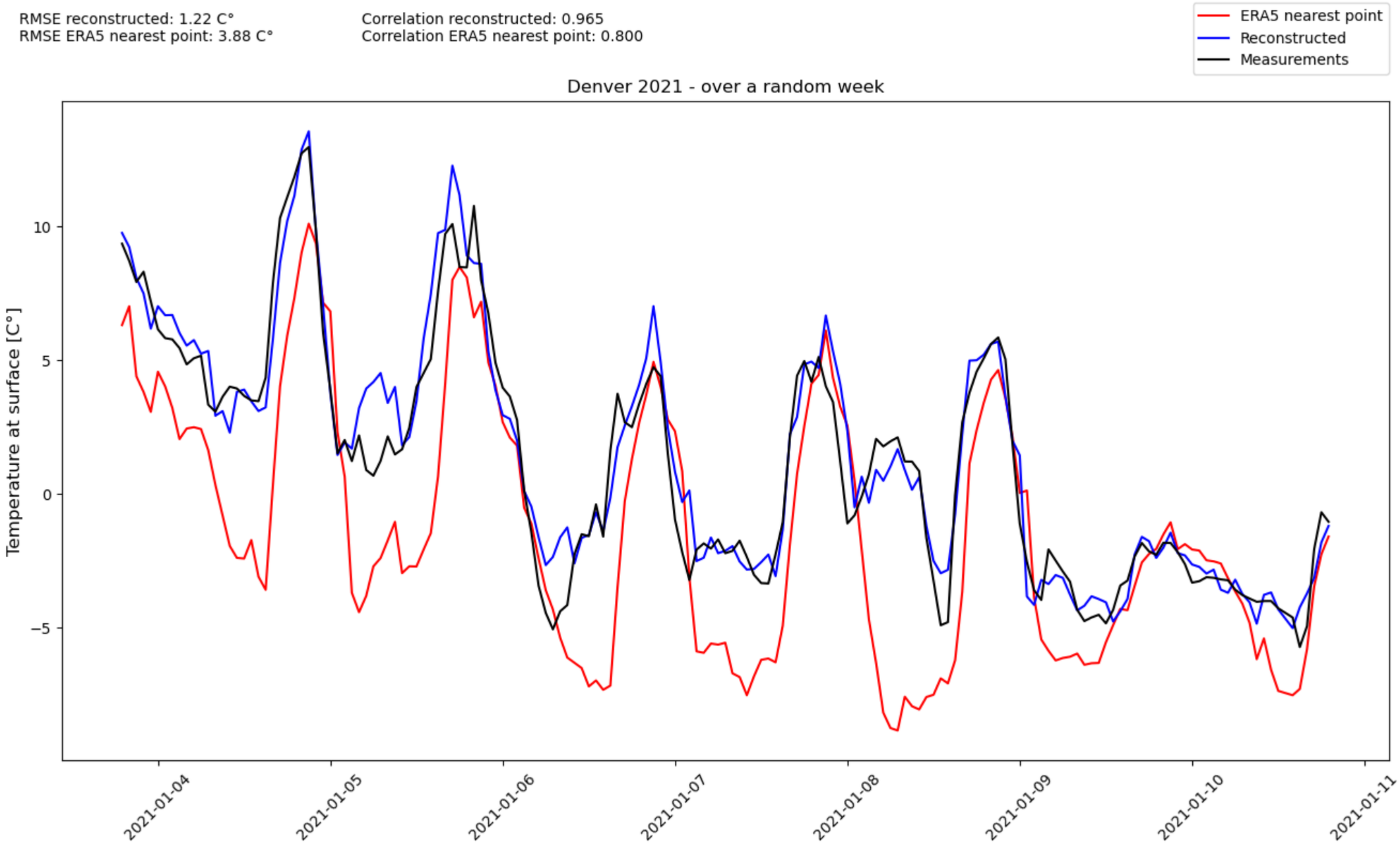
Plot Hourly (deltas), so errors against real measurements

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)
```

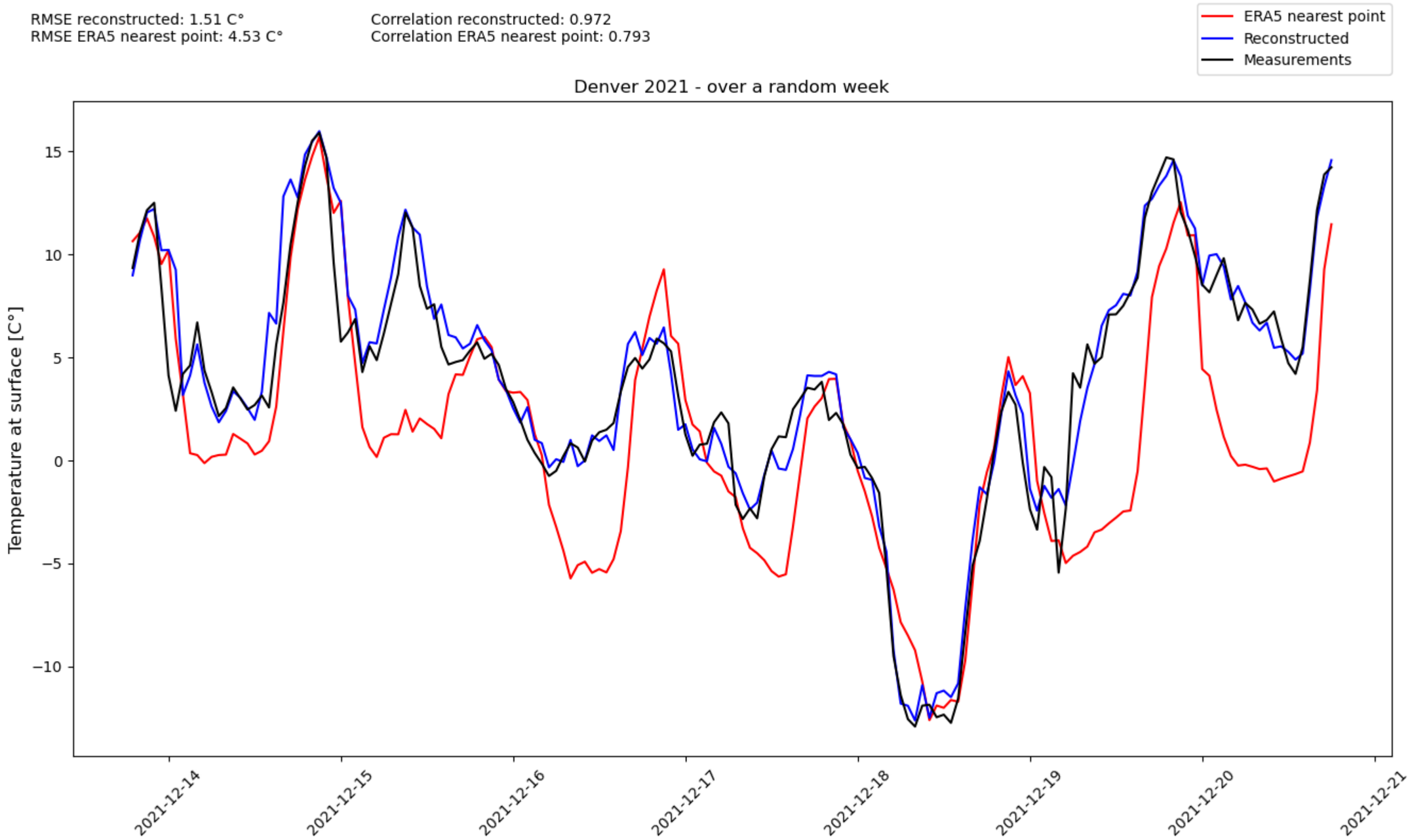
RMSE reconstructed: 1.22 C°
RMSE ERA5 nearest point: 3.88 C°

Correlation reconstructed: 0.965
Correlation ERA5 nearest point: 0.800



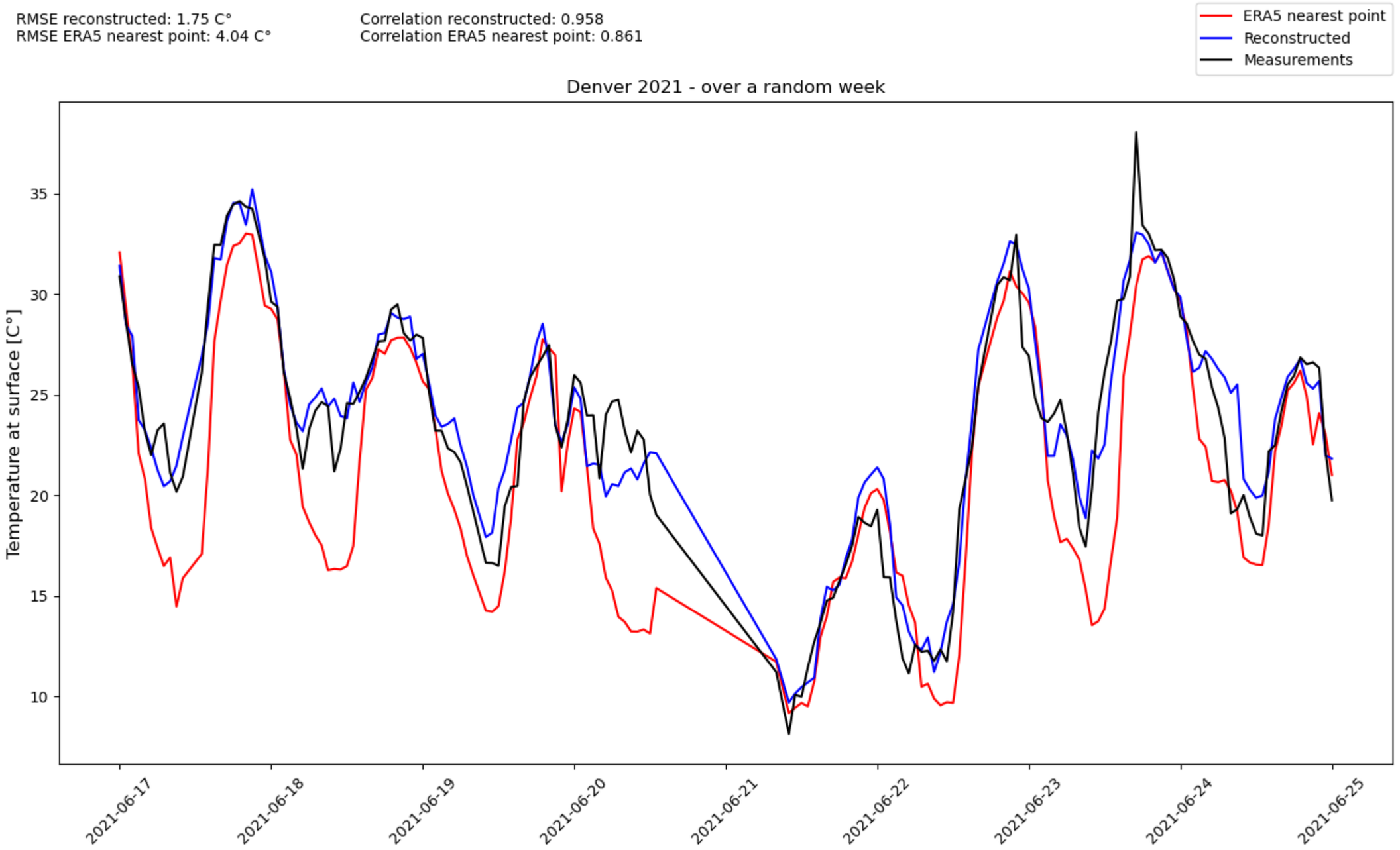
RMSE reconstructed: 1.51 C°
RMSE ERA5 nearest point: 4.53 C°

Correlation reconstructed: 0.972
Correlation ERA5 nearest point: 0.793



RMSE reconstructed: 1.75 C°
RMSE ERA5 nearest point: 4.04 C°

Correlation reconstructed: 0.958
Correlation ERA5 nearest point: 0.861

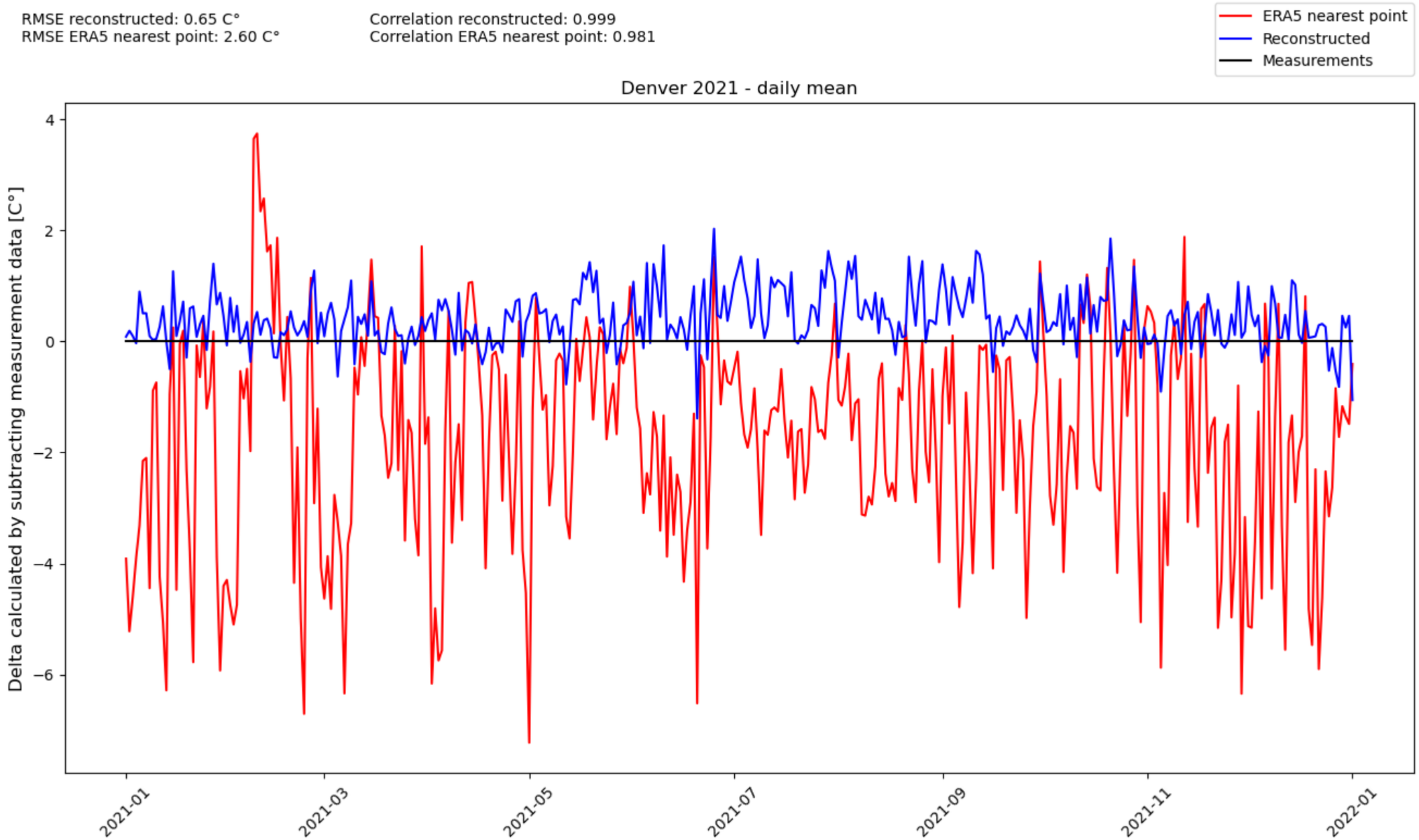


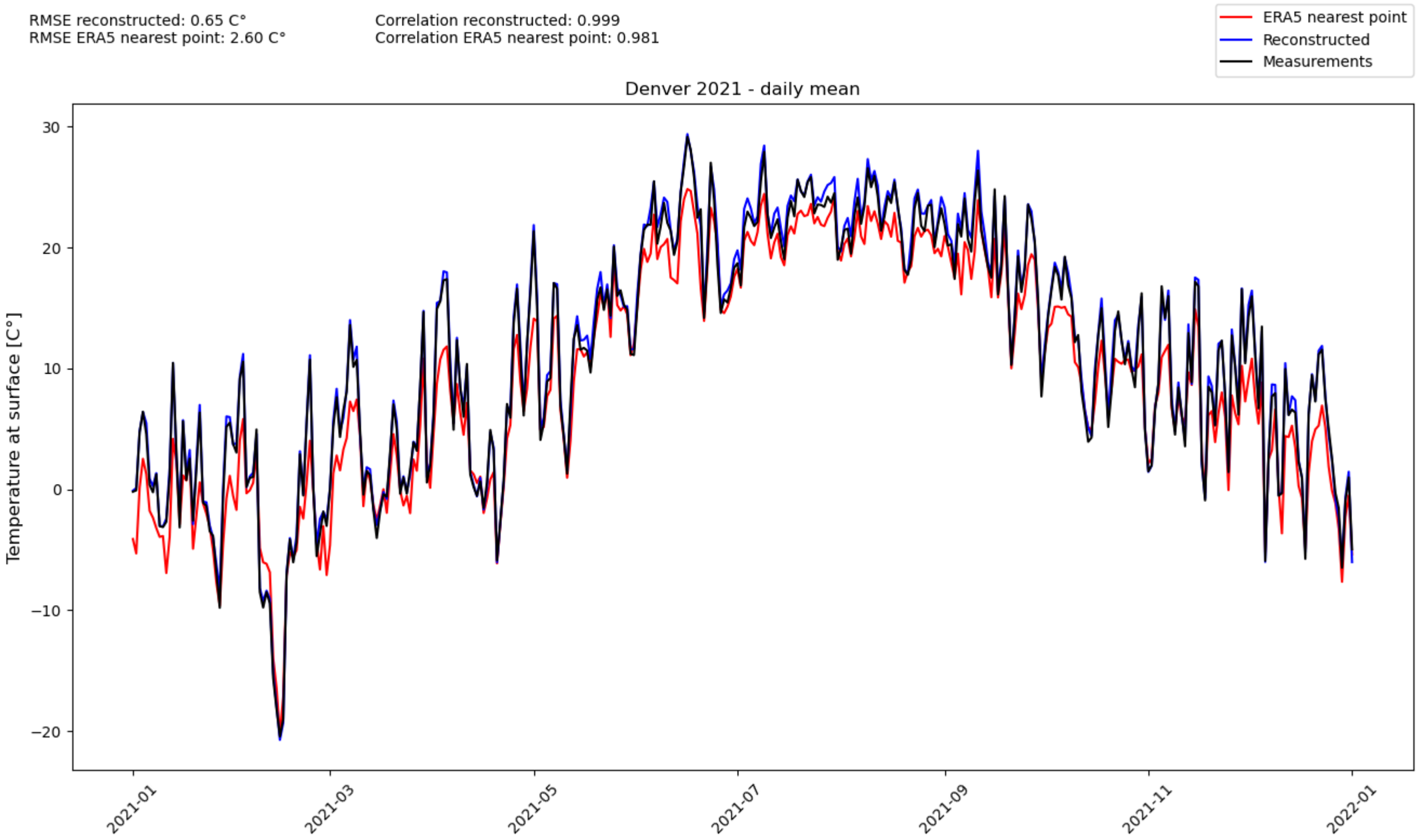
Resample Data from hourly to daily or monthly

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: 0.65 C°
RMSE ERA5 nearest point: 2.60 C°

Correlation reconstructed: 0.999
Correlation ERA5 nearest point: 0.981

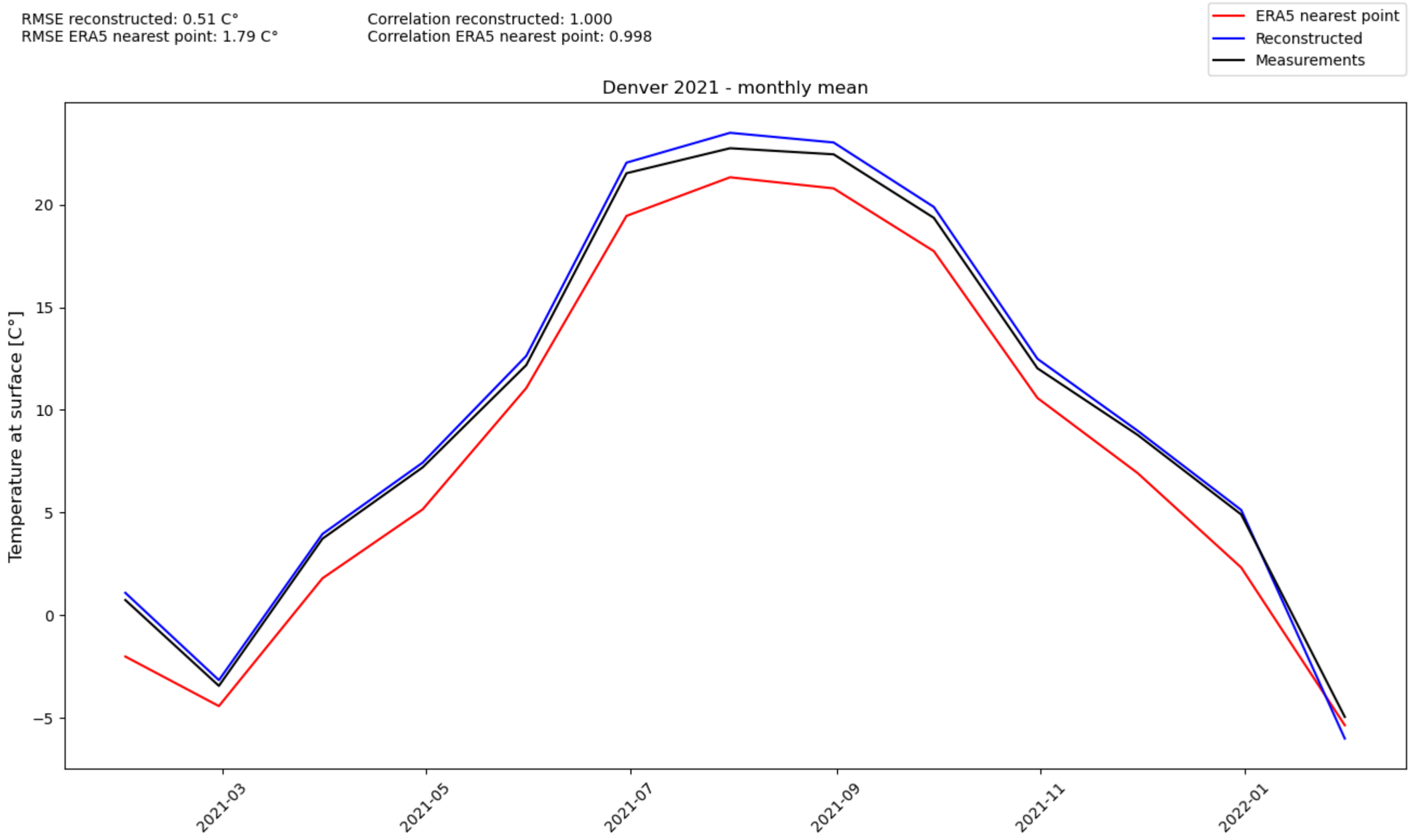




In [ ]:
```
# resample rows to monthly mean
df = hourly_df.resample("M").mean()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - monthly mean"
plot_n_steps_of_df(df, as_delta = False, title=title)
```

RMSE reconstructed: 0.51 C°
RMSE ERA5 nearest point: 1.79 C°

Correlation reconstructed: 1.000
Correlation ERA5 nearest point: 0.998

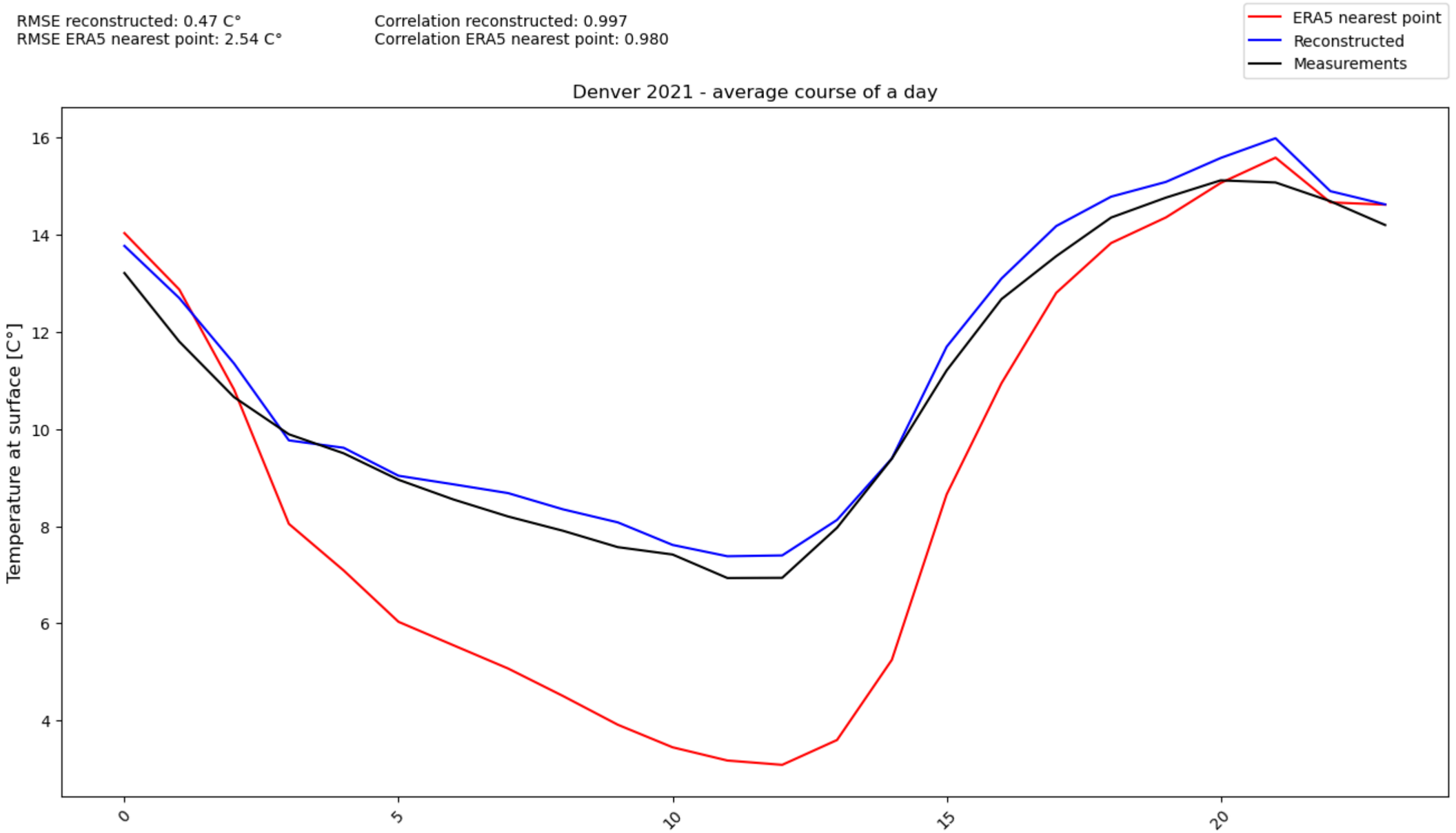


Average Course of the day

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)
```

RMSE reconstructed: 0.47 C°
RMSE ERA5 nearest point: 2.54 C°

Correlation reconstructed: 0.997
Correlation ERA5 nearest point: 0.980



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: 0.41 C°
RMSE ERA5 nearest point: 1.91 C°

Correlation reconstructed: 0.998
Correlation ERA5 nearest point: 0.959