Příloha 1: Matlab script Az 0-360°, El 0-90°

```
f_vz = 1;   % sampling frequency

% parameters of satellite
v_el = 0.167;       % angular rate of elevation of the satellite
t_max = 180/v_el;   % time it takes for the satellite to end its journey
azim = 30.067;      % initial azimuth of the satellite

t = 0:1/f_vz:t_max; % time vector
tlen = length(t);
elev = horzcat(v_el*t(1:tlen/2), -v_el*t(tlen/2+1:end) + 180);      % vector of elevations of
the satellite in time

figure(1)
subplot(311), plot(t,elev)
title("Az 0-360°, El 0-90°")
xlabel("t [s]")
ylabel("Elevation [°]")
legend("Sat")

% capabilities of the tracking system
ant_az_step = 1;      % min. angle step in deg - azimuth
ant_el_step = 1;      % min. angle step in deg - elevation
ant_v_az    = 4.5;    % max. angular rate of azimuth of the antenna
ant_v_el    = 4.5;    % max. angular rate of elevation of the antenna

ant_t_az    = ant_az_step/ant_v_az;     % time it takes for the antenna to change its azimuth
direction
ant_t_el    = ant_el_step/ant_v_el;     % time it takes for the antenna to change its elevation
direction

ant_azel = zeros(length(t),2);  % instructions for the tracker motors
ant_azel(1,:) = [azim,0];       % set first position
ant_azel(:,1) = ant_az_step*round(azim/ant_az_step)*ones(length(t),1); % set all azims to azim

simazel = zeros(length(t),2);   % only for vizualizing
simazel(1,:) = [azim,0];       % set first position
simazel(:,1) = ant_az_step*round(azim/ant_az_step)*ones(length(t),1); % set all azims to azim
simazel_prev = 0;
ant_is_moving = false;
for i = 2:length(t)
    % current position of the sattelite - [azim,elev(t)]
    if and(i >= tlen/2 , i <= tlen/2 + 40/f_vz)
        ant_azel(i,2) = 90;
        simazel(i,2) = 90;
        ant_is_moving = false;
        simazel_prev = 90;
    elseif i >= tlen/2

    % down
        ant_azel(i,2) = ant_el_step*(round(elev(i)/ant_el_step));
    if(ant_is_moving == true)
        simazel(i,2) = simazel(i-1,2) - ant_v_el/f_vz;
        if(simazel(i,2) <= ant_el_step*(floor(elev(i)/ant_el_step)))
            ant_is_moving = false;
            simazel(i,2) = ant_el_step*(floor(elev(i)/ant_el_step));
             if i < 700, sprintf("END Moving DOWN %i", i),end
        end
        simazel_prev = simazel(i,2);
        continue;
    end

    if(ant_is_moving == false)
        simazel(i,2) = simazel_prev;

        future_el = elev( min(i+floor((ant_t_el/2)*f_vz),length(elev)) );
```

```
        motor_el = ant_el_step*(ceil(elev(i)/ant_el_step)) - ant_v_el*(ant_t_el/2);
        if i < 650 * f_vz
                sprintf("%g, %g , %i", motor_el, future_el, i)
        end
        if(future_el <= motor_el)
            ant_is_moving = true;
            if i < 700
                sprintf("%g < %g => Moving DOWN %i", motor_el, future_el, i)
            end
        end
    end
    else
    % UP
    ant_azel(i,2) = ant_el_step*(round(elev(i)/ant_el_step));
    if(ant_is_moving == true)
        simazel(i,2) = simazel(i-1,2) + ant_v_el/f_vz;
        if(simazel(i,2) > ant_el_step*(floor(elev(i)/ant_el_step)+1))
            ant_is_moving = false;
            simazel(i,2) = ant_el_step*(floor(elev(i)/ant_el_step)+1);
        end
        simazel_prev = simazel(i,2);
        continue;
    end

    if(ant_is_moving == false)
        simazel(i,2) = ant_azel(i,2);

        future_el = elev(min(i+floor((ant_t_el/2)*f_vz),length(elev)));
        motor_el = ant_el_step*(round(elev(i)/ant_el_step)) + ant_v_el*(ant_t_el/2);
        if(future_el > motor_el)
            ant_is_moving = true;
            if i < 100
                %sprintf("%g > %g => Moving UP %i", motor_el, future_el, i)
            end
        end
    end
    end %if

end

figure(1)
subplot(312), stem(t,simazel(:,2)), hold on, plot(t,elev), hold off
xlabel("t [s]")
ylabel("Elevation [°]")
legend("Sim", "Ideal", "Konst.")
subplot(313), stem(t,simazel(:,2)), hold on, plot(t,elev), hold off
xlabel("t [s]")
ylabel("Elevation [°]")
legend("Sim", "Ideal", "Konst.")
f_vz = 1;   % sampling frequency
```

Příloha 2: Matlab script Az 0-360°, El 0-180°

```
% parameters of satellite
v_el = 0.167;       % angular rate of elevation of the satellite
t_max = 180/v_el;   % time it takes for the satellite to end its journey
azim = 0;      % initial azimuth of the satellite

t = 0:1/f_vz:t_max; % time vector
tlen = length(t);
elev = horzcat(v_el*t(1:tlen/2), -v_el*t(tlen/2+1:end) + 180);      % vector of elevations of
the satellite in time

figure(1)
subplot(311), plot(t,elev)
title("Az 0-360°, El 0-180°")
xlabel("t [s]")
ylabel("Elevation [°]")
legend("Sat")

% capabilities of the tracking system
ant_az_step = 1;      % min. angle step in deg - azimuth
ant_el_step = 1;      % min. angle step in deg - elevation
ant_v_az    = 4.5;    % max. angular rate of azimuth of the antenna
ant_v_el    = 4.5;    % max. angular rate of elevation of the antenna

ant_t_az    = ant_az_step/ant_v_az;     % time it takes for the antenna to change its azimuth
direction
ant_t_el    = ant_el_step/ant_v_el;     % time it takes for the antenna to change its elevation
direction

ant_azel = zeros(length(t),2);  % instructions for the tracker motors
ant_azel(1,:) = [azim,0];       % set first position
ant_azel(:,1) = ant_az_step*round(azim/ant_az_step)*ones(length(t),1); % set all azims to azim

simazel = zeros(length(t),2);   % only for vizualizing
simazel(1,:) = [azim,0];       % set first position
simazel(:,1) = ant_az_step*round(azim/ant_az_step)*ones(length(t),1); % set all azims to azim
ant_is_moving = false;
for i = 2:length(t)
    % current position of the sattelite - [azim,elev(t)]
    %ant_azel(i,2) = ant_el_step*round(elev(i)/ant_el_step); % simple version

    ant_azel(i,2) = ant_el_step*(round(elev(i)/ant_el_step));
    if(ant_is_moving == true)
        simazel(i,2) = simazel(i-1,2) + ant_v_el/f_vz;
        %ant_azel(i,2) = ant_el_step*(round(elev(i)/ant_el_step)+1);
        if(simazel(i,2) > ant_el_step*(floor(elev(i)/ant_el_step)+1))
            ant_is_moving = false;
            simazel(i,2) = ant_el_step*(floor(elev(i)/ant_el_step)+1);
        end
        continue;
    end

    if(ant_is_moving == false)
        simazel(i,2) = ant_azel(i,2);

        future_el = elev(min(i+floor((ant_t_el/2)*f_vz),length(elev)));
        motor_el = ant_el_step*(round(elev(i)/ant_el_step)) + ant_v_el*(ant_t_el/2);
        if(future_el > motor_el)
            ant_is_moving = true;
            if i < 100
                sprintf("%g > %g => Moving %i", motor_el, future_el, i)
            end
        end
    end

end
```

```
figure(1)
subplot(312), stem(t,simazel(:,2)), hold on, plot(t,elev), hold off
xlabel("t [s]")
ylabel("Elevation [°]")
legend("Sim", "Ideal", "Konst.")
subplot(313), stem(t,ant_azel(:,2)), hold on, plot(t,elev), hold off
xlabel("t [s]")
ylabel("Elevation [°]")
legend("Sim", "Ideal", "Konst.")
```

Příloha 3: Python aplikace

```
from astropy import units as u
from skyfield.api import Topos, load
from skyfield.api import EarthSatellite
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection
from matplotlib.dates import date2num
from matplotlib.patches import Patch
from pprint import pprint
from numpy import sign
from pathlib import Path
from dateutil import parser, tz
from functools import reduce
import json

# Load earth and time
planets = load("de421.bsp")
earth = planets["earth"]
ts = load.timescale()

def load_input():
    try:
        with open("./input.json") as f:
            data = json.load(f)
            now = datetime.now(tz.tzlocal())
            tomorrow = now + timedelta(days=1)
            t0  =  ts.from_datetime(parser.parse(data["startTime"],  default=now))
            t1  =  ts.from_datetime(parser.parse(data["endTime"],  default=tomorrow))
            antenna_position = Topos(data["antennaPosition"]["lat"],
data["antennaPosition"]["lon"],  elevation_m=data["antennaPosition"]["elevation"])
            return (t0, t1, data["satellite"], antenna_position, data["beamwidth"],
data["azimuthTurnRate"], data["elevationTurnRate"], data["sampleRate"],
data["bothAxis"], data["outputFile"])
    except FileNotFoundError:
        print("File \"input.json\" not found")
    except Exception as e:
        print(e)
        print("Error while parsing \"input.json\"")

(t0, t1, satellite_name, antenna_position, beamwidth, azimuth_turn_rate,
elevation_turn_rate, sample_rate, both_axis, output_file) = load_input()

# granularity for turning of the antena
azimuth_turn_granularity = 1
elevation_turn_granularity = 1

# loading TLE files from celestrak
stations_url = "https://www.celestrak.com/NORAD/elements/active.txt"
satellites = load.tle_file(stations_url, reload=False)
print("Loaded", len(satellites), "satellites")

by_name = {sat.name: sat for sat in satellites}
satellite = by_name[satellite_name]

if not satellite:
    print(f"Could not find satellite {satellite_name} in the celestrak TLE file")
    exit()

print(f"Selected satellite: {satellite.name}")
```

```
56
57 # Finding rising and setting times
58 t, events = satellite.find_events(antenna_position, t0, t1, altitude_degrees=0.0)
59 ranges = []
60 start = None
61 for ti, event in zip(t, events):
62     if event == 0:
63         start = ti
64     elif event == 2:
65         ranges.append((start,  ti))
66
67     name = ("rise above horizon", "culminate", "set below horizon")[event]
68
69 def  calculate_rotation_diff(sourceAngle,  targetAngle):
70     diff = sourceAngle - targetAngle
71     if (abs(diff) > 180):
72         diff = sourceAngle - (targetAngle + sign(diff) * 360)
73     return diff
74
75 def round_to_multiple(n, x):
76     return x * round(n / x)
77
78 class Antenna:
79     def __init__(self, position, beamwidth, azimuth_turn_rate, elevation_turn_rate,
   azimuth_turn_granularity,  elevation_turn_granularity,  both_axis):
80         self.position = position
81         self.beamwidth = beamwidth
82         self.azimuth_turn_rate = azimuth_turn_rate
83         self.elevation_turn_rate  =  elevation_turn_rate
84         self.azimuth_turn_granularity  =  azimuth_turn_granularity
85         self.elevation_turn_granularity  =  elevation_turn_granularity
86         self.azimuth = 0.0
87         self.elevation = 0.0
88         self.both_axis = both_axis
89
90     def turn(self, azimuth, elevation):
91         azimuth_step = round_to_multiple(min(abs(azimuth), self.azimuth_turn_rate),
   self.azimuth_turn_granularity)
92         elevation_step = round_to_multiple(min(abs(elevation),
   self.elevation_turn_rate), self.elevation_turn_granularity)
93
94         if both_axis or abs(azimuth) >= abs(elevation) and abs(azimuth) >=
   self.azimuth_turn_granularity / 2:
95             self.azimuth += sign(azimuth) * azimuth_step
96             self.azimuth %= 360
97
98         if both_axis or abs(elevation) > abs(azimuth) and abs(elevation) >=
   self.elevation_turn_granularity / 2:
99             self.elevation += sign(elevation) * elevation_step
100            self.elevation %= 360
101
102 antenna = Antenna(antenna_position, beamwidth, azimuth_turn_rate,
   elevation_turn_rate, azimuth_turn_granularity, elevation_turn_granularity,
   both_axis)
103 entries = []
104
105 # Simulating antenna tracking
106 for start, end in ranges:
107     tx = start
108
```

```
    print("from", start.utc_datetime().strftime("%Y-%m-%d %H:%M:%S"), "to", end.utc_datetime().strftime("%Y-%m-%d %H:%M:%S"))

    difference = satellite - antenna_position
    topocentric = difference.at(start)
    elevation_diff, azimuth_diff, distance = topocentric.altaz()

    # reset antenna azimuth and elevation
    antenna.azimuth = round_to_multiple(azimuth_diff.degrees, antenna.azimuth_turn_granularity)
    antenna.elevation = 0.0

    event_entry = []

    while tx.utc_datetime() < end.utc_datetime():
        # Calculate target azimuth and elevation
        difference = satellite - antenna_position
        topocentric = difference.at(tx)
        elevation, azimuth, distance = topocentric.altaz()

        # Calculate error and update antenna position
        azimuth_diff = calculate_rotation_diff(azimuth.degrees, antenna.azimuth)
        elevation_diff = calculate_rotation_diff(elevation.degrees, antenna.elevation)

        azimuth_in_beamwidth = bool(abs(azimuth_diff) < beamwidth / 2)
        elevation_in_beamwidth = bool(abs(elevation_diff) < beamwidth / 2)

        event_entry.append((tx.utc_datetime(), antenna.azimuth, antenna.elevation, azimuth.degrees, elevation.degrees, azimuth_diff, elevation_diff, azimuth_in_beamwidth, elevation_in_beamwidth))

        antenna.turn(azimuth_diff, elevation_diff)

        # next sample
        tx += timedelta(seconds = sample_rate)

    entries.append(event_entry)

def map_entry(entry):
    return {
        "time": entry[0].strftime("%Y-%m-%d %H:%M:%S"),
        "antennaAzimuth": entry[1],
        "antennaElevation": entry[2],
        "realAzimuth": entry[3],
        "realElevation": entry[4],
        "azimuthError": entry[5],
        "elevationError": entry[6],
        "azimuthInBeamwidth": entry[7],
        "elevationInBeamwidth": entry[8],
    }

if output_file:
    serializable_entries = [list(map(map_entry, entry)) for entry in entries]

    # dump data in json
    json_data = json.dumps(serializable_entries)
    with open('data.json', 'w') as f:
        f.write(json_data)
```

```
# create directory for plots if not exists
path = Path("./plots")
path.mkdir(parents=True,  exist_ok=True)

# plot the results
for index, entry in enumerate(entries):
    time, antenna_azimuth, antenna_elevation, real_azimuth, real_elevation, azimuth_diff, elevation_diff, azimuth_in_beamwidth, elevation_in_beamwidth = zip(*entry)
    time_outside_beamwidth = reduce(lambda acc, entry: acc + (sample_rate if not entry[7] or not entry[8] else 0), entry, 0)

    fig, (azimuth_ax1, azimuth_ax2) = plt.subplots(2, 1, sharex=True, dpi=400, figsize=(20, 10))
    plt.tight_layout(pad=4.0)

    azimuth_ax1.title.set_text("Azimuth")
    azimuth_ax1.plot(time,  real_azimuth,  label="ideal")
    azimuth_ax1.plot(time,  antenna_azimuth,  label="simulation")
    azimuth_ax1.legend()
    azimuth_ax1.set_ylabel("deg")

    azimuth_ax2.title.set_text(f"Azimuth  Error  (outside  beamwidth: {time_outside_beamwidth}s)")
    colors = ["b" if visible else "r" for visible in azimuth_in_beamwidth]
    lines = [((date2num(x0), y0), (date2num(x1), y1)) for x0, y0, x1, y1 in zip(time[:-1],  azimuth_diff[:-1], time[1:], azimuth_diff[1:])]
    line_collection = LineCollection(lines, colors = colors)
    azimuth_ax2.add_collection(line_collection)
    azimuth_ax2.autoscale_view()
    azimuth_ax2.legend(handles=[Patch(color='b', label='in beamwidth'), Patch(color='r', label='not in beamwidth')])
    azimuth_ax2.set_ylabel("deg")

    plt.setp(azimuth_ax2.get_xticklabels(), rotation=30, horizontalalignment='right')
    plt.savefig(f"./plots/azimuth-{index}.pdf")
    plt.close(fig)

    fig, (elevation_ax1, elevation_ax2) = plt.subplots(2, 1, sharex=True, dpi=400, figsize=(20, 10))
    plt.tight_layout(pad=4.0)

    elevation_ax1.title.set_text("Elevation")
    elevation_ax1.plot(time,  real_elevation,  label="ideal")
    elevation_ax1.plot(time,  antenna_elevation,  label="simulation")
    elevation_ax1.legend()
    elevation_ax1.set_ylabel("deg")

    elevation_ax2.title.set_text(f"Elevation  Error  (outside  beamwidth: {time_outside_beamwidth}s)")
    colors = ["b" if visible else "r" for visible in elevation_in_beamwidth]
    lines = [((date2num(x0), y0), (date2num(x1), y1)) for x0, y0, x1, y1 in zip(time[:-1], elevation_diff[:-1], time[1:], elevation_diff[1:])]
    line_collection = LineCollection(lines, colors = colors)
    elevation_ax2.add_collection(line_collection)
    elevation_ax2.autoscale_view()
    elevation_ax2.legend(handles=[Patch(color='b', label='in beamwidth'), Patch(color='r', label='not in beamwidth')])
    elevation_ax2.set_ylabel("deg")
```

```
212     elevation_ax2.set_xlabel("time")
213 
214     plt.setp(azimuth_ax2.get_xticklabels(),
    rotation=30, horizontalalignment='right')
215     plt.setp(elevation_ax2.get_xticklabels(),  rotation=30,
    horizontalalignment='right')
216     plt.savefig(f"./plots/elevation-{index}.pdf")
217     plt.close(fig)
218 
219 
```