worksheet_tutorial_02

May 30, 2022

Integrierte Navigation - Tutorial 2: Micro Electro Mechanical Systems (MEMS)

1 Description of the task

In this tutorial you'll be thought to post-process inertial measurement data. The lowest accuracy you'll get when you process data as they are. There are ways to get more accurate soltions (e.g. calibration of the IMU and Zero Velocity updates).

We provied necessary sensors so you could make your own measurements and evaluations in small groups. You should learn the complete workflow.

In the* moodle cours* you will find a guidance, how you connect the Arduino with the MPU and how you can receive and store data for further processing.

After installation from hardware and software as well as data acquisition there are severeal subtasks you have to do throughout the notebook. These tasks should help you to capture the complete workflow.

The notebook shows the workflow from data import, bias calculation and integration of the measured values to visualization and table length calculation.

Subasks are built into the individual sections. A large part of the source code is already given, which is to be supplemented by you. In task 7 you have a free hand to calculate your individual table length and to compare different sensor configurations.

2 Importing necessary modules

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import prepare_data
import mems
```

3 Tasks

3.1 Data Upload

The recorded measurement data in an ASCII-file must be uploaded in your personal Google Drive and your jupither notebook needs to be access to your Google Drive. 1. Google Drive - upload your file via drag and drop. 2. Google Colaboratory - click on the folder icon on the left and then on the the Google Drive folder icon. 3. Google Colaboratory - allow the access in the pop up

window. 4. Google Colaboratory - navigate to your uploaded file in the file explorer left. 5. Google Colaboratory - click on the three point next to your file and copy the file path. This you will need for Task 2.

3.1.1 Task 1

Do the upload according to the instructions above or use the jupiter notenbook offline.

3.2 Import data

After the upload, we have to load the data in the notebook. It is practically to store all data in one array. The values for time, acceleration and turn rates seperated in differend collums and for each timestamp in seperate rows.

> ***Note:*** The timestamp is not unixtime, rather it is the time in milliseconds from the beginning of the script.

3.2.1 Task 2

Programm the code for the import of your measurement data:

> ***Tipp:*** Change the [...] to your own values.
>
> ***Tipp:*** For that purpose you can use the function `loadtxt` from the `numpy` library.

[ ]:
```
# location = '...your file path from task 1 point 5...'
# data = np.loadtxt(location, delimiter='...', skiprows=...)
# data[:,0] = TIME | data[:,1] = ACC_X | data[:,2] = ACC_Y | data[:,3] = ACC_Z␣
↪| data[:,4] = GYR_X | data[:,5] = GYR_Y | data[:,6] = GYR_Z
files = ["data_track_01.csv",
         "data_track_02.csv",
         "data_track_03.csv",
         "data_track_04.csv",
         "data_track_05.csv",
         "data_track_06.csv",
         "data_track_07.csv",
         "data_track_08.csv",
         "data_track_09.csv",
         "data_track_10.csv",
         "data_track_12.csv",
         "data_track_13.csv",
         "data_turntable_01.csv",
         "data_turntable_02.csv",
         "data_turntable_03.csv",
         "data_turntable_04.csv",
         "data_turntable_05.csv",
         "data_turntable_06.csv",
         "data_turntable_07.csv",
         "data_turntable_08.csv",
         "data_turntable_09.csv",
```

```
        "data_turntable_10.csv",
        "data_turntable_11.csv",
        "data_turntable_12.csv"]
measurements = []
for filename in files:
    measurements.append(mems.import_data(filename))
```

3.3 Bias determination

After the import of the data, we have to determin the bias from a small dataset of the first values of the measurement. For that purpose you made the short initialisation break (approx. 10 to 20 sconds) in front of the movement along the table edge.

3.3.1 Task 3

Programm the code for the bias determination of your measurement data:

> ***Tipp:*** Change the [...] to your own values.
>
> ***Tipp:*** The `numpy` library has also for that task an easy solution.

[ ]:
```
# value = ...
# bias = np.array([np.mean(data[1:value,1]), np.mean(data[1:value,2]), ... ])
# bias[:,0] = ACC_X | bias[:,1] = ACC_Y | bias[:,2] = ACC_Z | bias[:,3] = GYR_X␣
 ↪| bias[:,4] = GYR_Y | bias[:,5] = GYR_Z
# Acceleration in [m/s^2]
# Turning rates in [°/s]

stationary_parts = [{"before": {"start": 15, "end": 710}, "after": {"start":␣
 ↪1125, "end": 1654}},
                    {"before": {"start": 7, "end": 597}, "after": {"start":␣
 ↪973, "end": 1320}},
                    {"before": {"start": 11, "end": 485}, "after": {"start":␣
 ↪923, "end": 1691}},
                    {"before": {"start": 7, "end": 522}, "after": {"start":␣
 ↪772, "end": 1467}},
                    {"before": {"start": 7, "end": 250}, "after": {"start":␣
 ↪821, "end": 1321}},
                    {"before": {"start": 16, "end": 300}, "after": {"start":␣
 ↪939, "end": 1391}},
                    {"before": {"start": 16, "end": 509}, "after": {"start":␣
 ↪1438, "end": 2041}},
                    {"before": {"start": 34, "end": 548}, "after": {"start":␣
 ↪1914, "end": 2338}},
                    {"before": {"start": 14, "end": 556}, "after": {"start":␣
 ↪1535, "end": 2229}},
                    {"before": {"start": 23, "end": 432}, "after": {"start":␣
 ↪1931, "end": 2294}},
```

```
                    {"before": {"start": 17, "end": 576}, "after": {"start":␣
↪1658, "end": 1999}},
                    {"before": {"start": 23, "end": 392}, "after": {"start":␣
↪1503, "end": 2348}},
                    {"before": {"start": 452, "end": 2574}, "after": {"start":␣
↪5262, "end": 5583}},
                    {"before": {"start": 38, "end": 1427}, "after": {"start":␣
↪3621, "end": 3802}},
                    {"before": {"start": 24, "end": 768}, "after": {"start":␣
↪2365, "end": 2840}},
                    {"before": {"start": 17, "end": 589}, "after": {"start":␣
↪2250, "end": 2406}},
                    {"before": {"start": 12, "end": 617}, "after": {"start":␣
↪1642, "end": 1879}},
                    {"before": {"start": 21, "end": 1967}, "after": {"start":␣
↪2901, "end": 3004}},
                    {"before": {"start": 7, "end": 140}, "after": {"start":␣
↪1405, "end": 1645}},
                    {"before": {"start": 4, "end": 213}, "after": {"start":␣
↪1249, "end": 1304}},
                    {"before": {"start": 10, "end": 818}, "after": {"start":␣
↪1762, "end": 1834}},
                    {"before": {"start": 9, "end": 389}, "after": {"start":␣
↪1760, "end": 1820}},
                    {"before": {"start": 6, "end": 354}, "after": {"start":␣
↪1245, "end": 1346}},
                    {"before": {"start": 6, "end": 202}, "after": {"start":␣
↪1158, "end": 1406}}]

accelerometer_all = []
gyroscope_all = []
timestamps_all = []
accelerometer_bias_all = []
gyroscope_bias_all = []
accelerometer_without_bias_all = []
gyroscope_without_bias_all = []

# Put Measurement-data into data-streams for gyroscope and accelerometer
for measurement_id, data in enumerate(measurements):
    accelerometer = {"x": [], "y": [], "z": []}
    gyroscope = {"x": [], "y": [], "z": []}
    timestamps = []
    for sensor_info in data:
        timestamps.append(sensor_info[0]/1000)
        for j, e in enumerate(["x", "y", "z"]):
            accelerometer[e].append(sensor_info[j+1])
```

```
            gyroscope[e].append(sensor_info[j+4])

    # Determine the biases before and after the movement for the accelerometer
    accelerometer_bias = {"before": {"x": 0.0, "y": 0.0, "z": 0.0},
                          "after": {"x": 0.0, "y": 0.0, "z": 0.0}}
    for i in ["before", "after"]:
        for xyz in ["x", "y", "z"]:
            accelerometer_bias[i][xyz] = mems.calc_offset(accelerometer[xyz],
                                                          ␣
↪stationary_parts[measurement_id][i]["start"],
                                                          ␣
↪stationary_parts[measurement_id][i]["end"])

    # Determine the biases before and after the movement for the gyroscope
    gyroscope_bias = {"before": {"x": 0.0, "y": 0.0, "z": 0.0},
                      "after": {"x": 0.0, "y": 0.0, "z": 0.0}}
    for i in ["before", "after"]:
        for xyz in ["x", "y", "z"]:
            gyroscope_bias[i][xyz] = mems.calc_offset(gyroscope[xyz],
                                                      ␣
↪stationary_parts[measurement_id][i]["start"],
                                                      ␣
↪stationary_parts[measurement_id][i]["end"])

    accelerometer_without_bias = {"x": [], "y": [], "z": []}
    gyroscope_without_bias = {"x": [], "y": [], "z": []}

    for xyz in ["x", "y", "z"]:
        accelerometer_without_bias[xyz] = mems.
↪remove_bias_advanced(accelerometer[xyz],
                                                              ␣
↪accelerometer_bias["before"][xyz],
                                                              ␣
↪accelerometer_bias["after"][xyz],
                                                              ␣
↪stationary_parts[measurement_id]["before"]["end"],
                                                              ␣
↪stationary_parts[measurement_id]["after"]["start"])
        gyroscope_without_bias[xyz] = mems.remove_bias_advanced(gyroscope[xyz],
                                                                ␣
↪gyroscope_bias["before"][xyz],
                                                                ␣
↪gyroscope_bias["after"][xyz],
                                                                ␣
↪stationary_parts[measurement_id]["before"]["end"],
```

```
↪stationary_parts[measurement_id]["after"]["start"])

    accelerometer_all.append(accelerometer)
    gyroscope_all.append(gyroscope)
    timestamps_all.append(timestamps)
    accelerometer_bias_all.append(accelerometer_bias)
    gyroscope_bias_all.append(gyroscope_bias)
    accelerometer_without_bias_all.append(accelerometer_without_bias)
    gyroscope_without_bias_all.append(gyroscope_without_bias)

# print("Biases of the Accelerometer:")
# for i in accelerometer_bias_all:
#     print(f'before: {i["before"]["x"]:15.10f} {i["before"]["y"]:15.10f}
↪{i["before"]["z"]:15.10f}    after: {i["after"]["x"]:15.10f}
↪{i["after"]["y"]:15.10f} {i["after"]["z"]:15.10f}')

# print("Biases of the Gyroscope:")
# for i in gyroscope_bias_all:
#     print(f'before: {i["before"]["x"]:15.10f} {i["before"]["y"]:15.10f}
↪{i["before"]["z"]:15.10f}    after: {i["after"]["x"]:15.10f}
↪{i["after"]["y"]:15.10f} {i["after"]["z"]:15.10f}')
```

3.4 Integration of acceleration

3.4.1 Velocity

Now we are going to integrate the accelerations to velocity by the given formula:

Formula for velocity:

$$v_i = v_0 + a_i * \Delta t$$

With: * v_i: velocity in $[m/s]$ * v_0: initial velocity $[m/s]$ * a_i: acceleration in $[m/s^2]$ * Δt: time difference to previous measurement $_{i-1}$ in $[s]$

3.4.2 Position

Formula for position:

$$s_i = s_0 + 0.5 * a_i * \Delta t^2 + v_0 * \Delta t$$

With: * s_i: distance $[m]$ * s_0: initial position $[m]$ * a_i: acceleration in $[m/s^2]$ * Δt: time difference to previous measurement $_{i-1}$ in $[s]$ * v_0: initial velocity $[m/s]$

3.4.3 Task 4

Programm the code for the integration of acceleration to velocity and positions by the formulaes above:

Tipp: Change the [...] to your own values.

Tipp: For this task with many iterations, `loops` makes your live easier.

[ ]:
```
"""
x_v = np.zeros((3, len(data[:,0])))
x_d = np.zeros((3, len(data[:,0])))

for i in range(1, len(data[:,0])):
    dt = ...

    # velocity [m/s]
    x_v[0,i] = ...
    x_v[1,i] = ...
    x_v[2,i] = ...

    # distances [m]
    x_d[0,i] = ...
    x_d[1,i] = ...
    x_d[2,i] = ...
"""
velocity_all = []
velocity_without_bias_all = []
position_all = []

for measurement_id, e in enumerate(timestamps_all):
    velocity_all.append(mems.
 ↪calc_velocity(accelerometer_without_bias_all[measurement_id], e))

    # Determine offset of velocity before and after movement
    velocity_offset = {"before": {"x": 0.0, "y": 0.0, "z": 0.0},
                       "after": {"x": 0.0, "y": 0.0, "z": 0.0}}
    for i in ["before", "after"]:
        for xyz in ["x", "y", "z"]:
            velocity_offset[i][xyz] = mems.
 ↪calc_offset(velocity_all[measurement_id][xyz],
                                                          ␣
 ↪stationary_parts[measurement_id][i]["start"],
                                                          ␣
 ↪stationary_parts[measurement_id][i]["end"])

    # Remove offset of velocity before and after movement
    velocity_without_bias = {"x": [], "y": [], "z": []}
    for xyz in ["x", "y", "z"]:
        velocity_without_bias[xyz] = mems.
 ↪remove_bias_advanced(velocity_all[measurement_id][xyz],
```

```
                                                                        ␣
↪velocity_offset["before"][xyz],
                                                                        ␣
↪velocity_offset["after"][xyz],
                                                                        ␣
↪stationary_parts[measurement_id]["before"]["end"],
                                                                        ␣
↪stationary_parts[measurement_id]["after"]["start"])
    velocity_without_bias_all.append(velocity_without_bias)

    # Calculate the position
    position_all.append(mems.
↪calc_position(accelerometer_without_bias_all[measurement_id], e,␣
↪velocity_without_bias))
```

3.5 Integration of turn rates

Next, we are going to intgrate the turn rates to angles with the formula for rotation angle:

$$r_i = r_{i-1} + \omega_i * \Delta t$$

With: * r_i: rotation angle * Δt: time difference to previous measurement $_{i-1}$ in $[s]$ * ω_i: rotation rate

3.5.1 Task 5

Programm the code for the integration of turn rates to angles by the formulaes above:

> ***Tipp:*** Change the [...] to your own values.
>
> ***Tipp:*** Also `loops` saves lifetime for this task.

[ ]:
```
# euler = np.zeros((3, len(data[:,0])))

# for i in range(1, len(data[:,0])):
#     dt = ...

#     # degree [°]
#     euler[0,i] = ... # phi
#     euler[1,i] = ... # theta
#     euler[2,i] = ... # psi

rotation_all = []

for measurement_id, e in enumerate(timestamps_all):
    rotation_all.append(mems.
↪calc_rotation(gyroscope_without_bias_all[measurement_id], e))
```

```
trajectory_all = []

for measurement_id, e in enumerate(timestamps_all):
    trajectory_all.append(mems.calc_trajectory(position_all[measurement_id],␣
 ↪rotation_all[measurement_id]))
```

3.6 Data visualisation

3.6.1 Task 6

Programm the code for the visualisation of your results. Make three seperate plots for the velocities, the positions and the turn rates. The plots should contain a title, axis labels and a legend.

> ***Tipp:*** Change the [...] to your own values and adapt it for the following plots.
>
> ***Tipp:*** The library `matplotlib` can be used to create fancy plots.

3.6.2 Angles

[ ]:
```
# plt.plot(data[:,0]/1000, euler[0,:], label='phi')
# plt.plot(...)
# plt.plot(...)
# plt.title('...')
# plt.xlabel('...')
# plt.ylabel('...')
# plt.legend()
# plt.grid()
# plt.show()

for measurement_id, e in enumerate(timestamps_all):
    mems.plot_results([rotation_all[measurement_id]["x"],␣
 ↪rotation_all[measurement_id]["y"], rotation_all[measurement_id]["z"]],
                      f'Rotation from {files[measurement_id]}',
                      "time [s]",
                      "rotation [°]",
                      ["x", "y", "z"],
                      e)
```

3.6.3 Velocity

[ ]:
```
for measurement_id, e in enumerate(timestamps_all):
    mems.plot_results([velocity_all[measurement_id]["x"],␣
 ↪velocity_all[measurement_id]["y"], velocity_all[measurement_id]["z"]],
                      f'Velocity from {files[measurement_id]} (without␣
 ↪offset-removal)',
                      "time [s]",
                      "velocity [m/s]",
                      ["x", "y", "z"],
                      e)
```

```
    mems.plot_results([velocity_without_bias_all[measurement_id]["x"],␣
↪velocity_without_bias_all[measurement_id]["y"],␣
↪velocity_without_bias_all[measurement_id]["z"]],
                      f'Velocity from {files[measurement_id]} (with the offset␣
↪removed)',
                      "time [s]",
                      "velocity [m/s]",
                      ["x", "y", "z"],
                      e)
```

3.6.4 Position

[ ]:
```
for measurement_id, e in enumerate(timestamps_all):
    mems.plot_results([position_all[measurement_id]["x"],␣
↪position_all[measurement_id]["y"], position_all[measurement_id]["z"]],
                      f'Position from {files[measurement_id]}',
                      "time [s]",
                      "position [m]",
                      ["x", "y", "z"],
                      e)
```

3.6.5 Other plots

[ ]:
```
for measurement_id, e in enumerate(timestamps_all):
    mems.plot_results([gyroscope_all[measurement_id]["x"],␣
↪gyroscope_all[measurement_id]["y"], gyroscope_all[measurement_id]["z"]],
                      f'Raw gyroscope-data from {files[measurement_id]}',
                      "time [s]",
                      "rotation change [°/s]",
                      ["x", "y", "z"],
                      e)
    mems.plot_results([accelerometer_all[measurement_id]["x"],␣
↪accelerometer_all[measurement_id]["y"],␣
↪accelerometer_all[measurement_id]["z"]],
                      f'Raw accelerometer-data from {files[measurement_id]}',
                      "time [s]",
                      "acceleration [m/s²]",
                      ["x", "y", "z"],
                      e)
    mems.plot_results([gyroscope_without_bias_all[measurement_id]["x"],␣
↪gyroscope_without_bias_all[measurement_id]["y"],␣
↪gyroscope_without_bias_all[measurement_id]["z"]],
                      f'Gyroscope-data from {files[measurement_id]} with bias␣
↪removed',
                      "time [s]",
                      "rotation change [°/s]",
                      ["x", "y", "z"],
```

```
                    e)
    mems.plot_results([accelerometer_without_bias_all[measurement_id]["x"],␣
↪accelerometer_without_bias_all[measurement_id]["y"],␣
↪accelerometer_without_bias_all[measurement_id]["z"]],
                    f'Accelerometer-data from {files[measurement_id]} with bias␣
↪removed',
                    "time [s]",
                    "acceleration [m/s²]",
                    ["x", "y", "z"],
                    e)
    mems.plot_results([trajectory_all[measurement_id]["x"]],
                    f'Trajectory from {files[measurement_id]}',
                    "Y",
                    "X",
                    ["trajectory"],
                    trajectory_all[measurement_id]["y"])
```

3.7 Table length determination

In the plot above regarding the position, hopefully you can see a straigt rising edge in one axis (x or y). This represents your moving direction along the table edge.

On basis of drifts, there is also a movement along the other axes possible and visible.

Now you can determin the table length with your measurement data. After that you can compare the result with the original length of your table. At least you can do an asumption regarding the quality of your result with respect to the limits of the used system.

3.7.1 Task 7

Programm the code for the determination of your table length.

> ***Tipp:*** Change the [...] to your own values.
>
> ***Tipp:*** Try again with the `numpy` library.

[ ]:
```
# table length with Pythagorean theorem
# table_length = np.sqrt(x_d[0,-1:]**2 + ... + ...)

# 'The length of my table is in real ... meter, but I have determined a length␣
↪with my measurement from {:2.3} meter!'.format(float(table_length))

distance_all = []

for measurement_id, e in enumerate(timestamps_all[0:12]):
    distance = mems.calc_distance(position_all[measurement_id],␣
↪stationary_parts[measurement_id]["before"],␣
↪stationary_parts[measurement_id]["after"])
    distance_all.append(distance)
```

```
    print(f'The distance for {files[measurement_id]} is {distance:.3f} m.')
```