

## **Course description Data Fusion Architectures (DFA)**

### **Content**

The course covers the following topics: sensor fusion architectures, common representational format, temporal and spatial alignment, error propagation analysis and device virtualization.

### **Learning Outcomes**

At the end of this module the student is able to:

- create data processing software for a sensor system which combines multi-sensor data into correct interpretations.
- analyse system requirements to identify applicable sensor fusion architectures.
- analyse data acquisition by a sensor system to construct a common representational format.
- adapt sensor system design to create correct temporal and spatial alignment and normalization of multi-sensor data.'
- design interfaces for sensor system management.
- apply error propagation
- explain where to use device virtualization.
- design and implement a system architecture using data stream management systems
- design and implement model-based reasoning in a stream observing expert system.

**Note:** These learning outcomes contribute to: SSE1: Giving meaning to Sensor Data and SSE2: Building Intelligent Architectures.

## Assignments

The class assignment contributes 10% to the final grade, while the practical assignment accounts for the remaining 90%. The practical assignment is divided into two parts:

### Part1: Data Stream Management System practical

#### Administrative matters:

due date:

you can work alone or in a pair. If you work together clearly state both your names at the top of each python, jupyter and markdown file and include them in your group name.

*This assignment constitutes 15% of the final grade.*

#### Before we begin:

##### Setting up MQTT (On Raspberry Pi)

MQTT installation

Specifies the MQTT broker version, check for a more recent version on: Eclipse Mosquitto download page

General website: <https://mosquitto.org>

##### Setting up Python API

General website: paho-mqtt documentation

General website: psutil documentation

#### What to do:

Process the CPU sensors (e.g., temperature) and Sense HAT data using MQTT.

**Program 1:** Write a program which publishes CPU performance data using paho-mqtt and psutil. It should publish a set of performance indicators during operation of the Bayesian model. Select a relevant subset of parameters from the psutil documentation. The program should publish readings every 10 ms (if possible) and each reading should be timestamped.

**Program 2:** Write a program which publishes Sense HAT data. It should publish the predicted location from the Bayesian algorithm. The

program should publish readings every 10 ms (if possible) and each reading should be timestamped.

**Program 3:** Write a program that subscribes to topics containing the CPU performance data (subscriber). Give a window for averaging and start two instances with different averaging windows.

**Program 4:** Write a program that subscribes to topics containing the CPU performance . Use (naive) Bernoulli sampling to use approximately 1/3 of the data points to estimate averages over the same averaging windows as used in the previous step.

**Finally:** Think of two simple rules to apply to detect malfunctioning of the Raspberry Pi.

## **Part2: IMU Assignment**

### **Administrative matters:**

due date:

you can work alone or in a pair. If you work together clearly state both your names at the top of each python, jupyter and markdown file and include them in your group name.

This assignment constitutes ***75% of the final grade.***

### **What you should do:**

#### **Reproducing a pedestrian inertial navigation system (reference 3)**

Localization in space and time is a key challenge when building sensor systems. In simple terms, it means determining where something is (its position in space) and when something happens (its position in time). Medical staff might call this “orientation in space and time,” referring to a person’s awareness of where they are and what time it is. For machines, this awareness comes from sensors. Outdoors, the

most reliable method for localization is GPS, as it provides both accurate position and timing information. However, GPS can experience latency or signal loss, and the initial acquisition of satellite data may take up to 30 seconds. Once operational, position

updates are typically available every second, but these can still be affected by noise or temporary outages. To obtain more precise and continuous estimates of both time and location, additional information from other sensors is required

In this project, we will use a Raspberry Pi equipped with a Sense HAT. The Sense HAT includes an Inertial Measurement Unit (IMU) that measures rotation (gyroscope) and acceleration (accelerometer). These measurements will be used by an algorithm, described in section II.C of the reference<sup>3</sup>, which will run directly on the Raspberry Pi. The goal of the algorithm is to estimate the most likely state of the system, for example, its position, orientation, and movement, based on the noisy sensor readings from the IMU.

This process of finding the most probable state is known as mode-seeking. It involves identifying the point in the state space where the probability distribution is highest, in other words, where the system is most likely to be. In Python, this can be done efficiently using functions from the `scipy.optimize` module, which are designed to find maxima or minima of functions. From a computational perspective, it is important to avoid calculating probabilities for every possible state, as this would be extremely inefficient and time-consuming. Instead, optimization techniques or probabilistic filters are used to narrow down the search and quickly identify the most probable state.

In addition, a particle filter should be implemented and compared to the algorithm from section II.C. While particle filters can be more accurate they also require significantly more computational resources. The comparison between the two approaches will focus on how they perform under the hardware constraints of the Raspberry Pi. In particular, we aim to explore the trade-off between accuracy and computational efficiency, identifying in which conditions one method outperforms the other, and how hardware limitations influence the achievable precision and real-time performance.

Please note that the algorithm outlined in the first paper is intended for a pedestrian and uses body movement to extract a stride. They used, Zero-Velocity Update (ZUPT) is a technique used in inertial navigation systems to reduce drift by detecting when the IMU (usually on the foot) is stationary during walking. The idea is When you walk the foot alternates between moving and stance phase. Whenever the foot is stationary, it resets the IMU's estimated velocity to zero, correcting accumulated drift from previous integration steps. This is out of scope for this assignment and should be replaced with a simple detection mechanism or simulated with a button push.

IMU orientation estimation, step detection, and velocity correction must be executed in real time. The same applies to the Bayesian map update, which should use a static floor plan probability density function (PDF) for heading correction and support live position monitoring.

Any limitations of the recursive Bayesian filter or particle filter that could slow down or prevent real-time operation must be clearly identified. Proposed solutions to these limitations should be implemented, tested on the board, and explained in detail.

In summary, the aim is to implement and compare at least two localization algorithms on the Raspberry Pi, the algorithm from section II.C and a particle filter to estimate orientation in space and time (and/or Kalman filter).

In addition, at least one linear Kalman filter should be included in the analysis to filter the data extracted from the accelerometer, gyroscope or other sensors.

**What to hand in:**

- Python code to run the experiment
- Jupyter notebook containing:

The visualized output

All the requirements discussed in the assignment description

Implementation of Bayes and particle filter

Implementation of linear Kalman filter

Clear indication where in the code the Bayesian rules are implemented. Inference equations also written in mathematical form and supported by a table of parameter values used.

Discussion of the impact of changes in the priors and likelihoods used on the final result.

For at least one of these the discussion should be illustrated with an experiment showing the impact.

You can also use the classical error analysis to evaluate the impact of some errors. Extra points from the rubric there.

## **References :**

1. A primer on Bayesian inference for biophysical systems, KE Hines, Biophysical Journal, (2015) ([link](#))
2. Data Analysis: A Bayesian Tutorial, (2nd Edition), D.S. Sivia with J. Skilling, Oxford Science Publications, 2006.
3. **Pedestrian inertial navigation with building floor plans for indoor environments via non-recursive Bayesian filtering, M. T. Koroglu and A. Yilmaz, IEEE SENSORS (2017)** ([link](#))
4. Indoor Pedestrian Navigation using an INS/EKF framework for Yaw Drift Reduction and a Foot-mounted IMU, A.R. Jiménez, F. Seco, J.C. Prieto and J. Guevara, 7th Workshop on Positioning, Navigation and Communication (2010) ([link](#))
5. Mean shift, mode seeking, and clustering, Y. Cheng, IEEE Transactions on Pattern Analysis and Machine Intelligence (1995) ([link](#))
6. Pedestrian Tracking with Shoe-Mounted Inertial Sensor, E. Foxlin, Intersense, (2005) ([link](#))

## Rubric IMU

<b>Assessment Aspect</b>	<b>Weight</b>	<b>Excellent (10)</b>	<b>Good (8)</b>	<b>Fair (5.5)</b>	<b>Insufficient (3)</b>
Create data processing software for a sensor system which combines multi-sensor data into correct interpretations.	0.35	Working code with comments correctly explaining what the code is trying to achieve. Where applicable pythonesque constructions are used for speeding up processing.	Working code with comments correctly explaining what the code is trying to achieve.	Working code.	Code writing attempted but visible errors are present.
Analyse system requirements to identify applicable sensor fusion architectures.	0.1	Three different architecture categorizations are used and applied correctly to describe proposed designs. The categorization are compared regarding their applicability for sensor fusion problem at hand.	Three different architecture categorizations are used and applied correctly to describe proposed designs.	Two different architecture categorizations are used and applied correctly to describe proposed designs .	Different architecture categorizations are used and applied but there are visible errors in the application .
Analyse data acquisition by a sensor system to construct a common representational format.	0.15	A correct common representational format is proposed and implemented in the code. Based on an evaluation of the implementation in the code improvements for a future implementation are proposed with correct motivation.	A correct common representational format is proposed and implemented in the code.	A correct common representational format is proposed.	A common representational format is proposed, but there are visible errors in the proposal.
Adapt sensor system design to create correct temporal and spatial alignment and normalization of multi-sensor data.	0.1	Correct and complete alignment. Normalization is attempted but visibly incorrect or incomplete, if normalization is not needed the absence of normalization is correctly motivated. Clear quantification of alignment differences before alignment is introduced. Different alignment strategies are discussed after which one is chosen.	Correct and complete alignment. Normalization is attempted but visibly incorrect or incomplete, if normalization is not needed the absence of normalization is correctly motivated. Clear quantification of alignment differences before alignment is introduced.	Correct and complete alignment. Normalization is attempted but visibly incorrect or incomplete, if normalization is not needed the absence of normalization is correctly motivated.	Alignment is attempted but visibly incorrect or incomplete. If needed normalization is attempted but visibly incorrect or incomplete, if not needed the absence of normalization is not motivated.
Design interfaces for sensor system management.	0.15	Configuration files are supported and allow for initialization of important initialization parameters. The configuration options	Configuration files are supported and allow for initialization of important initialization	Configuration files are supported and allow for initialization of important	Configuration files are supported but the function is visibly incomplete or broken.

		<p>are documented and clearly explained to facilitate the user in making configurational changes. It is documented how new configuration parameters can be added.</p>	<p>parameters. The configuration options are documented and clearly explained to facilitate the user in making configurational changes.</p>	<p>initialization parameters.</p>	
Apply error propagation techniques.	0.15	<p>Data from multiple sensors is used to correctly estimate physical parameters using Bayesian inference. An error analysis is provided showing how model assumptions influence the final outcome. The error analysis is related back to the implementation and possible improvements to the implementation are discussed.</p>	<p>Data from multiple sensors is used to correctly estimate a physical parameter using Bayesian inference. An error analysis is provided showing how model assumptions influence the final outcome.</p>	<p>Data from multiple sensors is used to correctly estimate a physical parameter using Bayesian inference.</p>	<p>Correct estimation of a physical parameter is lacking or the data is from one sensor only.</p>

## Rubric DSMS

Assessment Aspect	Weight in Final Grade	Excellent (10)	Good (8)	Fair (5.5)	Insufficient (3)
Design and implement a system architecture using data stream management systems.	0.5	<p>Correct design and implementation. Design decision are motivated and it is explained how they are translated to the implementation.</p> <p>Alternative designs are presented and it is explained why the implemented design is preferred over the other design.</p>	<p>Correct design and implementation.</p> <p>Design decision are motivated and it is explained how they are translated to the implementation.</p>	<p>Correct design and implementation.</p>	<p>Design and implementation attempted but visibly incomplete or incorrect.</p>
Design and implement model-based reasoning in a stream observing expert system	0.5	<p>Correct design and implementation. Design decision are motivated and it is explained how they are translated to the implementation.</p> <p>Alternative designs are presented and it is explained why the implemented design is preferred over the other design.</p>	<p>Correct design and implementation.</p> <p>Design decision are motivated and it is explained how they are translated to the implementation.</p>	<p>Correct design and implementation.</p>	<p>Design and implementation attempted but visibly incomplete or incorrect.</p>