[Industrial paper] Digital Twin Applications Using a Cloud Native Modelica Platform

Abhilash Kumar[1] Arunkumar Narasimhan[1] Tharrini Rajendran[1] Stéphane Velut[2]

[1]Modelon Engineering Pvt. Ltd., India, arunkumar.narasimhan@modelon.com
[2]Modelon AB, Sweden, stephane.velut@modelon.com

Abstract

This paper showcases how Modelica technology can be leveraged for real-time applications using a cloud native simulation platform, Modelon Impact™. The platform allows for real-time, two-way communication of data, from the IoT connected plant to a physical model and, from the physical model to a dashboard for plant monitoring and control. The communication relies on open standards and REST-API, which makes it possible to implement digital twins for various applications, such as plant monitoring, predictive maintenance, fault isolation or controls. The paper describes a state estimation workflow where data is transmitted back and forth to the simulation platform via Message Queuing Telemetry Transport (MQTT) and where Node-Red is used for the end-user interface.

Keywords: Digital twin, State estimation, Cloud native Modelica platform, MQTT, Node-RED, REST-API.

1 Introduction

A digital twin is a virtual representation of a real-world physical system or process (a physical twin) that serves for practical purposes, such as system simulation, integration, testing, monitoring, and maintenance. Modelica provides a clear separation between model and analysis definition which has proven successful in various applications over the years, also touching digital twins as illustrated in the following examples.

In motorsports, digital twins of the car, the track, and sometimes even the human driver are used in software-in-the-loop or hardware-in-the-loop configurations. The Modelon Vehicle Dynamics Library® (VDL, 2022) for instance has been used to define digital twins since well over a decade. Driven both by cost and regulations, a successful team in any of the higher leagues such as F1 or NASCAR have virtual representations of each of the cars they put on the racetrack. There are various applications with the common purpose to predict or estimate vehicle behavior beyond what is feasible or even allowed to investigate while the race car is driven on the racetrack.

Figure 1 shows a setup where a digital twin of the car is used to investigate the vehicle behavior on a certain part of the track in more detail. This model is used in offline as well as real-time applications. In the picture, the boundary conditions of the car are given from track and race data and includes for example track curvature, lateral acceleration, and throttle position. Since the model contains detailed representation of the race car's mechanics the workload of critical components such as tires, springs, and dampers can be estimated.

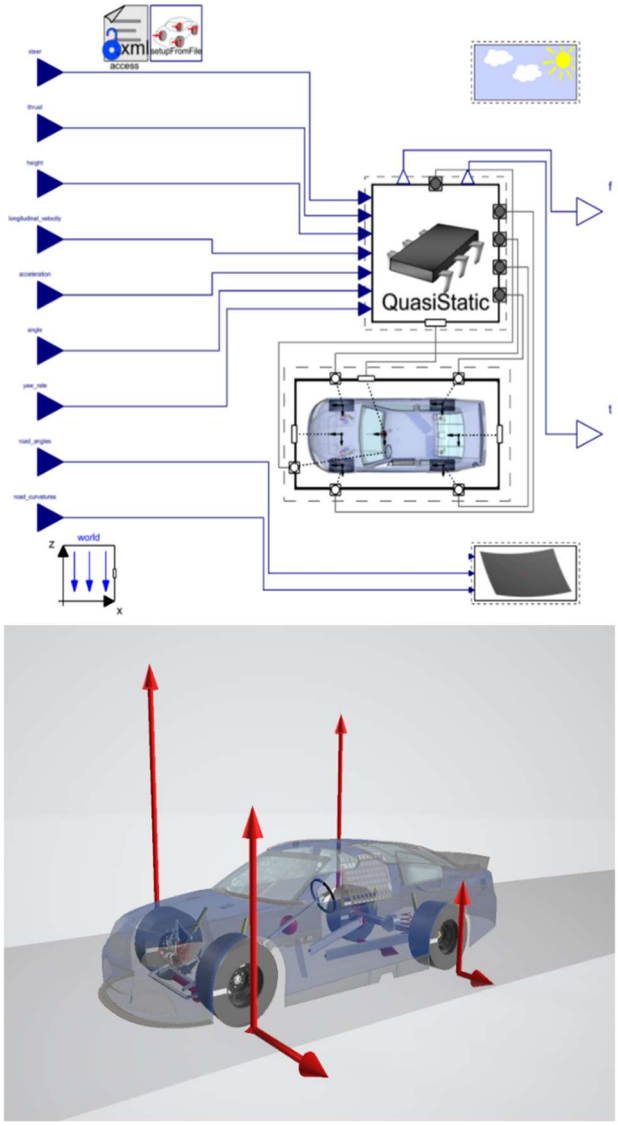

Figure 1. Digital twin of race car from the NASCAR series. Diagram view with race car and boundary conditions (top) and 3D visualization (bottom). The arrows show the estimated individual tire forces.

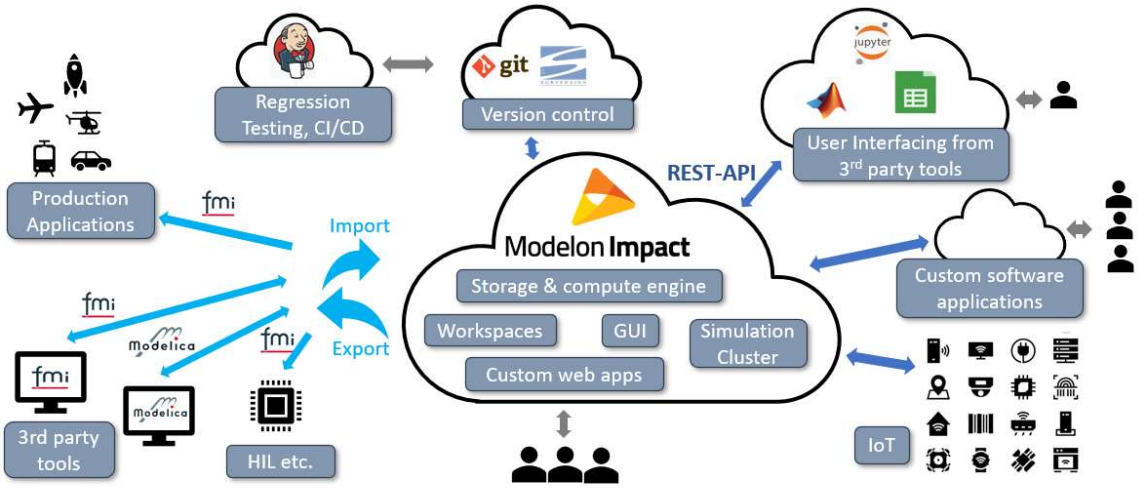

Figure 2. Modelon Impact is prepared to work in an eco-system with well-defined communication through public APIs.

In other applications, such as predictive maintenance or fault detection, the physical model needs to be used in combination with an algorithm to extract valuable information from the process in operation. This can be achieved by exporting the physical model to a scripting environment such as Matlab or Python using the Functional Mock-up Interface (FMI) standard. (Ruggaber and Brembeck, 2021; Gonzalez, et al., 2017; Andrén, et al., 2015) demonstrate how various variants of Kalman filters can be implemented for state and parameter estimation, also exploiting the directional derivatives defined by the FMI 2.0 standard. The combined usage of Modelica, FMI and the scripting environment has been proven to be successful for optimal start-up of power plants in offline mode (Dietl et al., 2014). For rapid testing and deployment of the state estimators without any need for scripting environment, the estimation algorithm can also be embedded according to the FMI standard as a Functional Mockup Unit (FMU), as shown in (Brembeck et al., 2011), (Bonvini et al, 2014), (Laughman and Bortoff, 2020). This however requires manual adaptation work for every considered plant model.

Data exchange is also an essential component that sets requirements on the digital twin implementation in terms of connectivity and openness. This can be achieved by integrating a plant model as FMU into a connected data management or control system as it was done in (ENGIE, 2022). Modelon Impact™ was used there to derive a digital twin of a solar photovoltaic power plant in Chile and to train fault detection algorithms. The model was run online, and its predictive nature permitted to detect and isolate component failures. In (Dietl and Link, 2018) the communication between the control system and the simulation platform relied on OPC-UA. The authors implemented and deployed a Moving Horizon state model predictive controller based on Modelon's optimization toolchain.

The mentioned examples have in common that they showcase the industrial value and the feasibility of digital twin type analyses based on Modelica and FMI technologies. They also illustrate the need for a framework that allow for a more systematic way of implementing and deploying models for real-time applications. The objective is to achieve a modular and flexible implementation to keep the model and the analysis separated and thereby facilitate code reuse for multiple applications.

This paper showcases how this can be achieved with Modelon Impact, a cloud-native Modelica-environment with public APIs. The paper is structured to first outline the relevant properties of Modelon Impact in Section 2, prediction, and correction in Section 3, followed by a set of select applications in Sections 4-5.

2 Enabling cloud infrastructure

Modelon Impact (Modelon Impact, 2022) is a cloud-native systems modelling and simulation environment that enables connectivity through public APIs. Modelon Impact generates and runs FMUs on the cloud. It is also possible to upload third party Modelica packages and use them either as dependencies or editable models. Modelon supports connectivity to version control software like Git and SVN. Modelon Impact also features installation on private clouds to ensure data security. Figure 2 shows an overview of the connectivity options that are offered. In this paper we will focus on the APIs that allow for 3rd party tools to communicate with the compute engine in Modelon Impact.

Modelon Impact communicates through a Representational State Transfer (REST) API, that enables remote controlling from other applications such as Microsoft Excel, Jupyter (Kluyver, et al., 2016), and custom web apps. The Modelon Help Center (Modelon Help Center, 2022) contains detailed information of the available REST API calls.

DOI
10.3384/ecp19375

Modelon Impact has client libraries in Python and JavaScript wrapping around the low-level web-interface (REST API) which makes it easy to programmatically connect and interact with a Modelon Impact server.

The client libraries help with:

- Defining and executing simulations on the Impact server.
- Compiling models on the server and downloading them as FMUs.
- Fetching results and do post-processing.
- Authenticate users against Modelon Impact.
- Creating and automating custom workflows in your favorite programming language

The client libraries enable the execution of workflows orchestrated on a client and executed on a Modelon Impact server, which may be running remotely. With sufficient login credentials and an API Key, Modelica models may be uploaded, compiled, and executed on a server. The results can be either processed on the server with a custom function or downloaded to the client for further analysis.

An analysis could be set up and executed and relevant trajectories plotted using the Python client library in a few lines of code as shown in Figure 3. Further information about the usage of the API is given for each application below.

```python
from modelon.impact.client import Client

client = Client(url=<impact-domain>)
workspace = client.create_workspace(<workspace-name>)

# Choose analysis type
dynamic = workspace.get_custom_function('dynamic')

# Compile model
model = workspace.get_model("Modelica.Blocks.Examples.PID_Controller")
fmu = model.compile(compiler_options=dynamic.get_compiler_options()).wait()

# Execute experiment
experiment_definition = fmu.new_experiment_definition(dynamic)
exp = workspace.execute(experiment_definition).wait()

# Plot Trajectory
import matplotlib.pyplot as plt

case = exp.get_case('case_1')
res = case.get_trajectories()
plt.plot(res['time'], res['inertia1.phi'])
plt.show()
```

Figure 3. Sample code to remotely operate Modelon Impact using a Python client library.

3 State Estimator implementation

As mentioned in the introduction, state estimation is an important component in digital twin applications. It can be used to filter noisy measurements, estimate key variables that cannot be reliably measured or estimate unknown parameters in the plant model. All state estimators, from standard to advanced Kalman Filters or Moving Horizon Estimators (MHE), share a similar structure as shown in Figure 4. They are driven by the plant inputs and measurements and generate estimates of key performance indicators. The physical plant model is often extended by a disturbance model to cope with modelling errors or unknown parameters.

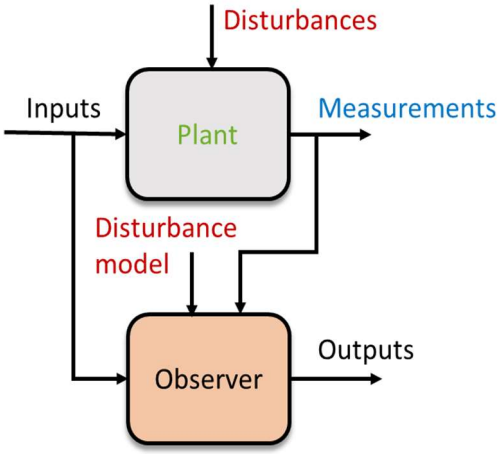

Figure 4. Schematics of an observer model that generates estimates of key variables from plant inputs, measurements, a plant model and eventually a disturbance model.

Such estimation workflows can be conveniently implemented and deployed using Modelon Impact and standard technologies and communication protocols. A dashboard for plant monitoring is built on Node-RED (NR) (Node-RED, 2022) where the data flow between the nodes of plant, digital twin/observer, and the NR dashboard visualized as shown in Figure 5.

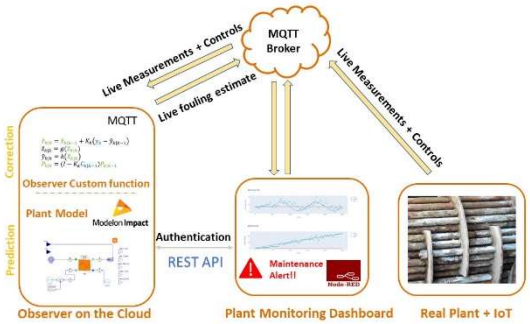

Figure 5. Modelon Impact and Node-RED based data flow for plant monitoring.

A combination of the Modelon Impact JavaScript client library and MQTT (MQTT Protocol, 2022), a publish/subscribe messaging protocol in the backend facilitates the two-way data exchange, where the plant measurement data are published on a specific topic to a central MQTT message broker and Modelon Impact acts as subscriber and listens to this topic. The measurement values received are fed to a custom function implementing an Extended Kalman filter as state estimator. The Kalman filter consists of two parts: a combined plant and disturbance model in Modelica for the prediction step and a Python script that implements the correction step. The plant estimates are then further broadcasted to the plants NR dashboard though the

MQTT broker. The authenticated user would call Modelon Impact to simulate the digital twin using the JavaScript APIs to Modelon Impact. Key performance indicators along with state variables and estimates would be published in the plant dashboard as shown in Figure 6 for fault prevention and predictive maintenance. NR dashboard compares model predicted state variables and corrected estimates from Extended Kalman Filter (EKF) along with live measurements from the real plant.

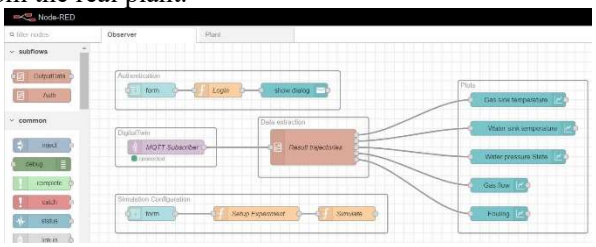

Figure 6. NR flow for plant dashboard.

The workflow is independent of the estimator type. If needed, one could implement a customer function for each estimation approach: Kalman filter, EKF, Unscented Kalman Filter (UKF) or MHE. The whole digital twin implementation is modular and flexible: plant model, algorithm for estimation, integration algorithms, monitoring dashboard are all separate, can be maintained and developed independently.

4 Digital Twin application 1: Heat exchanger fouling estimator

In this section, the state estimator workflow presented earlier is tested on a specific example: fouling estimation in a heat exchanger. In real time applications, fouling is always a major concern with the use of heat exchangers, which gradually degrades system performance and component life, while increasing the operational costs over time. But fouling cannot be measured and often cannot be identified early without leveraging live plant data. Fouling will here be estimated using a generic custom function implementing an Extended Kalman Filter, a heat exchanger model from Modelon Thermal Power Library and the framework described in the previous section. Plant data is here emulated using another heat exchanger model that runs on Modelon Impact, exchanging data using MQTT protocol.

Figure 7 shows a heat exchanger (HX) model from Modelon Thermal Power Library with open boundary conditions. The application in mind here is to estimate fouling on the gas side based on five noisy measurements, encircled in blue in Figure 7: all inlet and outlet temperatures as well as the liquid mass flow rate. It is assumed that the gas flow rate is not measurable, and it will also be estimated. The heat exchanger model is discretized in the flow direction according to a finite volume implementation. Each section has then three dynamic states, for liquid pressure, liquid temperature, and wall temperature. The plant model has been extended by a disturbance model to describe the unknown gas flowrate and the fouling factor (encircled in red in Figure 7). The disturbances are implemented in Modelica and assumed to be constant in time although they will be varied in the experiment.

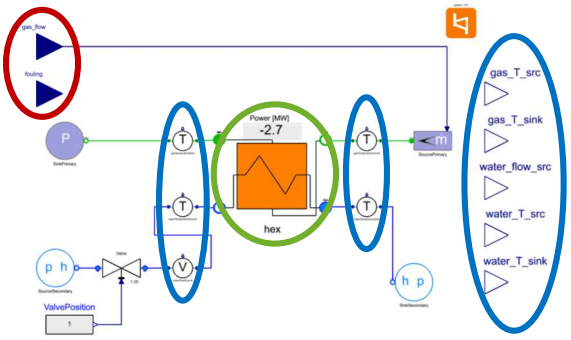

Figure 7. Physical plant model using the fouling estimator.

The measurements from the emulated plant and predictions of those measurements from the observer model are compared to validate the estimations. In Figure 8 showing the estimates, the fouling estimate in grey and the gas flow rate estimates are able to follow the true value. The offsets are due to the fast dynamics of the emulated fouling and gas flow changes, but they could be reduced with a more detailed disturbance model.

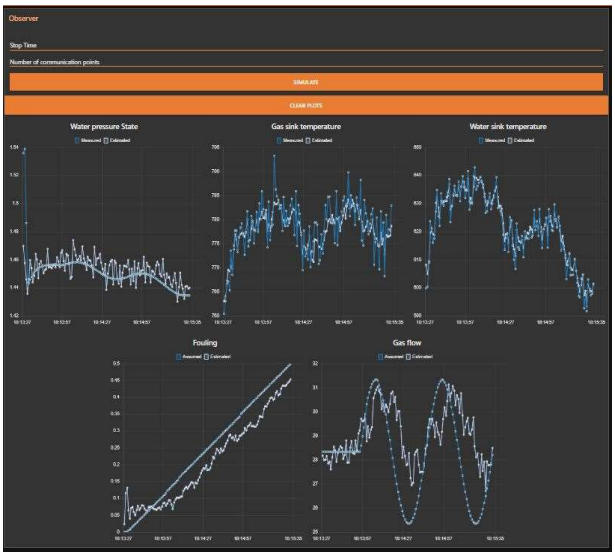

Figure 8. NR plant monitoring dashboard displaying measurements as well as estimates for fouling and gas flowrate.

5 Digital Twin application 2: Windmill Fleet

The previous section illustrates the application of Modelon Impact for performance monitoring of a single plant. There are a variety of cases where it is necessary to monitor a fleet of assets. For example, the growing

DOI
10.3384/ecp19375

number of wind farms demand highly reliable integrated systems to curtail Operation & Maintenance cost. The workflow described in section 4 can be extended to implement Digital Twins for online monitoring of a fleet of assets that are geographically spread as shown in Figure 9.

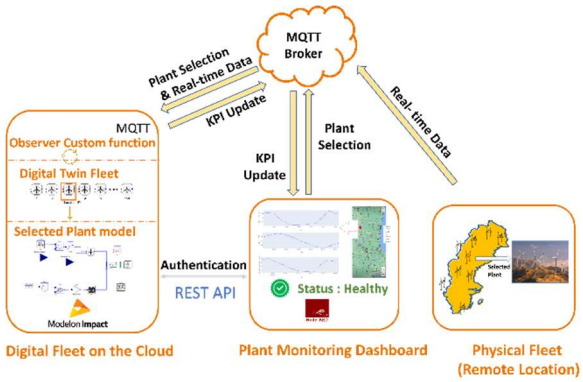

Figure 9. The overall digital twin workflow in the case of an asset fleet, here a Windmill Fleet.

The Digital Twin Fleet was implemented similarly as the single asset example, using three modules (i) Digital Twin Fleet based on a set of Modelica models built in Modelon Impact (ii) Real Time wind and plant data at various locations (iii) Interactive plant monitoring dashboard.

The plant monitoring dashboard needs to include an interactive map component that allows operators to select the windmill of interest in the windmill fleet. A proof-of-concept has been implemented using JavaScript. The map component in the dashboard is interactive and allows the user to select the location of the windmill of interest. It is a reaction based interactive map component containing map data through Google API. To the map, several different interface elements like overlays for point of interest, zoom in/out, highlight selection is added for best user interaction experience. Real time wind data for various windfarms in different locations were embedded into the map element through JavaScript. Real time data for the wind at any selected location was fetched from (Trafikeverket's open API, 2022). Additionally, the monitoring dashboard has been implemented to display the data of the windmill selected from the fleet. In this case study, no state estimation problem was solved. The goal of the demonstrator was instead to show the ability of the technology to deal with a fleet of digital twins with respect to plant selection, data visualization and bi-directional data exchange between the model and the plant data.

6 Conclusion

Digital twin applications based on Modelica models and FMI standard are not new. Different solutions have been suggested in literature and some tested in industrial applications. This clearly shows the feasibility and the potential of the approach. With Modelon Impact on the cloud and its public APIs, the path from systems modeling and simulation to digital twin in operation is significantly shortened. It also allows for a modular and flexible digital twin implementation where models and algorithms can be kept separate and be re-used for different applications. The NR dashboards powered by Modelon Impact Digital Twin are easy to setup and can present complex scenarios in easily understandable manner. They can also meet the connectivity requirements of simulation platform in digital twin applications by enabling bi-directional data exchange using standard communication protocols.

Acknowledgements

Special thanks to Johan Andreasson, Peter Sundström and John Griffin for their input on motorsports digital twin applications, and to Emil Fredriksson as well as Ola Flisback for the implementation of the windmill fleet example.

References

M. T. Andrén, and C. Wedding, (2015): Development of a Solution for Start-up Optimization of a Thermal Power Plant, M.Sc. thesis, Department of Automatic Control, Lund University, Sweden.

M. Bonvini, M. Wetter, and M. Sohn,(2014), An FMI-based Framework for State and Parameter Estimation. 10.3384/ecp14096647. *In Proceedings of the 10th International Modelica Conference Lund March 10-12.*

J. Brembeck, M. Ottera, and D. Zimmer,(2011): Nonlinear Observers based on the Functional Mockup Interface with Application to Electric Vehicles*, In Proceedings of the 10th International Modelica Conference Dresden March 20-22,2011.*

K. Dietl and K. Link, (2018): Startup optimization of Combined Cycle Power Plants: Controller development and real plant test results. *5th International Conference on Control, Decision and Information Technologies (CoDIT)*, pp. 599-604, doi: 10.1109/CoDIT.2018.8394850.

K. Dietl, S.G. Yances,A. Anna, J. Akesson, K. Link, and S. Velut, (2014): Industrial application of optimization with Modelica and Optimica using intelligent Python scripting. *In Proceedings of the 10th International Modelica Conference*, *Lund,March 10-12.*

ENGIE: URL https://modelon.com/support/engie-solar-power-plant-predicitive-maintenance-digital-twin/,2022.

M.Gonzalez, O. Salgado, J. Croes, B. Pluymers, and W. Desmet, (2017): Model-based virtual sensors by means of Modelica and FMI. *In Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017,* 337-344. 10.3384/ecp17132337.

T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and Jupyter development team, (2016): Jupyter Notebooks – a publishing format for reproducible computational workflows. *Loizides, Fernando and Scmidt, Birgit (eds.) In Positioning and Power in Academic*

Publishing: Players, Agents and Agendas. IOS Press. pp. 87-90. (doi:10.3233/978-1-61499-649-1-87)

C. Laughman, and S. Bortoff, (2020): Nonlinear State Estimation with FMI: Tutorial and Applications. *In Proceedings of the American Modelica Conference 2020 Boulder, Co, USA March 23-25* 186-195. 10.3384/ecp20169186

Modelon Impact: https://modelon.com/modelon-impact/, 2022.

Modelon Help Center: https://help.modelon.com, for API https://help.modelon.com/latest/guides/apidocumentation.html, 2022.

Modelon Vehicle Dynamics Library (VDL): https://modelon.com/library/vehicle-dynamics-library/, 2022.

MQTT Protocol, http://mqtt.org/, 2022

Node-RED, https://nodered.org/, 2022.

J. Ruggaber and J. Brembeck, (2021): A Novel Kalman Filter Design and Analysis Method Considering Observability and Dominance Properties of Measurands Applied to Vehicle State Estimation. *Sensors* 21, no. 14: 4750. https://doi.org/10.3390/s21144750

Trafikeverket's open API for traffic information with weather data: https://api.trafikinfo.trafikverket.se/, 2022.