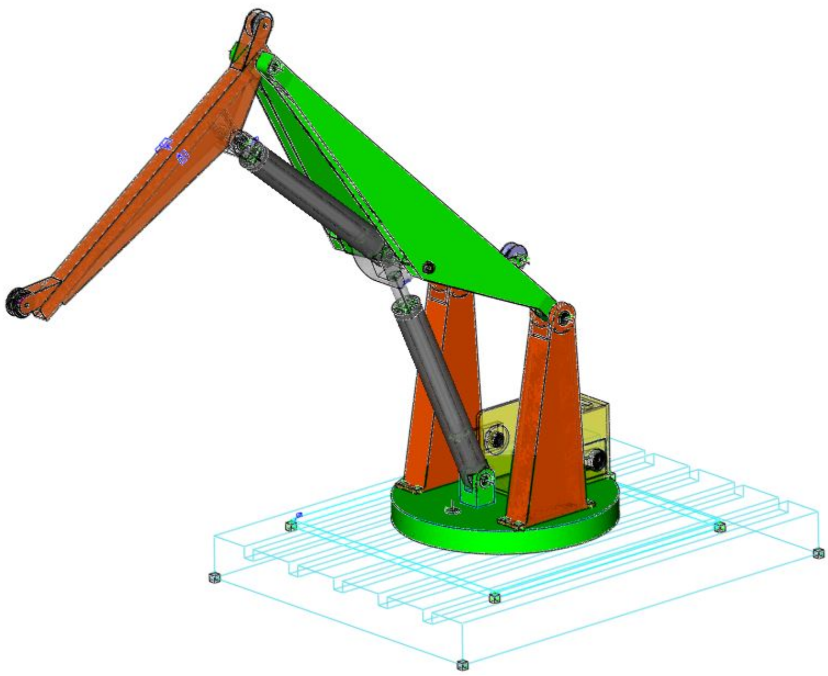
Christian Johansen

Digital Twin Of Knuckle Boom Crane

Master's thesis in Engineering & ICT
Supervisor: Terje Rølvåg
June 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering



NTNU
Norwegian University of Science and Technology

Christian Johansen

Digital Twin Of Knuckle Boom Crane

Master's thesis in Engineering & ICT
Supervisor: Terje Rølvåg
June 2019

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Sammendrag

I denne masteroppgaven har en digital tvilling av en miniatyrisert ”knucle boom” kran blitt utviklet. En digital tvilling er en ”finite element” (FE) modell av en fysisk gjenstand som gjennom FE-simuleringer basert på ekte data kan reprodusere gjenstandens oppførsel i sanntid.

For å fange kranens oppførsel har den blitt utstyrt med sensorer og et system for datainnsamling. Trådsensorer måler lengden på kranens aktuatorer og en enkoder montert på motoren som styrer baserotasjonen er brukt til å finne orienteringen til basen. Tre strekklapper er plassert på kranens øvre arm og målingene fra disse er brukt til å kalkulere den påførte belastningen på kranen.

I venteperioden for å få tilgang til kranen ble det også lagt inn arbeid i å utvikle programvare som vil hjelpe implementeringen av en digital tvilling til programvaren *Digital Twin Cloud Software*.

Abstract

In this thesis, a digital twin of a miniaturised knuckle boom crane has been developed. A digital twin is a finite element (FE) model of a physical asset that through FE simulations based on real sensor data can replicate the assets behaviour in real-time.

To capture the behaviour of the crane it has been equipped with sensors and a data acquisition system. Wire sensors capture the length of the crane actuators and an encoder mounted on the motor controlling the base rotation is used to keep track of the angle of the base. Three strain gauges are mounted on the crane's upper arm and their output is used to calculate the applied load.

While waiting to get access to the crane an effort was also put into creating software that would aid in implementing the digital twin to the *Digital Twin Cloud Software*.

Preface

This master's thesis marks the end of the master's program of Engineering & ICT at the Norwegian University of Science and Technology in Trondheim. The work aims to create a digital twin of a knuckle boom crane. The project has been conducted at the Department of Mechanical and Industrial Engineering (MTP) under the supervision of Professor Terje Rølvåg, to whom I would like to thank for supporting me throughout the entirety of this project and for creating the Fedem model of the crane.

I would like to extend a special thanks to senior engineer Frode Gran and Halvard Støwer, for their continuous support and guidance in the instrumentation process that is such a key element in this project.

Christian Johansen
Trondheim, 08.06, 2019

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Problem Description . . . 1
1.3 Approach . . . 3

2 Digital Twin **4**
2.1 Concept and Advantages . . . 4
2.2 Monitoring . . . 6

3 Instrumentation **7**
3.1 Sensor list . . . 7
3.2 Mounting . . . 7
3.2.1 Strain Gauges for Inverse Method . . . 7
3.2.2 Strain Gauges for Temperature Compensation . . . 8
3.2.3 Wire Sensors . . . 8
3.2.4 Encoders . . . 9
3.3 Sensor Mapping . . . 10
3.3.1 Wire Sensors . . . 10
3.3.2 Base Encoder . . . 10
3.4 Data Acquisition System . . . 12
3.4.1 Wiring . . . 12
3.5 Noise on Strain Gauge Signals . . . 13

4 Operation **15**
4.1 Driving the Crane . . . 15
4.2 Setting Up Catman . . . 17

5 Fedem Model **20**
5.1 External Functions . . . 20
5.2 Actuators . . . 20
5.3 Base Rotation . . . 23
5.4 Virtual Strain Gauges . . . 23
5.5 Monitoring . . . 24
5.5.1 Stress Analysis . . . 24
5.5.2 Fatigue . . . 26
5.5.3 Buckling and Instability . . . 26

6 Inverse Method **27**
6.1 Concept . . . 27

6.2 Performance . . . 29
6.2.1 Simulation With Static Load . . . 29
6.2.2 Simulation With Real Data . . . 31
6.3 Updated Inverse Method . . . 33
6.4 Challenges . . . 35
6.5 Simulation Script . . . 35

7 Digital Twin Cloud Software 37
7.1 Digital Twin Configuration Platform . . . 37
7.1.1 Front-End . . . 37
7.1.2 Back-End . . . 41
7.1.3 JSON-files . . . 41
7.2 Digital Twin JavaScript Class . . . 42
7.3 Results . . . 42
7.3.1 Old Digital Twin Cloud Software . . . 42

8 Functional Mock-Up Interface 44
8.1 What is FMI . . . 44
8.2 Digital Twin Cloud Software . . . 44

9 Further Work 45

Appendices 47

A Task Description 48

B Digital Twin Java Class 49

C Digital Twin Simulation Script 53

D Front-End 59
D.1 index.html . . . 59

E Back-End 70
E.1 Server.py . . . 70
E.2 ServerSupport.py . . . 74
E.3 parseExport.py . . . 80

F Modified Digital Twin Cloud Software 84
F.1 index.js . . . 84
F.2 index.html . . . 85

F.3 usg.ts 88

G Strain Gauge Data Sheets **93**
G.1 WFLA-6-11-1L 94
G.2 FLA-5-11-1L 95

H Wire Sensor Data Sheet **96**

I Encoder Data Sheet **101**

List of Figures

1 Crane . . . 2
2 Flow Chart . . . 5
3 Wire Sensor Placement . . . 8
4 Base Motor/Encoder . . . 9
5 Crane base . . . 11
6 Encoder Wiring . . . 13
7 Noise on Strain Gauges . . . 14
8 Actuator Control Panel . . . 15
9 Joystic Panel . . . 16
10 Catman Dashboard . . . 17
11 Wire Sensor Adaptation . . . 18
12 Strain Gauge Adaption . . . 19
13 Fedem model . . . 20
14 Actuator length . . . 22
15 Actuator length - Close Up . . . 22
16 Right view of model . . . 23
17 Stress Distribution - Front . . . 24
18 Stress Distribution - Back . . . 25
19 Stress Distribution - Close Up . . . 25
20 Crane . . . 27
21 Forces on the Crane . . . 28
22 Simulation Forces . . . 30
23 Simulation Strain . . . 30
24 Simulation Forces With Real Data . . . 31
25 Simulation Strain With Real Data . . . 32
26 Simulation Strain With Real Data . . . 33
27 Simulation Strain With Real Data . . . 34
28 Simulation Force With Real Data . . . 34
29 Create a Digital Twin Configuration file . . . 39
30 Digital Twin Configuration file . . . 40
31 Digital Twin Cloud Software . . . 43

List of Tables

1 List of Sensors . . . 7
2 Wire Sensor Calibration . . . 10
3 Wire Sensor Output . . . 10

4 Spider Channels 12
5 Actuator spring/damper coefficients 21

Listings

1 DigitalTwin.js 49
2 CraneShortUDP.py 53
3 Index.html 59
4 Server.py 70
5 ServerSupport.py 74
6 parseExport.py 80
7 index.js 84
8 index.html 85
9 usg.ts 88

1 Introduction

1.1 Background

During the fall of 2018 the authors of Johansen et al. (2018) developed a prototype of a *Digital Twin Cloud Software* as part of the project thesis for MTP. This prototype supported only a single digital twin, a Torsion Bar Suspension Rig which was the result of a project thesis from an previous student(Christiansen (2017)). The long term goal is to have a *Digital Twin Cloud Software* that is able to represent any arbitrary asset. However, the small number of digital twins available inhibits the development progress of the *Digital Twin Cloud Software*. To that end more assets should be made readily available, and that is what the work of this thesis aims to do.

1.2 Problem Description

The goal of this thesis is to make a digital twin of the crane seen in figure 1 and ultimately implement it to the *Digital Twin Cloud Software* (Johansen et al. (2018)) so that it can be used for structural monitoring. The crane is a scaled down version of a real knuckle boom crane and it is built and stationed NTNU's Department of Marine Technology (DMT).

A copy of the task description for this thesis can be found in appendix A. In agreement with supervisor Terje Rølvåg a big emphasis has been put into bulletin 2 and 6 of the task description. This is because monitoring the critical failure mode of the crane with a digital twin is only possible if the physical asset is equipped with sensors able to capture its behaviour.

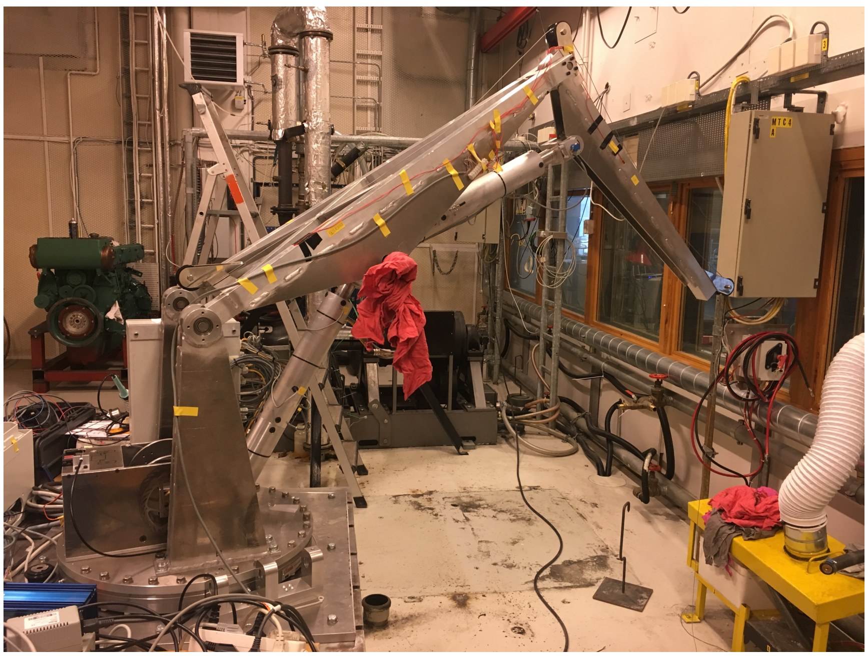


Figure 1: Crane

1.3 Approach

The work of this project has mainly followed an iterative approach:

1. Instrument crane with sensors
2. Run simulations on the digital twin with data from sensors
3. Check performance/validity of the simulation and look for errors
4. Improve:
   (a) Update/change the digital twin
   (b) Improve on sensors (e.g reduce noise)

However, the first two months of this project working with the crane was not possible. The crane was not operational and it was off-limits. During that period work was performed to further develop and improve on the *Digital Twin Cloud Software*.

2 Digital Twin

This section describes the concept of digital twins and how the crane is monitored.

2.1 Concept and Advantages

The concept of a digital twin is that of having a digital model of an physical asset capable of replicate the behaviour of the physical asset in real time. In this project the physical asset is the crane (Figure 1) and the digital twin is a finite element (FE) model in Fedem. Fedem can run real-time simulations based on real data, which makes it suitable for digital twin applications.

The advantage of using a digital twin for structural monitoring instead of traditional monitoring via sensors is that more information about the structure can be gathered with less sensors. This is because simulating with FE-models can give information about the entire structure, whereas using solely sensors only give information about the spot in which they are located. For instance in Fedem you can put virtual strain gauges on your model that gives the strain in that location through the entire simulation.

Figure 2 illustrates the concept of the digital twin in combination with the *Digital Twin Cloud Software*.

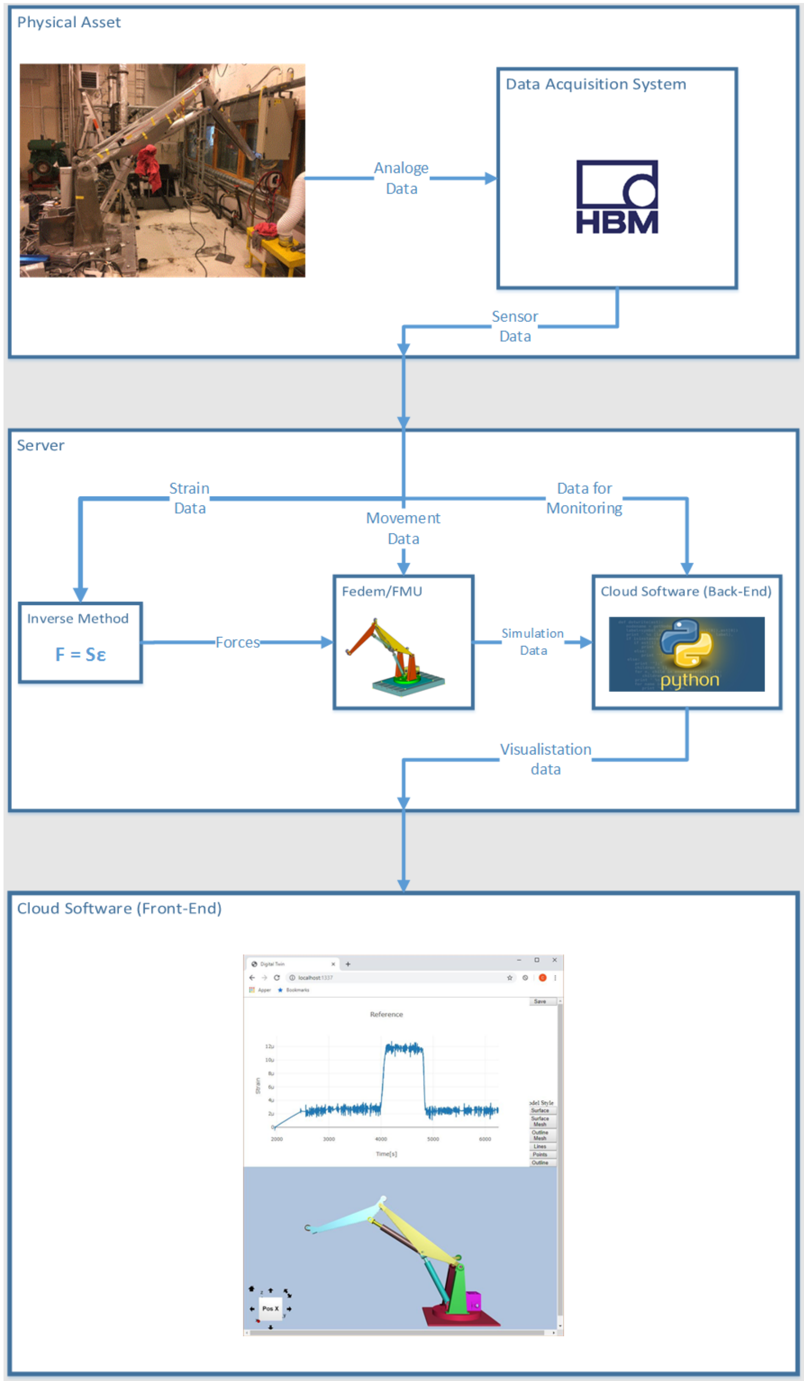


Figure 2: Flow Chart

2.2 Monitoring

To monitor the physical assets failure modes with a digital twin its behaviour must be captured. This way the digital twin will during simulation experience the same conditions as the asset, and when this is achieved one can start to explore the possibilities of monitoring failure modes with the digital twin.

Capturing the behaviour of the crane is done by equipping suitable sensors on it. The movement of the crane is captured by the use of two wire sensors measuring the length of the actuators and an encoder on the motor controlling the base rotation. The pay load applied to the crane is captured using three strain gauges. The values gathered by the strain gauges are used to recreate the forces with an inverse method and are then applied to the digital twin.

3 Instrumentation

This section describes the additional instrumentation that is done on the crane in order to achieve a digital twin of it.

3.1 Sensor list

Sensor type	Product	Manufacturer	Purpose	Amount	Data Sheet
Strain Gauge	WFLA-6-11-1L	TML	Inverse Method	2	Appendix G.1
Strain Gauge	WFLA-6-11-1L	TML	Reference	1	Appendix G.1
Strain Gauge	FLA-5-11-1L	TML	Inverse Method	1	Appendix G.2
Strain Gauge	FLA-5-11-1L	TML	Temperature Compensation	1	Appendix G.2
Displacement sensor	Celesco SP1-25	TE	Actuator Length	2	Appendix H
Encoder	HEDS 5540	Maxon Motors	Base Rotation	1	Appendix I

Table 1: List of Sensors

3.2 Mounting

3.2.1 Strain Gauges for Inverse Method

Three strain gauges are mounted on the upper arm of the crane and the values these produce are used as input to the inverse method developed by Torbjørn Moi. Each strain gauge is placed on a separate cross section 600 mm from the free end of the arm. One is placed at the top, one at the right and one at the left. These are marked "UT", "UR" and "UL" respectively. Right and Left are seen from the point of view of the crane. The placement was chosen after discussing the topic with Torbjørn Moi and considering previous work on another crane. (Section 3.1 in Christiansen (2018)).

A fourth strain gauge is placed on the upper cross section of the lower arm on the right side, 300 mm from the connection to the base of the crane. The position of this strain gauge is not in all that important, because its purpose is to verify the correctness of the inverse method. However, it is placed sufficiently far away from edges, welds and other sources of strain that is hard to account for in the FE-model. This strain gauge has been marked as "Reference".

3.2.2 Strain Gauges for Temperature Compensation

Strain gauges are very sensitive towards temperature changes, and changes in temperature will cause the signal to drift. To counteract this effect a fifth strain gauge is mounted to a separate piece of aluminium with the same thickness as the crane. The aluminium piece is then clamped to the crane. This fifth strain gauge is affected by the same temperature effects as the others, but experience none of the strain coming from loads applied to the crane. The values sampled from this strain gauge is subtracted from the others and a temperature compensation is achieved.

3.2.3 Wire Sensors

One wire sensor is mounted on each actuator. The base of the wire sensors are placed at the top part of their respective actuator with the wire stretching down. The wire is then fastened to a screw. An image of the setup is shown in figure 3.

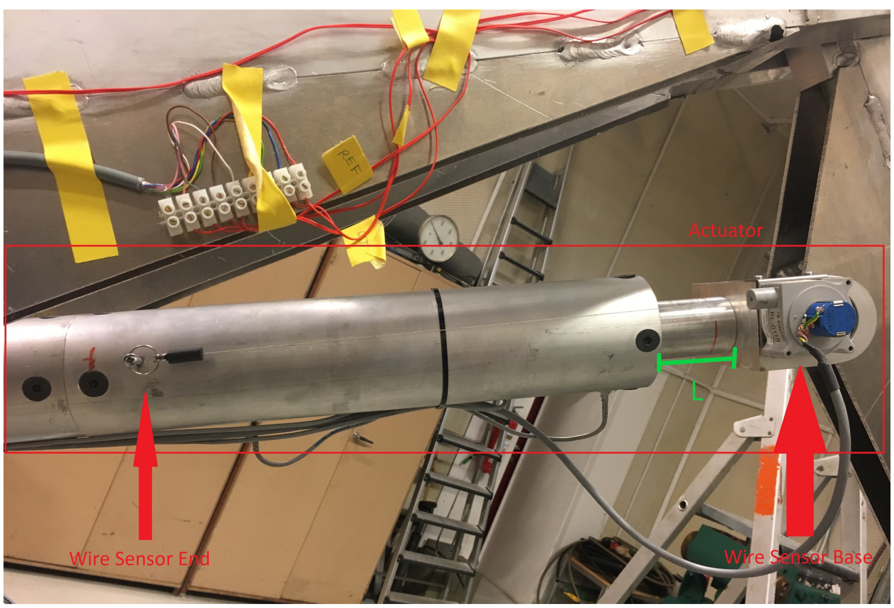


Figure 3: Wire Sensor Placement

3.2.4 Encoders

There are three encoders mounted on the crane, one for each motor except for the motor that drives the winch. These encoders were already fitted on the crane, but were not in use. Figure 4 shows where the encoders are mounted on the motors.

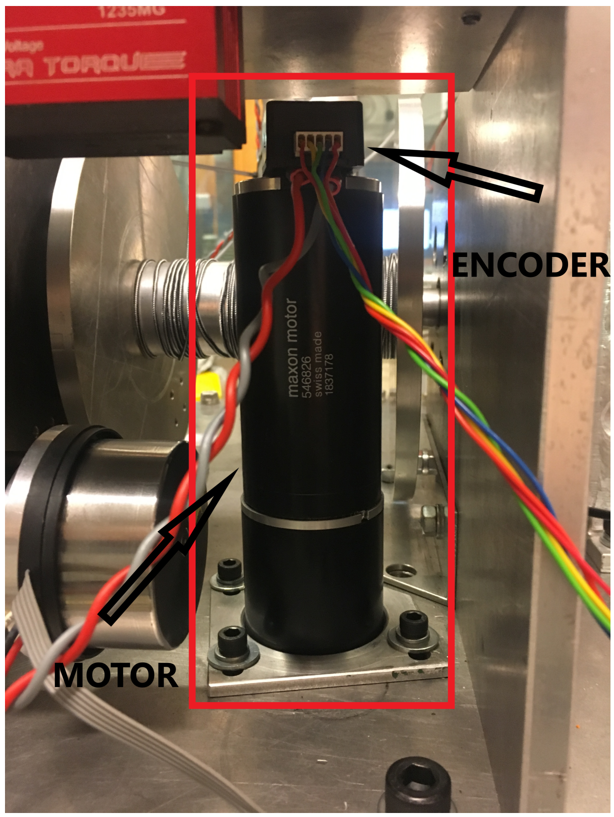


Figure 4: Base Motor/Encoder

3.3 Sensor Mapping

3.3.1 Wire Sensors

The mapping function for the wire sensors are found by measuring their output voltage when the actuators are configured with different lengths. The output voltage from the wire sensors is linear, hence only two points are needed to create a mapping function.

Voltage	**Actuator Length**
4.45 V	3.0 cm
7.64 V	24.5 cm
ΔV = 3.19 V	Δl = 21.5 cm

(a) Lower Arm

Voltage	**Actuator Length**
4.61 V	2.9 cm
25.2 V	7.92 cm
ΔV = 3.31 V	Δl = 22.3 cm

(b) Upper Arm

Table 2: Wire Sensor Calibration

The length of the actuator is defined as the length of its extension. The extension is marked green in figure 3. The values in table 2 are used to create the mapping function (See table 3). The mapping function is used directly in data acquisition software. (See figure 11 in section 4.2)

Arm	$V_{out}(l = 0cm)$	**ΔV/Δl**	**Length**
Lower	4 V	6.74 cm	$l = (V_{out} - 4V) * 6.74$
Upper	4.18 V	6.74 cm	$l = (V_{out} - 4.18V) * 6.74$

Table 3: Wire Sensor Output

3.3.2 Base Encoder

On the base of the crane there are screws evenly distributed along the circumference spaced apart by 10 degrees. This is illustrated in figure 5. Recording the number of impulses transmitted from the encoder when rotating the base 10 degrees gives the relation between impulses and rotation. The relation is showed in the equation below:

$$\phi = Impulses * \frac{10\,\text{deg}}{16500 Imp}[\text{deg}]$$

The equation is used in the Python script `CraneShortUDP.py`(See appendix C) before the data is used as input to the Fedem model.

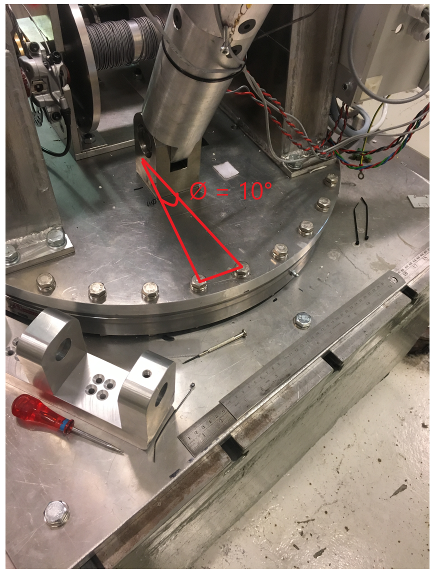


Figure 5: Crane base

3.4 Data Acquisition System

The data acquisition system used consists of:

- Data Acquisition Board: 2x HBM Spider8
- Data Acquisition Software: HBM Catman AP
- Computer with Catman AP

The spiders are connected to each other according to section "D.4.2" in: HBM (n.d.). This allows for the data acquisition software to see the two board as one unit with 16 hardware input channels.

Spider	Hardware Channel	Name	Sensor Type
1	0	Reference	S/G Half Bridge 120 OHM
1	1	UR	S/G Half Bridge 120 OHM
1	2	UL	S/G Half Bridge 120 OHM
1	3	UT	S/G Half Bridge 120 OHM
1	4	Wire Lower	Voltage +- 10DC
1	5	Wire Upper	Voltage +- 10DC
2	0	Encoder Base	Pulse Counter
2	1	Temperature Compensation	S/G Half Bridge 120 OHM

Table 4: Spider Channels

3.4.1 Wiring

The spiders have eight 15-pin sockets each and different sockets have different capabilities. E.g this means that not all 15-pin sockets support all sensor types. The wiring between the Spiders and the sensor will depend on which type of sensor that is connected. The sensor types used in this project can be found in table 4, and a detailed description of the wiring needed for the different types can be found in section "D" in HBM (n.d.).

The wiring for the HEDS 5540 encoder has some deviations from the description of wiring for the "Pulse Counter" found in the Spider reference manual. This is due to different functionality offered by the Spider and the encoder. Therefore a simple illustration of the real wiring is included and can be seen in figure 6.

The most important difference is that "Pin 3" is connected to a 5 voltage power supply. This is because it is an active low clear signal that will reset the impulse counter if it were to go from a high voltage to a low voltage.

There are two pins that are not utilised: "Pin 2" on the spider which is a 10 DC output voltage and "Channel I" on the HEDS encoder. "Channel I" is used to output an impulse when the encoder has done an entire revolution. This can serve as a reference signal for the counter that counts the impulses on "Channel A" and "Channel B". All three resistors "R1", "R2" and "R3" are 3 KΩ.

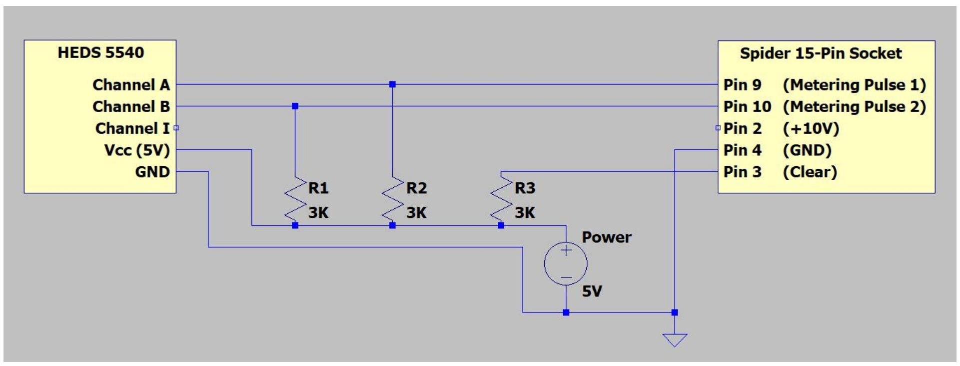


Figure 6: Encoder Wiring

3.5 Noise on Strain Gauge Signals

Strain gauges operate on very small voltages and are therefore prone to disturbances causing noise on the signal. Initially in the instrumentation phase there were issues with exceedingly high levels of noise on the signal, causing overflow errors in the data acquisition system. After a set of sessions with troubleshooting the source of the noise was identified as the crane control units.

In order to reduce the level of noise, the actuator control units were moved closer to the crane reducing the cable length connecting them and the motors. Figure 7 is a snapshot of a recording session in Catman AP taken after the control units were moved. The figure shows how the strain gauge signals are affected by the control units when they are enabled. The control units were enabled approximately six seconds into the recording.

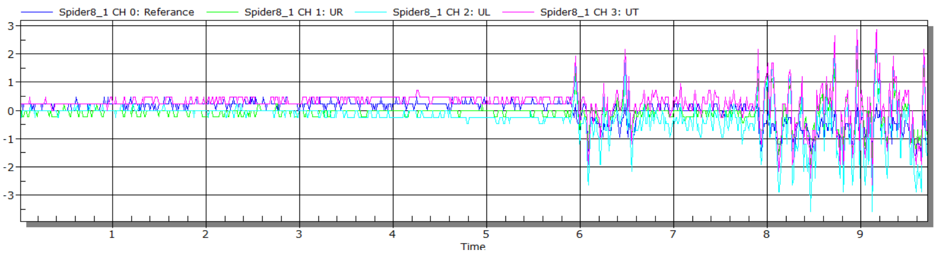


Figure 7: Noise on Strain Gauges

The noise was reduced further by cutting down on the cable length and number of connection points between the strain gauges and the Spiders. The noise was reduced to approximately ± 0.5 µ/µM after this measure was applied.

4 Operation

This section describes how to set up the crane in order to drive it and prepare for sending data to the *Digital Twin Cloud Software*.

4.1 Driving the Crane

There is a laptop labelled "Bachmann" at the laboratory which is connected to the crane control system. On this computer there is a Labview application named "Kran_03.19.vi" (See figure 8) that must be running. When this program is running you can click the button "Enable Actuators", this will allow for the operation of the actuators. The sliding panels at the right of the screen indicates the length of the actuators at the current time. These panels are meant as guiding tool to ensure that the user do not extend the actuators too far or compress them too much causing the screw inside the actuators to go outside its groves. If this event should occur, the only way to fix it is to disassemble the actuators and put the screw back into its place.

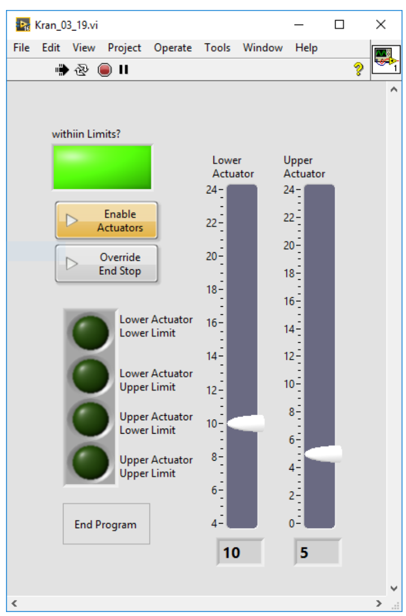


Figure 8: Actuator Control Panel

There are two measures added to avoid the screw inside the actuators to go outside their groves. The first is a rubber band placed around the moving part of the actuator, adding a physical block to prevent the actuators to compress too much. The second preventive measure is a control loop using the measured length of the actuators. Should the measured length be less than 4.5 cm or more than 20 cm the "Kran_03.19.vi" software will disable the actuators. The only way to get the actuators going again if this happens is to press and hold the "Override End Stop" button while driving the actuator out of the stop threshold.

There are also one physical switch for each motor that must be switched on for the operation of the crane. These switches are on the same panel as the joystick's used to control the crane. An image of this panel is showed below in figure 9 .

Figure 9: Joystic Panel

4.2 Setting Up Catman

There is a second laptop at the laboratory which is connected to the data acquisition system. On this computer Catman AP is installed and there is a Catman project file named "Krane1.MEP". To start recording or sending data, simply open the project file and press the "Start" button. Since there is no ethernet connection in the DMT laboratory where the crane is situated, it has not been possible to send live data to the server with the *Digital Twin Cloud Software*. Therefore, the Catman project file has not been configured for a remote connection. Should the crane be moved to a location with an ethernet connection a guide on how to set up Catman for this can be found in section 5.2.2 in Johansen et al. (2018).

The values from the strain gauges tend to drift over time. A temporary measure to counteract this phenomena is to zero out the values in Catman before commencing a "DAQ JOB". This can be achieved by selecting the sensors and the press the "Execute" button.

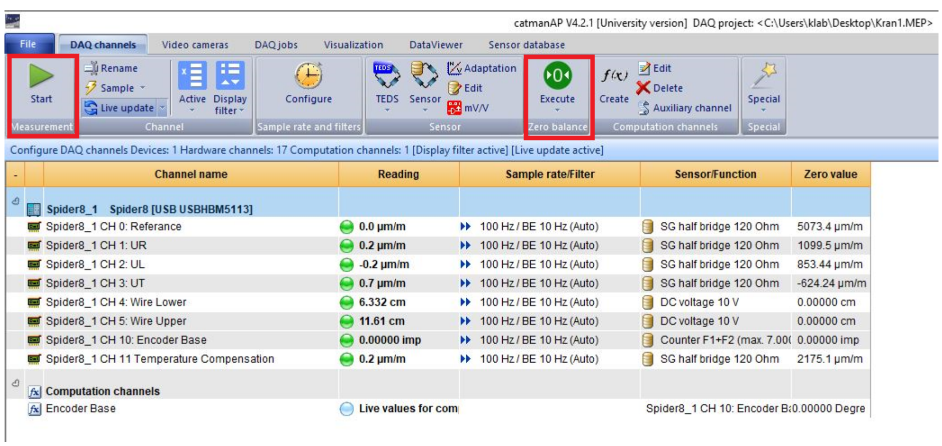


Figure 10: Catman Dashboard

Specific configurations for the different sensors are done with the "Sensor Adaptation" option in Catman. Figure 11 shows the sensor adaption for the lower wire sensor, where the values set are the same ones used in section 3.3.1. Figure 12 shows the configuration of the "Reference" strain gauge. The sensor adaption is the same for all the strain gauges. The only exception is that the strain gauges of the type "FLA-5-11-1L" have a gage factor of 2.13.

Note: If you choose to use the option "Temperature compensation using compensation S/G" in Catman for the strain gauges, the strain gauge chosen for temperature compensation has to be connected to hardware channel zero when using Spiders.

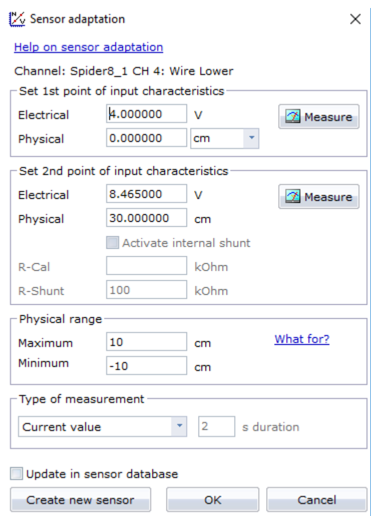


Figure 11: Wire Sensor Adaptation

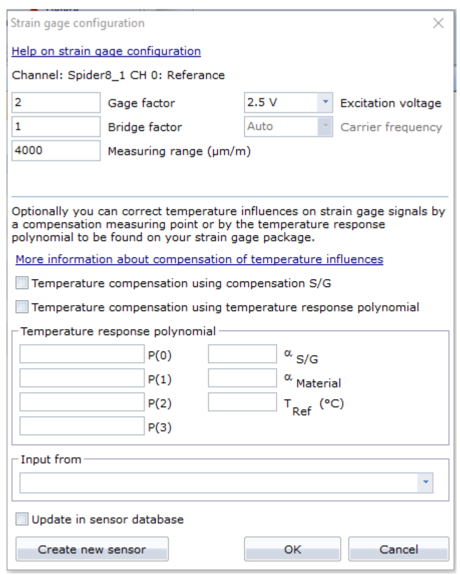


Figure 12: Strain Gauge Adaption

5 Fedem Model

This section describes the aspects of the Fedem model which is relevant for the it to work as a digital twin, and a subsection on how how the model can be used for monitoring. Figure 13 displays the model of the crane in Fedem.

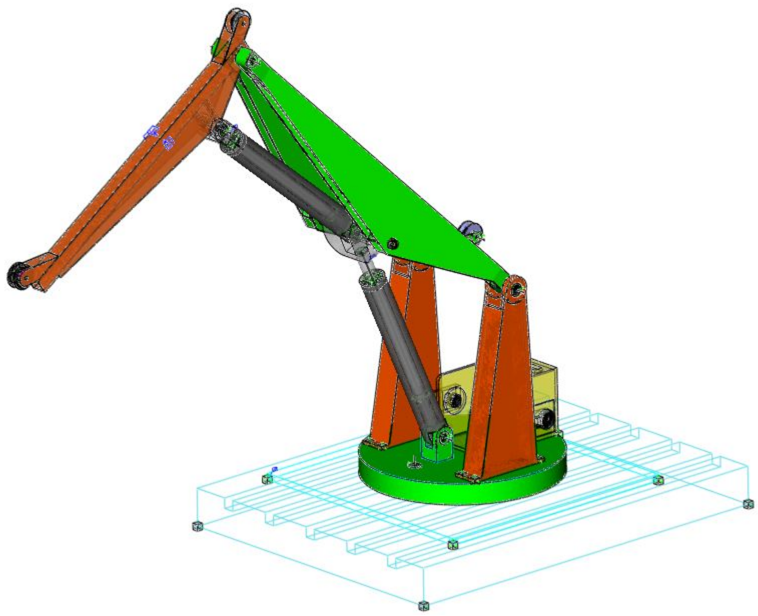


Figure 13: Fedem model

5.1 External Functions

External Functions in Fedem are functions that take their input from an external source such as a python script. To establish a link between a python script and the Fedem model one must use the methods provided by the `vpmSolver.py` and `vpmSolverRun.py` scripts. These scripts will not be discussed in detail in this report because of license restrictions.

5.2 Actuators

The actuators are modelled using prismatic joints that act as spring-dampers. The changes in the length of the actuators during simulation is controlled through the option ”stress free length change”. This is an option in Fedem that allows for springs to change length without contributing stresses. The length changes are controlled used external functions.

The spring and damper coefficients for the two actuators has been tuned by running simulations based on data recorded when driving the physical crane. The coefficients can be found in table 5. Figures 14 and 15 are plots of the length of the actuators in the model compared to the real recorded values.

Actuator	**Stiffness [Pa]**	**Damping**
Upper	1e7	1e3
Lower	1e7	1e3

Table 5: Actuator spring/damper coefficients

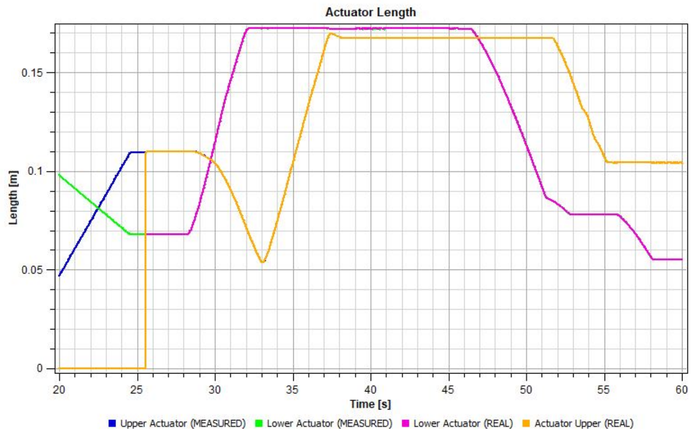


Figure 14: Actuator length

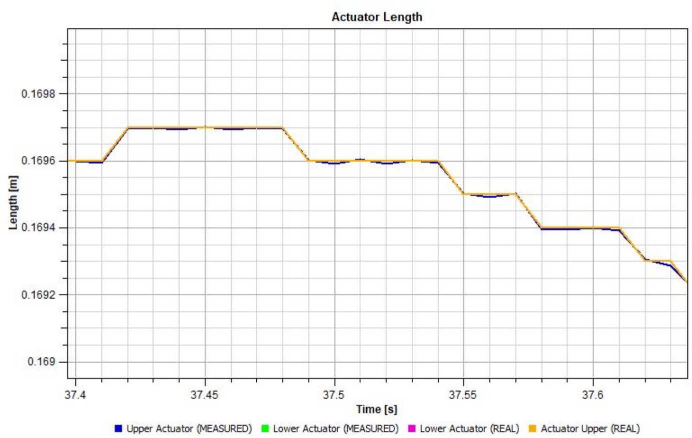


Figure 15: Actuator length - Close Up

5.3 Base Rotation

The rotation of the base is controlled using a revolute joint. Revolute joints in Fedem has the option for controlling angle changes with external functions.

5.4 Virtual Strain Gauges

Four virtual strain gauges are added to the model in the same position as the four real ones are placed on the real-life crane. In figure 16 the two upper part of the crane is shown from the right side and the virtual strain gauges are highlighted with red circles.

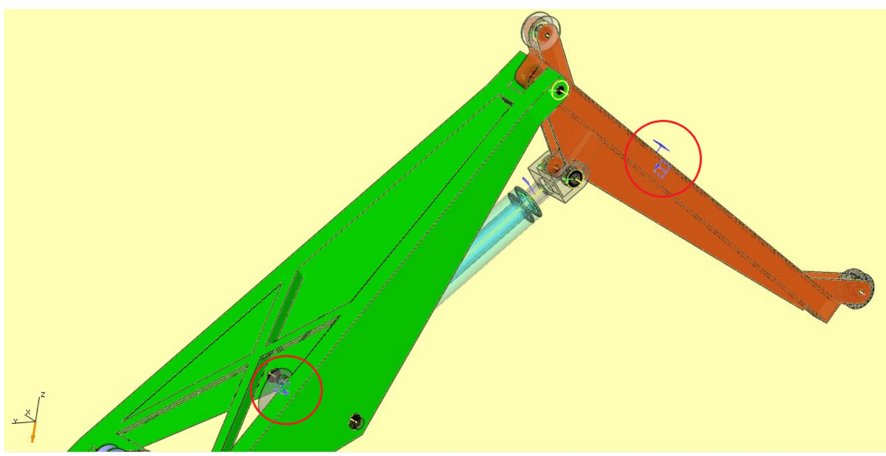


Figure 16: Right view of model

The functionality of these virtual strain gauges is to retrieve strain data from the digital model in these points so that it can be compared to the real captured strain levels. The three virtual strain gauges on the upper arm are also used in the calculation of the compliance- and component rotation compensation matrix used in the inverse method.

5.5 Monitoring

5.5.1 Stress Analysis

The purpose for performing a stress analysis is to locate areas on the crane that is most vulnerable. Thus, appropriate steps can be made to monitor these spots. For instance using virtual strain gauges in the Fedem model to monitor the stress in these locations and compare it to the yield stress for the material. From table 3.5 in Gyberg (2015) we have that the material for the crane arms is "Al 6082-T6" which has a yield stress of 260 MPa. An other possibility is to place an actual strain gauge on the crane in one or several of these positions.

A "max von Mises stress" analysis is performed on the results from a simulation that used real data and the inverse method to calculate the forces acting on the crane. For this case it was a 7.6 kg payload. The results from the analysis can be seen in figures 17, 18 and 19. The stresses are given in the unit pascal.

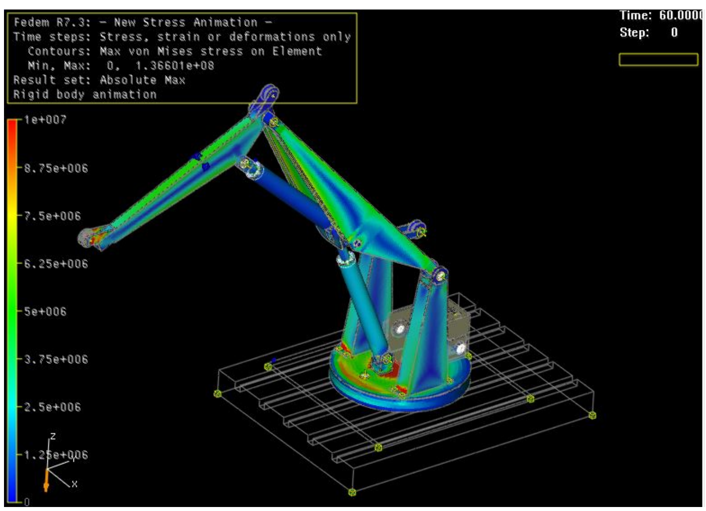


Figure 17: Stress Distribution - Front

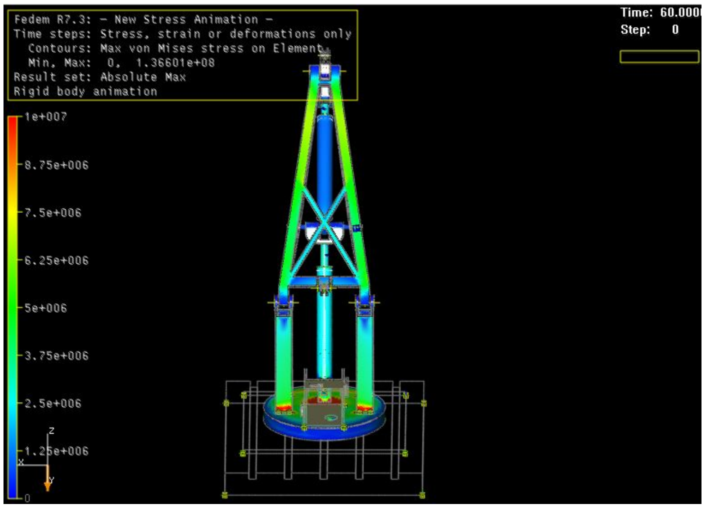


Figure 18: Stress Distribution - Back

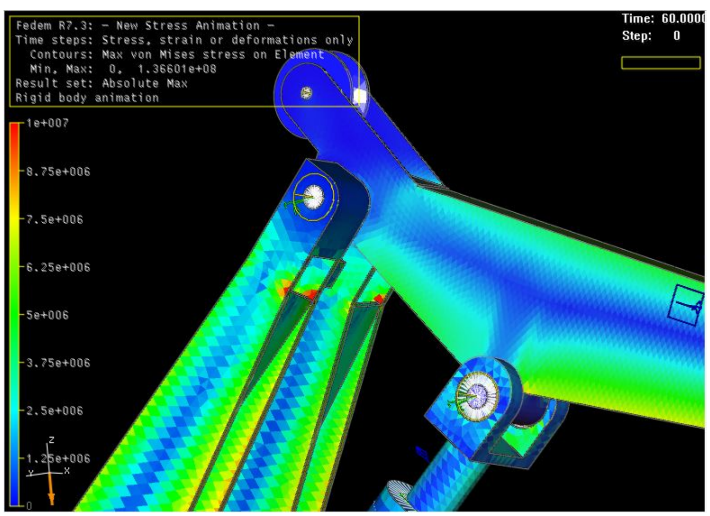


Figure 19: Stress Distribution - Close Up

5.5.2 Fatigue

Fatigue in Fedem can not be calculated in real time, however it has a post-processing functionality of calculating the fatigue in models using a rain-flow algorithm.

5.5.3 Buckling and Instability

Fedem has neither the option of calculating buckling or instability for a model. So to monitor these failure modes in Fedem expertise is needed to apply the correct measures.

6 Inverse Method

This section briefly describes the inverse method that is used to recreate the forces that the load applied to the crane creates, how it performs and challenges it presents.

6.1 Concept

The concept of the inverse method is to treat the free end of the upper arm of the crane (See figure 20) as a cantilever beam.

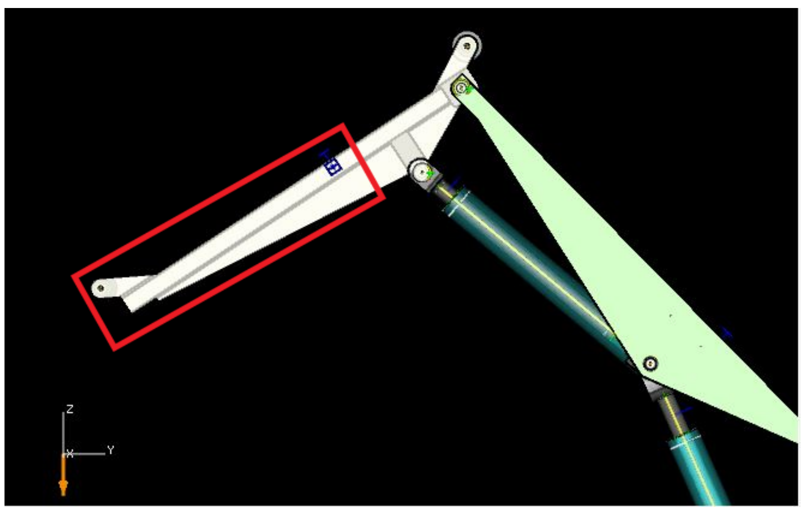


Figure 20: Crane

From Moi (2018) we have this relation for cantilever beams:

$$\boldsymbol{f} = \boldsymbol{S}(\hat{\boldsymbol{\epsilon}} - \boldsymbol{A}\boldsymbol{a}) \tag{1}$$

Where **f** is a force vector, **S** is a compliance matrix, $\hat{\boldsymbol{\epsilon}}$ is a strain vector, **a** is a vector containing the actuator movement and **A** is a component rotation compensation matrix. In section 3.2 in Moi (2018) a procedure for finding the compliance matrix using virtual strain gauges in Fedem is presented. Using this procedure the compliance matrix produced for this crane is:

$$
\boldsymbol{S} = \begin{bmatrix} 8.45226241e+03 & 1.45559342e+06 & -1.46645225e+06 \\ -1.59778721e+08 & 1.34931905e+08 & 1.34888745e+08 \\ -3.49330583e+07 & 2.58781733e+07 & 2.58700268e+07 \end{bmatrix}
$$

A similar procedure is also used to find the component rotation compensation matrix **A**:

$$
\boldsymbol{A} = \begin{bmatrix} 2.33136689e-05 & 3.37558959e-05 \\ 1.37893039e-05 & 1.95744406e-05 \\ 6.92998583e-06 & -3.69859663e-05 \end{bmatrix}
$$

Figure 21 shows where the forces are applied on the crane model.

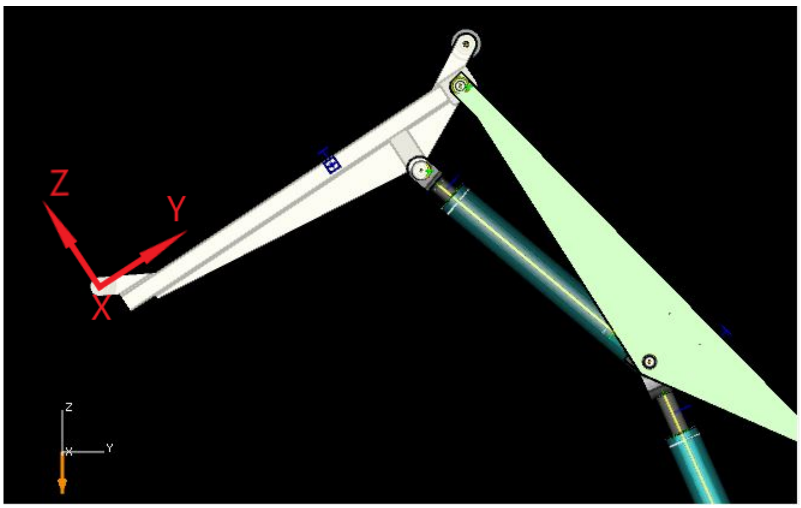


Figure 21: Forces on the Crane

6.2 Performance

6.2.1 Simulation With Static Load

To validate the inverse method two sets of simulations was performed. The procedure used is described below:

1. Run simulation with a known load case (-76 Newton in the Z-direction)
2. Retrieve results from virtual strain gauges
3. Use the virtual strain gauge results as input to the inverse method
4. Run a new simulation based on the output forces from the inverse method

The results can be seen in figure 22 and 23. Figure 22 shows the forces that are created with the inverse method. Figure 23 shows the strains that are produced in the model when applying the forces generated from the inverse method and the original strains measured in the first simulation.

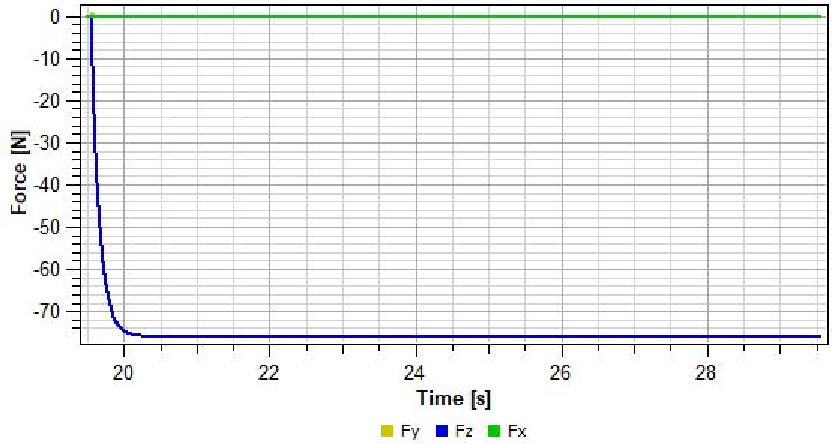


Figure 22: Simulation Forces

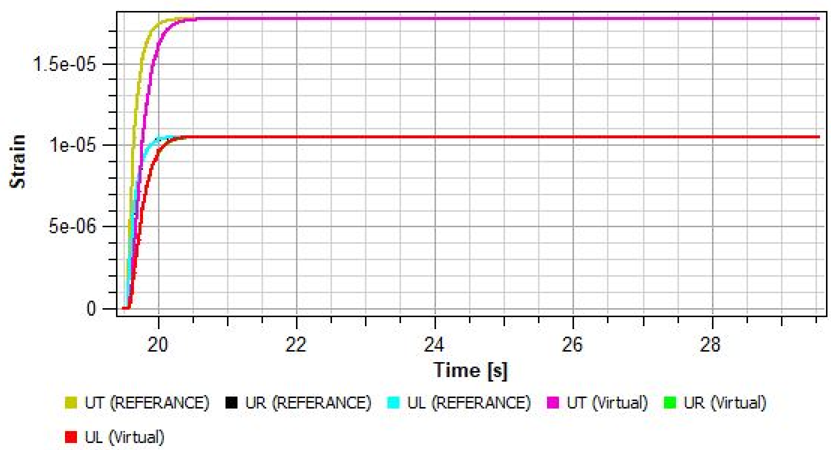


Figure 23: Simulation Strain

6.2.2 Simulation With Real Data

Figures 24 and 25 shows the simulation results when feeding the inverse method with real captured data. During this recording the crane was standing still and a few seconds into the recording a payload of 7.6 kg was applied to the crane for a few seconds and then removed again. As with the case of the simulation described in the previous section the strains are nicely reproduced, however the forces that the inverse method produces are not in any way representative of the real case.

Several simulations was performed using real data, where the data was recorded during standstill with and without payload and while the crane was hoisting a payload. Like the results shown in figures 24 and 25, the other simulations showed large forces in Y- and Z-direction, though the strains follows their reference.

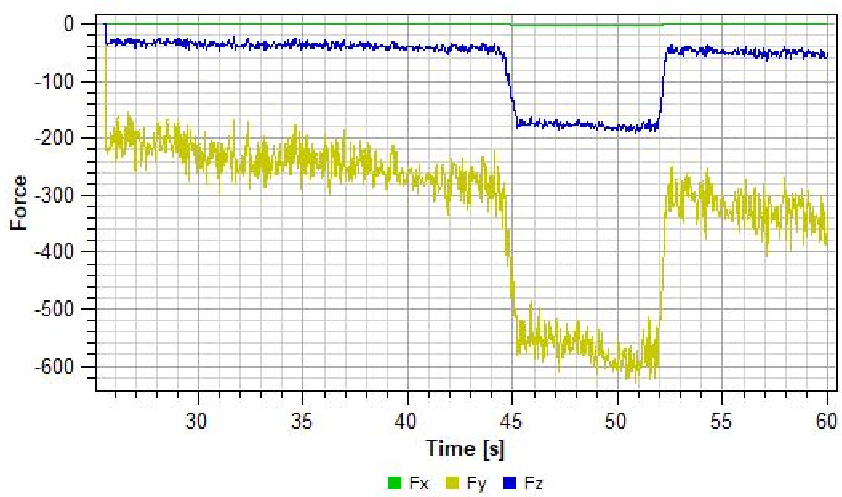


Figure 24: Simulation Forces With Real Data

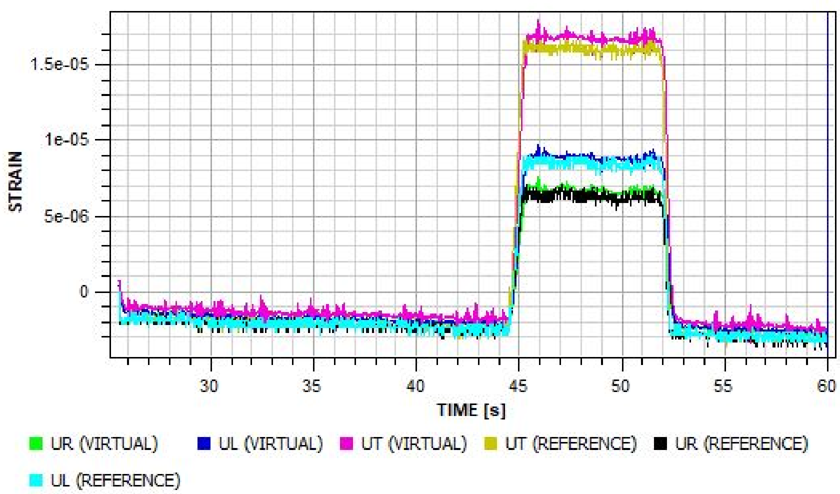


Figure 25: Simulation Strain With Real Data

6.3 Updated Inverse Method

In parallel to the work of this thesis, work has been done to improve on the inverse method. Section 4.4 in Moi (2019) suggests that a inter-component compliance matrix may be more favourable to the compliance matrix made from only one component.

A new compliance matrix **S** was calculated replacing the strain gauge marked "UL" with the strain gauge marked "Reference" (Noted as "REF" in figure 26) in the inverse method algorithm. The new compliance matrix is showed below:

$$\boldsymbol{S} = \begin{bmatrix} -1654561.7568496 & 2928120.39012839 & -260146.76598859 \\ -6809659.47885664 & -515511.90187169 & 23934323.05814316 \\ -5595408.22319956 & -99005.17780171 & 4590815.94145071 \end{bmatrix}$$

Simulations using the updated compliance matrix gives very promising results. Figures 26 and 27 shows that it performs well at recreating the strains and figure 28 shows a big improvement in the recreation of the forces. The data used in this simulation comes from a recording where the same procedure is used as in section 6.2.2.

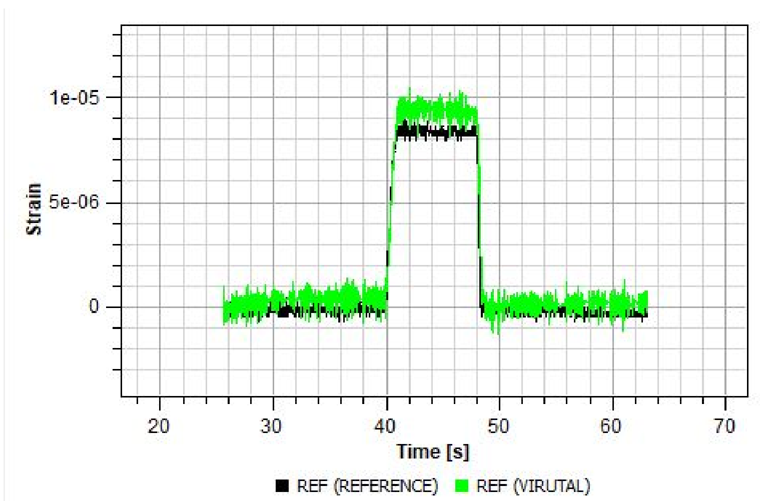


Figure 26: Simulation Strain With Real Data

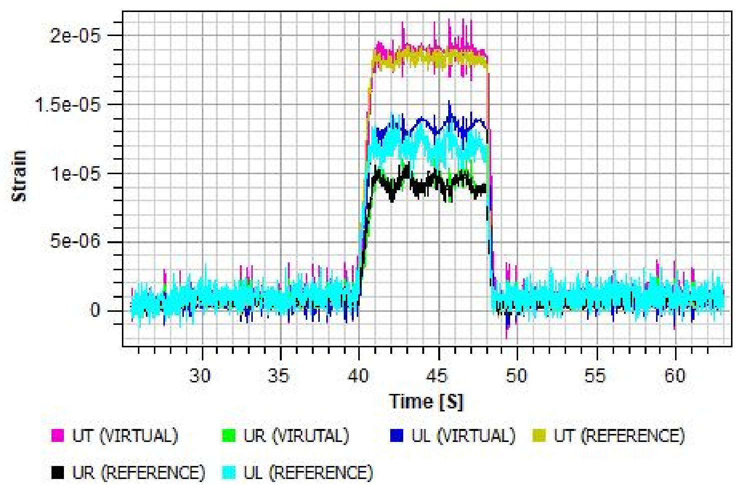


Figure 27: Simulation Strain With Real Data

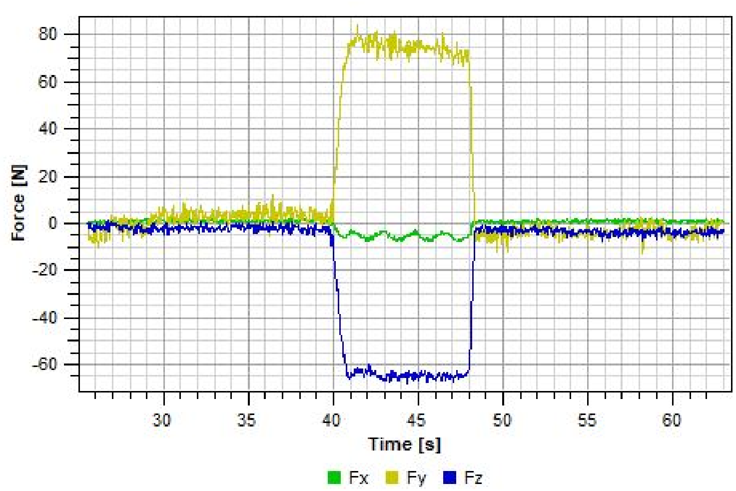


Figure 28: Simulation Force With Real Data

6.4 Challenges

Strain gauges operate on low voltage and are susceptible noise. Preventive measures can and has been applied, such as using low pass filters and improve cabling to reduce the noise. Adding a strain gauge for temperature compensation to reduce drifting is also done. However, noise cannot be removed wholly and the method proves notably vulnerable to noise when the elements in the compliance matrix for the structure is of a high degree. Noise of only a few micro strain can cause artificial forces in the magnitude of hundreds as seen in figure 24.

6.5 Simulation Script

A simulation script (Appendix C) has been developed for testing the digital twin of the crane with real data. The next paragraphs will briefly describe the different sections in the code.

Lines 8 to 12 are for initialising an UDP socket needed for sending data over UDP to the server.

Lines 14 to 62 make up the method `goToInitial()`, which ensures that the start position of the crane is the same in the model as it is in the recorded data. In other words, the length of the actuators are in the right position when the rest of the simulation commences.

Lines 80 to 112 is a routine for filtering the recorded data before it is used. The filter used is a eight order Butterworth low-pass filter with a 10 Hz cut-off frequency. The temperature compensation is also applied here.

Lines 128 to 172 is the code for calculating the compliance matrix of the structure needed for the inverse method. The pseudo-code for the procedure is described below:

1. Apply unit load in (X-, Y- and Z-Direction)
2. (a) Wait for structure to stabilise

   (b) Get strain vector from virtual strain gauges

   (c) Add strain vector as column in matrix S
3. Calculate inverse of S

Lines 172 to 205 is the code for calculating the component rotation compensation matrix of the structure.The pseudo-code for the procedure is described below:

1. Apply actuator length change (1 cm) for each actuator
2. (a) Wait for structure to stabilise
   (b) Get strain vector from virtual strain gauges
   (c) Add strain vector as column in matrix A
3. Divide A by 0.01 (1 cm)

Lines 209 to 255 contains a loop that goes through the recorded data, calculates forces with the inverse method and updates the Fedem model with these values for each time step.

7 Digital Twin Cloud Software

The *Digital Twin Cloud Software* (Johansen et al. (2018)) is still in development. As such there are parts of the system that needed reconfiguration for the implementation of new digital twin. This section describes software developed during this project that aids the automation of implementing a new digital twin to the *Digital Twin Cloud Software*.

7.1 Digital Twin Configuration Platform

The digital twin configuration platform is meant as a tool to aid implementing a digital twin to the *Digital Twin Cloud Software*. The purpose of this tool is to let the user generate a "Digital Twin Configuration File" that contains information needed for the server to properly map sensor input to the digital twin model. Also the platform enables the generation of VTFx-files which are needed for the 3D-representation in the web browser.

7.1.1 Front-End

The graphical user interface (GUI) is built using HTML and JavaScript. The code can be found in Appendix D. The GUI is simple and divided into three main parts. The division is illustrated in figure 29.

The box labelled "1" contains options that lets the user set the correct configurations for the digital twin. The options are:

1. **Choose File System**
   Supports Fedem models saved as ".fmm" or as a FMU.
2. **Fedem File / FMU**
   Select the ".fmm"-file or the FMU that the digital twin shall represent.
3. **Precision**
   Set the expected precision. This must be same as the precision set in the remote options in Catman.
4. **Generate VTFx-file**
   This option allows for the automatic generation of a set of JSON-files that is needed for 3D-Visualisation. See section 7.1.3 for more details on these files.

Note that if the selected "File System" is Fedem the ".ftl" files for all the parts must also be uploaded. An additional option box will appear for this purpose.

5. **Show functions**
   This option allows the user to see all the functions set in the Fedem model. (Box number three)

The box labelled "2" allows the user to specify all the sensors attached to the physical asset and specify what the server should do with the incoming values from that sensor. Note that the order in which the senors are added is important and must match the order in which the sensors are listed in Catman. A description of the columns can be seen in the list below.

1. **Name**
   Sensor identifier.
2. **Sensor Type**
   Set the sensor type. Instructs the server of the unit of the sensor values.
3. **Purpose**
   Instructs the server of what to do with the sensor values.
4. **Function ID**
   Maps the sensor values to the correct function in the Fedem model. Set to "-1" if not used.
5. **UDP Index**
   Sensor data is received on the server as a series of bytes. The "UDP Index" instructs the server of which bytes represents which sensors.

The box labelled "3" displays all the function found in the specified Fedem model, and it will only be shown if a ".fmm"-file or FMU is chosen and the "Show functions" box is ticked. The purpose of this section is to help the user map the correct external function in the Fedem model to the correct sensor input.

When all options are selected and all sensors are added the user can press "GENERATE FILE" which creates the Digital Twin Configuration file. Figure 30 is an illustrates how the configuration file may be formatted. However, the decision on how the configuration file should be formatted has yet to be taken.

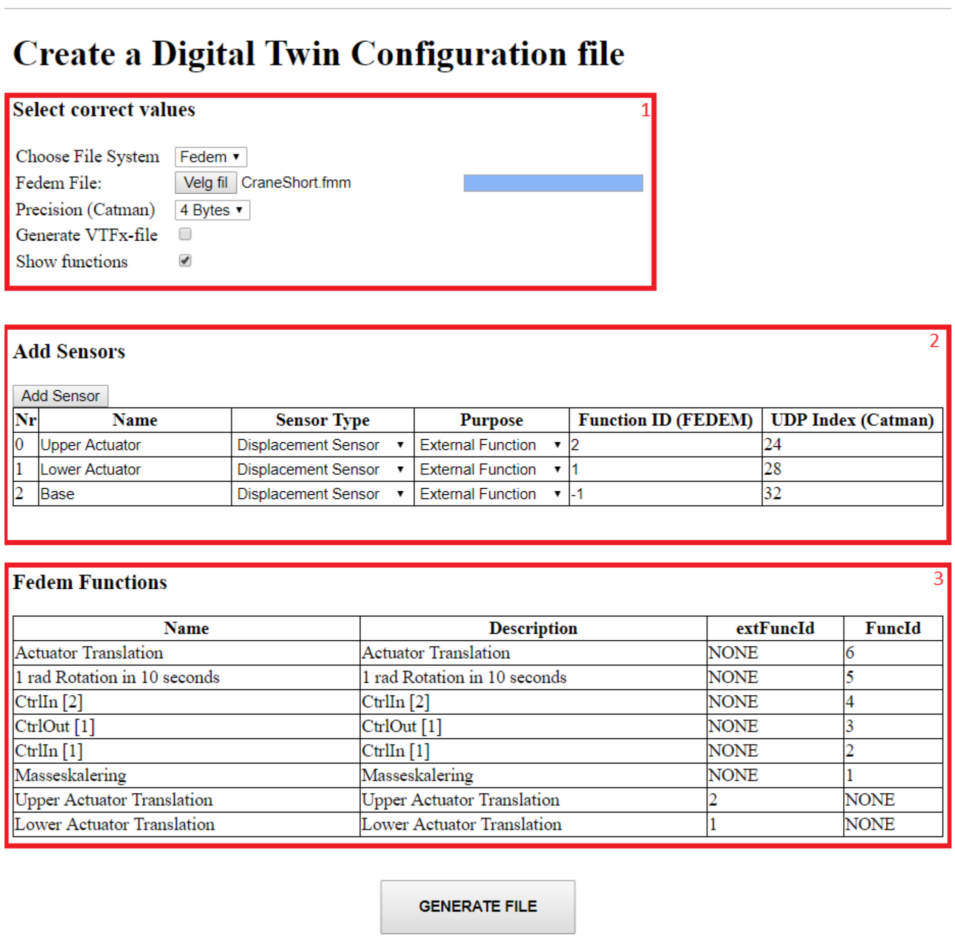


Figure 29: Create a Digital Twin Configuration file

```
Name: CraneShort
precision: 8 Bytes
sensor:
- '{"name":"Spider8_1 CH 0 Referance","sensorType":"Strain Gage","sensorPurpose":"Inverse
  Method","funcID":"13","UDPIndex":"24"}'
- '{"name":"Spider8_1 CH 1 UR","sensorType":"Strain Gage","sensorPurpose":"Inverse
  Method","funcID":"11","UDPIndex":"32"}'
- '{"name":"Spider8_1 CH 2 UL","sensorType":"Strain Gage","sensorPurpose":"External
  Function","funcID":"10","UDPIndex":"40"}'
- '{"name":"Spider8_1 CH 3 UT","sensorType":"Strain Gage","sensorPurpose":"Inverse
  Method","funcID":"9","UDPIndex":"48"}'
- '{"name":"Spider8_1 CH 4 Wire Lower","sensorType":"Displacement Sensor","sensorPurpose":"External
  Function","funcID":"12","UDPIndex":"56"}'
- '{"name":"Spider8_1 CH 5 Wire Upper","sensorType":"Displacement Sensor","sensorPurpose":"External
  Function","funcID":"1","UDPIndex":"64"}'
- '{"name":"Spider8_1 CH 6 Engine Lower UNUSED","sensorType":"Voltage","sensorPurpose":"Unused
    ","funcID":"-1","UDPIndex":"72"}'
- '{"name":"Spider8_1 CH 7 Engine Upper UNUSED","sensorType":"Voltage","sensorPurpose":"Unused
    ","funcID":"-1","UDPIndex":"80"}'
- '{"name":"Spider8_1 CH 8 UNUSED","sensorType":"Voltage","sensorPurpose":"Unused
    ","funcID":"-1","UDPIndex":"88"}'
- '{"name":"Spider8_1 CH 10 Encoder Base","sensorType":"Encoder","sensorPurpose":"External
  Function","funcID":"9","UDPIndex":"96"}'
- '{"name":"Spider8_1 CH 11 Temperature Compensation","sensorType":"Strain Gage","sensorPurpose":"
    Monitoring","funcID":"-1","UDPIndex":"104"}'
```


Figure 30: Digital Twin Configuration file

7.1.2 Back-End

Under the development of the Digital Twin Configuration platform the previous setup of the server for the *Digital Twin Cloud Software* was undergoing changes and was not available. Notably the server was rebuilt using Python instead of Node. Therefore a temporary server solution (`Server.py`) was made in Python for the development of the configuration platform. However, all functionality not dependent on the server setup is made modular and put in separate files (`ServerSupport.py` and `parseExport.py`) so that it easily can be implemented in the *Digital Twin Cloud Software* at a later stage. The code for the back-end can be found in appendix E.

`Server.py` contains code that handles the communication between the back-end and the front-end, and it contains the logic for handling user requests. Thus ensuring that information is sent back and forth between the front-end and the back-end is sent correctly.

`ServerSupprt.py` is the code that holds the logic that interprets user request and forms the proper response. Such as finding all the functions in a Fedem model.

`parseExport.py` contains methods that generate the JSON-files (See section 7.1.3) needed for 3D-visualisation in the *Digital Twin Cloud Software*. Note that the function `generatePart()` uses a class named `VTFXExporter` which can be found in a file called `Exporter.py`. This file is not included in the appendix because of license restrictions.

7.1.3 JSON-files

Each digital twin needs two types of JSON-files for the 3D-visualisation. The first type is a "master" file, and there exist only one such file for each digital twin. This file contains vital information about the parts the make up the model, how these parts relate to each other and an identifier. This information is gathered from either the ".fmm"-file or FMU describing the model.

The second type is a "part" file, and there exist one for each part that make up the digital twin. These files contains the description of the surface geometry of the parts and is needed for the visualisation of the digital twin. The surface geometry is retrieved using the `export.py` which exports a VTFx-representation from ".ftl" files.

7.2 Digital Twin JavaScript Class

The JavaScript class `DigitalTwin.js` (Appendix B) handles the data from the the JSON files (See section 7.1.3) and calls the `usg.ts` (Listing 3 in Johansen et al. (2018)) module which in turn creates the 3D-representation of the model.

7.3 Results

7.3.1 Old Digital Twin Cloud Software

Figure 31 shows the result of the digital twin using a slightly modified version of the old *Digital Twin Cloud Software*. The the most significant change in the code is the addition of the JavaScript class `DigitalTwin.js`. The updated versions of `index.js`, `index.html` and `usg.ts` (listing 2, 3 and 4 in Johansen et al. (2018)) can be found in appendix F.

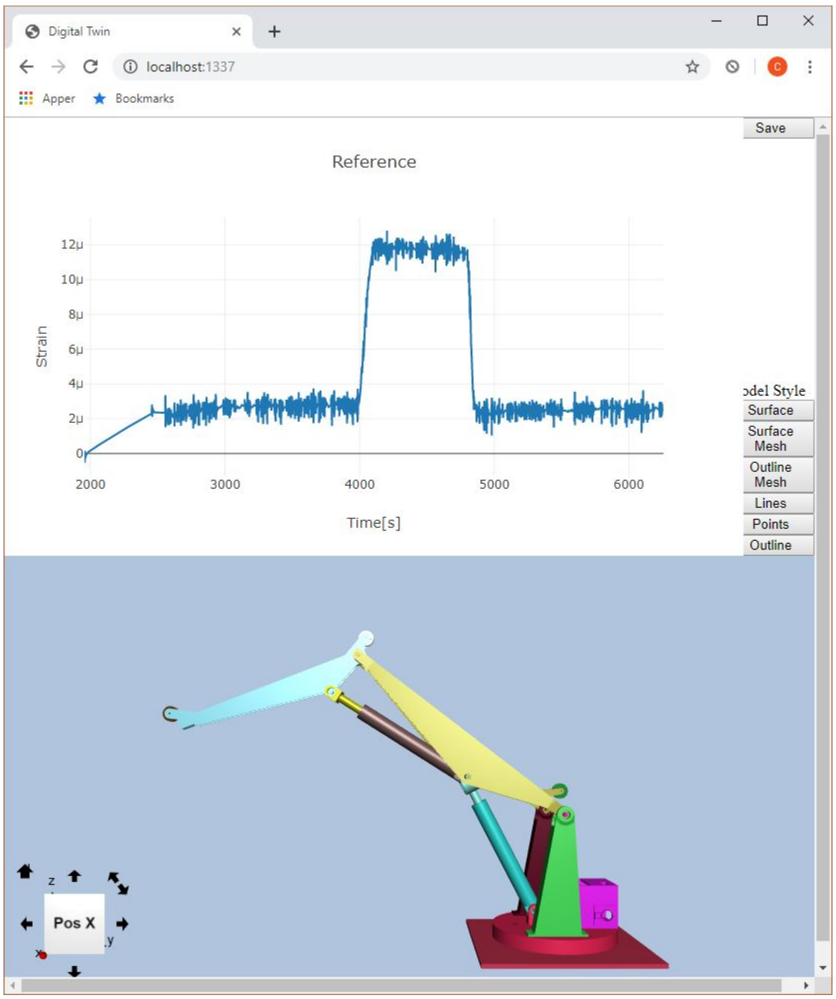


Figure 31: Digital Twin Cloud Software

8 Functional Mock-Up Interface

This section describes the Functional Mock-up Interface (FMI), why it is included in this project and the challenges that were introduced.

8.1 What is FMI

"Functional Mock-up Interface (FMI) is a tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of xml-files and compiled C-code." *FMI* (n.d.).

The FMI is designed so that engineers can generate a Functional Mock-up Unit (FMU) of their models (FE-, mathematical- model etc). A FMU is simply put a "black box" representation of the model with predefined input and output ports. This way a FMU can easily be exchanged between engineers and tested. For instance a control engineer for a car company can get FMU's from the mechanical- and the electrical department to test the performance of the control system.

8.2 Digital Twin Cloud Software

The new *Digital Twin Cloud Software* is built to primarily support FMU's. For this reason a FMU of the crane model has been generated using the script `buildFMU.py` provided in Fedem (Version R7.3).

The process of integrating the crane FMU into the *Digital Twin Cloud Software* has gone without any mayor issues. However, the crane FMU does not exhibit the same behaviour as the Fedem model of the crane.

9 Further Work

This section describes work that is needed for the digital twin to be fully operational and used for structural monitoring.

The cables connecting the strain gauges mounted on the crane and the spiders should be changed. The new cables should be as short as possible to minimise noise on the signal, but still be long enough so that they do not inhibit the movement of the crane.

As mentioned in section 5.5.2 it is not possible to calculate fatigue in real time in Fedem. Exploring the possibility to run the fatigue calculation from an external source and then retrieve the data is a task that should be done.

Fedem has no support for monitoring buckling or instability in the models. Since the crane is built for students and the there is a restriction on how much the students are allowed to load the crane buckling is not likely to happen. However, there is plans to place the crane on a raft in DMT's water laboratory. In those conditions stability is a factor, and therefore a solution for monitoring the stability should be explored. Either through Fedem or other means.

The work on the "Digital Twin Configuration Platform" should continue and be integrated with the *Digital Twin Cloud Software*, and decide on the format for the digital twin configuration file. This way the digital twin of the crane can be implemented to the *Digital Twin Cloud Software* in an automated way.

The work on the crane FMU should continue to find the source of the changes in behaviour.

References

Christiansen, E. E. (2017), 'Digital twin suspension system'. Project Thesis, NTNU.

Christiansen, E. E. (2018), Digital twin of crane, Master's thesis, NTNU.

FMI (n.d.), `https://fmi-standard.org/`. Accessed: 04.06.2019.

Gyberg, F. (2015), 'Development of physical crane model for use in student lab'. Project Thesis, NTNU.

HBM (n.d.), *Operating Manual: Spider8 and Spider8-30*.

Johansen, C., Jensen, S. N., Børhaug, A., Sande, O. H. S. & Brekke, K. (2018), 'Cloud software for digital twin modeling and monitoring'. Project Thesis, NTNU.

Moi, T. (2018), 'Inverse methods for digital twins using fedem vpm solver'. Project Thesis, NTNU.

Moi, T. (2019), Digital twin based structural monitoring of knuckle boom crane, Master's thesis, NTNU.

Appendices

A Task Description

Date

Faculty of Engineering
Department of Mechanical and Industrial Engineering

MASTER 2019
FOR
STUD.TECHN. CHRISTIAN JOHANSEN

CLOUD BASED DIGITAL TWIN MONITORING OF OFFSHORE KNUCKLE BOOM CRANE
Sky-basert Digital Twin monitorering av offshore knekk bom krane

ANSYS and SAP Engineering Center of Excellence are developing digital twin solutions for predictive maintenance and monitoring of structural integrity. The companies have therefore developed hardware and software solutions for instrumentation of physical structures and mechanisms. These solutions are currently in a prototype phase and we want to benchmark these systems on an offshore knuckle boom crane at NTNU/Marine based on the cloud solution developed by the candidate and other project students from MTP

Tasks include:

1. Identify offshore knuckle boom crane failure modes to be detected by the cloud based digital twin system (fatigue, yield, buckling, instability etc.).
2. Identify the functional requirements for monitoring of the most critical failure modes
3. Implement a generic configuration system in the cloud solution for easy adaption to other digital twin crane applications (other sensors, actuators, streaming analytics etc.)
4. Reconfigure the cloud based system based on outputs from the three previous tasks
5. Implement required software functionality in the cloud solution to support the requirements from task 2 (streaming analytics, curve plotting, 3D visualization, event trigging, report generation)
6. Instrument the physical crane based on results from task 2
7. Setup and benchmark the virtual and physical crane real time cloud communication

If time permits:

8. Write a scientific digital twin paper with the supervisors

Contact:
At the department (supervisor, co-supervisor): Terje Rølvåg and Bjørn Haugen
From Fedem Technology AS: Runar H. Refsnæs

Address: NO-7491 TRONDHEIM Norway
Org.nr. 974 767 880
Email: mtp-info@mtp.ntnu.no
https://www.ntnu.edu/mtp

B Digital Twin Java Class

```
class DigitalTwin{
    /*
        This is class retrieves all the necessary information needed for the USG.js module to create a 3D visualisation
        of the model.

        Two types of files are needed:
            - JSON Master file
                - List of parts
                - Base ID for each part
                - Local to global coordinate system transformation matrix for each part
            - JSON Part file (For each part in model)
                - Part geometry description
     */

    constructor(name){
        // Initiate variables
        this.name = name;
        this.fileName = this.name.concat(".json");
        this.directory = "/js/".concat(this.name); // /js/Name/
        this.parts = [];
        this.arrays = [];
        this.baseId = [];

        // Create Model
        this.createModel();
    }

    // Return part index
    getPartIndex(baseID){
        /*
            To properly select a part when updating its position we need the index it has been given. The index of a
            part corresponds to the parts position in the list of parts.
            e.g:
            this.parts = [arm, base, pulley, upperArm, ....] thus the index of "pulley" is 2.
         */

        var index = -1;

        for(var i = 0; i<this.baseId.length; i++){
```

```
            if(this.baseId[i] == baseID){
                index = i;
            }
        }
        return index;
    }

    // Return array with Base IDs
    getIDS(){
        return this.baseId;
    }

    // Create 3D model
    async createModel(){
        // Retrieve information from JSON master file
        await this.findParts();

        // Generate 3D visualisation for each part
        for(var i = 0; i<this.parts.length;i++){
            await this.loadParts(this.parts[i],this.arrays[i],i)
                ;
        }
    }


    loadParts(Name,Arr,index){
        /*
            This function loads part geometry and calls the
                function "addPartGeometry" which uses the Ceetron
                 USG module
            to create the 3D visualisation of this part.
         */

        return new Promise((resolve,reject) => {

            var temp = this.directory.concat("/").concat(Name)
            var oReq = new XMLHttpRequest();

            oReq.onload = () => {
                var data = JSON.parse(oReq.responseText);
                myApp.addPartGeometry(data,Arr[0],Arr[1],Arr[2],
                    index);
                resolve()
            }

            oReq.open("get",temp, true);
            oReq.send();
        });
    }
```

```
    findParts(){
        /*
             Retrieve information from JSON master file:
                - List of parts
                - Base ID for each part
                - Local to global coordinate system transformation matrix for each part
         */
        return new Promise((resolve,reject) => {

            const temp = this.directory.concat("/").concat(this.fileName);
            var oReq = new XMLHttpRequest();

            oReq.onload = () => {

                var data = JSON.parse(oReq.responseText);
                var files = data.ListOfFile;
                var coordinates = data.Coordinates;
                var IDs = data.baseID;
                var index = 0;
                const tempArray = [];

                for(var i = 0; i<files.length;i++){
                    index = i*3;
                    files[i] = files[i].replace(".ftl",".json");
                    this.parts.push(files[i]);
                    this.baseId.push(IDs[i]);
                    this.arrays.push([coordinates[index],coordinates[index+1],coordinates[index+2]]);
                }

                if(this.parts.length == files.length){
                    resolve();
                }else{
                    reject();
                }

            };

            oReq.open("get",temp, true);
            oReq.send();
        });
    }
```

```
}
```

Listing 1: DigitalTwin.js

C Digital Twin Simulation Script

```
from fedem.fedemdll.vpmSolverRun import VpmSolverRun
import socket
import numpy as np
from scipy import signal
import struct

# Configure UDP Socket
PHYSICAL_TWIN_ADDRESS = ("0.0.0.0", 7331)

WEB_SERVER_ADDRESS = ("localhost", 8001)#"129.241.90.187", 8001)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(PHYSICAL_TWIN_ADDRESS)

def goToInitial(actuatorLowerInit, actuatorUpperInit):
    # Taken from Model
    actuatorUpperModel = 0.040999981 # Prismatic joint 3
    actuatorLowerModel = 0.10099981  # Prismatic joint 1

    # Timing
    duration = 5  # SECONDS
    timeStep = 0.01
    numberOfSteps = round((duration/timeStep))

    # STEPS
    actuatorLower = 0
    actuatorUpper = 0

    # Interpolation
    stepUpper = (actuatorUpperInit - actuatorUpperModel) /
        numberOfSteps
    stepLower = (actuatorLowerInit - actuatorLowerModel) /
        numberOfSteps


    # Go to initial
    for i in range(0, numberOfSteps, 1):
        time = twin.getCurrentTime()
        twin.setExtFunc(1, time, actuatorLower)
        twin.setExtFunc(2, time, actuatorUpper)
        twin.solveNext()

        # Get transformation data for all triads and parts
        plotData = bytearray(struct.pack("d",twin.getFunction
            (35)))
        transformationData = twin.save_transformation_state()
        message = plotData + transformationData
```

```
        sock.sendto(message, WEB_SERVER_ADDRESS)


        actuatorLower = actuatorLower + stepLower
        actuatorUpper = actuatorUpper + stepUpper

    # One Second of Idle time
    for i in range(0, 100, 1):
        time = twin.getCurrentTime()
        twin.setExtFunc(1, time, actuatorLower)
        twin.setExtFunc(2, time, actuatorUpper)
        twin.solveNext()

        # Get transformation data for all triads and parts
        plotData = bytearray(struct.pack("d",twin.getFunction
            (35)))
        transformationData = twin.save_transformation_state()
        message = plotData + transformationData
        sock.sendto(message, WEB_SERVER_ADDRESS)

# Open file with sensor data
with open('C:\\Users\\chris\\Documents\\FedemProsjekt\\
    DT_EXAMPLE\\CraneRun\\Good_New\\Apply2.txt', 'r') as file:
    lines = file.readlines()

# Remove lines of meta-data
lines = lines[39:(len(lines)-1)]

# DT setup parameters
fedem_model_path = 'CraneShort.fmm'

# Declare variables
unfiltered_UT = []
unfiltered_UR = []
unfiltered_UL = []
unfiltered_REF = []


# FILTERING
for i in range(len(lines)):

    # Get sensor data
    line = lines[i]
    temp = line.split('\t')
    REF = float(temp[1]) * 10.0 ** -6
    UR = float(temp[2]) * 10.0 ** -6
    UL = float(temp[3]) * 10.0 ** -6
    UT = float(temp[4]) * 10.0 ** -6
    TempComp = float(temp[11]) * 10.0 ** -6
```

```python

    # Temperature compensation
    REF = REF - TempComp
    UR = UR - TempComp
    UL = UL - TempComp
    UT = UT - TempComp

    # Fill Arrays
    unfiltered_UT.append(UT)
    unfiltered_UR.append(UR)
    unfiltered_UL.append(UL)
    unfiltered_REF.append(REF)


# Get filter parameters
filter_a, filter_b = signal.butter(8, 10/50)

# Filter signals (Low Pass (Butterworth), Cutt-Off: 10Hz, Order: 8)
filtered_UT = signal.filtfilt(filter_a, filter_b, unfiltered_UT)
filtered_UR = signal.filtfilt(filter_a, filter_b, unfiltered_UR)
filtered_UL = signal.filtfilt(filter_a, filter_b, unfiltered_UL)
filtered_REF = signal.filtfilt(filter_a, filter_b, unfiltered_REF)


# Initiate VpmSolverRun object
with VpmSolverRun(fedem_model_path) as twin:
    # Initialization of solver (Needed for fedem functions)

    for n in range(2):
        twin.solveNext()

    # Get initial values for goToInit()
    line = lines[0]
    temp = line.split('\t')
    actuatorLower = (float(temp[5]) / 100)   # Prismatic Joint 1
    actuatorUpper = (float(temp[6]) / 100)   # Prismatic Joint 3

    # ---------------------------------- CALCULATE COMPLIANCE MATRIX START ----------------------------------
    # This part is only needed once, unless there is made structural changes to the Fedem model

    # Variable declarations
    out_def = [19, 20, 21]  # UT UR UL
    Sinv = np.zeros((3, 3))
    inp = 1                 # Input Force [N]
    inp_def = [6, 7, 8]     # Fx Fy Fz
```

```python

    for j in range(3):

        # Variable declarations
        out = [np.inf, np.inf, np.inf]  # Strain Output (From virtual strain gauges)

        # Apply Force
        time = twin.getCurrentTime()
        twin.setExtFunc(inp_def[j], time, inp)

        # Let Structure settle after applying force
        for k in range(600):
            twin.solveNext()

        # Get values from virtual strain gauges
        out[0] = twin.getFunction(out_def[0])   # UT
        out[1] = twin.getFunction(out_def[1])   # UR
        # out[2] = twin.getFunction(out_def[2]) # Inverse method using "UL"
        out[2] = twin.getFunction(35)           # REF

        time = twin.getCurrentTime()
        Sinv[:, j] = np.array(out)

        # Cancel Force
        twin.setExtFunc(inp_def[j], time, 0)
        twin.solveNext()

        # Half second of idle time
        for k in range(50):
            twin.solveNext()

    # Calculate compliance matrix S
    S = np.linalg.inv(Sinv)
    print("S")
    print(S)

    # ------------------------------ CALCULATE COMPLIANCE MATRIX END ------------------------------
    # ------------------------------ CRC Begin ------------------------------

    A = np.zeros((3, 2))
    strain = [np.inf, np.inf, np.inf]

    # Move Actuators
    for i in range(0, 2, 1):
        for j in range(0, 101, 1):
```

```
            time = twin.getCurrentTime()
            twin.setExtFunc(i, time, j*10**-4)
            twin.solveNext()

        # Idle Time
        for j in range(0, 101, 1):
            twin.solveNext()

        strain[0] = twin.getFunction(out_def[0]) - out[0]  # UT
        strain[1] = twin.getFunction(out_def[1]) - out[1]  # UR
        strain[2] = twin.getFunction(35)         - out[2]  # REF
        A[:, i] = np.array(strain)

    # Reset Actuators
    for i in range(0, 2, 1):
        for j in range(101, 0, -1):
            time = twin.getCurrentTime()
            twin.setExtFunc(i, time, j * 10 ** -4)
            twin.solveNext()

    print('A:')
    A = A/0.01
    print(A)

    # ---------------------------------------- CRC END ----------------------------------------

    goToInitial(actuatorLower, actuatorUpper)

    for i in range(0, len(lines), 1):
        time = twin.getCurrentTime()

        # Get sensor data
        line = lines[i]
        temp = line.split('\t')

        # Calculate Movement
        actuatorLower = (float(temp[5]) / 100) - 0.10099981  # Prismatic Joint 1
        actuatorUpper = (float(temp[6]) / 100) - 0.040999981  # Prismatic Joint 3
        base = float(temp[10]) * (3.14 / 180) * (10 / 16500)  # Radians
        a = np.array([actuatorLower, actuatorUpper])


        # CALCULATE FORCES
        # F = S @ (np.array([filtered_UT[i], filtered_UR[i], filtered_UL[i]]) -(A@a) # Inverse method using "UL"
```

```
        F = S @ (np.array([filtered_UT[i], filtered_UR[i],
            filtered_REF[i]]) -(A@a))

        # SET FORCE
        twin.setExtFunc(6, time, F[0])
        twin.setExtFunc(7, time, F[1])
        twin.setExtFunc(8, time, F[2])

        # PLOTTING STRAINS (REFERENCES)
        twin.setExtFunc(9, time, unfiltered_UT[i])
        twin.setExtFunc(10, time, unfiltered_UR[i])
        twin.setExtFunc(11, time, unfiltered_UL[i])
        twin.setExtFunc(12, time, unfiltered_REF[i])

        # Actuator movement
        twin.setExtFunc(1, time, actuatorLower)
        twin.setExtFunc(2, time, actuatorUpper)

        # Plotting Actuator movement (REFERENCE)
        twin.setExtFunc(3, time, actuatorLower + 0.10099981)
        twin.setExtFunc(4, time, actuatorUpper + 0.040999981)

        # Base Rotation
        twin.setExtFunc(5, time, base)

        twin.solveNext()

        # Get transformation data for all triads and parts
        plotData = bytearray(struct.pack("d",twin.getFunction
            (35)))
        transformationData = twin.save_transformation_state()
        message = plotData + transformationData
        sock.sendto(message, WEB_SERVER_ADDRESS)
```

Listing 2: CraneShortUDP.py

D Front-End

D.1 index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="shortcut icon" href="">
    <meta charset="UTF-8">
    <title>Create a Digital Twin</title>
    <style>
        #sensorTable, #FedemFunctionsTable,#FedemFunctionsTable
            th, #sensorTable th,#FedemFunctionsTable tr, #
            sensorTable tr,#FedemFunctionsTable td, #sensorTable
            td{
            border: 1px solid black;
            border-collapse: collapse;
        }
        label {
            width: 180px;
            clear: left;
            text-align: left;
            padding-right: 10px;
        }
        #fileGenButton{
            padding: 15px 32px;
            position: relative; left: 50%;
            transform: translateX(-50%);
        }
        .test{
            border-color: white;
            border-style: none;
            border-width: 0px;
            width: 99%;
            height: 100%;
        }
    </style>
</head>


<body>
    <h1>Create a Digital Twin Configuration file</h1>
    <h3>Select correct values</h3>
    <form>
        <table>
            <tr>
                <td><label>Choose File System</label></td>
                <td>
```

```
                    <select id="FMUorFMM" onchange="selectFileSystem()">
                        <option>FMU</option>
                        <option>Fedem</option>
                    </select>
                </td>
            </tr>
            <tr id="FMUDisplay">
                <td><label>FMU:</label></td>
                <td><input type="file" accept=".fmu,.zip" id="fmu" name="Chose File" onchange="loadFMU()"></td>
                <td><progress value="0" max="100" id="progressBarFMU"></progress></td>
            </tr>
            <tr style="display: none" id="FedemFileDisplay">
            <td><label>Fedem File:</label></td>
            <td><input type="file" accept=".fmm" id="fmm" name="Chose File" onchange="loadFmmFile()"></td>
            <td><progress value="0" max="100" id="progressBar"></progress></td>
            </tr>
            <tr style="display: none" id="showVTFx">
                <td><label>Choose ".ftl"-files</label></td>
                <td><input type="file" accept=".ftl" id="FTL" onchange="loadFTLFiles()" multiple></td>
                <td><progress value="0" max="100" id="progressBarFtl"></progress></td>
            </tr>
            <tr>
                <td><label>Precision (Catman)</label></td>
                <td>
                    <select id="Bytes">
                        <option>4 Bytes</option>
                        <option>8 Bytes</option>
                    </select>
                </td>
            </tr>
            <tr>
                <td><label>Generate VTFx-file</label></td>
                <td><input type="checkbox" onclick="generateVTFx()" id="VTFx"></td>
            </tr>
            <tr style="display: none" id="showFunctionsCheckbox">
                <td><label>Show functions</label></td>
                <td><input type="checkbox" onclick="showFmmFunctions()" id="showFmm"></td>
            </tr>
```

```
        </table>
    </form>
    <br>
    <br>
    <h3>Add Sensors</h3>
    <form onsubmit="return false">
        <input type="submit" value="Add Sensor" onclick="addSensor()">
    </form>
    <form>
    <table style="width: 100%" id="sensorTable" >
        <tr>
            <th>Nr</th>
            <th>Name</th>
            <th>Sensor Type</th>
            <th>Purpose</th>
            <th>Function ID (FEDEM)</th>
            <th>UDP Index (Catman)</th>
        </tr>
    </table>
    </form>
    <br>
    <br>
    <div style="display: none" id="divFedem">
        <h3>Fedem Functions</h3>
        <table style="width: 100%" id="FedemFunctionsTable">
            <tr>
                <th>Name</th>
                <th>Description</th>
                <th>extFuncId</th>
                <th>FuncId</th>
            </tr>
        </table>
    </div>
    <br>
    <br>
    <button onclick="sendFile()" id="fileGenButton"><b>GENERATE FILE</b></button>

<script>
    // VARIABLES
    var nSensors = 0;
    var udpIndex = 24;
    var sensors = [];
    var VTFxExport = false;
    var functionsLoaded = false;
    var FMU = true;
    var select =
                "<select id=\"sensorPurpose\" style='width: 
```

```
                    100%;' class='test'>\n" +
                    "<option>Inverse Method</option>\n" +
                    "<option>External Function</option>\n" +
                    "<option>Monitoring</option>\n" +
                    "<option>Unused</option>\n" +
                    "</select>";
    var sensorType =    "<select id=\"sensorType\" class='test'>\n" +
                        "<option>Accelerometer</option>\n" +
                        "<option>Displacement Sensor</option>\n" +
                        "<option>Strain Gage</option>\n" +
                        "<option>Encoder</option>\n" +
                        "<option>Voltage</option>\n" +
                        "</select>"
    var giveName = '<input type="text" value="Enter Name" class="test">'
    var functionID = "<input type=\"number\" name=\"funcID\" value=\"-1\", id=\"funcID\" class='test'>";


    function constructURL(){
        var URL = "?";
        var tempArray = [];
        var precision = "precision="+ document.getElementById("Bytes").options[document.getElementById("Bytes").selectedIndex].label;

        if(!FMU){
            var fileName = "fileName=" + document.getElementById("fmm").files[0].name;
        }

        tempArray.push(precision);
        for(var i = 0; i<nSensors; i++){
            tempArray.push("sensor="+JSON.stringify(sensors[i]));
        }

        for(var i = 0; i<tempArray.length; i++){
            if(i === tempArray.length-1){
                URL = URL + tempArray[i];
            }else{
                URL = URL + tempArray[i] + "&";
            }
        }
        return URL;
    }

```

```
function generateFile(){
    var xhr = new XMLHttpRequest();
    var URL = "GenerateConfigurationFile";
    var parameters = constructURL();

    xhr.open("POST",URL,true);
    xhr.onload = () =>{
        var response = JSON.parse(xhr.response);
            if(response.Message === "SUCCESS"){
                alert("A Digital Twin Configuration file was generated successfully");
            }else {
                alert("Something went wrong and the Digital Twin Configuration file was not generated");
            }
    }
    xhr.send(parameters);
}

function showFmmFunctions(){
    var checkbox = document.getElementById("showFmm");

    if(checkbox.checked == true){
        if(!functionsLoaded){
            getFmmFunctions();
        }else{
            document.getElementById("divFedem").style["display"] = "block";
        }
    }
    else{
        document.getElementById("divFedem").style["display"] = "none";
    }
}

function getFmmFunctions() {
    var xhr = new XMLHttpRequest();
    if(FMU){
        URL = "GET-FMM-FUNCTIONS?systemChoice=FMU";
    } else {
        URL = "GET-FMM-FUNCTIONS?systemChoice=FMM";
    }

    xhr.open("GET", URL, true);
    xhr.onload = () =>{
        var fedemFunctions = JSON.parse(xhr.response);
        if(fedemFunctions.Message === "SHIT"){
```

```
                alert("Could not find the file");
                functionsLoaded = false;
            }else{
                document.getElementById("divFedem").style["
                    display"] = "block";
                generateFedemFunctionsTable(fedemFunctions);
                functionsLoaded = true;
            }
        }
        xhr.send();
    }

    function generateFedemFunctionsTable(elements) {

        var found = elements.findIndex(function (element) {
           return element.funcId;
        });

        for(var i = 0; i<elements.length;i++){
            var table = document.getElementById("
                FedemFunctionsTable");
            var row = table.insertRow(1);
            var cell1 = row.insertCell(0);
            var cell2 = row.insertCell(1);
            var cell3 = row.insertCell(2);
            var cell4 = row.insertCell(3);
            cell1.innerHTML = elements[i].name;
            cell2.innerHTML = elements[i].description;
            if(i<found){
                cell3.innerHTML =  elements[i].extFuncId;
                cell4.innerHTML = "NONE";
            }
            else{
                cell3.innerHTML =  "NONE";
                cell4.innerHTML = elements[i].funcId;
            }
        }
    }

    function generateVTFx() {
        var checkbox = document.getElementById("VTFx");

            if(checkbox.checked == true){
                VTFxExport = true;
                if (!FMU){
                    document.getElementById("showVTFx").style["
                        display"] = "table-row";
                }
            }
```

```
            else{
                VTFxExport = false;
                document.getElementById("showVTFx").style["display"] = "none";
            }
     }

    function addSensor() {
        var precisionIndex = document.getElementById("Bytes").selectedIndex;
        var table = document.getElementById("sensorTable");
        var row = table.insertRow(nSensors+1);
        var cell1 = row.insertCell(0);
        var cell2 = row.insertCell(1);
        var cell3 = row.insertCell(2);
        var cell4 = row.insertCell(3);
        var cell5 = row.insertCell(4);
        var cell6 = row.insertCell(5);
        cell1.innerHTML = nSensors;
        cell2.innerHTML = giveName;
        cell3.innerHTML = sensorType;
        cell4.innerHTML = select;
        cell5.innerHTML = functionID;
        cell6.innerHTML = udpIndex;
        nSensors++;

        if(document.getElementById("Bytes").options[precisionIndex].label === "4 Bytes"){
            udpIndex = 24 + (nSensors*4);
        }else{
            udpIndex = 24 + (nSensors*8);
        }
    }

    function sendFile(){
        var table = document.getElementById('sensorTable');
        var temp = '';

        for(var i = 0, row; row = table.rows[i]; i++){
            temp = '';
            for (var j = 0, col; col = row.cells[j]; j++){
                if(i != 0){
                    switch(j){
                        case 1:
                            temp = '{"name":'+ '"' +  col.childNodes[0].value + '",' ;
                            break;
                        case 2:
                            temp = temp + '"sensorType":'+ '"' +
```

```
  col.childNodes[0].value + '",'
;
                        break;
                    case 3:
                        temp = temp + '"sensorPurpose":'+ '"
                        ' +  col.childNodes[0].value + '"
                        ,' ;
                        break;
                    case 4:
                        temp = temp + '"funcID":'+ '"' +  
                        col.childNodes[0].value + '",' ;
                        break;
                    case 5:
                        temp = temp +'"UDPIndex":'+ '"' +  
                        col.innerText + '"}';
                        break;
                }
            }
        }
        if(i != 0){
            sensors.push(JSON.parse(temp));
        }
    }

    var parameters = constructURL();
    var xhr = new XMLHttpRequest();
    xhr.onload = function () {
        if(xhr.status == 200){
            alert("Success! Upload Complete");
        } else {
            alert("Error! Upload failed");
        }
    }

    xhr.open("POST",'GENERATE-FILE',true);
    xhr.setRequestHeader("Content-Type","application/json")
    xhr.send(parameters);
}

function loadFmmFile(){
    var progressBar = document.getElementById("progressBar")
        ;
    var fileInput = document.getElementById("fmm");
    var URL = "LOAD-FILE?fileName="+fileInput.files[0].name;

    if(fileInput.files.length == 0){
        alert("Choose a file");
        return;
    }
```

```
        var xhr = new XMLHttpRequest();

        xhr.upload.onprogress = function(e){
            var percentComplete = (e.loaded / e.total)*100;
            progressBar.value = percentComplete;
        }

        xhr.onload = function () {
            if(xhr.status == 200){
                alert("Sucess! Upload Complete");
                document.getElementById("showFunctionsCheckbox").style["display"] = "table-row";
            } else {
                alert("Error! Upload failed");
            }
        }

        xhr.open("POST",URL,true);
        xhr.setRequestHeader("Content-Type","application/json")
        xhr.send(fileInput.files[0]);
    }

    function loadFTLFiles(){
        var progressBar = document.getElementById("progressBarFtl");
        var fileInput = document.getElementById("FTL");
        var URL = "LOAD-FTL-FILES?";
        var formData = new FormData();

        for(var i = 0; i<fileInput.files.length; i++){
            URL = URL + "fileName=" + fileInput.files[i].name + "&";
            formData.append("uploads",fileInput.files[i]);
        }

        URL = URL.slice(0,-1);
        console.log(formData);

        if(fileInput.files.length == 0){
            alert("Choose a file");
            return;
        }

        var xhr = new XMLHttpRequest();
        xhr.upload.onprogress = function(e){
            var percentComplete = (e.loaded / e.total)*100;
            progressBar.value = percentComplete;
        }
```

```
        xhr.onload = function () {
            if(xhr.status == 200){
                alert("Sucess! Upload Complete");
            } else {
                alert("Error! Upload failed");
            }
        }

        xhr.open("POST",URL,true);
        xhr.setRequestHeader("Content-Type","application/json")
        xhr.send(formData);
    }

    function  loadFMU() {
        var progressBar = document.getElementById("
            progressBarFMU");
        var fileInput = document.getElementById("fmu");
        var URL = "LOAD-FMU";

        if(fileInput.files.length == 0){
            alert("Choose a file");
            return;
        }

        var xhr = new XMLHttpRequest();
        xhr.upload.onprogress = function(e){
            var percentComplete = (e.loaded / e.total)*100;
            progressBar.value = percentComplete;
        }

        xhr.onload = function () {
            if(xhr.status == 200){
                alert("Sucess! Upload Complete");
                document.getElementById("showFunctionsCheckbox")
                    .style["display"] = "table-row";
            } else {
                alert("Error! Upload failed");
            }
        }

        xhr.open("POST",URL,true);
        xhr.setRequestHeader("Content-Type","application/json");
        xhr.send(fileInput.files[0]);
    }

    function selectFileSystem(){
        var URL = 'SELECT-FILE-SYSTEM';
        fileSystemIndex = document.getElementById('FMUorFMM').
```

```
selectedIndex;
        if (document.getElementById('FMUorFMM').options[fileSystemIndex].label === "FMU"){
            document.getElementById("FMUDisplay").style["display"] = "table-row";
            document.getElementById("FedemFileDisplay").style["display"] = "none";
            FMU = true;
            URL = URL + "?fileSystem=FMU" ;

        } else {
            FMU = false;
            document.getElementById("FMUDisplay").style["display"] = "none";
            document.getElementById("FedemFileDisplay").style["display"] = "table-row";
            URL = URL + "?fileSystem=FMM" ;
        }

        var xhr = new XMLHttpRequest();

        xhr.onload = function () {
            if (xhr.status != 200) {
                alert("OOPS Something went wrong");
            }
        }

        xhr.open("POST",URL,true);
        xhr.setRequestHeader("Content-Type","application/json");
        xhr.send();
    }
</script>
</body>

</html>
```

Listing 3: Index.html

E Back-End

E.1 Server.py

```python
from http.server import SimpleHTTPRequestHandler, HTTPServer
import ServerSupport
from urllib.parse import urlparse, parse_qs
import time
import zipfile


HOST_NAME = ""
PORT_NUMBER = 8080

# New container instance
container = ServerSupport.Storage()

# This class contains methods to handle our requests to different URIs in the app
class MyHandler(SimpleHTTPRequestHandler):


    def do_HEAD(self):
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()

    # Check the URI of the request to serve the proper content.
    def do_GET(self):

        if "GET-FMM-FUNCTIONS" in self.path:
            # Searches through a ".FMM" file and returns all functions

            systemChoice = container.getSystemChoice()

            if(systemChoice == 'FMM'):

                if(container.getStatus()):

                    try:
                        content = ServerSupport.getFmmFunctions(container.getContent())
                        self.respond(content)
                    except:
                        self.respond('{"Message":"ERROR" }')
            else:

```

```python
                zFile = zipfile.ZipFile('C:\\Users\\chris\\Downloads\\testrig.fmu.zip')
                content = zFile.open('resources/model/response.bak.fmm')
                response = ServerSupport.getFmmFunctions(content.read().decode('utf-8'))
                self.respond(response)

        else:
            super(MyHandler, self).do_GET()  # serves the static src file by default

    def do_POST(self):

        if("SELECT-FILE-SYSTEM" in self.path):

            # Selects the correct system Choice ( e.g FMU or FMM )

            systemChoice = parse_qs(urlparse(self.path).query)['fileSystem'][0]
            container.setSystemChoice(systemChoice)

            self.send_response(200)
            self.send_header("Content-Type", 'application/json')
            self.end_headers()

        if("GENERATE-FILE" in self.path):

            # Writes ".FMM" file and ".FTL" files to disc
            # Generate a configuration file (YAML)

            try:
                ServerSupport.generateFile(container)
                parameters = parse_qs(urlparse(self.rfile.read(int(self.headers['Content-Length'])).decode('utf-8')).query)

                for key, value in parameters.items():

                    if key != 'sensor':

                        parameters[key] = value[0]
                        parameters[key] = value[0]

                ServerSupport.writeYAML('C:\\Users\\chris\\OneDrive\\Master\\temp\\TEST.yaml', parameters)

```

```
                self.send_response(200)
                self.send_header("Content-Type", 'application/json')
                self.end_headers()

            except:

                self.send_response(1)
                self.send_header("Content-Type", 'application/json')
                self.end_headers()

        if("LOAD-FMU" in self.path):
            # Writes FMU to disc

            content_length = int(self.headers['Content-length'])
            content = self.rfile.read(content_length)
            fmuPath = 'C:\\Users\\chris\\OneDrive\\Master\\temp\\test.zip'
            container.setFMUPath(fmuPath)

            ServerSupport.writeFMU(fmuPath, content)

            self.send_response(200)
            self.send_header("Content-Type", 'application/json')
            self.end_headers()


        if ("LOAD-FILE" in self.path):
            # Loads ".FMM" file to memory
            try:
                content_length = int(self.headers['Content-length'])
                fmmFileName = parse_qs(urlparse(self.path).query)['fileName'][0]
                container.setFileName(fmmFileName)
                container.setContent(self.rfile.read(content_length).decode('ascii'))
                container.setPath('C:\\Users\\chris\\OneDrive\\Master\\UDPplotter\\TwinGen\\' + fmmFileName.replace('.fmm','') + '\\')
                container.setStatus(True)

                self.send_response(200)
                self.send_header("Content-Type", 'application/json')
                self.end_headers()

            except:
```

```
                self.send_response(1)
                self.send_header("Content-Type", 'application/json')
                self.end_headers()

        if("LOAD-FTL-FILES" in self.path):
            # Loads ".FTL" files to memory

            content_length = int(self.headers['Content-length'])
            content = self.rfile.read(content_length)

            try:
                numberOfFiles = ServerSupport.parseFormData(content, container)

                for i in range(0, numberOfFiles, 1):
                    container.addFtlFileName(parse_qs(urlparse(self.path).query)['fileName'][i])

                self.send_response(200)
                self.send_header("Content-Type", 'application/json')
                self.end_headers()

            except:
                self.send_response(1)
                self.send_header("Content-Type", 'application/json')
                self.end_headers()


    def handle_http(self, data):
        self.send_response(200)
        # set the data type for the response header. In this case it will be json.
        # setting these headers is important for the browser to know what  to do with
        # the response. Browsers can be very picky this way.
        self.send_header('Content-type', 'application/json')
        self.end_headers()
        return bytes(data, 'UTF-8')

    # store response for delivery back to client. This is good to do so
    # the user has a way of knowing what the server's response was.
    def respond(self, data):
        response = self.handle_http(data)
        self.wfile.write(response)
```

```


# This is the main method that will fire off the server.
if __name__ == '__main__':
    server_class = HTTPServer
    httpd = server_class((HOST_NAME, PORT_NUMBER), MyHandler)
    print(time.asctime(), 'Server Starts - %s:%s' % (HOST_NAME,
        PORT_NUMBER))
    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        pass
    httpd.server_close()
    print(time.asctime(), 'Server Stops - %s:%s' % (HOST_NAME,
        PORT_NUMBER))
```

Listing 4: Server.py

E.2 ServerSupport.py

```
import json
import re
import os
import yaml
import zipfile
from parseExport import createDigitalTwin, generatePart
from pathlib import PurePath

class Storage:
    # Container class that stores vital information until
        configuration process is completed

    # ---------- Variables ----------------
    def __init__(self):
        self.fmmFileName = ''
        self.fmmFilePath = ''
        self.fmmFileContent = ''
        self.fmmFileLoaded = False
        self.ftlFiles = []
        self.ftlFileNames = []
        self.systemChoice = 'FMU'
        self.fmuPath = ''

    # ---------- GET FUNCTIONS ----------------
    def getFMUPath(self):
        return self.fmuPath

    def getSystemChoice(self):
        return self.systemChoice
```

```python

    def getFileName(self):
        return self.fmmFileName

    def getPath(self):
        return self.fmmFilePath

    def getContent(self):
        return self.fmmFileContent

    def getStatus(self):
        return self.fmmFileLoaded

    def getftlFile(self, index):
        return self.ftlFiles[index]

    def getftlFileName(self, index):
        return self.ftlFileNames[index]

    def getFTLLen(self):
        return len(self.ftlFiles)

    # ---------- SET FUNCTIONS ----------------
    def setFMUPath(self, fmuPath):
        self.fmuPath = fmuPath

    def setFileName(self, fileName):
        self.fmmFileName = fileName

    def setPath(self, path):
        self.fmmFilePath = path

    def setContent(self, content):
        self.fmmFileContent = content

    def setStatus(self, status):
        self.fmmFileLoaded = status

    def setSystemChoice(self, systemChoice):
        self.systemChoice = systemChoice

    # ---------- ARRAY FUNCTIONS ----------------

    def addFtlFile(self, file):
        self.ftlFiles.append(file)

    def addFtlFileName(self, name):
        self.ftlFileNames.append(name)

```

```python

# ------------- FUNCTIONS -------------

def getFmmFunctionsFromFile(fmm_file, path):

    # This method reads through a ".FMM" file stored on disc and returns the functions for this Fedem model
    # THE CODE FOR THE METHOD IS BASED ON buildFMU.py

    input_functions = []
    output_functions = []

    # Read ".FMM" file to retrieve the model functions

    with open(''.join(path)+"\\"+''.join(fmm_file)) as file:
        line = file.readline()
        while line:
            if line.startswith("ENGINE"):
                line = file.readline()
                if line.startswith("{"):
                    descr = ""
                    id = -1
                    extId = -1
                    while line:
                        if line.startswith("}"):
                            break
                        if line.startswith("DESCR"):
                            descr = (line.split("=")[-1].strip()).split("\"")[1]
                        if line.startswith("ID"):
                            id = int(re.search(r'\d+', line.split("=")[-1].strip()).group())
                        if line.find("FcfEXTERNAL_FUNCTION") != -1:
                            extId = line.split(" ")[2].strip()
                        line = file.readline()
                    if extId != -1:
                        function = {"name": descr, "description": descr, "extFuncId": extId}
                        input_functions.append(function)
                    else:
                        function = {"name": descr, "description": descr, "funcId": id}
                        output_functions.append(function)
            line = file.readline()

    input_functions.extend(output_functions.copy())

    # Store all input and output functions in JSON format
```

```
    x = json.dumps(input_functions)

    return x

def getFmmFunctions(content):

    # This method reads through a ".FMM" file loaded to memory
        and returns the functions for this Fedem model
    # THE CODE FOR THE METHOD IS BASED ON buildFMU.py

    input_functions = []
    output_functions = []

    # Read ".FMM" file to retrieve the model functions

    lines = content.splitlines()
    i = 0
    while(i<len(lines)):
        line = lines[i]
        if line.startswith("ENGINE"):
            i = i+1
            line = lines[i]
            if line.startswith("{"):
                descr = ""
                id = -1
                extId = -1
                while line:
                    if line.startswith("}"):
                        break
                    if line.startswith("DESCR"):
                        descr = (line.split("=")[-1].strip()).
                            split("\"")[1]
                    if line.startswith("ID"):
                        id = int(re.search(r'\d+', line.split("=
                            ")[-1].strip()).group())
                    if line.find("FcfEXTERNAL_FUNCTION") != -1:
                        extId = line.split(" ")[2].strip()
                    i = i + 1
                    line = lines[i]
                if extId != -1:
                    function = {"name": descr, "description":
                        descr, "extFuncId": extId}
                    input_functions.append(function)
                else:
                    function = {"name": descr, "description":
                        descr, "funcId": id}
                    output_functions.append(function)
        i = i + 1

```

```python


    input_functions.extend(output_functions.copy())

    # Store all input and output functions in JSON format
    x = json.dumps(input_functions)

    return x

def generateFile(container):

    if(container.getSystemChoice() == 'FMM'):

        # Writes ".FMM" file and ".FTL" files to disc

        if (container.getStatus()):

            # CHECK DIRECTORY
            os.makedirs(os.path.dirname(container.getPath()), exist_ok=True)

            # WRITE ".FMM" to disc
            with open(container.getPath() + container.getFileName(), 'w', newline='\n')as file:

                file.write(container.getContent())

            # Generate a JSON Master file for the 3D visualisation
            listOfFiles = createDigitalTwin(container.getFileName(), container.getPath())

            # CHECK FOR ".FTL" files in memory

            if (container.getFTLLen() > 0):

                for i in range(0, container.getFTLLen(), 1):

                    # CHECK DIRECTORY
                    os.makedirs(os.path.dirname(container.getPath() + container.getftlFileName(i)), exist_ok=True)

                    # WRITE ".FTL" files to disc
                    with open(container.getPath() + container.getftlFileName(i), 'w', newline='\n') as file:

                        file.write(container.getftlFile(i).
```

```
                                decode('utf8'))

                # Generate JSON files from the ".FTL" files. Used for 3D visualisation
                generatePart(listOfFiles, container.getPath(), container.getFileName())
    else:

        fmuPath = 'C:\\Users\\chris\\Downloads\\testrig.fmu.zip' # container.getFMUPath()

        # Get file name
        fileString = fmuPath.split('\\')[-1]
        fileString = fileString.split('.')[0]

        # Open FMU(ZIP) file
        zFile = zipfile.ZipFile(fmuPath)
        fmmContent = zFile.open('resources/model/response.bak.fmm').read().decode('utf-8')

        path = 'C:\\Users\\chris\\OneDrive\\Master\\UDPplotter\\TwinGen\\' + fileString + '\\'

        # CHECK DIRECTORY
        os.makedirs(os.path.dirname(path), exist_ok=True)

        # Write ".FMM" file to disc
        with open(path +  fileString + '.fmm', 'w', newline='\n') as file:
            file.write(fmmContent)

        # Locate all ".FTL" files (All parts)
        for name in zFile.namelist():
            pathCheck = PurePath(name)

            # Check for ".FTL" files in the directory "resources/link_DB"
            if pathCheck.parent.parts == ('resources', 'link_DB') and pathCheck.suffix == '.ftl':
                ftlContent = zFile.open(name)
                ftlFileName = name.split('/')[-1]


                # Write ".FTL" files to disc
                with open(path + ftlFileName, 'w', newline='\n') as file:
                    file.write(ftlContent.read().decode('utf-8'))

```

```
        # Generate a JSON Master file for the 3D visualisation
        # listOfFiles = createDigitalTwin(path,fileString)
        # Generate JSON files from the ".FTL" files. Used for 3D visualisation
        # generatePart(listOfFiles, path, fileString)


def parseFormData(formData,container):

    # Reads multiple ".FTL" files sent with a POST Request in the FormData-format

    # Find start of ".FTL" files
    startOfFile = re.finditer(b'FTLVERSION',formData)
    startIndices = [m.start(0) for m in startOfFile]

    # Find end of ".FTL" files
    endOfFile = re.finditer(b'#_End_of_file',formData)
    endIndices = [m.start(0) for m in endOfFile]

    numberOfFiles = len(startIndices)

    for i in range(0,numberOfFiles,1):

        # Stores the ".FTL" files in memory (AS BYTES)
        container.addFtlFile(formData[startIndices[i]:endIndices[i]+len(b'#_End_of_file')])

    return numberOfFiles

def writeFMU(path, content):
    # Writes a ".ZIP" file to disc

    with open(path,'wb') as file:
        file.write(content)

def writeYAML (path, content):
    # Writes a ".YAML" file to disc

    with open(path,'w') as file:
        yaml.dump(content, file)
```

Listing 5: ServerSupport.py

E.3 parseExport.py

```
from Exporter import VTFXExporter
import json

```

```
def createDigitalTwin(fmm_file,fmm_path):
    # Reads through a ".FMM" file

    # Generate a JSON Master file which contains:
    #   - Information on which parts the Fedem model contains
    #   - BASE ID for each part
    #   - Local to global coordinate system transformation matrix for each part

    # This method returns a list of the parts used in the Fedem model

    listOfFiles = []
    coordinates = []
    baseID = []

    fileString = fmm_file
    path = fmm_path + fmm_file

    # Open ".FMM" file
    with open(path) as file:
        content = file.readlines()

    isPart = False

    # Read ".FMM" file
    for i in range(0, len(content), 1):
        line = content[i]
        if(line == 'PART\n'):
            isPart = True
        if(isPart):
            if ('BASE_FTL_FILE' in line):
                line = line.split('"')
                listOfFiles.append(line[1])
            if( 'BASE_ID' in line):
                temp = line.split()
                temp = temp[2]
                temp = temp[:-1]
                baseID.append(temp)
            if(line == 'COORDINATE_SYSTEM = \n'):
                test = content[i+1]
                test = test.split()
                coordinates.append(test)
                test = content[i+2]
                test = test.split()
                coordinates.append(test)
                test = content[i+3]
                test = test.split()
```

```
                test[3] = test[3][:-1]
                coordinates.append(test)
        if(isPart and line == '}\n'):
            isPart = False



    jsonPath = 'C:\\Users\\chris\\OneDrive\\Master\\UDPplotter\\js\\' + fileString.replace('.fmm','') + '\\'


    # Write Master JSON file to disc
    with open(jsonPath + fileString.replace('fmm', 'json'), 'w') as file:
        file.write(json.dumps({'ListOfFile': listOfFiles, 'Coordinates': coordinates, 'baseID': baseID}))

    # Return list of files
    return listOfFiles



def generatePart(listOfFiles,ftlPath,fileName):
    exporter = VTFXExporter()
    exporter.initialize()
    j = 0

    # For each part
    for part in listOfFiles:

        path = ftlPath + part
        print(path)
        temp = part.split('.')
        print(temp[0])
        parameter1 = bytes(path, 'utf-8')
        parameter2 = bytes(temp[0], 'utf-8')
        j = j +1

        # Arg(Path, Name, baseId)
        exporter.add_fe_part(parameter1, parameter2, j)

        # Get number of elements
        numElems = exporter.get_number_of_elements(j)

        # Total number of elements-to-no. E.g 2 quad elements gives 8
        numElemNodes = exporter.get_number_of_element_nodes(j)

        #Lenght of vertex-array = noder*3
```

```
        numNodes = exporter.get_number_of_nodes(j)

        # Get element type, vertex and element-to-node connection
        # In Ceetron: cee.usg.Mesh(nodeArr,elementTypeArr,elementNodeIndexArr)
        elementTypeArr = exporter.get_element_types(j, numElems)
        nodeArr = exporter.get_nodes(j,numNodes)
        elementNodeIndexArr = exporter.get_elements(j, numElemNodes)

        # Write geometry data to JSON-file
        jsonFileName = temp[0]+'.json'
        with open('C:\\Users\\chris\\OneDrive\\Master\\UDPplotter\\js\\'+ fileName.replace('.fmm', '')+'\\' + jsonFileName, 'w') as file:
            file.write(json.dumps({
                'nodeArr': nodeArr,
                'elementTypeArr': elementTypeArr,
                'elementNodeIndexArr': elementNodeIndexArr
            }))
```

Listing 6: parseExport.py

F Modified Digital Twin Cloud Software

F.1 index.js

```
// Import and initialise libraries
const express = require('express');
const app = express();
const http = require('http').Server(app);
const io = require('socket.io')(http);
const dgram = require('dgram');
const struct = require('python-struct');


// Serve index.html when users visits the page
app.get('/', function(req, res) {
    res.sendFile(__dirname + '/index.html');
});

app.use('/ceetron', express.static('ceetron'));
app.use('/js', express.static('js'));

// Start the http server for serving index.html
http.listen(1337, function(){
    console.log('listening on *:1337');
});

// Create socket listening for new data from solver
fedemSocket = dgram.createSocket('udp4');

// Print to console when ready to listen for new data
fedemSocket.on('listening', function(){
    const address = fedemSocket.address();
    console.log('listening on ' + address.address + ':' + address.port);
});

// Function for parsing new data from solver
fedemSocket.on('message', function(message, remote){


    // Extract time and strain data
    const Reference = struct.unpack('<d', message.slice(0))[0];
    const timestamp = struct.unpack('<d', message.slice(8))[0];
    // Send the vertical displacement to the client
    io.emit('new_data', [timestamp, Reference]);



```

```
    // Skip 32 first bytes
    // See section 3.3.2 in "Cloud Software For Digital Twin Modeling And Monitoring" (Johansen et al,(2018)) for
    // more details on data pack structure
    for (var i = 32; i < message.length-111; i += 112) {
        // CHECK IF PART
        if(struct.unpack('<d', message.slice(i+8)) == 2);
        {
            // Read the baseId as the second of the 14 doubles
            const baseId = struct.unpack('<d', message.slice(i+8));
            const t = struct.unpack('<12d', message.slice(i+16));
            const m = [
                t[0], t[1],  t[2],  0,
                t[3], t[4],  t[5],  0,
                t[6], t[7],  t[8],  0,
                t[9], t[10], t[11], 1
            ];

            io.emit('transformation', [m,baseId[0]]);
        }



    }

});

// Start listening for new data from solver
fedemSocket.bind(8001, '0.0.0.0');
```

Listing 7: index.js

F.2 index.html

```
<!doctype html>
<html lang="en">
<head>
    <title>Digital Twin</title>
    <link rel="style.css">
    <script src="/socket.io/socket.io.js"></script>
    <script src="https://cdn.plot.ly/plotly-latest.js" charset="utf-8"></script>
</head>
<body style="margin: 0; height:100vh; display: grid; grid: minmax(400px, 50%) minmax(200px, 50%) / minmax(400px, 100%)">

<div style="display: flex">
```

```
    <div id="chartContainer" style="width: 100%"></div>
    <div style="display: flex; flex-direction: column">
        <button onclick="save()">Save</button>
        <div style="flex-grow: 1"></div>
        <span>Model Style</span>
        <button onclick="myApp.setDrawStyle('surface')">Surface</button>
        <button onclick="myApp.setDrawStyle('surface_mesh')">Surface Mesh</button>
        <button onclick="myApp.setDrawStyle('outline_mesh')">Outline Mesh</button>
        <button onclick="myApp.setDrawStyle('lines')">Lines</button>
        <button onclick="myApp.setDrawStyle('points')">Points</button>
        <button onclick="myApp.setDrawStyle('outline')">Outline</button>
    </div>
</div>

<div style="line-height: 0">
    <canvas id="CeetronCanvas"></canvas>
</div>


<script src="ceetron/require.js"></script>
<script>



    // Initialise connection to server
    var socket = io();

    var Crane;

     // Initialise USG module
    var myApp = null;
    require(["js/usg","js/DigitalTwin"],function(appModule) {
        myApp = appModule.startApp("CeetronCanvas");
        Crane =  new DigitalTwin("CraneShort");
    });


    // Store reference to container for plot
    var graphContainer = document.getElementById('chartContainer');

    // Container for displacement plot data
    var displacements = {x:[[]], y:[[]]};
```

```
    // Initialise plot
    Plotly.newPlot(
        graphContainer,
        [{y:[]}],
        {
            title: 'Reference',
            yaxis: {
                title: 'Strain'
            },
            xaxis: {
                title: 'Time[s]'
            }
        },
        {responsive: true}
    );

    // Counter for how many data points has been received
    var dataRecievedCount = 0;

    // Update plot with new data for every 100 new data points
    socket.on('new data', function(msg){
        // displacements.x[0].push(new Date(msg[0]));
        displacements.x[0].push(msg[0]);
        displacements.y[0].push(msg[1]);
        // If 100 data points recieved since last update
        if (dataRecievedCount++ % 100 === 0) {
            // Remove points received more than 1000 points ago
            displacements.x[0] = displacements.x[0].slice
                (-100000);
            displacements.y[0] = displacements.y[0].slice
                (-100000);
            // Update plot
            Plotly.restyle(graphContainer, displacements);
        }
    });

    // Update transformation of model for every 100 new data
        points
    socket.on('transformation', function(msg){

        if (myApp !== null) {

            console.log(msg[1]);
            var partIndex = Crane.getPartIndex(msg[1]);
            if(partIndex >=0){
                myApp.updateDisplacement(msg[0],partIndex);
            }

```

```
        }
    });



    // Create download dialog for currently plotted data
    function save() {
        var saveData = "Timestamp, displacement(mm)\r\n";
        for (var i = 0; i < displacements.x[0].length; i++) {
            saveData += displacements.x[0][i].valueOf() + ", " +
                displacements.y[0][i] + "\r\n"
        }
        download(saveData, "twin_" + new Date().toISOString() +
            ".csv", "text/csv");
    }

    // Downloading data to a file
    function download(data, filename, type) {
        var file = new Blob([data], {type: type});
        if (window.navigator.msSaveOrOpenBlob) // IE10+
            window.navigator.msSaveOrOpenBlob(file, filename);
        else { // Others
            var a = document.createElement("a"),
                url = URL.createObjectURL(file);
            a.href = url;
            a.download = filename;
            document.body.appendChild(a);
            a.click();
            setTimeout(function() {
                document.body.removeChild(a);
                window.URL.revokeObjectURL(url);
            }, 0);
        }
    }

</script>
</body>
</html>
```

Listing 8: index.html

F.3 usg.ts

```
import * as cee from "../ceetron/CeeCloudClientComponent";

// Initialiser for Ceetron module of application
export function startApp(canvasElementId: string): App {
    let canvas = document.getElementById(canvasElementId);
    if (!(canvas instanceof HTMLCanvasElement)) {
```

```
        throw("Could not get canvas element");
    }
    return new App(canvas);
}

// Class containing Ceetron Cloud Client Component state
export class App {

    // Ceetron Cloud Client Component state
    private cloudSession: cee.CloudSession;
    private view: cee.View;
    private model: cee.usg.UnstructGridModel;
    private state: cee.usg.State;

    // Canvas containing visualisation
    private canvas: HTMLCanvasElement;

    constructor(canvas: HTMLCanvasElement) {
        this.canvas = canvas;

        // Initialise Ceetron Cloud Client Component
        this.cloudSession = new cee.CloudSession();
        let viewer = this.cloudSession.addViewer(canvas);
        if (!viewer) {
            throw("No WebGL support");
        }
        this.view = viewer.addView();
        this.model = new cee.usg.UnstructGridModel();
        this.view.addModel(this.model);
        this.state = this.model.addState();
        this.state.geometry = new cee.usg.Geometry();

        // Hide infoBox initially
        this.view.overlay.infoBoxVisible = false;

        // Listen for resize events
        window.addEventListener('resize', () => this._handleWindowResizeEvent());

        // Manually run resize function once
        this._handleWindowResizeEvent();

        // Update view every browser frame
        window.requestAnimationFrame((time: number) => this._myAnimationFrameCallback(time));
    }

    // Adjust view dimension (called when window is resized)
    private _handleWindowResizeEvent() {
```

```
        let canvasWidth = window.innerWidth;
        let canvasHeight = this.canvas.parentElement.offsetHeight;
        this.cloudSession.getViewerAt(0).resizeViewer(canvasWidth, canvasHeight);
    }

    // Update view (called every browser frame)
    private _myAnimationFrameCallback(highResTimestamp:number) {
        this.cloudSession.handleAnimationFrameCallback(highResTimestamp);
        window.requestAnimationFrame((time: number) => this._myAnimationFrameCallback(time));
    }

    // Create the torsion rod geometry
    addRodGeometry(data) {
        let geometry = this.state.geometry.addPart();
        geometry.mesh = new cee.usg.Mesh(data.nodeArr, data.elementTypeArr, data.elementNodeIndexArr);
        geometry.settings.color = new cee.Color3(.8, .8, .8);

        // Transform to global coordinate system
        const c = cee.Mat4.fromElements(
            1, 0, 0, -0.02407066,
            0, 1, 0, -0.02722985,
            0, 0, 1, 0.27199998,
            0, 0, 0, 1
        );
        this.state.setPartTransformationAt(1, c);
    }

    addPartGeometry(data,a1,a2,a3,index){
        let geometry = this.state.geometry.addPart();
        geometry.mesh = new cee.usg.Mesh(data.nodeArr, data.elementTypeArr, data.elementNodeIndexArr);
        geometry.settings.color = new cee.Color3(Math.random(),Math.random(),Math.random());


        const c = cee.Mat4.fromElements(
            a1[0], a1[1], a1[2], a1[3],
            a2[0], a2[1], a2[2], a2[3],
            a3[0], a3[1], a3[2], a3[3],
            0, 0, 0, 1
```

```
        );

        this.state.setPartTransformationAt(index, c);
        //this.showStatistics(this.state.geometry);
    }

    private showStatistics(geometry) {
        // Generate statistics on geometry

        let nodeCount = 0;
        let elementCount = 0;
        for (let part of geometry.getPartArray()) {
            nodeCount += part.mesh.nodeCount;
            elementCount += part.mesh.elementCount;
        }

        // Log generated statistics
        console.log("Initial state loaded, nodeCount=" +
            nodeCount + ", elementCount=" + elementCount);

        // Draw generated statistics in bottom right corner
        this.view.overlay.infoBoxVisible = true;
        this.view.overlay.setInfoBoxContent(`Elements: ${
            elementCount} elements \nNodes: ${nodeCount} nodes`);
    }

    updateDisplacement(transformationMatrix: number[],baseID) {
        // Create Ceetron matrix from transformation data
        const m = cee.Mat4.fromArray(transformationMatrix);
        const localToGlobalTransformation = cee.Mat4.
            fromElements(
            1, 0, 0, 0,
            0, 1, 0, 0,
            0, 0, 1, 0,
            0, 0, 0, 1
        );
        const transformation = cee.Mat4.multiply(m,
            localToGlobalTransformation);

        // Apply transformation to armGeometry
        this.state.setPartTransformationAt(baseID,
            transformation);
    }

    // Change drawing style for geometries
    setDrawStyle(ds: string) {
        const geometry = this.model.getStateAt(0).geometry;
        for (let part of geometry.getPartArray()) {
            if      (ds === "surface")              part.
```

```
                settings.drawStyle = cee.usg.DrawStyle.SURFACE;
            else if (ds === "surface_mesh")          part.settings.drawStyle = cee.usg.DrawStyle.SURFACE_MESH;
            else if (ds === "outline_mesh")          part.settings.drawStyle = cee.usg.DrawStyle.SURFACE_OUTLINE_MESH;
            else if (ds === "lines")                 part.settings.drawStyle = cee.usg.DrawStyle.LINES;
            else if (ds === "points")                part.settings.drawStyle = cee.usg.DrawStyle.POINTS;
            else if (ds === "outline")               part.settings.drawStyle = cee.usg.DrawStyle.OUTLINE;
        }
    }
}
```

Listing 9: usg.ts

G Strain Gauge Data Sheets

G.1 WFLA-6-11-1L

WATERPROOF STRAIN GAUGES series WF

M metal

Operating temperature range: 0°C ~ +80°C

Temperature compensation range: +10°C ~ +80°C

Suffix code for temperature compensation materials
-11: Mild steel -17: Stainless steel -23: Aluminium

For ordering, the above suffix code should be added to the basic gauge type.

Applicable adhesives

CN	0 ~ +80ºC
P-2	0 ~ +80ºC
EB-2	0 ~ +80ºC

WATERPROOF STRAIN GAUGES

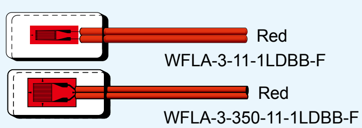
Gauge pattern

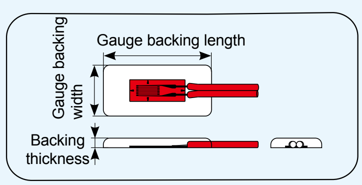
These gauges eliminate the need for moisture-proofing coating, which is sometimes troublesome in a field test. They have an integral vinyl leadwire, and whole area of the strain gauges and the leadwire junction are coated with epoxy resin. The coating is transparent and flexible, so the positioning and bonding works are very easy. By merely bonding the gauges with CN or P-2 adhesive, outdoor or underwater measurement for a short-term becomes possible. These gauges are also effective in omitting primary coating in case of applying a multi-layer coating.

Single element : WFLA

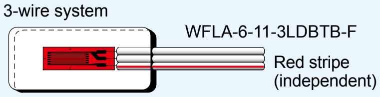
0.08mm² integral vinyl leadwire
Total leadwire resistance per meter : 0.44Ω

2-wire system



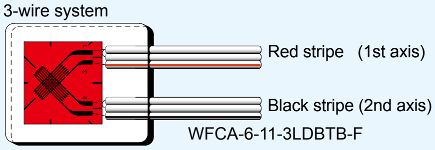


3-wire system



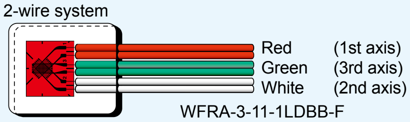
0°/90° 2-element stacked Rosette WFCA

3-wire system



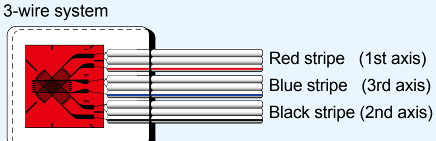
0°/45°/90° 3-element stacked Rosette WFRA

2-wire system



94

3-wire system



CE

Example of type nubmer designation

WFLA -3 -350 -11 -3LDBB-F

- Length in meter and the specified integral leadwire with CE compliant
- Self-temperature-compensation number
- Gauge resistance (blank for 120Ω)
- Gauge length
- Basic gauge type

2-wire system

Single element

Basic type		Gauge size L	W	Backing L	W		Resistance Ω
WFLA-3	-_LDBB-F _ filled with length	3	1.7	17	8	1.5	120
WFLA-3-350		3	3.2	17	8	1.5	350
WFLA-6		6	2.2	25	11	1.5	120

0°/90° 2-element Rosette Stacked

Basic type		Gauge size L	W	Backing L	W		Resistance Ω
WFCA-3	-_LDBB-F _ filled with length	3	1.7	19	16	1.5	120
WFCA-6		6	2.3	25	21	1.5	120

0°/45°/90° 3-element Rosette Stacked

Basic type		Gauge size L	W	Backing L	W		Resistance Ω
WFRA-3	-LDBB-F _ filled with length	3	1.7	19	16	1.5	120
WFRA-6		6	2.3	25	21	1.5	120

Minimum order is 10 gauges or more.

3-wire system

Quarter bridge 3-wire system is usable to avoid an unexpected effect of resistance change with temperature.

Slingle element

Basic type		Gauge size L	W	Backing L	W		Resistance Ω
WFLA-3	-_LDBTB-F _ filled with length	3	1.7	17.0	8	1.5	120
WFLA-6		6	2.2	25.0	11	1.5	120

0°/90° 2-element Stacked

Basic type		Gauge size L	W	Backing L	W		Resistance Ω
WFCA-3	-_LDBTB-F _ filled with length	3	1.7	19	16	1.5	120
WFCA-6		6	2.3	25	21	1.5	120

0°/45°/90° 3-element Rosette Stacked

Basic type		Gauge size L	W	Backing L	W		Resistance Ω
WFRA-3	-_LDBTB-F _ filled with length	3	1.7	19	16	1.5	120
WFRA-6		6	2.3	25	21	1.5	120

Minimum order is 10 gauges or more.

G.2 FLA-5-11-1L

FOIL STRAIN GAUGE series "F"

M metal

Compatible adhesive & Operational temperature
CN : −20〜+80℃
P-2 : −20〜+80℃ EB-2 : −20〜+80℃

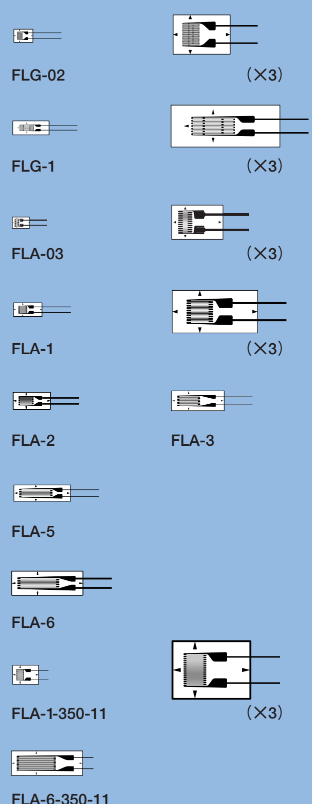
Operational temperature −20〜+80℃
Temperature compensation range +10〜+80℃

GENERAL USE

Gauge pattern

This gauge employs alloy foils which are 0.003 to 0.007 mm thick. Its gauge backing is made of epoxy resin with thickness of 0.03 mm which exhibits excellent electrical insulation performance. The backing is color coded for distinction of object specimen material for self temperature compensation.

■Single-element (G.F. 2.1 approx.)



L : length W : width (Unit : mm)

	Type	Gauge size L	Gauge size W	Backing L	Backing W	Resistance in Ω
Single-element	FLG-02-11 -17 -23	0.2	1.4	3.5	2.5	120
	FLG-1-11 -17 -23	1	1.1	6.5	2.5	120
	FLA-03-11 -17 -23	0.3	1.4	3.0	2.0	120
	FLA-05-11 -17 -23	0.5	1.2	5.0	2.2	120
	FLA-1-11 -17 -23	1	1.3	5.0	2.5	120
	FLA-2-11 -17 -23	2	1.5	6.5	3.0	120
	FLA-3-11 -17 -23	3	1.7	8.8	3.5	120
	FLA-3-60-11 -17 -23	3	1.2	8.0	3.0	60
	FLA-5-11 -17 -23	5	1.5	10.0	3.0	120
	FLA-6-11 -17 -23	6	2.2	12.5	4.3	120
350Ω	FLA-1-350-11 FLA-1-350-17 FLA-1-350-23	1	2.0	5.0	4.0	350
	FLA-2-350-11 FLA-2-350-17 FLA-2-350-23	2	1.9	6.1	3.5	350
	FLA-3-350-11 FLA-3-350-17 FLA-3-350-23	3	3.2	8.5	5.0	350
	FLA-6-350-11 FLA-6-350-17 FLA-6-350-23	6	2.6	12.5	4.5	350

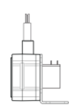
Each package contains 10 gauges.

H Wire Sensor Data Sheet

TE connectivity SENSOR SOLUTIONS

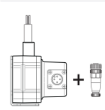
This compact stringpot with "voltage divider" output, provides ease-of-use and flexibility for measurement ranges up to 50 inches. Made of rugged polycarbonate, the SP1 fits in small spaces, doesn't need perfect alignment and ships with a stainless steel mounting bracket to let the user easily orient this sensor in just about any direction imaginable.
The SP1 is available with a connector, mating plug and sensor cover to protect against IP67 (wet) environments and a lower cost, "open" sensor version priced for both the budget conscious single piece user and the OEM alike.

Ordering Information



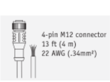
includes sensor & mounting bracket.

Part Number	full stroke range	accuracy	max. acceleration	measuring cable tension (± 25%)	cycle life
SP1-4	4.75 in (120 mm)	1.00%	15 g	7oz. (1,9 N)	2.5M
SP1-12	12.5 in (317 mm)	.25%	15 g	7oz. (1,9 N)	500K
SP1-25	25 in (635 mm)	.25%	15 g	7oz. (1,9 N)	500K
SP1-50	50 in (1270 mm)	.25%	15 g	7oz. (1,9 N)	250K



includes sensor, mounting bracket & mating connector.

Part Number	full stroke range	accuracy	max. acceleration	measuring cable tension (± 25%)	cycle life
SP1-4-3	4.75 in (120 mm)	1.00%	11 g	11.5 oz. (3,2 N)	2.5M
SP1-12-3	12.5 in (317 mm)	.25%	11 g	13.9 oz. (3,7 N)	500K
SP1-25-3	25 in (635 mm)	.25%	11 g	11.5 oz. (3,2 N)	500K
SP1-50-3	50 in (1270 mm)	.25%	11 g	11.5 oz. (3,2 N)	250K



Optional Cordset

Part No.
9036810-0040 for all SP1-xx-3 versions, an optional 13-ft. cordset with 4-pin M12 connector

SP1

Compact String Pot • Voltage Divider

Linear Position to 50 inches (1270 mm)

Rugged Polycarbonate Enclosure • IP67 Optional

Mounting Bracket & Optional Sensor Cover w/Connector

IN STOCK for Quick Delivery!

Complete Specifications

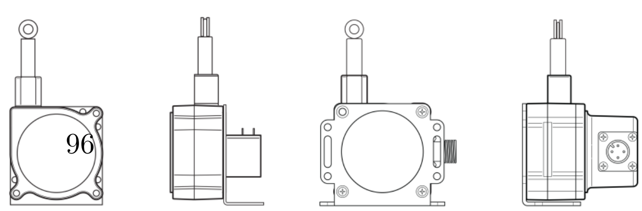
Available Stroke Ranges	0-4.75, 0-12.5, 0-25, 0-50 inches
Output Signal	voltage divider (potentiometer)
Accuracy	±0.25 to ±1.00% (see ordering info)
Repeatability	± 0.05% full stroke
Resolution	essentially infinite
Measuring Cable	0.019-in. dia. nylon-coated stainless steel
Measuring Cable Tension	see ordering information
Maximum Cable Acceleration	see ordering information
Enclosure Material	Polycarbonate
Sensor	plastic-hybrid precision potentiometer
Weight, max. (includes bracket)	.4 lbs (.19 kg)

Electrical

Input Resistance	10K ohms, ±10%
Power Rating, Watts	2.0 at 70°F derated to 0 at 250°
Recommended Maximum Input Voltage	30 V (AC/DC)
Output Signal Change Over Full Stroke Range	94% ±4% of input voltage
Electrical Connection, SP1-xx	solder terminals
Electrical Connection, SP1-xx-3	4-pin, M12 connector

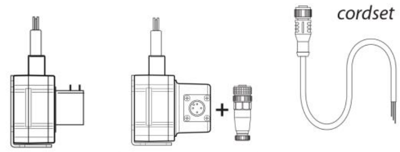
Environmental

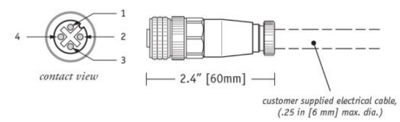
Enclosure	IP 50 (SP1-xx), IP67 (SP1-xx-3)
Operating Temperature, SP1-xx	0° to 160°F (-18° to 70°C)
Operating Temperature, SP1-xx-3	-40° to 160°F (-40° to 70°C)
Vibration	up to 10 g to 2000 Hz maximum



offset

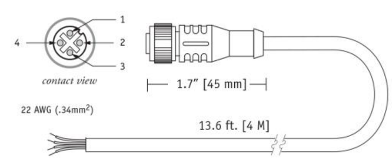
Electrical Connection



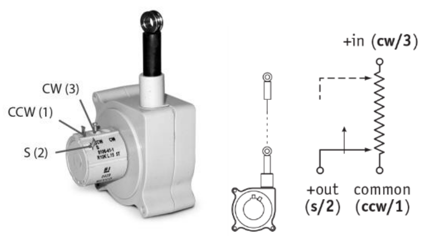


Signals	*solder terminals*	*pin*	*pin - colorcode*
+in	#3 (cw)	1	1 - brown
common	#1 (ccw)	2	2 - white
+out	#2 (s)	3	3 - blue
n/c		4	4 - black

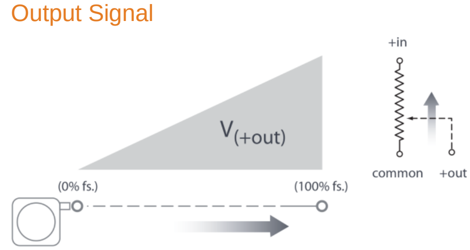
cordset, part no. 9036810-0040 (optional)



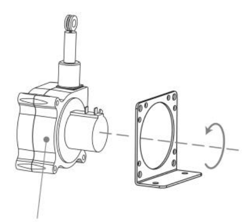
Terminal/Pin Location (SP1-xx)



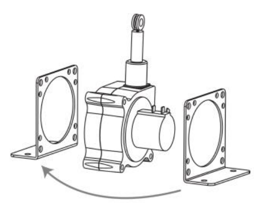
Output Signal



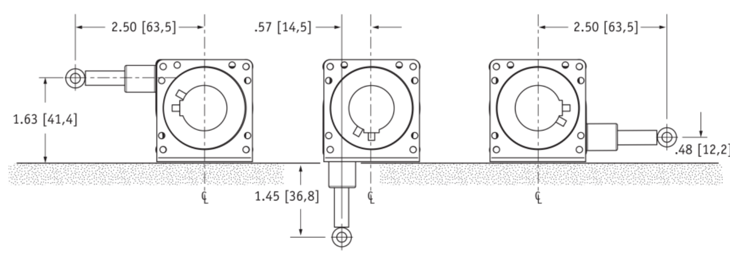
Mounting Options for SP1-xx:



To change measuring cable direction simply remove the 3 bracket attaching screws and rotate sensor body to desired direction.

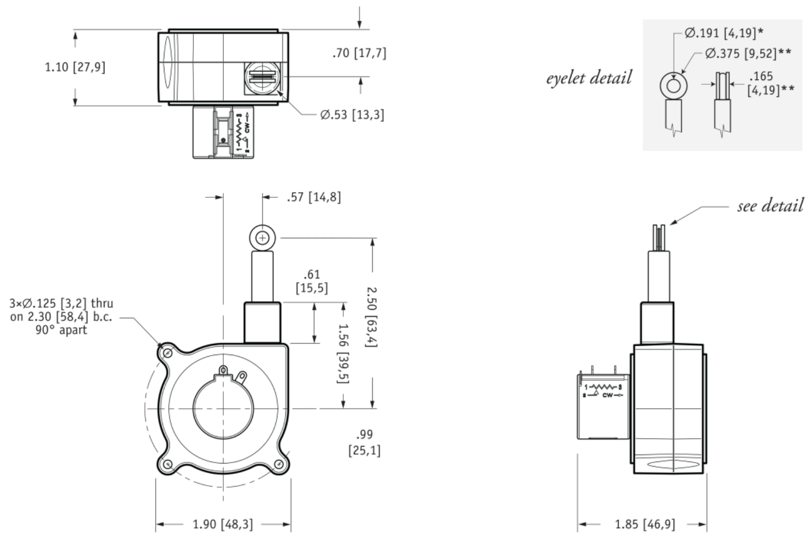


For added flexibilty, mounting bracket can easily be switched to the opposite side.



offset

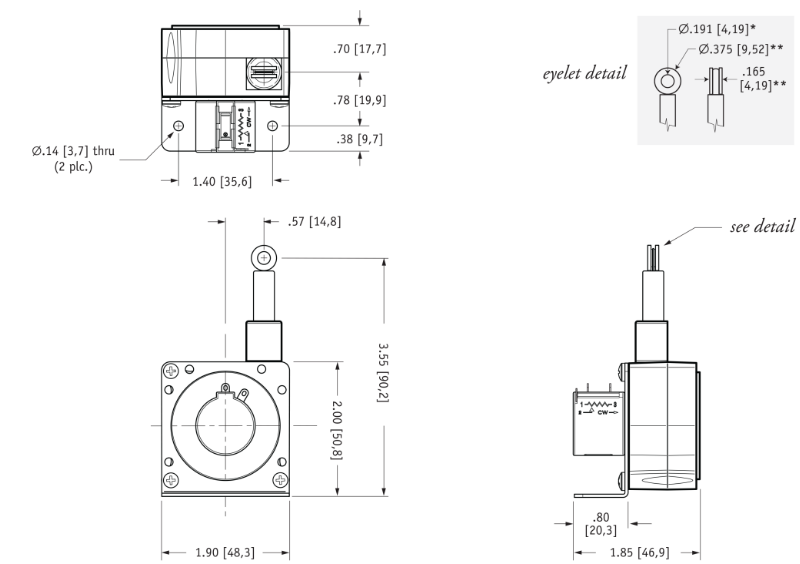
SP1-xx w/o Mounting Bracket



DIMENSIONS ARE IN INCHES [MM]
tolerances are 0.04 IN. [1,0 MM] unless otherwise noted.

* tolerance = +.005 –.001 [+0,1 –0,0]
** tolerance = +.005 –.005 [+0,1 –0,1]

SP1-xx w/ Mounting Bracket

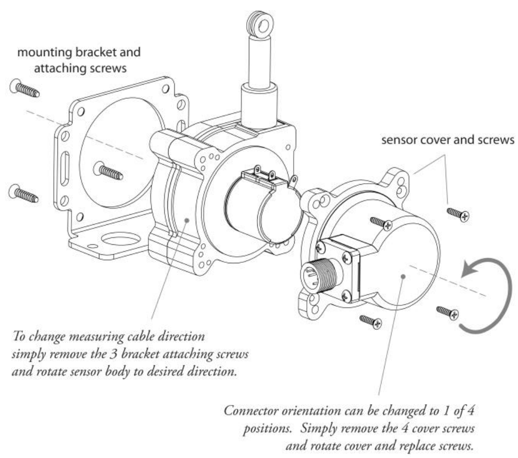


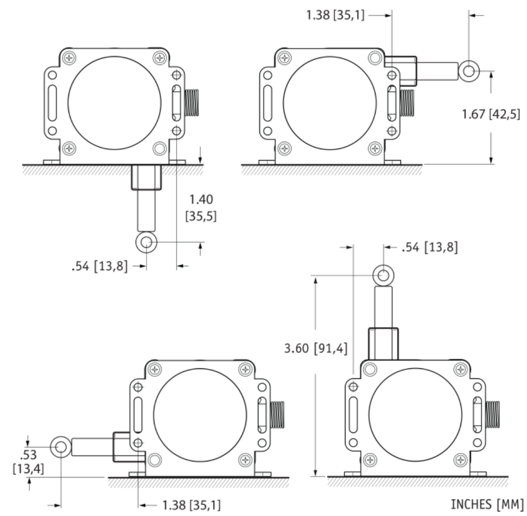
DIMENSIONS ARE IN INCHES [MM]
tolerances are 0.04 IN. [1,0 MM] unless otherwise noted.

* tolerance = +.005 –.001 [+0,1 –0,0]
** tolerance = +.005 –.005 [+0,1 –0,1]

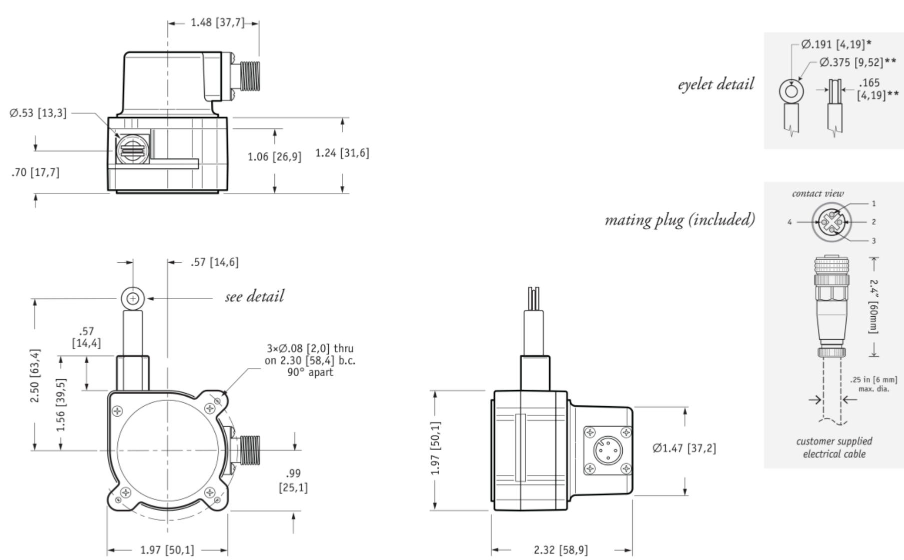
offset

Mounting Options for SP1-xx-3





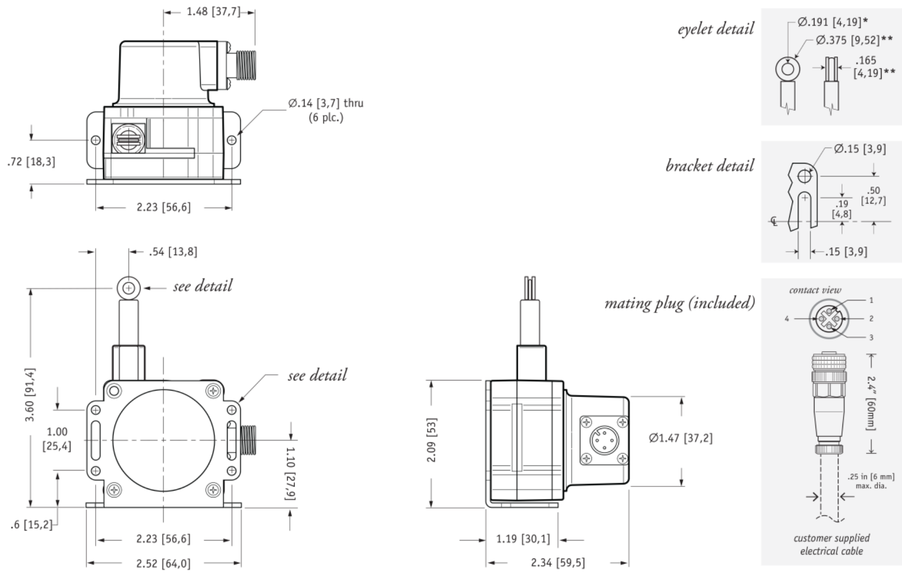
SP1-xx-3 w/o Mounting Bracket



DIMENSIONS ARE IN INCHES [MM]
tolerances are 0.04 IN. [1,0 MM] unless otherwise noted.

* tolerance = +.005 –.001 [+0,1 –0,0]
** tolerance = +.005 –.005 [+0,1 –0,1]

offset SP1-xx-3 w/ Mounting Bracket



DIMENSIONS ARE IN INCHES [MM]
tolerances are 0.04 IN. [1,0 MM] unless otherwise noted.

* tolerance = +.005 –.001 [+0,1 –0,0]
** tolerance = +.005 –.005 [+0,1 –0,1]

NORTH AMERICA

Measurement Specialties, Inc.,
a TE Connectivity company
20630 Plummer Street
Chatsworth, CA 91311
Tel +1 800 423 5483
Tel +1 818 701 2750
Fax +1 818 701 2799
info@celesco.com

TE.com/sensorsolutions

Measurement Specialties, Inc., a TE Connectivity company.

Measurement Specialties, TE Connectivity, TE Connectivity (logo) and EVERY CONNECTION COUNTS are trademarks. All other logos, products and/or company names referred to herein might be trademarks of their respective owners.

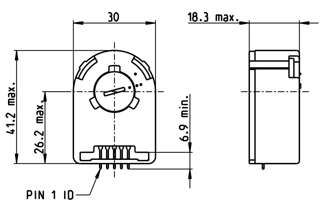
The information given herein, including drawings, illustrations and schematics which are intended for illustration purposes only, is believed to be reliable. However, TE Connectivity makes no warranties as to its accuracy or completeness and disclaims any liability in connection with its use. TE Connectivity's obligations shall only be as set forth in TE Connectivity's Standard Terms and Conditions of Sale for this product and in no case will TE Connectivity be liable for any incidental, indirect or consequential damages arising out of the sale, resale, use or misuse of the product. Users of TE Connectivity products should make their own evaluation to determine the suitability of each such product for the specific application.

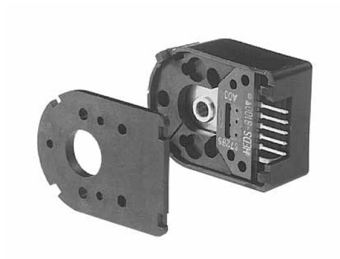
© 2015 TE Connectivity Ltd. family of companies All Rights Reserved.

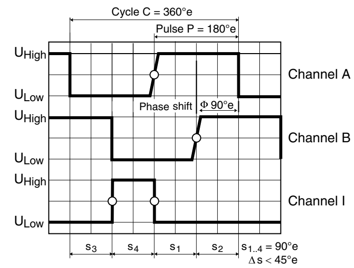
SP1 12/01/2015

I Encoder Data Sheet

Encoder HEDS 5540 500 CPT, 3 Channels







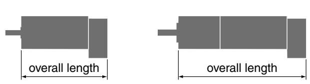
Direction of rotation cw (definition cw p. 60)

maxon sensor

Stock program
Standard program
Special program (on request)

Part Numbers

Type	110511	110513	110515	X drives
Counts per turn	500	500	500	500
Number of channels	3	3	3	3
Max. operating frequency (kHz)	100	100	100	100
Max. speed (rpm)	12 000	12 000	12 000	12 000
Shaft diameter (mm)	3	4	6	2-4



maxon Modular System

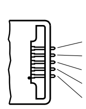
+ Motor	Page	+ Gearhead	Page	+ Brake	Page	Overall length [mm] / ● see Gearhead			
RE 25	125/127					75.3			
RE 25	125/127	GP 26, 0.75 - 4.5 Nm	340			●			
RE 25	125/127	GP 32, 0.75 - 6.0 Nm	342-346			●			
RE 25	125/127	KD 32, 1.0 - 4.5 Nm	352			●			
RE 25	125/127	GP 32 S	374-379			●			
RE 25, 20 W	127			AB 28	480	105.8			
RE 25, 20 W	127	GP 26, 0.75 - 4.5 Nm	340	AB 28	480	●			
RE 25, 20 W	127	GP 32, 0.75 - 6.0 Nm	342-346	AB 28	480	●			
RE 25, 20 W	127	KD 32, 1.0 - 4.5 Nm	352	AB 28	480	●			
RE 25, 20 W	127	GP 32 S	374-379	AB 28	480	●			
RE 30, 15 W	128					88.8			
RE 30, 15 W	128	GP 32, 0.75 - 4.5 Nm	344			●			
RE 30, 60 W	129					88.8			
RE 30, 60 W	129	GP 32, 0.75 - 6.0 Nm	342-349			●			
RE 30, 60 W	129	KD 32, 1.0 - 4.5 Nm	352			●			
RE 30, 60 W	129	GP 32 S	374-379			●			
RE 35, 90 W	130						91.7		
RE 35, 90 W	130	GP 32, 0.75 - 8.0 Nm	342-350				●		
RE 35, 90 W	130	GP 42, 3.0 - 15 Nm	354				●		
RE 35, 90 W	130	GP 32 S	374-379				●		
RE 35, 90 W	130			AB 28	480	124.3			
RE 35, 90 W	130	GP 32, 0.75 - 8.0 Nm	342-350	AB 28	480	●			
RE 35, 90 W	130	GP 42, 3.0 - 15 Nm	354	AB 28	480	●			
RE 35, 90 W	130	GP 32 S	374-379	AB 28	480	●			
RE 40, 25 W	131						91.7		
RE 40, 150 W	132						●		
RE 40, 150 W	132	GP 42, 3.0 - 15 Nm	354				●		
RE 40, 150 W	132	GP 52, 4.0 - 30 Nm	359				●		
RE 40, 150 W	132			AB 28	480	124.3			
RE 40, 150 W	132	GP 42, 3.0 - 15 Nm	354	AB 28	480	●			
RE 40, 150 W	132	GP 52, 4.0 - 30 Nm	359	AB 28	480	●			
DCX 22 S	80-81								online
DCX 22 L	82-83								online
DCX 26 L	84-85								online
DCX 32 L	86								online
DCX 35 L	87								online

Technical Data

Supply voltage V_{CC}	5 V ± 10%
Typical current draw	55 mA
Output signal	TTL compatible
Phase shift Φ	90°e ± 45°e
Signal rise time (typically, at C_L = 25 pF, R_L = 2.7 kΩ, 25 °C)	180 ns
Signal fall time (typically, at C_L = 25 pF, R_L = 2.7 kΩ, 25 °C)	40 ns
Index pulse width (nominal)	90°e
Operating temperature range	-40…+100 °C
Moment of inertia of code wheel	≤ 0.6 gcm²
Max. angular acceleration	250 000 rad s^{-2}
Output current per channel	min. -1 mA, max. 5 mA

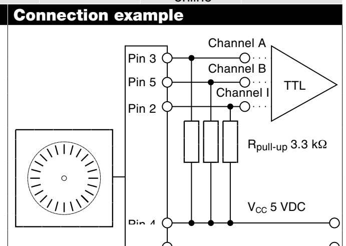
The index signal I is synchronized with channel A or B.

Pin Allocation



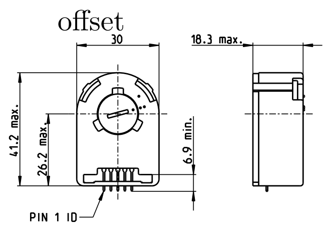
Encoder	Description	Pin no. from 3409.506
Pin 5	Channel B	1
Pin 4	V_{CC}	2
Pin 3	Channel A	3
Pin 2	Channel I	4
Pin 1	GND	5

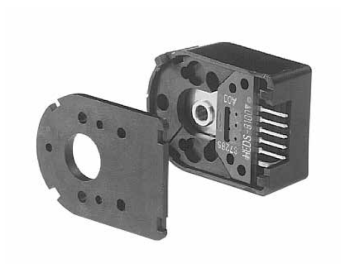
Connection example

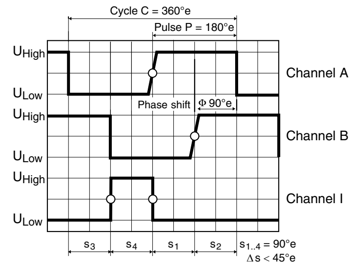


Ambient temperature range ϑ_U = 25 °C

Encoder HEDS 5540 500 CPT, 3 Channels







Direction of rotation cw (definition cw p. 60)

- Stock program
- Standard program
- Special program (on request)

Part Numbers

Type	110511	110513	110515	110517
Counts per turn	500	500	500	500
Number of channels	3	3	3	3
Max. operating frequency (kHz)	100	100	100	100
Max. speed (rpm)	12000	12000	12000	12000
Shaft diameter (mm)	3	4	6	8

maxon Modular System

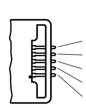
+ Motor	Page	+ Gearhead	Page	+ Brake	Page	Overall length [mm] / ● see Gearhead			
RE 25, 20 W	126					63.8			
RE 25, 20 W	126	GP 26, 0.75 - 4.5 Nm	340			●			
RE 25, 20 W	126	GP 32, 0.75 - 4.5 Nm	342			●			
RE 25, 20 W	126	GP 32, 0.75 - 6.0 Nm	343/346			●			
RE 25, 20 W	126	KD 32, 1.0 - 4.5 Nm	352			●			
RE 25, 20 W	126	GP 32 S	374-378			●			
RE 25, 20 W	126			AB 28	480	94.3			
RE 25, 20 W	126	GP 22, 0.5 Nm	334			●			
RE 25, 20 W	126	GP 26, 0.75 - 4.5 Nm	340	AB 28	480	●			
RE 25, 20 W	126	GP 32, 0.75 - 4.5 Nm	342	AB 28	480	●			
RE 25, 20 W	126	GP 32, 0.75 - 6.0 Nm	343/346	AB 28	480	●			
RE 25, 20 W	126	KD 32, 1.0 - 4.5 Nm	352	AB 28	480	●			
RE 25, 20 W	126	GP 32 S	374-378	AB 28	480	●			
RE 50, 200 W	133								128.7
RE 50, 200 W	133	GP 52, 4 - 30 Nm	360						●
RE 50, 200 W	133	GP 62, 8 - 50 Nm	361						●
RE 65, 250 W	134								157.3
RE 65, 250 W	134	GP 81, 20 - 120 Nm	362						●
A-max 26	152-158					63.1			
A-max 26	152-158	GP 26, 0.75 - 4.5 Nm	340			●			
A-max 26	152-158	GS 30, 0.07 - 0.2 Nm	341			●			
A-max 26	152-158	GP 32, 0.75 - 4.5 Nm	342			●			
A-max 26	152-158	GP 32, 0.75 - 6.0 Nm	343/347			●			
A-max 26	152-158	GS 38, 0.1 - 0.6 Nm	353			●			
A-max 26	152-158	GP 32 S	374-378			●			
A-max 32	160/162						82.3		
A-max 32	160/162	GP 32, 0.75 - 6.0 Nm	342-347				●		
A-max 32	160/162	GS 38, 0.1 - 0.6 Nm	353				●		
A-max 32	160/162	GP 32 S	374-378				●		
EC 32, 80 W	212						78.4		
EC 32, 80 W	212	GP 32, 0.75 - 6.0 Nm	342-349				●		
EC 32, 80 W	212	GP 32 S	374-378				●		
EC 40, 170 W	213							103.4	
EC 40, 170 W	213	GP 42, 3.0 - 15 Nm	354					●	
EC 40, 170 W	213	GP 52, 4.0 - 30 Nm	359					●	

Technical Data

Supply voltage V_{CC}	5 V ± 10%
Typical current draw	55 mA
Output signal	TTL compatible
Phase shift Φ	90°e ± 45°e
Signal rise time (typically, at C_L = 25 pF, R_L = 2.7 kΩ, 25 °C)	180 ns
Signal fall time (typically, at C_L = 25 pF, R_L = 2.7 kΩ, 25 °C)	40 ns
Index pulse width	90°e
Operating temperature range	-40…+100 °C
Moment of inertia of code wheel	≤ 0.6 gcm²
Max. angular acceleration	250 000 rad s⁻²
Output current per channel	min. -1 mA, max. 5 mA

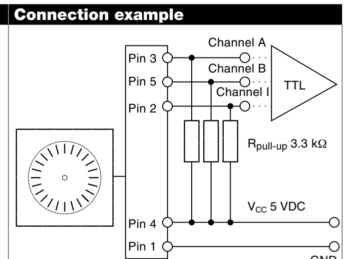
The index signal I is synchronized with channel A or B.

Pin Allocation



Encoder	Description	Pin no. from 3409.506
Pin 5	Channel B	1
Pin 4	V_{CC}	2
Pin 3	Channel A	3
Pin 2	Channel I	4
Pin 1	GND	5

Connection example



Ambient temperature range ϑ_U = 25 °C

NTNU

Norwegian University of
Science and Technology