Distributed Control Architecture for Real-time Model Predictive Control for System-level Harmonic Mitigation in Power Systems

Espen Skjong, Tor Arne Johansen, *Senior Member, IEEE*, and Marta Molinas, *Member, IEEE*

***Abstract*—It can be challenging to design and implement Model Predictive Control (MPC) schemes in systems with fast dynamics. As MPCs often introduce high computational loads, it can be hard to assure real-time properties required by the dynamic system. An understanding of the system's behavior, to exploit system properties that can benefit real-time implementation is imperative. Moreover, MPC implementations on embedded local devices rarely allows flexibility to changes in model and control philosophy, due to increased complexity and computational loads. A change in control philosophy (run-time) can be quite relevant in power systems that can change from an integrated to a segregated state. This paper proposes a distributed control hierarchy with a real-time MPC implementation, designed as a higher-level control unit, to feed a lower-level control device with references. The higher-level control unit's objective in this paper is to generate the control reference of an Active Power Filter for system-level harmonic mitigation. In particular, a novel system architecture, which incorporates the higher-level MPC control and handles distribution of control action to low-level controllers, as well as receiving measurements used by the MPC, is proposed to obtain the application's real-time properties and control flexibility. The higher-level MPC control, which is designed as a distributed control node, can be swapped with another controller (or control philosophy) if the control objective or the dynamic system changes. A standard optimization framework and standard software and hardware technology is used, and the MPC is designed on the basis of repetitive and distributed control, which allows the use of relatively low control update rate. A simulator architecture is implemented with the aim of mimicking a Hardware-In-Loop (HIL) simulator test to evaluate the application's real-time properties, as well as the application's resource usage. The results demonstrates that the implementation of the harmonic mitigation application exhibits the real-time requirements of the application with acceptable resource usage.**

***Index Terms*—Real-time, model predictive control, harmonic mitigation, system architecture, distributed hierarchical control, repetitive control**

This work has been carried out at the Centre for Autonomous Marine Operations and Systems (AMOS). The Norwegian Research Council is acknowledged as the main sponsor of AMOS. This work was supported by Ulstein Power & Control AS and the Research Council of Norway, Project number 241205, and Ulstein Blue Ctrl AS.

E. Skjong (corresponding author) is with the Department of Engineering Cybernetics, Norwegian University of Science and Technology, 7034 Trondheim, Norway, with the Centre for Autonomous Marine Operations and Systems (NTNU-AMOS), Norwegian University of Science and Technology, 7052 Trondheim, Norway, and with Ulstein Blue Ctrl AS, 6018 Ålesund, Norway (e-mail: espen.skjong@ulstein.com)

T. A. Johansen is with the Department of Engineering Cybernetics, Norwegian University of Science and Technology, 7034 Trondheim, Norway, with the Centre for Autonomous Marine Operations and Systems (NTNU-AMOS), Norwegian University of Science and Technology, 7052 Trondheim, Norway (e-mail: tor.arne.johansen@itk.ntnu.no)

M. Molinas is with the Department of Engineering Cybernetics, Norwegian University of Science and Technology, 7034 Trondheim, (e-mail: marta.molinas@ntnu.no)

NOMENCLATURE

Symbol	Description	Unit
$\mathbf{i}_j$	Current in *abc* frame, electrical segment j	A
$\mathbf{v}_j$	Voltage in *abc* frame, electrical segment j	V
L_j	Impedance, electrical segment j	H
C_j	Capacitance, electrical segment j	F
R_j	Resistance, electrical segment j	Ω
$\mathbf{x}$	Dynamic states vector	-
$\mathbf{z}$	Algebraic states vector	-
$\mathbf{u}$	Control vector	-
$\mathbf{g}(\cdot)$	Equality constraints function	-
$\mathbf{h}(\cdot)$	Inequality constraints function	-
$\mathbf{l}(\cdot)$	Stage cost function	-
$\mathbf{V}(\cdot)$	Objective function	-

I. INTRODUCTION

Model predictive control (MPC), which is founded on optimization, utilizes a model of the system to online forecast system behavior and optimize the forecast to produce the best control decision at the current time instance [1], [2]. The model, which is an approximation of the physical system that represent the dynamics under investigation, is initialized by measurements, or estimates, of the system's current state. A cost function, defining the objective of the control and constraints, may be applied to reflect the system's physical and operational limitations. At each sampling interval the future control action is obtained by solving online a finite horizon optimal control problem. A range of different MPC schemes have been developed for systems with different properties and requirements, including deterministic as well as stochastic, linear and nonlinear systems. Hence, MPC is not one single method but rather a set of methods and algorithms that forms a control philosophy [3]. A general, but simplified, illustration of MPC is portrayed in Fig. 1.

Since the early advents of MPC in the process industries, thousands of successful MPC applications have been implemented in the same industries [1], [4], [5]. A lot of research has been directed to the MPC's area of application, and MPC has been investigated within several industries and fields of research in the pursuit of smart control schemes. The desired outcome of this research has been to improve existing non-optimal control strategies, or to solve challenging control problems where conventional control theory alone does not provide a sufficient solution. In this regard, MPC is often used

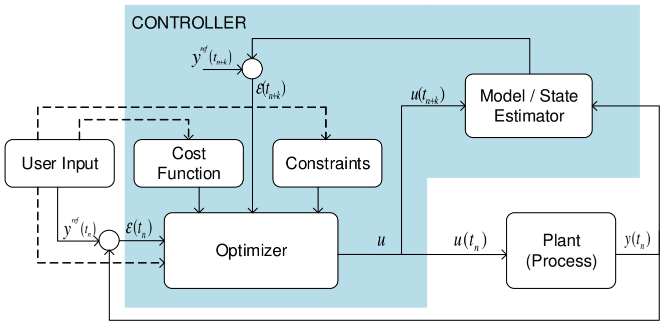


Fig. 1. Simplified illustration of model predictive control, with measurements $y(\cdot)$, references $y^{ref}(\cdot)$ controls $u(\cdot)$, error (difference between references and measurements) $\varepsilon(\cdot)$, and time step t_n.

as a higher-level controller feeding one or multiple lower-level controllers with references, or set-points, to be tracked.

In the field of electrical power engineering, MPC has been frequently investigated as a vital option for optimal control of power converters [6]–[14], where the switching of the Power Electronics (PE) devices has been the main focus of control. As examples, in [9] an indirect Finite Control Set (FCS) MPC is investigated for the optimal control of the Modular Multilevel Converter's (MMC) switching. In [15] MPC is applied to power system protection schemes, ship energy management [16], control of batteries in a peak-shaving application is discussed in [17], frequency control in [18], control of distributed energy resources in [19], [20], and mitigation of harmonic distortions in [21]–[26]. MPCs do often introduce high computational loads that might require the computational loads to be shared among multiple distributed controller units. [27], [28] do not utilize MPCs, however, present interesting applications using multi-layered and distributed optimization-based control strategies for optimal power flow in transmission and distribution systems. Even though simultaneous real-time optimization and control is one of the most desirable properties of MPC, there is still a vast area of applications in electrical power engineering where multi-layered control is common practice, utilizing ad-hoc offline optimization strategies [29], [30]. An example of such an application is mitigation of harmonic distortions.

Harmonic distortions, which are any deviation from the pure sinusoidal voltage or current waveform, introduce active power losses and contributes to reactive power in the system [31]. Methods for mitigating harmonic distortion include the use of passive and active filters. Unlike passive filters, the active filters can be controlled, and, depending on the control philosophy, be able to adapt to changes in the harmonic distortion spectra. This is a desirable functionality, especially in power systems with dynamic load profiles. The most applied control philosophy for active filters involves the mitigation of harmonic distortion at a specific location in the power system (e.g. [31], [32]). However, as active filters can be controlled to dynamically track a current reference, a single active filter can be designed to track a current reference that can optimize the harmonic profile of the entire system. This task can be performed in real-time by a tailor-designed MPC.

This paper proposes a scheme for real-time MPC implementation in a case-study of system-level harmonic profile optimization, where the common practice has been the use of offline optimization for the choice of set-points for the converter controllers. The main contribution and novelty in this paper lies in the real-time system framework and implementation of an MPC designed for such a task, in contrast with the state of the art solution based on offline optimization for set-point definition. In specific, the novelty lies on the use of a standard hardware and software platform for the real-time implementation of a Continuous Control Set (CCS) MPC application for optimal mitigation of harmonic distortions, as discussed in [21]–[25]. By exploiting the periodic nature of the voltage and current waveforms to use relatively low control update rate, a repetitive MPC control philosophy is selected and a dedicated real-time framework is proposed. The MPC implementation is split in two levels, by exploiting the architecture of this dedicated hardware-software platform. In the higher level, the MPC is designed as a higher-level distributed control node that feeds a lower-level (local) controller with references, or control set-points. The MPC, or the higher-level distributed control node, can be swapped with another controller on-the-fly if the control philosophy or the dynamical system changes. Hardware-in-Loop (HIL) simulation experiments are conducted to verify the system architecture with regards to the MPC's execution cost and the time delay introduced by the framework and the communication link. The novel framework enables a reliable and fast nonlinear MPC to be implemented in this challenging application by using standard optimization frameworks and standard software and hardware technology without resorting to hard real-time systems implemented on embedded devices, such as FPGAs and PLCs, and formal verification.

The paper is organized as follows: The problem formulation and adopted control philosophies are addressed in section II, section III presents the system architecture and the implementation of the MPC and its framework and middleware. Furthermore, section IV presents a HIL test of the system architecture. Finally, section V concludes the work.

II. Problem Formulation

The MPC uses a model, or a state estimator, of the system to predict future behavior and be able to calculate the best possible control action to control the system to meet a desired objective. At the same time, the MPC has to comply with the system's physical and operational constraints. In the following, the derivation of the MPC and its model on standard form for the optimal harmonic mitigation application, as introduced in [24], [25] for a two-bus shipboard power system, will be discussed. The different hardware layers and adopted control philosophy will be introduced.

A. MPC Formulation

For simplicity of presentation, although the concept is general, a simplified model of a two-bus shipboard power system, which is illustrated in Fig. 2, is used in the design of the MPC's internal model. According to Kirchhoff's laws, the model's dynamic equations can be stated as

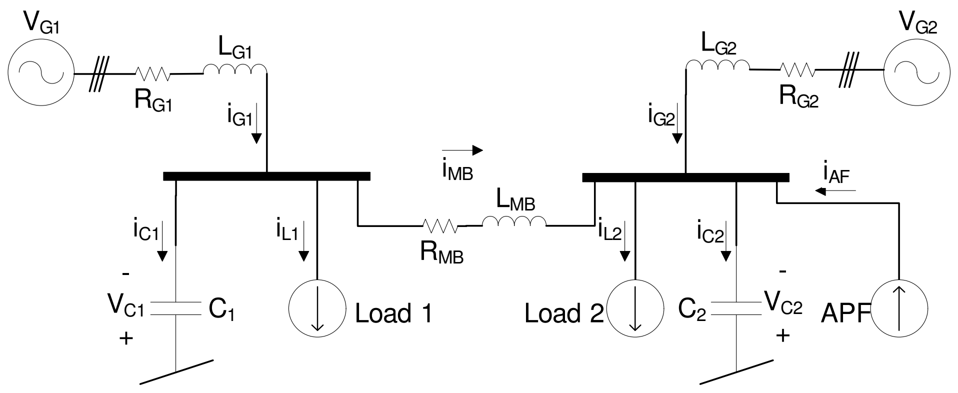


Fig. 2. Simplified model of a two-bus shipboard power system: Propulsion loads and Active Power Filter (APF) modeled as ideal current sources, generators modeled as ideal voltage sources. Shunt capacitors are included for the purpose of modeling cable capacitance and provide bus voltages [24].

$$L_{S1}\frac{d\mathbf{i}_{S1}}{dt} = -R_{S1}\mathbf{i}_{S1} - \mathbf{v}_{C1} \tag{1a}$$

$$C_1\frac{d\mathbf{v}_{C1}}{dt} = \mathbf{i}_{S1} - \mathbf{i}_{MB} - \mathbf{i}_{L1} \tag{1b}$$

$$L_{MB}\frac{d\mathbf{i}_{MB}}{dt} = \mathbf{v}_{C1} - \mathbf{v}_{C2} - R_{MB}\mathbf{i}_{MB} \tag{1c}$$

$$C_2\frac{d\mathbf{v}_{C2}}{dt} = \mathbf{i}_{MB} + \mathbf{i}_{S2} - \mathbf{i}_{L2} + \mathbf{i}_F \tag{1d}$$

$$L_{S2}\frac{d\mathbf{i}_{S2}}{dt} = -R_{S2}\mathbf{i}_{S2} - \mathbf{v}_{C2}, \tag{1e}$$

where t represents the continuous time. The vectors $\mathbf{v}$ and $\mathbf{i}$ represent the three-phase voltages and currents, respectively, given in the abc frame. Assuming the generators are not sources of harmonic distortion, the fundamental components (voltages and currents) are left out of (1), as only the dynamics originating from the harmonic distortion introduced by the loads are subjects for optimization. The propulsion loads ($\mathbf{i}_{L1}$ and $\mathbf{i}_{L2}$) can be modeled as Fourier series,

$$\mathbf{i}_{Lj}(t) = \begin{bmatrix} \sum_i I^a_{L,j,i}\sin\left(i\left(\omega t + \phi^a_{L,j,i}\right)\right) \\ \sum_i I^b_{L,j,i}\sin\left(i\left(\omega t + \phi^b_{L,j,i} - \frac{2\pi}{3}\right)\right) \\ \sum_i I^c_{L,j,i}\sin\left(i\left(\omega t + \phi^c_{L,j,i} + \frac{2\pi}{3}\right)\right) \end{bmatrix}, \quad \forall i \in \mathcal{H}, j \in \{1,2\}, \tag{2}$$

where $\mathcal{H}$ is the set of harmonic orders to be mitigated, $\omega = 2\pi f$ with f as the fundamental frequency, $I^k_{j,i}$ and $\phi^k_{L,j,i}$ are harmonic amplitudes and phases, respectively, for phases $k \in \{a,b,c\}$. The active filter's current constraints can be stated as

$$i^k_{\text{min}} \le i^k_F \le i^k_{\text{max}}, \tag{3}$$

with phases $k \in \{a,b,c\}$. Assuming a balanced filter yields $i^k_{\text{max}} = -i^k_{\text{min}}\ \forall k$, and $i^k_m = i^l_m\ \forall(k,l)|_{k\neq l} \in \{a,b,c\}$ with $m \in \{\text{min},\text{max}\}$. The harmonic mitigation problem can now be written on standard MPC form,

$$\begin{aligned} \min_{\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t)} \quad & V(\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t)) = \\ & \int_{t_0}^{t_0+T} l\left(\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t)\right)dt \\ \text{s.t.} \quad & \\ & \dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t)\right), \\ & \mathbf{g}\left(\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t)\right) = 0, \\ & \mathbf{h}\left(\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t)\right) \le 0, \\ & \forall t \in [t_0, t_0+T] \end{aligned} \tag{4}$$

with initial time instance t_0 and horizon length T, dynamic equations $\mathbf{f}(\cdot)$, algebraic equations $\mathbf{g}(\cdot)$ and inequality constraints $\mathbf{h}(\cdot)$. By dropping the time notation t, the dynamic state vector $\mathbf{x}$, algebraic state vector $\mathbf{z}$ and control vector $\mathbf{u}$ are stated as

$$\begin{aligned} \mathbf{x} &= \left[\mathbf{i}^\top_{S1}, \mathbf{i}^\top_{S2}, \mathbf{i}^\top_{MB}, \mathbf{v}^\top_{C1}, \mathbf{v}^\top_{C2}\right]^\top \\ \mathbf{z} &= \left[\mathbf{i}^\top_{L1}, \mathbf{i}^\top_{L2}\right]^\top \\ \mathbf{u} &= \mathbf{i}_F. \end{aligned} \tag{5}$$

The objective function, which specifies the objective of the optimization, is $V(\cdot)$ with the convex stage cost function

$$l\left(\mathbf{x},\mathbf{z},\mathbf{u}\right) = \mathbf{i}^\top_{S1}\mathbf{Q}_1\mathbf{i}_{S1} + \mathbf{i}^\top_{S2}\mathbf{Q}_2\mathbf{i}_{S2} + \mathbf{u}^\top\mathbf{Q}_u\mathbf{u}. \tag{6}$$

The first and second term in (6) represent the quadratic contribution of harmonic currents drawn from the generators, while the last part is included to penalize the use of large (high amplitude) active filter currents. $\mathbf{Q}_1$, $\mathbf{Q}_2$ and $\mathbf{Q}_\mathbf{u}$ are diagonal weight matrices, where $Q_{1,ii}, Q_{2,ii} \ge Q_{\mathbf{u},ii}$ for all $i \in \{1,2,3\}$ as minimizing the harmonic currents is of greater importance than penalizing the utilization of large active filter currents. As evident from (1)-(6), all three phases are decoupled from each other, which allows the use of independent distributed MPCs, one for each phase, according to [33]. This is a desired property, which might be crucial in the pursue of meeting the application's real-time demands. In the rest of this work the MPC will be treated as a single-phase

MPC, according to the phase decoupling of the three-phase MPC formulation presented by (1)-(6).

B. Control and Hardware Layers

MPC is often used as a higher-level controller feeding one or multiple lower-level controllers with setpoints or references (trajectories) to track, which forms a multi-layered solution involving both hardware and software. A simplified schematic of the control- and hardware layers for the optimal harmonic mitigation application discussed in this work is showcased in Fig. 3. As can be seen in the figure, the MPC is part of the higher-level control layer and interacts with the lower-level control layer. The lower-level control layer, which consists of the Active Power Filter (APF) controller and measurement processing, interacts with the MPC in the higher-level control layer and the power system in the hardware layer. The amplitude and phase information from the harmonics to be mitigated are provided by FFT, or a suitable estimator which can be realized on a suitable hardware platform such as FPGAs (Field-Programmable Gate Arrays), DSPs (Digital Signal Processor), or PLCs (Programmable Logic Controller). Other available measurements are also provided by the lower-level control layer (MEAS. SAMPLING block in Fig. 3). As the MPC utilizes a simplified model of the power system for predictive purposes, all available measurements should be used to update and initialize the MPC model to minimize the modelling errors. This includes voltage and current measurements as well as impedance measurements. The measurement gathering and synchronization can be realized using standard hard-real-time industrial hardware and software solutions, e.g. FPGAs, DSPs and PLCs, that utilizes estimators and filters along with time synchronization mechanisms such as PTP (Precision Time Protocol). Furthermore, the APF controller is fed with an optimal current reference calculated by the MPC. As the higher-level control layer exists of distributed (general) control nodes running higher-level control (MPC) to generate instantaneous current references for the lower-level (embedded) controller to track, the time scale for the higher- and lower-level control layer is quite different. The higher-level control layer, which is a software-oriented control layer, works in a time scale corresponding to the time horizon spanned by the MPC, while the lower-level control layer, which is a hardware-oriented control layer, adopts the time scale of the APF, thus is more time sensitive. In this work the design of the higher-level control layer will be treated, in which incorporates the lower-level control layer as well as the hardware layer.

C. Control Philosophy and Real-time Classification

Maybe the most used MPC control philosophy, where MPC is involved as a higher-level controller, is to use the first, or first few, points from the MPC's output control vector, which are fed to one or multiple lower-level controllers and used as setpoints. This control philosophy is not suitable for the harmonic mitigation application presented in this work due to the inherent fast dynamics. The MPC must be able to provide a new setpoint/reference to the APF control at a

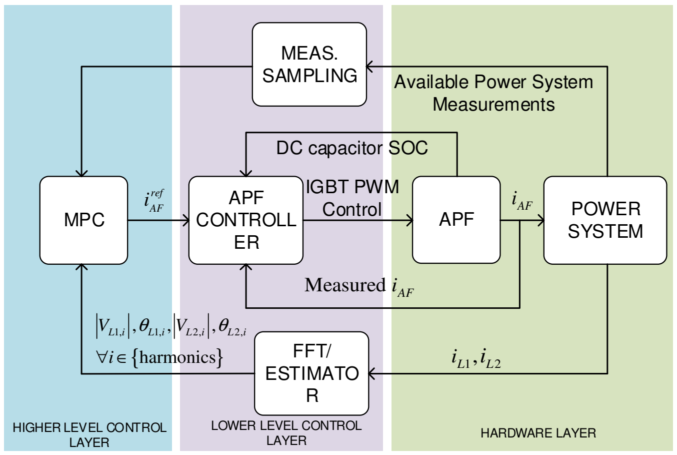


Fig. 3. Multi-layered control- and hardware architecture for the optimal harmonic mitigation application using MPC: The MPC in the higher-level control layer interacts with the lower control layer, while the lower layer control layer interacts with the hardware layer.

specific time instance to assure harmonic mitigation properties. If the setpoint update is too slow, the active filter may inject harmonics contributing to a higher THD, thus failing to meet the designed control objective of harmonic mitigation.

Another control philosophy, that is supported by Continuous Control Set (CCS) MPCs, which is adopted in this work, is repetitive control. Instead of using only one or the first few points from the MPC's optimal future control vector, the whole vector is used to form a reference for the active filter to track. The MPC's optimization horizon can be designed to span one fundamental period, in which, due to the optimization problem's nature, gives interesting properties that can be used for fault handling: If for instance the MPC fails to deliver a new control vector to the APF control within a finite deadline, the old control vector might be used once more for the next period. How to utilize this property of repetitive control will be discussed in more details later on.

To be able to define success criteria for the implementation, where required real-time properties are assured, a real-time system must be defined. There exist many definitions classifying real-time systems. In general, a real-time system is *a system that is required to react to stimuli from the environment (including the passage of physical time) within time intervals dictated by the environment* [34]. A real-time system is in terms any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period. Furthermore, real-time systems can be split into two groups, *soft* and *hard*, with the following definitions, [34]:

- *Hard* real-time systems are those where it is absolutely imperative that responses occur within the specified deadline.
- *Soft* real-time systems are those where response times are important but the system will still function correctly if deadlines are occasionally missed.

With the definitions of *hard* and *soft* real-time systems one can really wonder if a MPC would be hard real-time (or real-time at all). It all comes down to the application area and the requirements (success criteria) defined by the application. For many MPC applications it would be a requirement to include an additional control logic (probably not based on optimization) as backup if the MPC is not able to finish its calculations in time, [4]. This is especially important for hard (embedded) real-time systems with *firm* deadlines.

In this work the MPC can be classified as being part of a *soft* real-time system, due to the repetitive nature of the calculated control vector (applied control vector equals one fundamental period in length). If the MPC is not able to finish in time, the previous control vector can be used. In addition, alternative mitigation approaches, that do not rely on optimization can be used, as discussed in [24], as backup controllers in case the MPC overshoots its deadline. Since the commands from the MPC are synchronized within the lower-level control layer in Fig 3, the consequence of the MPC not finishing its calculations in time is not a failure of the control system, but a limited degradation in control performance with limited increase in THD for a short period of time. Furthermore, the MPC scheduling is strongly dependent on receiving new (and updated) measurements. If the MPC does not receive new measurements, the previous control vector should be used as control action. Hence, the MPC should never be *re-scheduled* based on *old* measurement. In this term, the scheduling of the MPC can be said to be a *reactive* and *event-triggered (sporadic) soft* real-time system.

III. System Architecture and Implementation

To obtain the required real-time properties for an MPC controlling a stiff system with fast dynamics might prove to be a challenge. Even if the MPC is able to meet the necessary real-time demands, the middleware, which is responsible for connecting the MPC to the rest of the system, might introduce additional critical latency which combined with the MPC's computational costs fail to assure the real-time demands. As important as thorough MPC design and tuning, the middleware should be designed to interact with the MPC and the rest of the system without unnecessary overhead. In the following the MPC implementation and the system architecture, including the middleware, is discussed. Moreover, a simple simulator architecture is proposed for the purpose of verifying the control system's real-time properties.

A. MPC Implementation

There exist many suitable software solutions and libraries for nonlinear MPC implementations, and two examples are CasADi [35] and the ACADO toolkit [36]. These simplifies the implementation since they provide an abstraction layer between the MPC specification and the numerical optimization software. In this work the ACADO toolkit is used due to its fast prototyping properties and real-time support [37]. ACADO comes with a high-level C++ interface, where the MPC's model and specifications are written on standard form. From this C++ interface, a highly efficient C code can be generated. This approach has been adopted in this work using the MPC on standard form in (4), and the generated C code has been embedded in a larger system which will be discussed below.

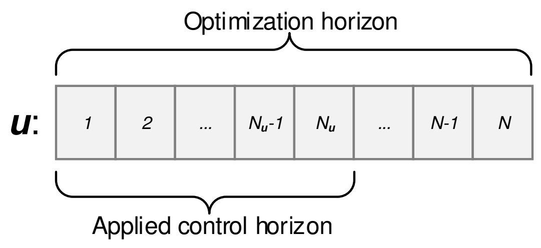


Fig. 4. Visual representation of the control vector length: Optimization horizon discretization steps N, applied control steps $N_{\mathbf{u}}$, $N > N_{\mathbf{u}}$.

To achieve the needed real-time properties, the discretization and optimization horizon, as well as integrator and Nonlinear Programming (NLP) solver, have to be chosen with care. These design parameters will be treated separately in the following:

*1) **Optimization Horizon**:* The control philosophy adopted in this work is repetitive control. Repetitive control means that controls reappear in a repetitive manner, which indeed is the case for harmonic currents. Harmonic currents introduce fast nonlinear dynamics, which require a lot of computational effort. Therefore, instead of using one single step from each optimal future control vector calculated by the MPC, which is an often applied practice in MPC design, a whole fundamental period of control steps (i.e. active filter currents) will be used. This requires the optimization horizon to be larger than one fundamental period (20ms for 50Hz), enabling an overlap between fundamental horizons. Such an overlap is important for keeping future changes in account, and assure optimality between fundamental periods. Moreover, by using a control vector which spans one fundamental period, the repetitive nature of the harmonic currents allows to reuse the same control vector (assuming approximately constant fundamental frequency) if the MPC fails to deliver a new control vector within the required deadline. If the fundamental frequency is not constant, the optimization horizon should be long enough to enclose the freqency variations' fundamental periods. This work assumes a fundamental frequency close to 50Hz, thus the optimization horizon is chosen to be 22ms and the applied control horizon is 20ms. A visualized representation of the applied control horizon and optimization horizon is given in Fig. 4.

*2) **Discretization**:* The discretization should be chosen to represent the fastest dynamics treated in the control application, while, on the other hand, be chosen to satisfy real-time demands as an increased number of discretization steps introduce additional computation costs. Assuming a fundamental frequency of 50Hz, and assuming harmonics up to the 50th order, the Shannon-Nyquist sampling theorem (to avoid aliasing) states that the sampling frequency should be chosen as $2 \cdot 50\text{Hz} \cdot 50 = 5000\text{Hz}$. This gives a step size of 0.2ms, and for a 22ms optimization horizon a discretization of 110 steps is needed.

The discretization type (or method) is responsible to convert the MPC on standard form to NLP form, which can be solved by a suitable NLP solver. The most common discretization methods are single shooting, multiple shooting and collocation [3]. Multiple shooting is a refinement of single shooting.

Unlike single shooting, which integrates a differential state throughout the horizon as one trajectory, multiple shooting divides the optimization horizon into elements. The elements are integrated separately, which gives better numerical stability and robustness due to decoupling of the elements. State constraints are enforced on each segment junction to ensure continuity between the elements throughout the horizon. Multiple shooting forms a larger NLP problem than single shooting, but, on the other hand, enables parallelization of element integration routines, which might give an advantage in the pursuit of real-time properties. Collocation, as with multiple shooting, divides the optimization horizon into elements, however, the state trajectories in a collocation scheme are approximated by polynomials on each control interval within the optimization horizon. Each polynomial is parametrized by interpolating points, which have the same dimension as the state space formulation and are extra decision variables in the NLP scheme. Even though the size of the NLP problem increases compared to multiple shooting, the polynomial approximations of the state trajectories often become easier to solve [3], [22], especially with highly nonlinear system equations, and the system matrices are often sparse which could be exploited by a sparse QP solver.

Both multiple shooting and collocation are good candidates for discretization type for the MPC application presented in this work, however, collocation is not yet supported by the code generation feature in ACADO. Hence, multiple shooting is chosen as the discretization type.

*3) **Integrator**:* The problem formulation presents a stiff nonlinear system, thus using a common integrator such as the Runge-Kutta of order 4 (RK4) will require a high number of integration steps. The RK4 integrator was implemented for the problem formulation in (4), and required 1500 integration steps to converge. Even with that high number of steps the solution was not sufficiently accurate. In addition, the high number of integration steps destroyed the real-time properties of the MPC. In this work the implicit Runge-Kutta Radau IIA of order 3 (RIIA3), which is an integrator that is able to handle stiff systems [38], is chosen with $2N$ (220) integrator steps.

*4) **NLP Solver**:* There exist a range of different NLP solvers with different properties that might fit the MPC proposed in this work. ACADO's code generation feature is currently supporting qpOASES [39] and FORCES [40], which are both Quadratic Programming (QP) solvers. qpOASES is an active set online QP solver, and ACADO provides different condensing techniques when using qpOASES to exploit the structure of the system matrices. FORCES is an interior point QP solver that exploit sparsity in the system matrices. As only qpOASES is open source, with available source code that can easily be embedded in a larger framework, the qpOASES solver is chosen in this work. Table I summarizes the implementation details of the MPC.

TABLE I
MPC IMPLEMENTATION DETAILS.

Parameter	Value
Software	ACADO
Optimization horizon (T)	22ms
Applied control horizon	20ms
Discretization steps (N)	110
Applied control vector discretization steps ($N_{\mathbf{u}}$)	100
Discretization type	Multiple Shooting
Hessian approximation ($\nabla^2\mathbf{f}$)	Gauss-Newton
Integrator type	Implicit Runge Kutta Radau IIA 3 (IRK RIIA3)
Number of integration steps	220 ($2N$)
NLP solver	qpOASES
Number of iterations	2

B. MPC Framework and Architecture

In the design of a system architecture and framework, which comply with real-time demands, aspects such as threading, communication (middleware), scheduling and execution of tasks with cross-thread synchronization need to be considered. In the wake of *Industrial Internet of Things* (IIoT), event-based architectures have gained a lot of attention. Unlike cyclic execution, which runs with a predefined cycle frequency, event-based architectures trigger on events, or signals, meaning that an event-based thread is in hibernation (latent) until an event arrives and triggers execution of the tread. Both cyclic and event-based architectures have desired properties that can be exploited in the design of the system architecture for the harmonic mitigation application addressed in this work. An example of desired property of the event-based architecture is minimal resource use, i.e. memory and CPU, while for a cyclic architecture is fast response. This is because an event-based thread that is latent (sleeping) does not consume processing resources, while a cyclic thread is running whether it is doing any work or not, which adds to the resource use. However, as the cyclic thread is constantly running, it does not have any invoking delay, which might be the case for an event-based thread, depending on the occupied system resources and processor scheduling at the time instant the event mechanism calls for task scheduling.

Fig. 5 portrays the system architecture for the main controller, i.e. the controller running the MPC with suitable middleware and framework. In this work the term middleware is defined as communication between devices, while framework is defined as internal mechanisms that constitutes cross-thread communication and synchronization, internal memory allocation and information sharing. The blocks in Fig. 5 represents threads, and for each tread a state-machine based on Unified Modelling Language (UML) [41] is presented. Cross-thread signals (events) are presented as connections between the threads. The communication (interaction) to the rest of the system (Fig. 3) is also presented as arrows to/from the main controller block. In the following the blocks (threads) in Fig. 5 will be treated separately. Thread names are referred to with bold font, while states and signals use italic font.

*1) **Engine**:* The **Engine** thread is the main component in the MPC framework; it is an event-based thread and act as an event manager, meaning receiving events from the other threads and determines appropriate actions. The appropriate actions are then distributed to the other threads as new events. As can be seen from Fig. 5, the initial state in the **Engine**'s

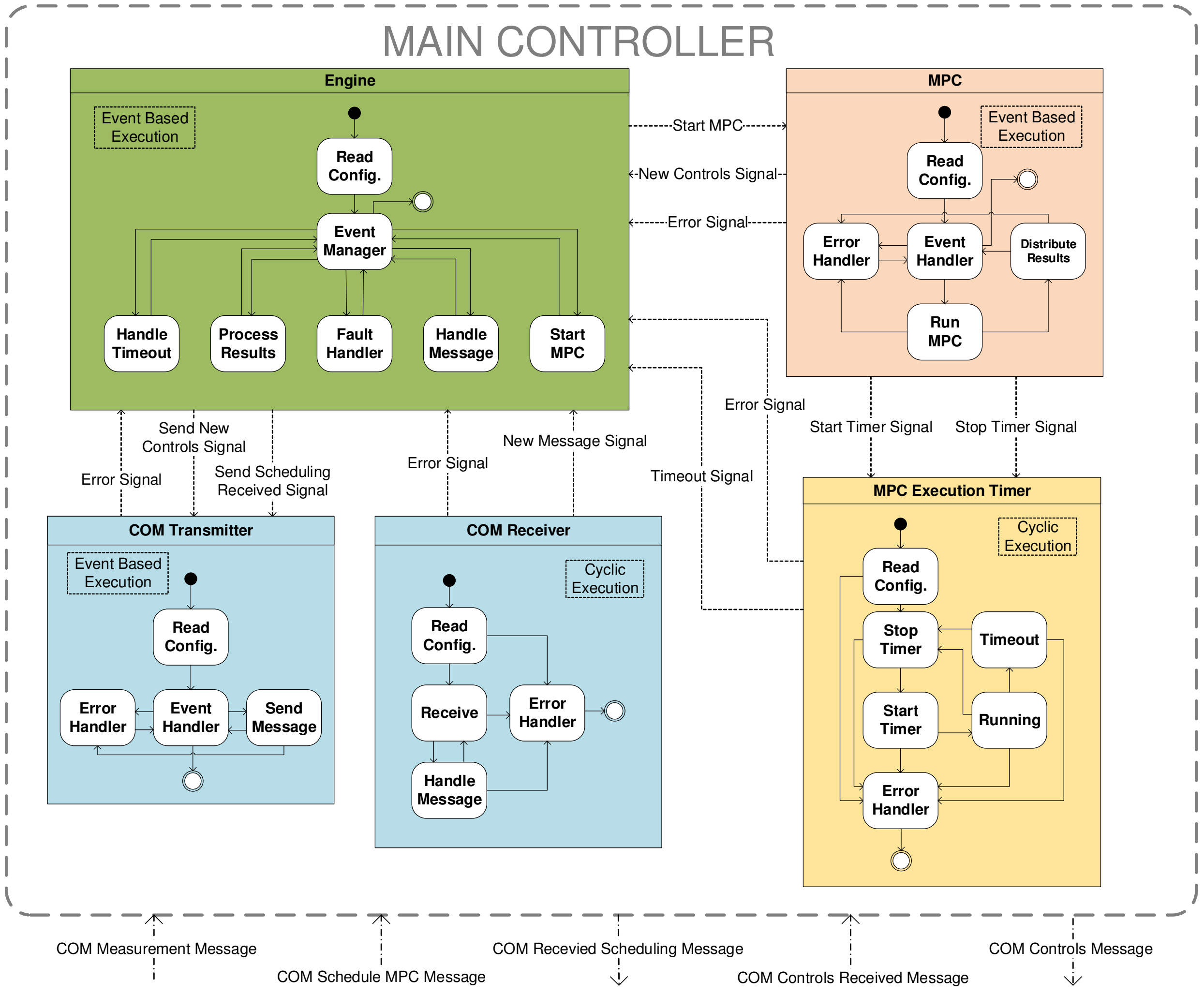


Fig. 5. Simplified schematic of the main controller architecture. The different blocks (either cyclic or event-based execution) represents threads. The signals between the blocks are internal event-based signals. The arrows connected to the main controller block represents communication links with other devices, such as the simulator in Fig. 6.

state-machine is responsible for reading configuration. This configuration is given as an XML file during startup, and includes configuration related to the MPC (parameters), the communication link (**COM Transmitter** and **COM Receiver** thread) and the **MPC Execution Timer** thread. The read configuration is then stored in the shared memory to be loaded by the other threads. As the **Engine** thread is the only thread reading the XML configuration file, all other threads are initialized after the Engine thread finishes reading and storing the configuration. After reading the configuration, the **Engine** goes to the *Event Manager* state and awaits events. Depending on the received events, the **Engine** may take different states. The *Handle Timeout* state handles the timeout of the MPC, i.e. if the execution of the MPC takes longer time than specified in the configuration file. An appropriate action is then to send the previously computed control vector to the active filter, which can be reused due to the repetitive nature of the filter currents and the fact that the applied horizon length equals one fundamental period. Another state is the *Process Results* state, which handles the resulting control vector after the MPC finishes. The control vector is then checked (length, discretization and amplitude) before being sent to the active filter controller through the **COM Transmitter** thread. The *Fault Handler* state handles faults from the different threads. Examples of faults could be communication error or that the MPC is unable to provide a new control vector due to infeasibility. An appropriate action for communication error would be to check the return codes (error messages) from the erroneous sockets for further diagnostics to find the reason behind the errors. As a last resort the communication sockets might be closed and the communication threads reinitialized. For MPC infeasibility, depending on the cause, an appropriate action could be to rerun the MPC with new measurements, while sending the previously computed control vector to the active filter controller. The *Handle Message* state handles all messages received over the communication link, which is delivered by the **COM Receiver thread**. If a *Schedule MPC* message is received, a reply message is made and signalled to the **COM Transmitter** thread to be sent. This reply message can be omitted if a reliable protocol, such as

TCP over Ethernet, is used. The last state, *Start MPC*, signals the MPC thread to start (if new measurements are available), and reallocates memory for a new control vector.

*2) **MPC**:* The **MPC** thread is also an event-based thread which is started when receiving a *Start MPC* signal from the **Engine** thread. As can be seen from Fig. 5, the **MPC** thread has also a *Read Configuration* state, which reads the configuration from the internal memory stored by the **Engine** thread. After the configuration is read, the *Event Handler* state is invoked. When receiving a *Start MPC* signal from the **Engine** thread, the state *Run MPC* is invoked. In this state a *Start Timer* signal is sent to the **MPC Execution Timer** thread before the MPC is run. When the MPC finishes, a *Stop Timer* event is signalled the **MPC Execution Timer** thread, before invoking the *Distribute Results* state. In this state the resulting control vector is stored in the controller's internal memory before signalling to the **Engine** thread that new controls are available. After this, a transition to the *Event Handler* state is made. In case of errors, which may result from the MPC (infeasibility, solver failure, etc.), the **MPC** thread enters the *Error Handler* state, which performs local diagnostics and signals an error message to be handled by the **Engine** thread. After the error message is sent, the **MPC** thread transitions back to the *Event Handler* state.

*3) **MPC Execution Timer**:* This thread is a cyclic (suspend-resume) thread which is started when the **MPC** thread starts, and stopped when the **MPC** thread stops. Its main function is to time the execution of the MPC using a high resolution monotonic timer. As with the previous threads, the **MPC Execution Timer** thread reads the configuration during startup, then enters the *Stop Timer* state. When signalled by the **MPC** thread, the **MPC Execution Timer** thread enters the *Start Timer* state, in which resets and starts the timer before entering the *Running State*. If a *Stop Timer* signal is sent from the **MPC** thread before the timer times out, a transition to the *Stop Timer* state is made. On the other hand, if the timer times out, according to a predefined setpoint in the configuration, a transition to the *Timeout* state is made, which signals a *Timeout* signal to the **Engine** thread. Also this thread has an *Error Handler* state, which handles errors related to the timer object used. If an error is not solved locally, an *Error* signal is sent to the **Engine** thread for further investigation and appropriate actions.

*4) **COM Transmitter**:* This event-based thread is responsible for sending information to other devices, i.e. handles external outgoing communication. As with the other threads this thread has also a *Read Configuration* state, which reads configuration related to the communication link. The communication link itself could be i.e Ethernet or serial communication. After the configuration is read, the thread enters the *Event Handler* state, and awaits events sent by the **Engine** thread. If events are received, i.e. a *Send New Controls* or *Send Scheduling Received* signal, the *Send Message* state is entered, which sends the message before transitioning back to the *Event Handler* state. Examples of messages are *COM Received Scheduling* and *COM Controls*, as depicted in Fig. 5. As communication links may break down or fail, the thread also includes an *Error Handler* state. If the error is not solved locally, an *Error* signal is sent to the *Engine* thread for further investigation.

*5) **COM Receiver**:* This thread is a cyclic (suspend-resume) thread that checks the communication link for new messages in a cyclic behavior. As with the other thread this thread also has a *Read Configuration* state which is entered after the thread initialization. After the configuration is read, the thread transitions to the *Receive* state. If a new message is received over the communication link, the thread enters the *Handle Message* state, which parses the message and copies its content to an appropriate data structure which is stored in the controller's internal memory. A *New Message* event is then signalled to the **Engine** thread, in which processes the message. Example of messages are *COM Measurement* and *COM Schedule MPC*, as depicted in Fig. 5. As with the **COM Transmitter** thread, also this thread has an *Error* state in which communication errors will be handled. If the error is not resolved, an *Error* signal is sent to the **Engine** thread for further action.

C. Simulator Architecture

To test and verify the architecture of the main controller, discussed in the previous section, a simulator architecture is proposed in Fig. 6. As in the previous section, the different threads (blocks) will be separately discussed in the following. The **COM Transmitter** and **COM Receiver** threads adopt the same functionality as for the communication threads in the main controller in Fig. 5.

*1) **Engine**:* The simulator's **Engine** thread is like the **Engine** thread in Fig. 5, although simpler. It is an event-based thread and acts as an event manager. The states *Read Configuration* and *Handle Message* work just like the coinciding threads in the main controller's **Engine** thread, except that the *Handle Message* state has functionality devoted for the types of messages that are received by the simulator's **COM Receiver** thread. As an example, if a *COM Controls* Message is received, depending on the communication protocol, a reply message should be sent to the main controller indicating the new control vector was received. Furthermore, the *Fault Handler* state is responsible for resolving errors that are not resolved locally by corresponding threads. The **Engine** thread also has a *Handle Timeout* state, in which is entered if the **Control Message Timer** thread distributes a *Timeout* event. Such an event is distributed if the time difference between sending a *COM Schedule MPC* message and receiving a *COM Controls* message exceeds a predefined threshold.

*2) **Measurement Simulator**:* The **Measurement Simulator** thread is a cyclic thread responsible for generating measurements that are distributed to the main controller through the **COM Transmitter** thread. The first state is the *Read Configuration* state, which reads the configuration that specifies how the measurements should be generated, e.g. which harmonic orders to generate, amplitude bands, phase bands and rate of change. After the configuration is read, a transition to the *Simulate Measurements* state is made, and in this state the measurements are generated. After the measurements are generated the *Distribute Measurements* state is entered, which

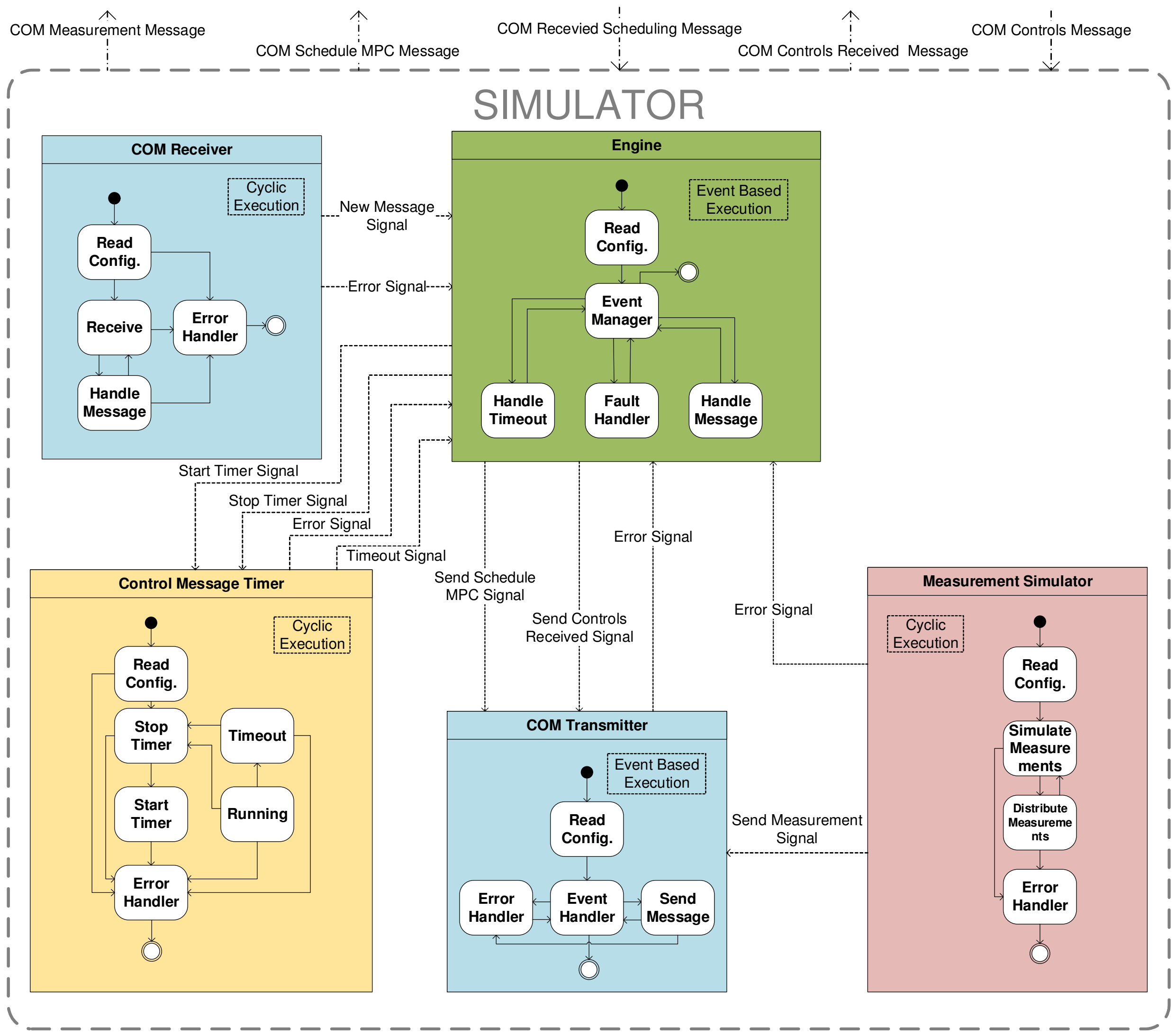


Fig. 6. Simplified schematic of the simulator architecture, which includes measurement generation. The different blocks (either cyclic or event-based execution) represents threads. The signals between the blocks are internal event-based signals. The arrows connected to the simulator block represents communication links with other devices, such as the main controller in Fig. 5.

packs the measurements in a suitable data structure to be sent over the communication link by the **COM Transmitter** thread by invoking a *Send Measurement* event signal. The **Measurement Simulator** thread also has an *Error Handler* state, in which handles errors related to the timer used for generating the measurements. If errors are not solved locally an *Error* signal is sent to the **Engine** thread for further investigation.

3) Control Message Timer: The **Control Message Timer** thread is also a cyclic (suspend-resume) thread. Its main function is to calculate the time difference between sending a *COM Schedule MPC* message over the communication link and receiving a new *COM Controls* message using a high resolution monotonic timer. The thread's state machine has the same structure as the **MPC Execution Timer** in Fig. 5.

D. Synchronization of Measurements

Available measurements, which are sampled different places in the grid, are sent to the main controller after proper processing (noise suppressing and validation) to be used by the MPC. However, if not all the measurements are consistent, i.e. all the measurements are not sampled synchronously, leading to the MPC receives and uses some new measurement along with old measurements, the controls obtained from the MPC cannot be guaranteed to be valid. Thus, synchronization of the measurements are quite important for the MPC to provide a valid control vector. A proper synchronization procedure of the measurement devices might result in unnecessary high communication traffic and communication delay, thus might lead to measurements being invalid when reaching the MPC. As the measurements should be filtered to suppress measurement noise in the lower-level control layer in Fig. 3, an estimator such as a Kalman filter [42] can be used, which has both predictive as well as noise suppression capabilities. The filter's prediction capabilities allow to predict measurements at a desired time instance when measurements do not arrive simultaneously. As the design of measurement processing systems falls outside the scope of this work, this will not be discussed any further.

E. Communication Link

The communication link, which is supervised by the **COM Receiver** and **COM Transmitter** threads, is quite important for this type of application. The communication link must allow fast distribution of much data. For instance, if 100 active filter reference points (float representation with 32 bits) are distributed at least every 20ms, this means $\frac{100 \cdot 32\text{b}}{0.02\text{s}} = 160\text{kb/s}$, or 20kB/s. Even though this transmission rate does not include additional message overhead, which is protocol dependent, an RS-232 serial communication link is excluded. An alternative serial link that can be used is RS-485, but a more appropriate solution that has the needed transmission rate, and at the same time offers flexibility and N-to-N connection, is Ethernet with protocols such as TCP or UDP. Unlike the TCP protocol, in which can guarantee that the messages arrive their destination as long as the communication link is alive, UDP is a *best effort* protocol, where the arrival of important messages, such as the *COM Schedule MPC* and *COM Controls* messages in Fig. 5 and 6, must be confirmed by separate reply messages (for this example the *COM Received Scheduling* and *COM Controls Received* messages). UDP is widely used in the industry for communication between distributed control nodes and systems, and plays an important role in cloud based IIoT middleware without centralized servers. However, both TCP and UPD will have overheads due to collisions and back-off of the Ethernet protocols in access to the transmission medium. This problem can be minimized by using switched Ethernet, or totally eliminated by using a TDMA (Time-Division Multiple Access) based real-time communication layer, such as RTNET, above the UDP layer.

F. Implementation Aspects

In this work, the MPC framework in Fig. 5 and the simulator in Fig. 6 were implemented in C++ with libraries from Qt [43], [44] for event management, with Linux (Ubuntu 16.04 with low-latency kernel patch) as target platform. The kernel used, 4.4.0-X, does not feature a real-time (RT) scheduler, as this was removed from the official Ubuntu distribution after kernel version 2.6.X. Hence, the non-RT scheduler CFS (Completely Fair Scheduler), which is part of the official kernel release from Ubuntu, was used in this work to satisfy the requirements of realizing the application on a standard hardware and software platform. The threads are implemented using the thread abstraction layer in Qt, and given a high priority `(QThread::HighPriority)`. External priority grouping, i.e. task priority and scheduling policy directly from the kernel, was not adopted in this work. The timers used to log the latencies are implemented as high resolution monotonic timers in the Qt framework (`QtElapsedTimer` class). The communication link is realized using UDP over Ethernet. The message protocol is designed using JSON, which offers great properties in the design and prototyping of communication structure. JSON is promoted as a low-overhead alternative to XML, with great debugging and logging properties due to human-readable text to transmit data objects consisting of attribute-value pairs. JSON messages are also easily parsed and processed, and corrupt messages can easily be detected due to the JSON message identifiers, which encloses one message structure.

IV. Hardware in the Loop Test

Hardware in the Loop (HIL) simulation tests are conducted with two computers connected to a local Ethernet network, see Fig. 7. One of the computers acts as the main controller running the MPC and its framework, while the other computer runs the simulator, as discussed in the previous section. The specifications of the two computers are listed in Table II, and the HIL setup is showcased in Fig. 7. As showcased in Fig. 7 the two controllers are connected to a local dedicated Ethernet and communicates through a gigabit switch. The architecture for the higher-level control, as described in Fig. 5, is implemented on the main controller, where the software runs as a common process in the Linux operation system. The lower-level control is mimiced by a simulator, with architecture described in Fig. 6, and runs as a common process in the simulator controller's Linux operation system. For generality, the computers used in this HIL-setup are common desktop computers.

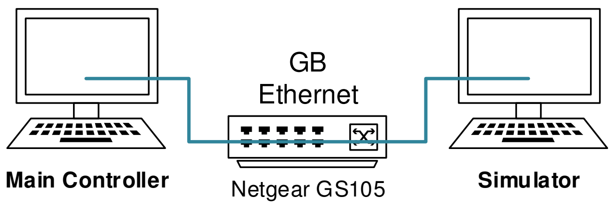
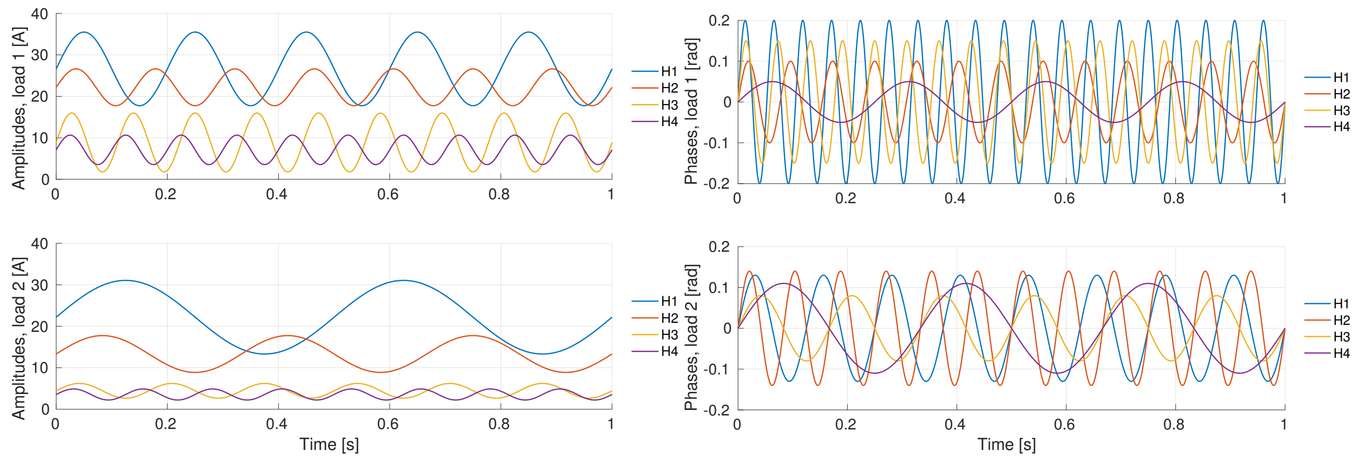


Fig. 7. Hardware in loop setup: Two computers, one acting as the main controller running the MPC and its framework and the other running the simulator, connected to a local Gb Ethernet network.

TABLE II
Details of Main Controller and Simulator used in HIL test.

Main Controller	Simulator
Lenovo Thinkpad T440s 8GB memory (DDR3, 1600MHz) Intel® Core™ i7-4600U CPU @ 2.10GHz × 4 Graphics: Intel® Haswell Mobile 64-bit Ubuntu 16.04 LTS Low latency kernel: 4.4.0-22 x86_64	Lenovo Thinkpad P50 24GB memory (DDR4, 2133MHz) Intel® Core™ i7-6820HQ CPU @ 2.70GHz × 8 Graphics: Quadro M2000M/PCIe/SSE2 64-bit Ubuntu 16.04 LTS Low latency kernel: 4.4.0-22 x86_64

The parameters in the MPC's internal model are, according to Fig. 2, listed in Table III. As can be seen, the APF's power rating is set to 10% of the generator rating, which is a relative small filter. With a voltage level of 690V this corresponds to current limits of $i_{\max} = -i_{\min} = \sqrt{2} \cdot \frac{50\text{kVA}}{690\text{V}} \approx 102.48\text{A}$ (peak current) in (3). Furthermore, the fundamental frequency is set to 50Hz, and the harmonics to be mitigated are the first four significant harmonic orders in a 6-pulse rectifier, i.e. 5th, 7th, 11th and 13th. The other electrical parameters are adopted from [24], [25].



(a) FFT amplitudes. (b) FFT phases.

Fig. 8. Simulated FFT amplitudes and phases for the four harmonic orders (H) to be mitigated: H1=5th, H2=7th, H3=11th and H4=13th.

TABLE III
POWER SYSTEM MODEL PARAMETERS, ACCORDING TO FIG. 2.

Parameter	Value
RMS voltage	690V
Generator ratings	500kVA
APF rating	50kVA
L_{G1}, L_{G2}	30.309mH
R_{G1}, R_{G2}	9.512mΩ
L_{MB}	60.619µF
R_{MB}	1.904mΩ
C_1, C_2	2µF
Fundamental frequency (f)	50Hz
Harmonic orders to be mitigated	5th, 7th, 11th, 13th

The simulator, with architecture showcased in Fig. 6, is responsible for generating the measurements the MPC is using for optimal harmonic conditioning. The generated FFT amplitudes and phases, for the load currents in Fig. 2, are shown in Fig. 8. The measurements are designed to provide a dynamic spectra of the harmonics to be mitigated, with the intention to provide both low and high levels of harmonic pollution that challenges the MPC in different ways that might affect the MPC's computational costs. Hence, the measurements are not extracted from a physical (or simulated) power system, but designed to stress test the MPC and challenge the the system architecture and the MPC's real-time properties. The measurements are generated as sine waves, with amplitudes and phases for load 1 and load 2 in Fig. 2 given as

$$i_h = \left(A_{i,s}^h + A_{i,b}^h \cdot \sin\left(2\pi \cdot \frac{1}{T_i^h} \cdot t\right)\right) \cdot I_G$$
$$\phi_h = A_\phi^h \cdot \sin\left(2\pi \cdot \frac{1}{T_\phi^h} \cdot t\right). \tag{7}$$

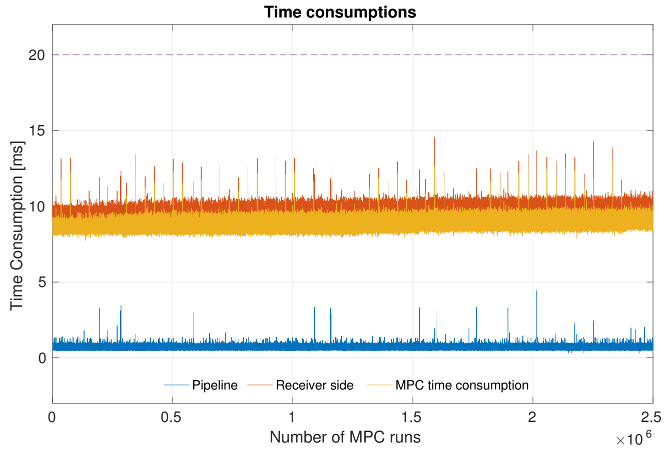
$A_{i,s}^h$ is the amplitude setpoint, $A_{i,b}^h$ is the amplitude band, $\frac{1}{T_i^h}$ is the amplitude frequency, A_ϕ^h is the phase amplitude and $\frac{1}{T_\phi^h}$ is the phase frequency for each harmonic order h. $I_G = \frac{\sqrt{3}}{\sqrt{2}} \cdot \frac{500\text{kVA}}{690\text{V}}$ is the rated generator current. Table IV lists the parameters used in (7) to generate the measurements in Fig. 8.

TABLE IV
AMPLITUDE AND PHASE MEASUREMENT PARAMETERS USED TO GENERATE THE MEASUREMENTS IN FIG. 8. LOAD 1 AND 2 ACCORDING TO FIG. 2.

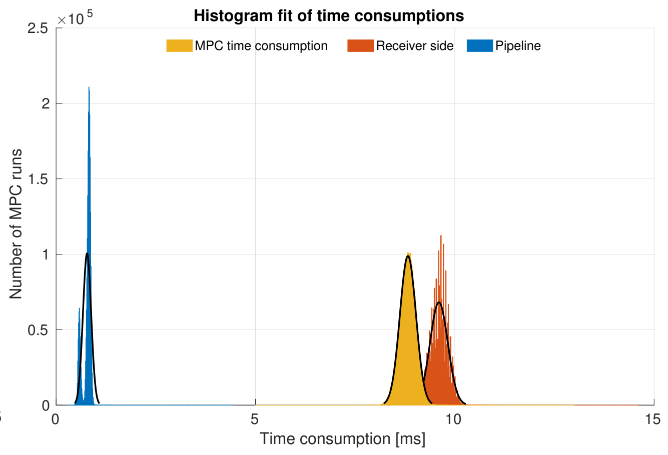
Harmonic order (load 1)	$\mathbf{A_{i,s}^h}$	$\mathbf{A_{i,b}^h}$	$\mathbf{T_i^h}$	$\mathbf{A_\phi^h}$	$\mathbf{T_\phi^h}$
5th	0.03	0.01	5	0.2	19
7th	0.025	0.005	7	0.1	13
11th	0.01	0.008	9	0.15	17
13th	0.008	0.004	10	0.05	4
Harmonic order (load 2)	$\mathbf{A_{i,s}^h}$	$\mathbf{A_{i,b}^h}$	$\mathbf{T_i^h}$	$\mathbf{A_\phi^h}$	$\mathbf{T_\phi^h}$
5th	0.025	0.015	2	0.13	8
7th	0.01	0.005	3	0.14	12
11th	0.005	0.002	6	0.08	6
13th	0.004	0.0015	8	0.11	3

As the simulator does not provide closed-loop control, due to the fact that the HIL test is designed to test the MPC's real-time properties and not the harmonic mitigation capabilities (which has been thoroughly explored in [24], [25]), the state trajectories (voltages and currents) from the previous run of the MPC is used to initialize the states before a new run. Hence, the MPC should be in a worst-case situation regarding convergence and execution time (*cold-start conditions*), compared to an industrial situation with closed-loop measurements and model re-initialization based on results and values from the previous cycle.

A HIL test was performed with 2.5 million MPC runs, and the results are shown in Fig. 9 and summarized in Table V. Fig. 9a shows the MPC's time consumption, which was calculated by the **MPC Execution Timer** thread in Fig. 5, and the time between scheduling an MPC run and receiving the control vector (indicated in the figure by *Receiver side*) calculated by the **Control Message Timer** thread in Fig. 6. The difference between these timers represents the pipeline in the figure, including transmission delays and framework delays. The additional latency experienced by the receiver side,



(a) Time consumptions of the HIL test.



(b) Histogram fit of time consumptions from the HIL test.

Fig. 9. Results of the HIL test: Time consumption of the MPC, of the pipeline and the time between scheduling an MPC run and receiving the control vector from the simulator side.

which is shown as spikes in Fig. 9a for the pipeline, might be related to other high priority background processes running on the controllers, additional latency introduced by the framework and/or by the switch in the HIL setup in Fig. 7. In Fig. 9a one can see a pattern with slightly increasing time consumption. This is due to logging the results (time consumption measurements) to files, where the log files become quite large. Hence, the time spent to opening, writing, and closing the log files after each MPC run increases with increasing log file sizes.

Fig. 9b portrays a histogram of the time consumptions in Fig. 9a. As can be seen in the figure, the time consumption of the MPC, the receiver side and the pipeline all give single characteristic peaks in the histogram, which represent consistent time consumption within stochastic distributions. Table V summarizes Fig. 9, and as the receiver side's maximum time consumption is below 20ms, there is no need to reuse any control vectors. Hence, the results from this HIL simulations indicate that the MPC, with architecture shown in 5, is able to fulfill the real-time requirements, i.e. deliver a new control vector every 20ms, for the proposed optimization-based system-level harmonic mitigation application. This validates the main result in this paper, that the proposed real-time MPC architecture fulfills the requirements to resource usage and real-time performance.

TABLE V
HIL RESULTS SUMMARY OF FIG. 9.

Time measurement	Avg. [ms]	Max [ms]	Min [ms]	Histogram Peak [ms]
MPC	8.825	12.990	4.991	8.823
Receiver side	9.602	14.584	8.478	9.605
Pipeline	0.777	4.445	0.309	0.775

Fig. 10 shows the resource use of the main controller during the HIL test, sampled at 1Hz. The upper plot shows the percentage of CPU time used by the application, the plot in the middle shows the physical memory currently used by the application (RSS), while the lower plot shows the total memory the application has allocated for its execution (VSZ). From the plots it is evident that the application running on the main controller is quite steady in its resource usage. The % CPU time settles around 52.2%. The RSS and VSZ are constant throughout the HIL test, 12.2MB and 373.6MB respectively.

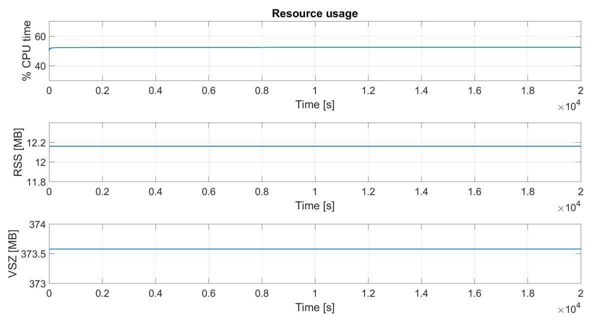


Fig. 10. The main controller's resource usage during HIL test, sampled at 1Hz. From above: i) % CPU time, which is the CPU time used by the application divided by the time the application has been running, ii) RSS (resident set size) is the non-swapped physical memory (RAM) that the application currently is using, iii) VSZ (virtual set size) is the memory size assigned to the application and represents how much memory the application has available for its execution usage (allocated address space).

V. CONCLUSION

MPC applications for systems with fast dynamics are challenging, and put stringent requirements on the implementation, which relate to the design and the internal mechanisms of the MPC as well as its framework and middleware connecting the MPC application to the physical system. In this work a novel MPC implementation for optimal harmonic mitigation, that is based on a standard hardware and software platform, has been presented, and the system design and architecture for obtaining the necessary (soft) real-time properties have been discussed and implemented. To obtain the required real-time properties, the design of the MPC has been centered around the repetitive control philosophy, which enables the utilization of larger parts of the calculated control vector compared to conventional MPC designs, which uses only one or few steps from the obtained control vector. The proposed system architecture uses both cyclic and event-based threads with the aim of minimizing the resource usage. To mimic a practical implementation of this architecture, a simulator was designed to verify the MPC's and the framework's real-time properties, for which a HIL test using two computers connected to a dedicated Ethernet link was conducted. The results indicate that the proposed system architecture is able to meet the system's soft real-time demands with consistent and relatively low resource usage. The main results can be listed as follows:

- A scenario for system-level harmonic mitigation suitable to a marine vessel's electric system utilizing MPC and multi-layered distributed control has been explored, and a HIL test has been conducted to quantify computational loads and resource usage
- During the HIL test the MPC never failed to deliver a new control vector within deadline
- The resource usage on the higher-level controller running the MPC experienced a stable computational load; CPU time settles around 52.2%, and RAM usage around 12.2MB

By definition, as the MPC never failed to deliver a new control vector during the HIL test, the proposed system architecture and conceptual implementation was able to meet hard real-time requirements as well, although this can not be guaranteed for other potential simulation scenarios. Even though the results indicate that the application, with the proposed architecture, exhibits the required real-time properties, this work is only centered around the higher-level control layer in Fig. 3. Hence, future work has to be conducted for realizing the lower-level control layer, thus enabling possibilities for experimental tests where the complete system is considered. Potential applications for this control architecture are envisioned in the marine vessel power system, where operation and configuration of the power system can change demanding control flexibility that can be met by the scheme presented in this paper.

Acknowledgment

This work has been carried out at the Centre for Autonomous Marine Operations and Systems (NTNU-AMOS) whose main sponsor is The Research Council of Norway. The work was supported by Ulstein Power & Control AS and The Research Council of Norway, Project number 241205, and Ulstein Blue Ctrl AS.

References

[1] J. B. Rawlings and D. Q. Mayne, *Model predictive control: Theory and design*. Nob Hill Pub., 2009.

[2] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Survey paper constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, pp. 789–814, 2000.

[3] L. Biegler, *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*, ser. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2010.

[4] T. A. Johansen, "Toward dependable embedded model predictive control," *IEEE Systems Journal*, vol. 11, pp. 1208–1219, 2017.

[5] H. Peyrl, H. J. Ferreau, and D. Kouzoupis, "A hybrid hardware implementation for nonlinear model predictive control," *IFAC-PapersOnLine*, vol. 48, no. 23, pp. 87 – 93, 2015, 5th IFAC Conference on Nonlinear Model Predictive Control NMPC, Seville, Spain, 17-20 September 2015.

[6] C. Bordons and C. Montero, "Basic Principles of MPC for Power Converters: Bridging the Gap Between Theory and Practice," *IEEE Industrial Electronics Magazine*, vol. 9, no. 3, pp. 31–43, Sept 2015.

[7] J. Rodriguez, M. P. Kazmierkowski, J. R. Espinoza, P. Zanchetta, H. Abu-Rub, H. A. Young, and C. A. Rojas, "State of the art of finite control set model predictive control in power electronics," *IEEE Trans. Ind. Informat.*, vol. 9, no. 2, pp. 1003–1016, 2013.

[8] S. Kouro, P. Cortés, R. Vargas, U. Ammann, and J. Rodríguez, "Model predictive controla simple and powerful method to control power converters," *IEEE Trans. Ind. Electron.*, vol. 56, no. 6, pp. 1826–1838, 2009.

[9] M. Vatani, B. Bahrani, M. Saeedifard, and M. Hovd, "Indirect finite control set model predictive control of modular multilevel converters," *IEEE Trans. Smart Grid*, vol. 6, no. 3, pp. 1520–1529, May 2015.

[10] P. Cortés, M. P. Kazmierkowski, R. M. Kennel, D. E. Quevedo, and J. Rodríguez, "Predictive control in power electronics and drives," *IEEE Trans. Ind. Electron.*, vol. 55, no. 12, pp. 4312–4324, 2008.

[11] P. Cortés, G. Ortiz, J. I. Yuz, J. Rodríguez, S. Vazquez, and L. G. Franquelo, "Model predictive control of an inverter with output filter for ups applications," *IEEE Trans. Ind. Electron.*, vol. 56, no. 6, pp. 1875–1883, 2009.

[12] P. Cortes, J. Rodríguez, P. Antoniewicz, and M. Kazmierkowski, "Direct power control of an afe using predictive control," *IEEE Trans. Power Electron.*, vol. 23, no. 5, pp. 2516–2523, 2008.

[13] S. Richter, S. Marithoz, and M. Morari, "High-speed online MPC based on a fast gradient method applied to power converter control," in *Proceedings of the 2010 American Control Conference*, June 2010, pp. 4737–4743.

[14] M. Rivera, A. Wilson, C. A. Rojas, J. Rodriguez, J. R. Espinoza, P. W. Wheeler, and L. Empringham, "A Comparative Assessment of Model Predictive Current Control and Space Vector Modulation in a Direct Matrix Converter," *IEEE Trans. Ind. Electron.*, vol. 60, no. 2, pp. 578–588, Feb 2013.

[15] L. Jin, R. Kumar, and N. Elia, "Model predictive control-based real-time power system protection schemes," *IEEE Trans. Power Syst.*, vol. 25, no. 2, pp. 988–998, May 2010.

[16] T. V. Vu, D. Gonsoulin, F. Diaz, C. S. Edrington, and T. El-Mezyani, "Predictive control for energy management in ship power systems under high-power ramp rate loads," *IEEE Trans. Power Conversion*, vol. 32, pp. 788 – 797, 2017.

[17] T. I. Bøand T. A. Johansen, "Battery power smoothing control in a marine electric power plant using nonlinear model predictive control," *IEEE Transactions on Control Systems Technology*, vol. 25, pp. 1449–1456, 2017.

[18] A. M. Ersdal, D. Fabozzi, L. Imsland, and N. F. Thornhill, "Model predictive control for power system frequency control taking into account imbalance uncertainty," in *Proc. IFAC World Congr*, vol. 19, 2014, pp. 981–986.

[19] E. Mayhorn, K. Kalsi, J. Lian, and M. Elizondo, "Model Predictive Control-Based Optimal Coordination of Distributed Energy Resources," in *2013 46th Hawaii International Conference on System Sciences (HICSS)*, Jan 2013, pp. 2237–2244.

[20] E. D. Mehleri, H. Sarimveis, L. G. Papageorgiou, and N. C. Markatos, "Model predictive control of distributed energy resources," in *2012 20th Mediterranean Conference on Control Automation (MED)*, July 2012, pp. 672–678.

[21] E. Skjong, M. Ochoa-Gimenez, M. Molinas, and T. A. Johansen, "Management of harmonic propagation in a marine vessel by use of optimization," in *2015 IEEE Transportation Electrification Conference and Expo (ITEC)*. IEEE, 2015, pp. 1–8.

[22] E. Skjong, M. Molinas, and T. A. Johansen, "Optimized current reference generation for system-level harmonic mitigation in a diesel-electric ship using non-linear model predictive control," in *2015 IEEE International Conference on Industrial Technology (ICIT)*. IEEE Conference Publications, 2015, pp. 2314–2321.

[23] E. Skjong, M. Molinas, T. A. Johansen, and R. Volden, "Shaping the current waveform of an active filter for optimized system level harmonic conditioning," in *Proceedings of the 1st International Conference on Vehicle Technology and Intelligent Transport Systems*, 2015, pp. 98–106.

[24] E. Skjong, J. A. Suul, A. Rygg, M. Molinas, and T. A. Johansen, "System-wide harmonic mitigation in a diesel electric ship by model predictive control," *IEEE Trans. Ind. Electron.*, vol. 63, no. 7, pp. 4008–4019, July 2016.
[25] E. Skjong, J. A. Suul, M. Molinas, and T. A. Johansen, "Optimal compensation of harmonic propagation in a multi-bus microgrid," in *International Conference on Renewable Energies and Power Quality (ICREPQ'16), Renewable Energy and Power Quality Journal (RE&PQJ)*, 2016, pp. 1–6.
[26] E. Skjong, "Optimization-based control in shipboard electric systems," PhD thesis, Norwegian University of Science and Technology (NTNU), 2017, 2017:139.
[27] A. Mohammadi, M. Mehrtash, and A. Kargarian, "Diagonal quadratic approximation for decentralized collaborative tso+ dso optimal power flow," *IEEE Transactions on Smart Grid*, 2018.
[28] M. H. Amini, S. Bahrami, F. Kamyab, S. Mishra, R. Jaddivada, K. Boroojeni, P. Weng, and Y. Xu, "Chapter 6 - decomposition methods for distributed optimal power flow: Panorama and case studies of the dc model," in *Classical and Recent Aspects of Power System Optimization*, A. F. Zobaa, S. H. A. Aleem, and A. Y. Abdelaziz, Eds. Academic Press, 2018, pp. 137 – 155.
[29] W. M. Grady, M. J. Samotyj, and A. H. Noyola, "Survey of active power line conditioning methodologies," *Power Delivery, IEEE Transactions on*, vol. 5, no. 3, pp. 1536–1542, 1990.
[30] W. Grady, M. Samotyj, and A. Noyola, "Minimizing network harmonic voltage distortion with an active power line conditioner," *Power Delivery, IEEE Transactions on*, vol. 6, no. 4, pp. 1690–1697, 1991.
[31] H. Akagi, E. Watanabe, and M. Aredes, *Instantaneous Power Theory and Applications to Power Conditioning*, ser. IEEE Press Series on Power Engineering. Wiley, 2007.
[32] N. A. Al-Emadi, C. Buccella, C. Cecati, and H. A. Khalid, "A novel dstatcom with 5-level chb architecture and selective harmonic mitigation algorithm," *Electric Power Systems Research*, vol. 130, pp. 251–258, 2016.
[33] A. N. Venkat, I. A. Hiskens, J. B. Rawlings, and S. J. Wright, "Distributed mpc strategies with application to power system automatic generation control," *IEEE Trans. Control Syst. Technol*, vol. 16, no. 6, pp. 1192–1206, Nov 2008.
[34] A. Burns and A. J. Wellings, *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. Pearson Education, 2001.
[35] J. Andersson, "A General-Purpose Software Framework for Dynamic Optimization," PhD thesis, Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, October 2013.
[36] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.
[37] ——, "An Auto-Generated Real-Time Iteration Algorithm for Nonlinear MPC in the Microsecond Range," *Automatica*, vol. 47, no. 10, pp. 2279–2285, 2011.
[38] E. Hairer and G. Wanner, "Stiff differential equations solved by radau methods," *Journal of Computational and Applied Mathematics*, vol. 111, no. 12, pp. 93 – 111, 1999.
[39] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpoases: a parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.
[40] A. Domahidi and J. Jerez, "FORCES Professional," embotech GmbH (http://embotech.com/FORCES-Pro), Jul. 2014.
[41] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.
[42] R. G. Brown and P. Y. C. Hwang, *Introduction to random signals and applied kalman filtering: with MATLAB exercises and solutions; 3rd ed.* New York, NY: Wiley, 1997.
[43] M. Summerfield, *Advanced Qt Programming: Creating Great Software with C++ and QT 4*, ser. Prentice Hall open source software development series. Addison-Wesley, 2011.
[44] S. Huang, *Qt 5 Blueprints*, ser. Community experience distilled. Packt Publishing, 2015.

Espen Skjong received his MSc degree in Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 2014, specializing in model predictive control (MPC) for autonomous control of UAVs. He is currently employed in Ulstein Power & Control AS (Ålesund, Norway) as an industrial PhD candidate. His research topic is optimization in power management systems for marine vessels. His industrial PhD fellowship is within the Center of Excellence on Autonomous Marine Operations and Systems (AMOS) at NTNU.

Tor Arne Johansen (M'98, SM'01) received the MSc degree in 1989 and the PhD degree in 1994, both in electrical and computer engineering, from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway. From 1995 to 1997, he worked at SINTEF Information and Communication Technology as a researcher before he was appointed Associated Professor at NTNU in Trondheim in 1997 and Professor in 2001. He has published several hundred articles in the areas of control, estimation and optimization with applications in the marine, automotive, biomedical and process industries. In 2002 Johansen co-founded the company Marine Cybernetics AS where he was Vice President until 2008. Prof. Johansen received the 2006 Arch T. Colwell Merit Award of the SAE, and is currently a principal researcher within the Center of Excellence on Autonomous Marine Operations and Systems (AMOS) and director of the Unmanned Aerial Vehicle Laboratory at NTNU.

Marta Molinas (M'94) received the Diploma degree in electromechanical engineering from the National University of Asuncion, Asuncion, Paraguay, in 1992; the Master of Engineering degree from Ryukyu University, Japan, in 1997; and the Doctor of Engineering degree from the Tokyo Institute of Technology, Tokyo, Japan, in 2000. She was a Guest Researcher with the University of Padova, Padova, Italy, during 1998. From 2004 to 2007, she was a Postdoctoral Researcher with the Norwegian University of Science and Technology (NTNU) and from 2008-2014 she has been professor at the Department of Electric Power Engineering at the same university. From 2008 to 2009, she was a Japan Society for the Promotion of Science (JSPS) Research Fellow with the Energy Technology Research Institute, National Institute of Advanced Industrial Science and Technology, Tsukuba, Japan. In 2014, she was Visiting Professor at Columbia University and Invited Fellow by the Kingdom of Bhutan working with renewable energy microgrids for developing regions. She is currently Professor at the Department of Engineering Cybernetics, NTNU. Her research interests include stability of power electronics systems, harmonics, oscillatory phenomena, and non-stationary signals from the human and the machine. Dr. Molinas has been an AdCom Member of the IEEE Power Electronics Society. She is Associate Editor and Reviewer for *IEEE Transactions on Power Electronics* and *PELS Letters*.