Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Cloud-based model predictive control with variable horizon [⋆]

Per Skarin [*,**] **Johan Eker** [*,**] **Karl-Erik Årzén** [**]

[*] *Ericsson Research*
[**] *Department of Automatic Control, Lund University*

Abstract: A novel method using the cloud to implement a variable horizon model predictive controller is presented. In case of sudden long delays and downtime, a graceful degradation is used. Robust, best effort strategies allow industrial grade use of the powerful, efficient, and quickly improving cloud ecosystems. The variable horizon strategy finds use in, for example, non-linear control problems, and the proposed method can be generalized to implement robust and scalable controllers that benefit from cloud technology. We show results from two horizon selection strategies, service degradation and connectivity issues.

Keywords: Industry automation, Predictive control, Optimal control, Systems concept, System architectures, Networks, Adaptive systems, Parallelism

1. INTRODUCTION

The cloud has come to represent the availability of computing infrastructure and services in resemblance with a public utility such as electricity or water. For many, it is an ubiquitous everyday necessity, accessible at all times. This ecosystem of web services and compute and storage resources, available over remote networks and paid per usage, gained momentum about a decade ago (Armbrust et al. (2010)). Since then the cloud has revolutionized the software industry and is rapidly becoming an intrinsic part of our infrastructure. The cloud supports the design and operation of highly automated and elastic services that can scale to meet demands.

Recent advances in cloud technology include concepts such as edge, fog and osmotic computing (Yousefpour et al. (2019); Villari et al. (2016)). An edge node can be a server on the factory floor which is connected to a centralized cloud or it can be a radio base station that offers compute capabilities. The edge node is integrated as part of the cloud and provides the advantages of lower latency and reduced dependencies on remote service, e.g. in the case of connectivity issues the edge node can continue to provide the cloud service, but at reduced capacity. Combined with rich sensor environments and the edge, cloud technology is gaining traction in traditional industries including logistics, transport, factory automation, manufacturing, and even for critical closed loop control systems.

In this paper, we present a Model Predictive Control (MPC) architecture that is designed for cloud deployment and makes, from an MPC perspective, novel use of the "infinite" compute capacity of the cloud while providing robustness to loss of connectivity. The assumption is that the controller requires the cloud to operate, possibly due to computational restrictions on the device or the need to access data that is not locally available. In the approach, the prediction horizon is decided dynamically; and stability and feasibility are enforced in the optimization. Robustness towards loss of connectivity and extensive computational delays is obtained by combining the MPC with a local controller. We define two modes of operation. In assisted mode, the controller uses the resources of the cloud to solve the optimization problem over a large set of horizons in parallel. In local mode, a simpler controller stabilizes the system and provides a basic degree of performance. The resulting system is convenient to implement and has a built-in capacity to scale with the problem. The approach naturally extends to edge clouds, which combine the compute capacity of a centralized cloud with the low latency access of local nodes. It also puts in perspective the use of flexible, cost effective, best effort control systems as opposed to traditional, costly and static systems. Reliability of best effort systems could provide a viable cloud alternative when upgrading industrial processes and the study of these systems is highly relevant for Industry 4.0.

The outline of the paper is as follows. Section 2.1 to 2.6 describe the problem and its solution. Section 2.7 and 2.8 briefly presents the latency and plant models. Section 2.9 outlines the design steps necessary to implement the controller. In Section 3.1 the cloud service is benchmarked while Section 3.2 shows simulated results based on these observations. Finally, Section 4 concludes the paper.

1.1 Related Work

There is a lack of work on control design that considers the possibilities and implications of the elastic, and virtually unlimited compute resources of the cloud. Previous work on cloud based control systems (Pelle et al. (2019); Mubeen et al. (2017); Givehchi et al. (2014); Hegazy and Hefeeda (2015); Heilig et al. (2015)) focus on a high level of abstraction and replacing existing designs with cloud coun-

[⋆] This work has been partially funded by the Wallenberg AI, Autonomous Systems and Software Program (WASP), the ELLIIT strategic research area on IT and mobile communications, and the Nordforsk university hub on Industrial IoT (HI2OT).

Copyright lies with the authors

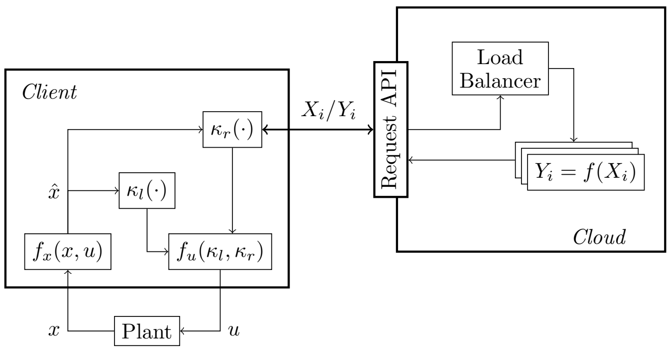

Fig. 1. The system setup where the control law is divided into a local part $\kappa_l(\cdot)$ running in the client and a remote part $\kappa_r(\cdot)$ that is off-loaded to the cloud. The remote part $\kappa_r(\cdot)$ is realized by running multiple parallel instances in the cloud.

terparts. Work on distributed MPC over networks include collaborative agent systems, distributed optimizations, hierarchical control and optimal control in the presence of delay (for instance Zheng et al. (2016); Christofides et al. (2013); Alessio and Bemporad (2007); Lu et al. (2014); Scattolini (2009); Stewart et al. (2010); Camponogara et al. (2002)), but work that considers the cloud as a separate paradigm is largely missing. One example which does make an explicit case for MPC in the cloud is Heilig et al. (2015) which presents an application for intelligent transport systems. A multi-agent system comprising many MPCs executing in the cloud is proposed conceptually as a flexible method to handle heavy computations and large amounts of data, but there is no consideration as to how the control structure itself can be improved using the cloud.

Variable horizon MPC as introduced for nonlinear systems by Michalska and Mayne (1993) adds minimization of the horizon to the optimization problem in order to reach a target set in the shortest possible amount of steps. In our work, we utilize the cloud to evaluate a set of horizons and define a separate operation which selects the horizon, or more generally an MPC definition, from the available responses when the output is applied. A different reason for variable horizons (or multiple horizon) MPC is used in Park et al. (2015) where the optimal horizon varies and is determined by evaluating the gait of a quadruped as it prepares for jumping an obstacle.

2. METHOD

2.1 Targeted system

The targeted system is illustrated in Figure 1. The system is composed of a plant controlled by our proposed *assisted* controller, here operated by the local device (denoted *Client* in the figure). The client continuously executes a local on-board controller $\kappa_l(\cdot)$ and a remote controller $\kappa_r(\cdot)$, which performs most of its work in the cloud. Using a cloud service it processes several potential solutions in parallel, filters the responses and forwards the best selection to a function $f_u(\kappa_l,\kappa_r)$ that merges the output of the two controllers. The function $f_x(x,u)$ predicts the plant state a number of sampling periods into the future. This is used to set a deadline for the remote controller, i.e., any response from the remote controller must arrive to the local client before the deadline.

A key element of the design is the use of the cloud to execute multiple instances of a model predictive controller. In Figure 1 this is represented by the stacked boxes labeled $Y_i = f(X_i)$. In relation to Section 2.2, X_i is the input pair (x_0, N_i), while Y_i is a predicted state trajectory and a control input vector $(\mathbf{x}_i, \mathbf{u}_i)$ Cloud services specifically designed for executing functions are referred to as Function-as-a-Service (FaaS). These services provide the user with a convenient way to implement and execute single operations in the cloud. Such services automatically scale to handle load, allowing a large number of requests at a given time.

2.2 Controller

The control strategy uses the cloud to implement a variable prediction horizon MPC. An MPC works by repeatedly solving a numerical optimization problem to find an optimal control signal. The optimization problem can take considerable time to evaluate. The control strategy must take this delay into account in combination with the communication latency between the client and the cloud. The load of the network and in the cloud service varies over time and there is an inherent risk of loosing connectivity. When a request is made to the cloud it is assumed that it can fail, or that it does not provide the service that was expected and aimed for. Altogether this requires a local backup strategy. For the purpose of the work herein it is assumed that the local device is limited to a degree such that it is not possible to execute an MPC controller locally, not even an explicit MPC (Bemporad et al. (2002)). Instead, a Linear Quadratic (LQ) regulator is used, derived from the specifications of the MPC.

The selection of the horizon in the MPC can be non-trivial. A short horizon can lead to infeasibility while a long horizon can take too long to evaluate. A scalable controller should not be bound by a predetermined horizon. The cloud allows us to run the optimization over many horizons and select the best alternative that provides a response within the deadline. To ensure stability and recursive feasibility of the responses the MPC uses a terminal cost and terminal state constraints, referred to as the terminal set Rawlings and Mayne (2009). The terminal requirements force responses which ensure constraint satisfaction in future states and, as the system gets close to the setpoint, its response approaches that of the local LQ controller. When the state is inside the terminal set the plant can be operated using the local controller.

The function that executes in the cloud is a MPC as defined in Equation (1).

$$\begin{aligned} \min_{\mathbf{u}} f(x_0, N) = &\sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i + x_N^T P x_N \\ \text{subject to } \quad & x_{i+1} = A x_i + B u_i \\ & G x_i \le g,\ H u_i \le h,\ x_N \in \mathcal{X}_f \end{aligned} \tag{1}$$

This is a discrete time, constrained controller with a quadratic cost function defined by the matrices Q and R, applied to the state and control signal respectively. N is the horizon. Equation (1) includes a terminal constraint set $\mathcal{X}_f$ and a terminal cost P. A local control law, $\kappa_l(\cdot)$, i.e., the LQ regulator, is defined for which $\mathcal{X}_f$ is a positive invariant set. That is, if the starting state is in $\mathcal{X}_f$ then the system, under control of $\kappa_l(\cdot)$, will remain in $\mathcal{X}_f$. Formally, $f(x, \kappa_l(x)) \in \mathcal{X}_f, \forall x \in \mathcal{X}_f$, where $f(x, u)$ defines the state evolution of the system. It is also required that all states in $\mathcal{X}_f$ satisfy the state constraints, that $\mathcal{X}_f$ is a closed set, and that $\mathcal{X}_f$ contains the origin in its interior. Finally, the input constraints must also be satisfied, $\kappa_l(x) \in \mathbb{U}, \forall x \in \mathcal{X}_f$, where $\mathbb{U}$ are the control input constraints. Since the selected MPC is a linear quadratic problem we derive from it a local linear state feedback controller $\kappa_l(x) = Kx$ and find a terminal set $\mathcal{X}_f$ for which the conditions above hold. This local LQ regulator is the solution to (1) with an infinite horizon and no constraints. The constraint satisfaction is ensured through the terminal set $\mathcal{X}_f$. The asymptotic and unconstrained solution to the LQ regulator problem provides a cost P which is a local Lyapunov function for $\kappa_l(\cdot)$. Inserting P and $\mathcal{X}_f$ into (1) ensures a stabilizing state feedback controller. It is assumed that the system state is fully observable.

To create a time frame for the optimization a dead-time is forced into the controller by generating a state approximation $\hat{x}_{k+1}$ using the available system model

$$\hat{x}_{k+1} = Ax_k + Bu_k \tag{2}$$

This state approximation is inserted into the optimization at time k and the resulting controller output is applied at the next sampling instant $k + 1$, i.e., the control signal is delayed until the next sampling instant. Note that this delay is not part of the MPC problem and therefore not included in (1). The delay of one sample is the time frame available for the optimizations in the cloud. Any result returned later will be discarded.

We are now ready to define the cloud assisted, elastic controller. First, if we can guarantee a result from the optimization in (1) for all samples then the local control $\kappa_l(\cdot)$ is not explicitly needed. It is introduced implicitly into the MPC by adding its cost P as the terminal cost and enforcing the terminal set. An important property of our cloud controller is that the availability of the MPC output is not guaranteed due to connectivity, latency or feasibility issues. Therefore there must be a local device controller available to ensure uninterrupted control of the plant. For this purpose the stabilizing LQ regulator from the MPC formulation is used to provide the baseline performance of the cloud assisted controller. In a nominal system, this controller is optimal and does not violate constraints, as long as the state error is inside the terminal set. To ensure that the local controller is inside $\mathcal{X}_f$ it is necessary to shape the setpoint, something which is returned to in the description of the *local mode*.

Next, we consider the MPC *functions* that are requested from the cloud and how we define admissible responses. Let $\mathcal{M}$ be the set of all applicable models and $\mathcal{C}$ the set of all applicable constraints. N is as usual the horizon. Assuming that the cost function is the same then Φ is the set of all possible model predictive controllers for our problem.

$$\mathbf{\Phi} = \left\{\kappa_{N,m,c} | N \in \mathbb{Z}^+, m \in \mathcal{M}, c \in \mathcal{C}\right\} \tag{3}$$

Given the theoretical idealization of the cloud as a place of infinite computational resources all controllers in Φ could be evaluated with every sample. In practice, however, the controller must select a subset of MPCs, $\phi_k \subseteq \Phi$, to execute in the cloud. The set ϕ_k is referred to as the request set. The set of admissible results ψ_k at time k are all the MPC responses that are returned within the deadline. It is assumed that the solver only returns results that are feasible, i.e. that the solver either aborts in case of infeasibility or returns an indication causing the local device to discard the result. Denoting $\tau^i_{N,m,c}$ as the latency for controller $\kappa_{N,m,c}$ evaluated at time i, δ the deadline, and $g(\kappa_{N,m,c})$ the result obtained when solving the optimization, the response set is defined as

$$\psi_k = \left\{g(\kappa_{N,m,c}) | \tau^i_{N,m,c} < \delta, \delta \in \mathbb{R}^+\right\} \tag{4}$$

Note that $g(\kappa_{N,m,c})$ corresponds to Y_i in Figure 1 and ϕ_k defines the stacked functions in the cloud. Restricting the setup to a single model and a single constraint set, and restricting δ to the time between two control actions we get the reduced sets

$$\begin{aligned} \mathbf{\Phi} &= \{\kappa_N | N \in \mathbb{Z}\} \\ \psi_k &= \left\{g(\kappa^{k-1}_N) | \tau^{k-1}_N < h\right\} \end{aligned} \tag{5}$$

where h is the sampling period $t(k) - t(k-1)$. In this setup all requested MPCs have the same model, constraints and cost function but different horizons. Only responses which arrive within one sample are valid. In the following this setup will be the focus.

We now define two modes for the controller, *assisted mode* and *local mode*, and the switching strategies.

2.3 Assisted mode

In the assisted mode, the device is connected to the network. At each sample a new set of requests, ϕ_k, is sent to the cloud service. This set includes state information and the horizons to evaluate. Up until the deadline admissible MPC responses in ψ_k are received from the cloud. Several horizons are evaluated to increase the chance of receiving an admissible response in time. Results may be lost or late due to network delay, computation time, admission time into the cloud services, packet loss, connectivity loss and machine failure. Short horizons may not provide feasible solutions and long horizons can take too long to evaluate. Late responses are discarded.

At the start of a sampling period the local control system selects one of the arrived responses. Since all the responses in ψ_k guarantee stability, feasibility and no constraint violation any of the responses can be selected and used. The set operation

$$\Psi : \psi_k \to g(\kappa_r(\cdot)) \tag{6}$$

defines a function that selects one of the available responses. The selection criteria is arbitrary. It could be a simple criteria such as always selecting the smallest or largest horizon or a more involved method which looks at the system state or the predicted cost. The selected $g(\kappa_r(\cdot))$ contains a sequence of control inputs over the length of the horizon

$$\vec{u}_k = \{u_k, u_{k+1}, ..., u_{k+N-1}\} \tag{7}$$

When operating in the assisted mode, and hence continuously receiving responses from the cloud, the controller acts as an ordinary MPC and applies to the plant only the first value in the sequence, $\vec{u}_k(0)$.

2.4 Local mode

In local mode, the control is achieved using the LQ controller obtained from the costs and model of the MPC. Local mode is entered when connectivity is lost, the cloud is unable to provide admissible results or when the state error lies within the terminal set. The latter implies that resources are not requested from the cloud when the state error is small. To operate reliably setpoint shaping is used for the local controller, e.g. a large setpoint change is replaced by a sequence if smaller step changes. Setpoints are also restricted to operate at a safe distance from constraints. Limiting the magnitude of errors perceived by the controller makes the local mode limited in performance but in return provide stability and satisfies constraints.

Figure 2 illustrates the effect of limiting the controller in this manner; showing the initial open loop prediction, Linear Quadratic Regulator (LQR), set-point shaped LQR and a MPC response to a step change in the reference of a second order system (see Skarin et al. (2019) for details). The state constraints are contracted for robustness. The terminal set and tightened constraints are marked in the figure while the outer gray box illustrate the original constraints. The trajectory of the setpoint shaped, conservative LQR is far from the optimal path but stays well within the constraints. The open loop sequence does not correspond with the closed loop optimal path and violates constraints, which shows the severity of the model error. The unrestricted LQR largely violates the constraint, which shows the reaction without setpoint shaping.

2.5 Switching from local to assisted mode

The switch from local to assisted mode is instantaneous. When $\psi_k \neq \emptyset$, i.e. the controller has received one or more responses in time, it will apply one of those results as the next control output. Similarly, if the controller is in transition from assisted to local mode, that process is immediately interrupted.

2.6 Switching from assisted to local mode

The switch to local mode can happen when the system has entered into the invariant set for the local controller or because no response was received from the network. This switching is critical since the local control does not handle constraints. In the first scenario, when having entered into the invariant set, it is straight forward to hand over control to the local controller. Being in the invariant set ensures that the local controller can act without violating the constraints. In the second scenario connectivity is lost or the admissible set becomes empty. If local mode is entered immediately when this happens the system can experience erratic behavior or large constraint violations because of the limitations of the local controller. To avoid this, the control sequence from the latest selected response is used to provide a sequence of inputs used in combination with the local controller.

Since the LQ is built into the MPC the control signal can be considered as the output of a linear, unconstrained controller summed with a perturbation from the constrained optimization, i.e., as

$$f_u(\kappa_l, \kappa_r) = \vec{u}_k(k-i) - K\hat{x}_k|x_{i-1} + K\hat{x}_k|x_{k-1}, \tag{8}$$

where i is the time step in which the MPC with output sequence $\vec{u}_k$ was selected. This decomposition is strongly related to the tube-based robust MPC proposed in Rawlings and Mayne (2009). The state estimate $\hat{x}_k|x_{i-1}$ is the prediction of the current state used by the the MPC to generate $\vec{u}_k$ while $\hat{x}_k|x_{k-1}$ is the current state prediction from the previous sample, i.e., using the latest information as opposed to the information available when the MPC was requested. The expression in (8) removes the predicted LQ response from the MPC output and re-adds the LQ using updated state information. In the first step, when $i = k$, the two last terms cancel and the closed loop MPC of (1) is obtained. In assisted mode this happens repeatedly as $i = k$ when the response set is non-empty, $\psi \neq \emptyset$. When $i < k$ the states may not match and the LQ regulator, working in closed loop, is allowed to compensate for prediction errors. This strategy assumes that the MPC, in addition to the control signal sequence, also returns the corresponding state sequence, i.e., that this is also contained in the response set ψ_k. We do not formally address constraint satisfaction on connectivity loss but (8) is a better alternative than using only the open loop sequence or unconstrained LQ.

Using (8), the local LQ regulator is active at all times and applied as a residual to the MPC open loop trajectories during the mode switch. As a further measure to create robustness and a smooth transition to local control, (8) is extended with an averaging term, α_i, which gradually decreases the impact of the MPC control signal when switching from assisted to local mode. To make explicit that the local controller uses setpoint shaping, $\hat{x}_k^{lim}$ is introduced to denote the limited state after shaping.

$$f_u(\kappa_l, \kappa_r) = \alpha_i(\vec{u}_k(k-i) - K\hat{x}_k^{lim}|x_{i-1}) + K\hat{x}_k^{lim}|x_{k-1} \tag{9}$$

Equation (9) is performed by the client (inside f_u in Figure 1) and is not part of the MPC problem. From here on (9) is referred to as α-switching. Figure 3 shows again the scenario in Figure 2 but this time with two examples of α-switching. These two trajectories have suffered connection loss and are switching to the local mode using the exponential decay given by Equation (10) but with different values of N. An exponential decay is chosen from the observation that model errors in the MPC produce an exponential growth in the error over the predicted path. The exponent γ and starting condition β_0 are set to change the influence of the MPC from 99 % to 1 % over the horizon.

$$\begin{aligned} \alpha_i &= 1 - \beta_i, \quad i = \{1, .., N\} \\ \beta_i &= \beta_{i-1}e^{\gamma}, \quad \beta_0 = 0.01,\ \gamma = log(99)/N \end{aligned} \tag{10}$$

There are several things to note in this figure. First, the closed loop MPC represents the assisted mode. This is the typical path of the cloud-assisted MPC controller. Second, the α-switch with $N = 20$ violates the original constraints while the the α-switch with $N = 9$ does not. To handle connectivity issues it is a more robust strategy to select short horizons since they are forced into the terminal set within fewer steps. Third, the two dashed lines shows paths with $\alpha_i = 1$. The effect of α-switching is clearly visible when $N = 9$. The effect is reduced as N increases since α_i remains large outside the terminal set, which retains the MPC output until the two modes begin to coincide. It should be noted also that (8) already provides some

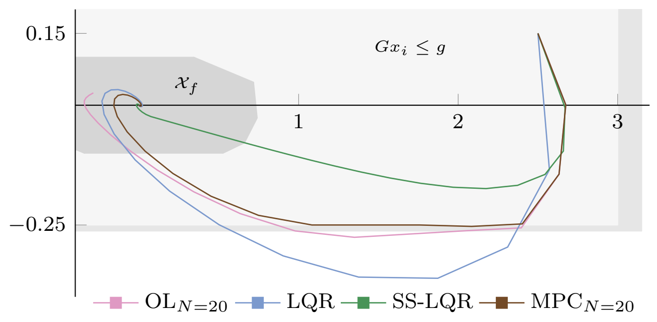

Fig. 2. Response of the setpoint shaped LQR (SS-LQR), open loop prediction, unrestricted LQR, and MPC in a system with a model error. Terminal set and tightened constraints are marked in the figure. The outer gray box shows the critical, original constraints.

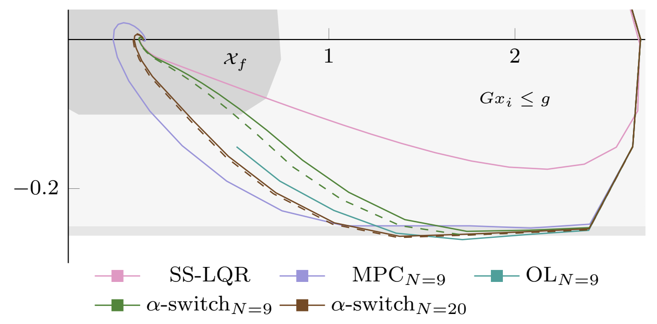

Fig. 3. Same scenario as in Figure 2, with different controllers. The local, setpoint shaped LQR (SS-LQR), a fully connected MPC$_{N=9}$, initial open loop prediction and two, disconnected, α-switching controllers, using (9) with the decay in (10). Dashed lines show the paths for a fixed $\alpha_i = 1$, i.e. (8).

robustness to model errors as seen by comparing to the open-loop sequence. Fourth, the α-switching initially pulls the path towards the conservative path of the setpoint shaped LQR, but as the system approaches the terminal set the $N = 9$ solid path again begins to merge with its corresponding dashed path. This happens even though the local controller becomes more and more dominant because the paths of the two modes begin to coincide. The strategy pulls the path slightly towards SS-LQR, with the result that the path stays within the original constraints. Finally, on close inspection, the path of the closed loop MPC temporarily violates the constraints at one point. This is because of the model error. At this point the MPC will be infeasible but control is recovered by temporarily entering the switching mode. The switching brings the state back inside the constraints where in turn the assisted mode, and therefore MPC efficiency, is regained.

2.7 System delay model

The latency of a single optimization, i.e., one configuration κ_N in Φ, is modeled as

$$\tau_{rt}(N, \epsilon) = X_e(N, \epsilon) + X_s \qquad (11)$$

where τ_{rt} is the round trip delay and X_e and X_s are random variables referred to as the execution time and service delay. The execution time depends on the controller horizon and the current state error ϵ. Repeating a request is expected to not provide an identical execution time due to the expectancy of executing on different machines and alongside other applications in the cloud. The function and variability governing the execution time is unknown. To obtain an estimate of the execution times as a function of state and horizon, and variability due to the cloud, an efficient optimizer (QPgen, Giselsson (2015)) is deployed in the cloud service. However, this optimizer lacks the necessary support for terminal conditions and therefore in the evaluations a Matlab implementation is used.

The service delay includes networking and other overhead such as queuing for access to the cloud service. It is assumed that the distributions are relatively stable and therefore they do not depend on the time step k. In practice, if the load on the system temporarily causes the designed for distributions to be wrong, the loss is managed by the switching and local modes. X_s is calculated through measurements of $\tau_{rt}(N, \epsilon)$ and $X_e(N, \epsilon)$ when running QPgen optimizations on real cloud services. This is what is used in Section 3.1.

2.8 Plant model

The approach in this paper is exemplified using the linearized triple integrator process in (12) and (13). This system controls a ball rolling on a tilting beam. x_1 is the ball's position, x_2 the velocity of the ball and x_3 the angle of the beam. This system is observable, controllable, and open-loop unstable. The state $x = [x_1 \ x_2 \ x_3]$ is limited by the explicit constraints given in (13).

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) \end{aligned} \qquad (12)$$

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -\frac{5}{7}\cdot g \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 0.44 \end{bmatrix}, C = 10 \begin{bmatrix} \frac{1}{0.55} & 0 & 0 \\ 0 & 0 & \frac{4}{\pi} \end{bmatrix} \qquad (13)$$
$$x = \begin{bmatrix} x_1 & \dot{x}_1 & x_2 \end{bmatrix}, |x_1| \le 0.55, |x_2| \le 1.5, |x_2| \le \pi/4$$

The aim is to use a sample rate of 20 Hz. This is a reasonable frequency for the plant and for the cloud control system, as previously observed by Skarin et al. (2018).

2.9 Designing an assisted controller

The controller is defined as the linear MPC in (1). The local regulator is obtained as the solution to the unconstrained LQ problem as $N \to \infty$. The model matrices A, B, the constraint pair G, g, cost matrices Q, R and the terminal set $\mathcal{X}_f$ must be specified. No horizon is defined for the controller but as specified in Section 2.2 there needs to be a strategy for choosing the request set ϕ_k. Here this strategy is defined as using a fixed set of horizons which are evaluated with every sample. The terminal cost P follows as the asymptotic cost in the LQ problem, the solution to which is standard procedure (Kalman (1960)). A brief introduction to the terminal set follows.

The positive invariant set is a set of starting conditions which ensures that the local controller does not violate constraints. A large terminal set reduces the need for large horizons but can also become complex and hard to calculate. Based on the assumptions and purpose for creating the cloud controller it is not necessary to find the largest possible invariant set. A reasonably small, non-optimal, but preferably robust, subset will suffice.

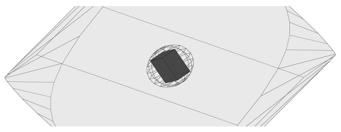

Fig. 4. The maximal invariant set (outer hull), an invariant ellipse and the contained invariant set cube represented by three linear constraints. The local controller is constrained to operate in this limited space.

Polytopes can represent the invariant set arbitrarily well but become complex. An alternative method to find a reasonable non-optimal subset is to use elliptical constraints. However, this requires the use of cone programming for the optimization. To stay with the linear constraints in Equation (14) a rectangular polytope is defined which fits inside the elliptical subset.

$$X_f = \{x | H_t x \leq h\} \tag{14}$$

This is illustrated in Figure 4[1]. The outer hull in this figure is the maximal invariant set for the LQ controller. The much smaller cube shows the final terminal constraints which are contained inside the ellipse. A thorough introduction to the use and calculation of invariant sets is found in Blanchini (1999).

3. RESULTS

In this section, we first look at an example of the cloud in practice and evaluate it for our application. The data collected is used for the simulations in Section 3.2. The simulations use Matlab, and Simulink, with a continuous-time model of the plant. Matlab's *quadprog* is used for the optimizations, with support for the terminal conditions.

3.1 Cloud

First, the proposition to use FaaS (Section 2.1) is investigated. Figure 5 shows results from executing the MPCs on three different platforms. The setup is as in Figure 1 but using different cloud providers and configurations. While the details of cloud service changes, the request API and the implementation on the client remains unaffected. The tests repeatedly evaluate a step change in the setpoint for the system in (13). For a set of horizons $N = 5 - 120$, the step is evaluated a total of 1000 times using each service. The services γ, α, and β_1 are continuously loaded with ten optimizations in parallel while β_2 must handle twice as many, i.e. 20, parallel requests. With each new request, the horizon is selected randomly from the above set.

γ is the result of a function-based implementation using a representative public FaaS. Here, we have submitted the function, i.e. the optimization software, to the cloud but do not manage the service architecture. The default configuration of the service such as number of concurrent requests and timeouts are not restrictive for this scenario. Examples α, β_1, and β_2 use the infrastructure service of the cloud (IaaS) to create a custom service. This uses Flask and HAProxy to provide the request API and a load balancer. The data center provider and virtual machine configuration are different for α and β_1 but both have one load balancer and the same number of worker cores (eight).

[1] Using the Multi-Parametric Toolbox by Herceg et al. (2013).

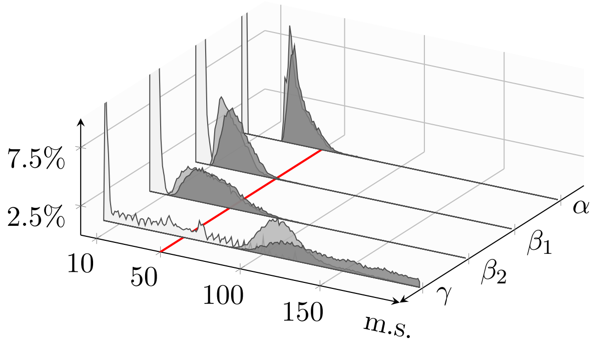

Fig. 5. Delay distributions from three different services. Dark gray is round-trip time. Gray is service time. Light gray is execution time. β_2 is loaded by twice as many parallel requests.

β_2 was given more cores to account for the additional load. The data center of β is a much smaller, municipal data center, while α uses the same, more distant, public data center also used for γ. The inclusion of α is for comparison with γ and the performance of the municipal data center. In all cases, the same Python code is used for the optimization function.

As stated in Section 2.8 the aim is to use a sampling period of 50ms (the red line). This is not possible for γ since service times, and sometimes also execution times, end up beyond the red line. The large spread of the execution times, $\mathcal{X}_e(N, \epsilon)$, seems due to that the service penalizes extended executions. That is, the performance per time unit decreases when the complexity of the problem increases. This translates to long execution times for large horizons and when the system state requires many iterations for the optimization to converge.

The results prompt the necessity to try something different, which is why α and β_1 are introduced. The response of α is much better than that of γ. Since they use the same cloud provider, with the same locality, this shows that the long delays of γ, caused by execution times, are due to the FaaS. The figure also shows that using the local cloud provider, β_1, gives a response similar to α. The more distant provider has faster execution times but a larger minimum service time. This is attributed to network delay. Finally, β_2 was loaded by more concurrent requests, to allow for more horizons per sample. Although the number of workers was scaled up to account for the load, the service time increased. Thus, the assumption of an ideal system capable of an arbitrary amount of concurrent requests does not fully hold for this configuration and load. Still, this setup has a higher utility, with more successful requests per sample.

β_2 is chosen as the baseline to study control performance. The service time distribution, X_s, is obtained through maximum likelihood fitting of the data in Figure 5 onto a log-normal distribution. Execution times, $\mathcal{X}_e(N, \epsilon)$, are obtained by executing the functions on the cloud service.

3.2 Control

The performance of a nominal system is shown in Figure 6, Figure 7, and Figure 8. The simulations are run in Matlab but draws the execution delays from the cloud as presented

Table 1. Admissible responses (percent)

N	3	5	7	9	11	20	29	38	47	56	66	75	84	93	102	111	120
A	26	40	51	60	69	94	94	96	94	92	90	93	91	92	87	90	85
B	23	36	47	56	65	95	98	98	99	99	96	95	92	95	94	91	88
C	7	13	19	20	28	39	25	31	26	27	24	21	24	20	23	16	19
D	6	8	11	12	12	17	22	21	17	15	15	14	10	4	11	8	9

Table 2. Horizon selection (percent)

N	0	3	5	7	9	11	20	29	38	47	56	66	75	84	93	102	111	120
A	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	24	73
B	55	15	5	4	4	4	11	2	0	0	0	0	0	0	0	0	0	0
C	40	4	10	11	6	8	9	3	4	2	1	1	1	1	0	0	0	0
D	20	5	5	5	5	2	11	10	15	7	4	2	3	2	0	1	1	0

in Section 2.7. In examples A and B the service times are drawn from a log-normal distribution, $\mu = 2.54, \sigma = 0.48$, offset $= 6.8$, which represents the service of β_{20} in the previous section. For examples C and D, problematic network/service conditions are simulated by increasing the mean to $\mu = 3.93$.

Examples A and B in Figure 6 illustrate two different controller modes. In A (left) Ψ is defined to select the longest horizon from the admissible set ψ_k. In B (right) Ψ is instead defined to select the shortest available horizon. Table 1 shows the percentage of times each horizon provides an admissible result, i.e. is feasible and responds in time. In examples A and B service conditions are good and most requests respond in time. Looking at Table 1 the decreasing values for horizons 11 to 3 are not due to service or execution times but to the horizons being too short to provide feasible solutions. Horizons from 20 and above always provide feasible solutions but a few responses are lost due to delays. The lower values for the longer horizons are attributed to execution time delays. Table 2 shows how often each horizon is selected by the controller. Here $N = 0$ refers to the local mode of operation. Since the service performance in β_{20} is good, selecting the long horizon effectively translates to using $N = 120$ or $N = 111$ almost always, i.e., this system behaves similar to a standard MPC with a long constant horizon. Service latency is the reason why $N = 120$ is not always selected in example A. In B however, the selected horizon will decrease as the state error gets smaller and the controller therefore operates over a range of horizons by design. In example A the local mode was almost never used since the system does not have time to stabilize around the setpoint before it changes. In difference to A, example B is able to often operate in the local mode. This is due to the use of short horizons near the set point. The difference in behaviour between the two modes is clearly visible as the system approaches a setpoint, emphasized by the encircled red areas in the figure. Both system stay within the constraints, which is not the case for the optimal LQ regulator shown in the dotted line, and both are more efficient than the setpoint-shaped LQ regulator shown in the dashed line.

Example C in Figure 7 also uses short horizon selection but the service delays are now longer. In Table 1 there is a clear decline in admissible responses as seen in the lower numbers and reduced blue color. The effect of combined service time, execution time and feasible horizon also becomes clearer with a peak of admissible responses at horizon 20. In Table 2 the use of horizons above 20 shows that on occasion, the network conditions of C causes unnecessarily long horizons to be the only choice available. This translates into some of the used trajectories resembling example A and some example B. Overall the controller continues to perform well but the occasional longer horizon is an issue for the robustness aspect of choosing shorter horizons. For the switching strategy to be effective the MPC must move quickly away from constraints and/or the shaped local controller must be switched in early. When using the mode switching, selecting the shortest feasible horizon creates these conditions but due to service latency and potential execution variability, the shortest feasible result may not arrive to enter the admissible set.

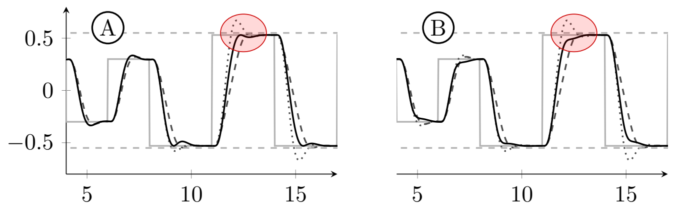

Fig. 6. (A): Assisted controller selecting the longest horizon. (B): Assisted mode controller selecting the shortest horizon. The gray trajectory is the setpoint and the state constraint is shown as the dashed gray line. A dotted gray line shows the unconstrained LQR and the dashed black line show an LQR with setpoint shaping to not violate the constraints.

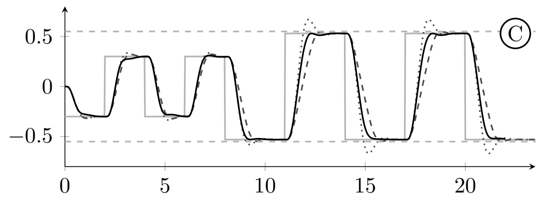

Fig. 7. The scenario in Figure 6-B with degraded service.

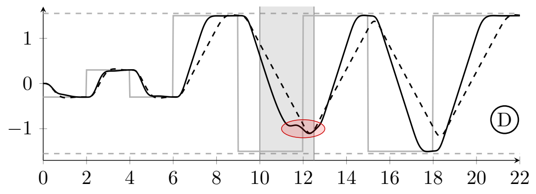

Fig. 8. Assisted control with degraded network and modified constraint. In the gray area the controller has no connectivity to the cloud and transitions to local mode. Performance is regained when connectivity is restored. Dashed line is a setpoint shaped LQR.

In example D, Figure 8, a change has been made to the controller constraints. The range of state x_1 has more than doubled which allows for a larger range of setpoints. At time $t = 5$ this is used to request larger state changes than what was previously possible. This example also uses short horizon selection. The larger setpoint changes may require longer horizons for feasible solutions. Up until now the controller could limit itself to use horizons 3 to 20. When it now needs longer horizons they may not be admissible due to delay. If they become available they can be used, if they do not, the local mode will eventually bring the system to a state were the shorter horizons are feasible. The larger setpoint in D is observed in Table 2 as the increased selection of horizons above 20. The peak in admissible responses is not as clear as in example C but there seems to be a slight shift upwards in Table 1 which is expected with the larger setpoint changes. In addition to modified constraints and the degraded service conditions, example C also introduces connectivity loss at $t = 10$. When this happens, the controller must switch from assisted to local mode using the open loop data of the latest selected MPC. After connectivity is restored the controller enters the assisted mode again as soon as a feasible MPC is returned in time. The encircled red area shows the switch to and from local mode. It may seem that the trajectory leaves the optimal path prematurely but this is part of the α-switching and short horizon strategy which prioritizes robustness over the potential performance of the open loop data from the MPC.

4. CONCLUSION

We have shown the implementation of a variable horizon model predictive controller using the cloud. The performance of the cloud services and the controller were shown in a combination of benchmarks performed on the real services in the cloud and controller simulations driven by the observed data. Through a combination of local LQ control and cloud supported MPC an improved MPC design was obtained which allows flexible performance while remaining stable and reliable. Constraint satisfaction is made possible in the nominal case and a strategy combining short horizon selection and gradual mode switching provides smooth operation, performance and a degree of robustness. The idea of elastic control extends to complex and non-linear systems, and the strategy of evaluation many controllers concurrently should lend itself to many problems for finding cost minimizing solutions online and for creating robust best effort solutions in control.

REFERENCES

Alessio, A. and Bemporad, A. (2007). Decentralized model predictive control of constrained linear systems. In *2007 European Control Conference (ECC)*, 2813–2818. IEEE.

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. *Commun. ACM*, 53(4), 50–58.

Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E.N. (2002). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1), 3 – 20.

Blanchini, F. (1999). Set invariance in control. *Automatica*, 35(11), 1747 – 1767.

Camponogara, E., Jia, D., Krogh, B.H., and Talukdar, S. (2002). Distributed model predictive control. *IEEE Control Systems*, 22(1), 44–52.

Christofides, P.D., Scattolini, R., de la Pena, D.M., and Liu, J. (2013). Distributed model predictive control: A tutorial review and future research directions. *Computers & Chemical Engineering*, 51, 21–41.

Giselsson, P. (2015). QPgen. http://www.control.lth.se/fileadmin/control/Research/Tools/qpgen/index.html, Accessed May 2020.

Givehchi, O., Imtiaz, J., Trsek, H., and Jasperneite, J. (2014). Control-as-a-service from the cloud: A case study for using virtualized PLCs. In *10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*, 1–4.

Hegazy, T. and Hefeeda, M. (2015). Industrial automation as a cloud service. *IEEE Transactions on Parallel and Distributed Systems*, 26(10), 2750–2763.

Heilig, L., Negenborn, R.R., and Voß, S. (2015). Cloud-based intelligent transportation systems using model predictive control. In *International Conference on Computational Logistics*, 464–477. Springer.

Herceg, M., Kvasnica, M., Jones, C., and Morari, M. (2013). Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, 502–510. Zürich, CH.

Kalman, R.E. (1960). Contributions to the theory of optimal control. *Boletin de la Sociedad Matematica Mexicana*, 5(2), 102–119.

Lu, Q., Sun, Y., Zhou, Q., and Feng, Z. (2014). New results on robust model predictive control for time-delay systems with input constraints. *Journal of Applied Mathematics*, 2014.

Michalska, H. and Mayne, D.Q. (1993). Robust receding horizon control of constrained nonlinear systems. *IEEE Transactions on Automatic Control*, 38(11), 1623–1633.

Mubeen, S., Nikolaidis, P., Didic, A., Pei-Breivold, H., Sandström, K., and Behnam, M. (2017). Delay mitigation in offloaded cloud controllers in industrial IoT. *IEEE Access*, 5, 4418–4430.

Park, H.W., Wensing, P., and Kim, S. (2015). Online planning for autonomous running jumps over obstacles in high-speed quadrupeds. In *Proceedings of Robotics: Science and Systems*. Rome, Italy.

Pelle, I., Czentye, J., Dóka, J., and Sonko, B. (2019). Towards latency sensitive cloud native applications: A performance study on aws. *IEEE Services*.

Rawlings, J. and Mayne, D. (2009). *Model Predictive Control: Theory and Design*. Nob Hill Pub.

Scattolini, R. (2009). Architectures for distributed and hierarchical model predictive control – a review. *Journal of Process Control*, 19(5), 723 – 731.

Skarin, P., Tärneberg, W., Årzen, K.E., and Kihl, M. (2018). Towards mission-critical control at the edge and over 5G. In *2018 IEEE International Conference on Edge Computing (EDGE)*, 50–57.

Skarin, P., Eker, J., Kihl, M., and Årzén, K.E. (2019). An assisting model predictive controller approach to control over the cloud. *arXiv e-prints*, arXiv:1905.06305.

Stewart, B.T., Venkat, A.N., Rawlings, J.B., Wright, S.J., and Pannocchia, G. (2010). Cooperative distributed model predictive control. *Systems & Control Letters*, 59(8), 460 – 469.

Villari, M., Fazio, M., Dustdar, S., Rana, O., and Ranjan, R. (2016). Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6).

Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., and Jue, J.P. (2019). All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*.

Zheng, Y., Li, S.E., Li, K., Borrelli, F., and Hedrick, J.K. (2016). Distributed model predictive control for heterogeneous vehicle platoons under unidirectional topologies. *IEEE Transactions on Control Systems Technology*, 25(3), 899–910.