Dynamic Offloading of Control Algorithms to the Edge using 5G and WebAssembly

Ahmed Al Bayati
Dept. of Automatic Control
LTH, Faculty of Engineering
Lund, Sweden
ahmed.al_bayati@control.lth.se

Karl-Erik Årzén
Dept. of Automatic Control
LTH, Faculty of Engineering
Lund, Sweden
karl-erik.arzen@control.lth.se

***Abstract*—The aim of this work was to test if WebAssembly universal byte-code and wireless 5G technology, together with edge computing, is suitable in the context of offloading real-time control applications. To test these tools a dynamic offloading strategy was implemented and tested on an inverted Furuta pendulum, an inherently unstable and time-critical process, using an edge node as the offloading target. The implementation is considered dynamic because: 1) The local device which interacts with the I/O of the process dynamically also sends the code of the offloaded control application to the edge node. 2) The local device dynamically decides on which controller should control the process, either the local fallback Linear Quadratic Regulator (LQR) or the remote Model Predictive Control (MPC) solver compiled into Ahead-of-Time (AOT) WebAssembly (Wasm) format. The work concluded that Wasm run in interpreted form was too slow to control the process, while AOT compiled Wasm worked well with an execution time close to the native speed. It was also concluded that data transfer using 5G technology without URLCC was fast enough to balance the pendulum and is suitable for offloading, other communication techniques were also tested in this work including WiFi and wired Ethernet. In the study we also developed a simple control quality measure for decision making on when to use the offloaded controller and when to use the local controller.**

***Index Terms*—Dynamic Offloading, Edge Computing, 5G Network, WebAssembly, MPC, CVXGEN**

I. Introduction

There are many reasons for executing an application or parts of it in the cloud or at the edge. The most obvious one is the access to the compute power that the cloud provides.

More compute power means that it is possible to solve larger control problems or solve the same problem more often and/or faster. The controller type that is considered here is Model-Predictive Control (MPC), where a quadratic optimization problem is solved every sampling period. This can be quite time-consuming if a standard off-the-shelf optimization solver is used. Moreover, the amount of time it takes varies from sample to sample. An assumption that one, sometimes implicitly, make then is that it is simply not possible to execute the controller in the local device, at least not at the desired frequency. This may be true if one uses a small microcontroller as local device. However, in our case the local device is a modern Linux laptop which has almost the same capacity as the servers in the edge node. Hence, although it would be possible to execute the MPC at this particular local device, we pretend here that we have a much less powerful local device.

Offloading a controller can potentially give better control performance due to the increased compute power. However, this may be counteracted by the increased latency caused by the communication between the local device and the cloud or edge. Hence, there is often a sweet spot where the control performance is maximized. Offloading the controller to a potentially very powerful cloud data center that is far away will decrease the performance and executing the controller in the local device, if at all possible, will also decrease the performance compared to the performance at the sweet spot. The sweet spot can in many cases be the edge node.

Whenever a controller is offloaded there is always the risk that no control action is returned to the local device. There are several reasons for this. An optimization-based controller may fail to obtain a feasible solution, i.e., a solution that meets the optimization constraints. A wireless connection link can lose the packet being transmitted. Hence, there is a need to have a fallback controller in the local device to switch to if the control signal is not returned within a certain deadline. In the paper the fallback controller is a LQR controller, i.e., a state feedback controller that only requires a few multiplications and additions to implement, and which without any problem can execute also on a very resource-constrained local device.

A. Research Questions

This work aims to explore dynamically offloading parts of a control algorithm compiled to Ahead-of-Time WebAssembly to an edge node and testing different communication technologies such as 5G, WiFi, and wired Ethernet by controlling a Furuta pendulum in the upward position. More concretely, this work aims to address the following research questions:

- How would a dynamic offloading strategy utilizing WebAssembly look like?
- What are the expected delays and control qualities of the different communication mediums?
- How to evaluate different controllers during runtime?

This work has received funding from Vinnova through the AORTA project. The author is WASP affiliated and also a member of ELLIIT.

B. Papers Outline

Section I introduces the paper and Section II presents an overview of relevant topics used in this work. The offloading implementation is presented in Section III and in Section IV the obtained results are presented and discussed. Finally the paper is concluded in Section V

C. Related Work

There has been extensive research conducted on various aspects of offloading. Some researchers have concentrated on implementing offloading techniques to conserve energy and/or computational power [14][5][4]. Others have focused on achieving code portability and migration between different computing systems and architectures [13][21][11]. Some have delved into the decision-making process of offloading, determining when to offload [11][3], while others have sought formal proofs of stability for different offloading aspects [32][29]. The networking and infrastructure enabling offloading have also been extensively studied, with some researchers examining existing infrastructure while others aim to formalize proposed networks for offloading [12] [25]. Given the breadth of research on offloading, some scholars have focused on summarizing the field through surveys. These surveys may be specific to particular areas or more general in nature [19] [1] [26] [31].

II. Background

A. Furuta Pendulum

The Furuta pendulum is a two-link system, as can be seen in Figure 1, where one controls the base in order to balance the pendulum attached to it [9]. It is characterized by the angles, θ, and ϕ, representing the pendulum angle and base angle, respectively. The angular velocities associated with these angles are denoted as $\dot{\theta}$ and $\dot{\phi}$ respectively, which collectively are the states of the process, x.

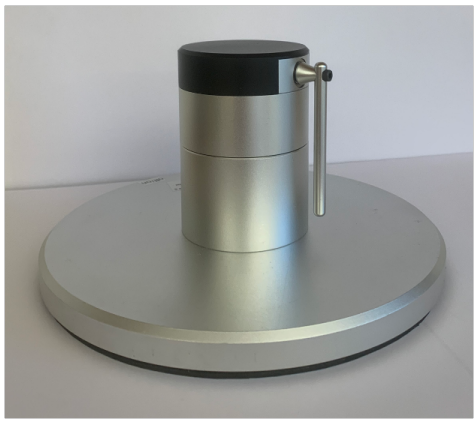

Fig. 1. The process used in this work was a desktop Furuta pendulum, where the angle between the pendulum and the normal is denoted θ while the angle of the base is denoted ϕ.

In this paper, only control in the upward position, i.e., the balancing, is considered. It is assumed that a human user holds the pendulum in the correct upward position when the controller is started.

The theoretical model of the Furuta pendulum used here was derived by Gäfvert, [10], using Euler-Lagrange equations.

B. MPC and LQR

Model Predictive Control (MPC) is a control strategy which solves an online optimization problem every control instance to obtain a control signal. This control signal both respects constraints on the process states and the control signal; and minimizes the cost defined by the problem formulation [28].

The MPC solver used in this paper was CVXGEN, [23]. CVXGEN automatically generates a solver written in C given a small QP problem formulation.

The Linear Quadratic Regulator (LQR) is also an optimal controller [20]. The main difference between it and the MPC controller is that LQR has an analytic solution obtained using the discrete-time (algebraic) Riccati equation, i.e., it does not require solving an online optimization problem every control instance, which reduces the computation time significantly. This makes it suitable as a fallback controller in the local device. The drawback with LQR is that the states and control signals cannot be constrained.

The LQR solver used in this work was the `dlqr` function provided by Matlab. `dlqr` returns the state feedback law for the infinite horizon problem,[22].

C. WebAssembly

WebAssembly (Wasm) is a binary-code format executed by a stack-based virtual machine [36]. Wasm was initially developed for modern web browsers, but is now increasingly used in cloud contexts. WebAssembly byte code is obtained by coding in languages such as C, C++, or Rust and then compile the code to Wasm [24]. The resulting binary code can then be executed by a Wasm runtime such as WasmEdge.

WasmEdge can be used for executing Wasm code in cloud-native environments, edge applications or integrated into programs written in languages such as C, Rust, or Go. WasmEdge runs Wasm code in a sandbox environment to ensure a secure execution environment independent of the underlying operating system [35].

In addition, WasmEdge features a compiler that translates WebAssembly into native machine code. This capability enables WasmEdge to operate in Ahead-of-Time (AOT) mode, leading to improved execution speed [34].

D. 5G and Edge Technology

5G is the fifth generation of cellular networks which is said to be up to 100 times faster than 4G, have low latency and greater throughput capacity than 4G [7] . 5G wireless communication system is expected to support a broad range of emerging applications, especially since the ultra-reliable and low-latency communication mode (URLLC) is being introduced. URLLC is one of the standout features of 5G technology and is designed to ensure ultra-reliable and low-latency communication, especially crucial for mission-critical applications. URLLC guarantees good quality of service (QoS),

offering low radio latency of 1 millisecond (ms) along with high reliability [16]. URLLC was, however, not used in this work since wireless 5G routers with URLLC support are rare and expensive.

Edge computing is a distributed framework which brings processing and storage resources for applications closer to where data is generated and/or consumed, in this paper at the 5G base station. It opens up the opportunity for cloud applications to be more responsive and faster to response. This can make it possible to control mission-critical systems which requires fast sampling [8].

E. Containers and Kubernetes

Since bother containers and Kubernetes where used in this work they are presented here for completion.

Containers are a virtualization technology that can encapsulates all necessary software and dependencies required to execute a program. Each container possesses its own file system, process space, and network stacks, which are isolated from the hosting operating system [15].

Kubernetes (K8s) is a container orchestration platform that automates the deployment, scaling, and management of containerized applications, by abstracting the underlying infrastructure, making it easier to manage complex, multi-container applications, [17][18].

III. Implementation

This section contains an overview of how the experimental setup looked like, a conceptual overview of how the implementation works, and a detailed description of the four major components in the implementation: Control Quality Measure, CVXGEN, Local Device, and Remote Solver.

A. Experimental Setup

The devices used in the experiments presented in this work are:

Furuta Pendulum: The Furuta pendulum is a mechanical process developed by Ben Katz and modified by the Department of Automatic Control at Lund University [27]. To interact with the process the Moberg API was used, wish is an I/O interface for communicating with various lab processes in the Department [2].

Local device (Host): The local device, also referred to as the host, is the computer that is connected to the pendulum. In this paper it was a Lenovo T15 laptop with an Intel Core i7-10510U CPU @ 1.80GHz with a RAM of 16GB, running Fedora Linux 38.

Remote solver: The remote system, i.e, the offloading target, is a bare-metal Kubernetes edge node with seven servers connected to the 5G base station using local IP breakout. It is located in the Department of Electrical and Information Technology at Lund University. The hardware information is given in Table I.

5G router: The 5G router used to communicate with the edge from the local device was an Askey 5G Sub6 with model name NDQ1300-RoHS dongle.

TABLE I
Edge Node Hardware Information

Node	CPU	# CPUs	RAM
1	Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz	20	15.8G
2	Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz	20	15.8G
3	Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz	20	15.8G
4	Intel(R) Xeon(R) Silver 4208 CPU @ 2.10GHz	16	15.8G
5	Intel(R) Xeon(R) Silver 4208 CPU @ 2.10GHz	16	15.8G
6	Intel(R) Xeon(R) Silver 4208 CPU @ 2.10GHz	16	15.8G
7	Intel(R) Xeon(R) Silver 4208 CPU @ 2.10GHz	16	15.8G

WiFi router: For the evaluations of WiFi and wired Ethernet the WiFi router used to communicate with the edge was an Archer Ax72 AX5400 Wi-Fi 6 Router.

B. Conceptual overview

To understand how the offloaded implementation works an overview of how it runs is presented. The details are omitted to make the basic idea clear. The details are presented in the subsections that follow.

Upon start the remote solver does not have the MPC algorithm needed to perform the MPC control. hence, it simply waits for the local device to send the application. The local device sends the MPC code to the remote solver while controlling the Furuta using its local LQR controller as illustrated in Figure 2. When the remote solver receives the control application, it becomes ready to receive offloading requests from the local device.

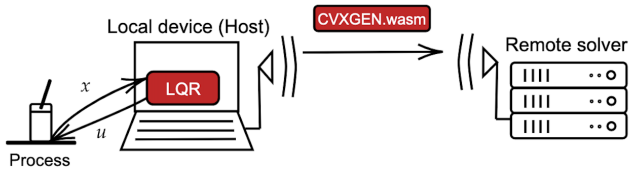

Fig. 2. When the local device starts it controls the process using its local controller while sending the control application to the remote solver. In the figure the LQR is the local controller and the CVXGEN.wasm is the control application.

After sending the application the local device will monitor the control quality of the pendulum. If the control quality is deemed low, the local device will onload, i.e, control the process using its local LQR controller to ensure stability. Otherwise, if the control quality is deemed good the local device will offload. When the remote solver receives an offloading request it uses the MPC code to calculate a control signal and sends it back to the local device.

If the response from the remote solver is revived by the local device within a predefined time, the local device will actuate the process with that control signal as illustrated in Figure 3. Otherwise, if the deadline is missed the local device will default to the local LQR controller as illustrated in Figure 4. This offloading logic is then repeated until the program is terminated.

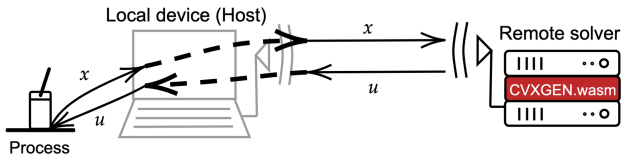

Fig. 3. When the local device chooses to offload it wait for the remote solver to return a control signal, if the remote solver returns a control signal within a predefined deadline it will use that control signal to actuate the process.

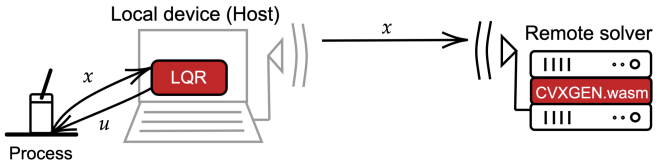

Fig. 4. When the local device chooses to offload but does not get any control signal in time it will onload the control task and control the process using the local LQR.

C. CVXGEN

The control application used in this paper is the CVXGEN MPC solver.

Every time the CVXGEN solver is invoked, one must first define a parameter list consisting of the states of the process, the state matrix, the input matrix, and the MPC parameters, such as the cost on the states, the cost on the control signal, the maximum and minimum value of the control signal, and the maximum rate change of the control signal. Since the process and the MPC parameters remain unchanged in our case, they are stored in a list and are reused every time the CVXGEN solver is called along with the new states.

1) Wasm & WasmEdge: The CVXGEN solver is compiled to Wasm and WasmEdge runtime is used to run it. To interact with the code in Wasm format, a file named `bridge.c` was implemented. This addition acts as a bridge to the CVXGEN solver. It defines the global structures and creates an interface between the CVXGEN solver and the program that interacts with it. Subsequently, the source code of the solver and the complementary `bridge.c` where compiled to Wasm using Emcc, which is a compiler toolchain that compiles C, or other programming languages that uses LLVM, to WebAssembly [6]. When testing the Wasm CVXGEN controller on the process it was observed that Wasm in interpreted form was too slow for our use case. Therefore the CVXGEN Wasm file was further compiled to AOT compiled Wasm using the WasmEdge compiler [34].

D. Control Quality

In order to make an informed decision on when to offload, it is important to determine the control quality of the process. The underlying logic follows a straightforward principle: If the cost, a measure of the control quality, is lower than a predefined threshold the local device will offload the control task to the remote solver. Otherwise, if the cost is higher than the threshold, the local device will use the local fallback controller to control the process and guarantee stability.

Various measures already exist to quantify the control quality of a controller, such as Integral Square Error (ISE) and Integral Time Square Error (ITSE) [30]. While these control quality measures are widely used, they suffer from a limitation relevant for offloading: they integrate the error over the entire time span. Implementing this approach would lead to a monotonically increasing cost which will eventually causing the remote controller to be unused.

To address this limitation, we wrote our own simple control quality cost function, represented mathematically as:

$$J(k) = 0.5(\theta^2 + \dot{\theta}^2 + \phi^2 + \dot{\phi}^2) + 0.5J(k-1). \quad (1)$$

This cost consists of two components. Firstly, it incorporates the squared deviation of the four states from their reference value (which is zero). Secondly, it includes a fraction of the cost from the previous time step. This dual-component structure ensures the cost function's responsiveness to dynamic adjustments while maintaining a degree of stability for noise.

The scaling of the states in Equation 1 is 1, meaning that the deviation of any of the states from zero contribute equally to the control quality measure.

In the control quality measure presented, the control signal is excluded to have similar structure as the already existing control quality measures presented in [30].

E. Local device (Host)

The local device has four functions to perform:

- Sending the control application to the remote solver.
- Monitoring the control quality of the process.
- Deciding to either offload or onload.
- Preforming the offloading or controlling the process using the local controller.

This is realized by four Posix threads (pthreads): Application Sender, Control Quality Monitor, Controller, and Main. Every shared resource between the different threads are protected by semaphores to ensure mutual execution.

1) Application Sender: When the program is started one thread called `sender_thread` is created. This thread does the following tasks and terminates:

- Establishes a TCP connection with the remote solver.
- Sends the control application, which in this case is a AOT Wasm compiled CVXGEN solver, to the remote solver.
- Sends the constant parameters that are needed by the CVXGEN solver to the remote solver.
- When receiving a confirmation from the remote solver it notifies the rest of the program to indicate that the remote solver is ready to receive offloading requests.

2) Control Quality Monitor: The quality control monitoring is implemented in `control_quality_thread`. This thread performs the following tasks at a frequency of 1000 Hz. This frequency is larger than the frequency of the controller, the rationale for sampling the control quality monitor faster than the controller is to capture disturbances that occur between the control samples.

- The states of the process are sampled.
- The measured states and the old control quality measure are used to calculate the new control quality measure using Equation 1.

3) Controller: The controller is implemented in the `regul_thread` thread. This thread does the following tasks with a frequency of 66.67 Hz.

- Actuate the process with the previously calculated control signal.
- Samples the states of the process.
- If the control application and the model parameters are sent to the remote solver and the control quality is good the thread will offload the control task to the remote solver using a UDP socket. Otherwise, the thread will onload the control task.
- If the thread chooses to offload it waits for the response from the remote solver. If the thread receives a response within 14 ms it will use the control signal in its next actuation. If the deadline is missed the thread will use the local controller to control the process.

The reason behind offloading when the control quality is already high is that the process is robust enough and can afford to offload without risking its stability due to sub optimal control signals from, e.g., bit flips.

To clarify the timing, two diagrams showing the two possible outcomes that could happen when the `regul_thread` decides to offload are presented. In Figure 5 the remote solver returns a control signal before the timeout deadline of 14 ms while Figure 6 shows the timing when that deadline is missed. The blocks in the time line are not drawn to scale and not all details of the program are presented in the figures.

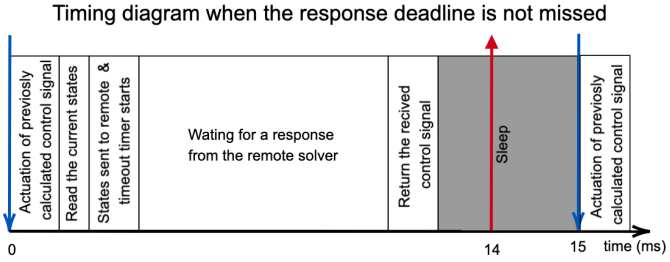

Fig. 5. The timing diagram of the local device when it decided to offload and get a control signal within its timeout deadline of 14 ms.

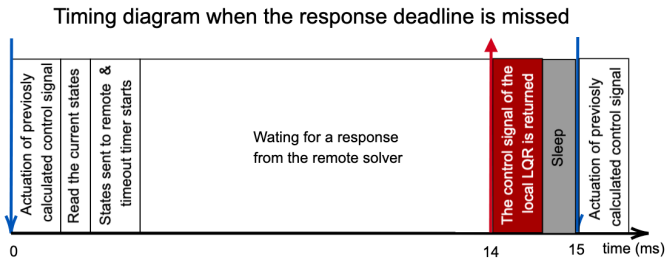

Fig. 6. The timing diagram of the local device when it decided to offload and does not get a control signal within its timeout deadline of 14 ms.

4) Main: The main thread starts up the other threads upon launch and remains inactive until the end of the execution of the program.

F. Remote Solver

The remote solver is the offloading target for the local device and has two functionalities:

- Receive the control application from the local device using a TCP connection.
- When a request is received, it uses the control application to calculate the control signal and sends it back to the local device using an UDP socket.

The remote solver is implemented using three pthreads.

1) The application receiver: When the remote solver program is started a thread called `receiver_thread` is created. This thread blocks the rest of the program until the control application is received. The AOT-compiled Wasm file is then saved locally while the parameters are stored in program memory.

2) The offloading request handler: When the remote solver program saves the control application it starts two threads: `intermediary_thread` and `solver_thread`, which together handle offloading requests.

- `solver_thread`. This thread invokes the control application and returns the control signal to the requester. It does it by using the WasmEdge runtime to run the AOT Wasm compiled solver.
- `intermediary_thread`. This thread acts as the intermediary between the local device and the CVXGEN solver. It opens a UDP socket and starts listening for incoming requests from the local device. When a request from the local device is received it forwards it to the `solver_thread` using UNIX sockets. The thread also forwards the response from `solver_thread` thread to the local device. This structure was chosen to asynchronously stop the execution of the solver inside of `solver_thread` when a new request is revived from the local device, since the execution of the solver is blocking.

Ideally, one would compile all the remote solver code to WebAssembly and use WasmEdge to run it as Wasm container [33]. The reason behind using Wasm is that the remote solver

and the control algorithm should be as platform agnostic as possible since the offloading target might not be known in advance. However, this approach was not implemented because WasmEdge currently lacks support for threads and UDP/TCP sockets. Instead, the entire remote solver is encapsulated in a container along with the WasmEdge SDK and library to allow some flexibility in where the code could be run even though the introduction of containers introduces a bit of overhead.

IV. Results

Different types of experiments were conducted to explore different aspects of the offloading strategy. The results of these experiments are divided into four sections: Round-Trip Time (RTT) delays for the control signal request, Timing of File Sending, Control Quality, and Demonstration.

- **RTT delays for the control signal request.** The total time it took from sending a control request, calculate a control signal, and sending back the control signal to the requester is presented for various configurations.
- **Time for sending the solver file.** The time it took to send the solver file to the remote solver in different configurations is presented.
- **Control Quality.** A presentation of how the different configurations affected the control quality of the process.
- **Demonstration.** The pendulum angle, the control quality measure and which controller is used are presented for when the remote solver is deployed in an edge node and the communication medium is the 5G network.

A. Timing for the Control Signal

Controlling an unstable system, such as the Furuta pendulum, usually requires short delays. This Subsection presents the delays across different configurations, classifying them into two categories: Integrated control and Kubernetes control.

- **Integrated control.** All controllers tested in this Subsection are implemented in the local device without offloading. This forms a baseline for how fast the controllers are.
- **Kubernetes control.** In this Subsection the results of the implementation written about in section III are presented. The offloading implementation composed of the host and a remote solver is tested. In this Subsection the remote solver is the edge which is made up out of a Kubernetes cluster.

In the Kubernetes control Section the RTT delay time is the duration from when the local device request a control signal from the remote solver until the local device received a control signal as illustrated in Figure 3. In the Integrated control Section the RTT delay time is simply the execution time of the code since no offloading is performed.

1) Integrated Control: Integrated control refers to the scenario where the pendulum is controlled by the host computer without any offloading. In this Subsection four controllers are presented: LQR, CVXGEN MPC written in C, Wasm CVXGEN MPC and AOT Wasm CVXGEN MPC.

The time presented here is simple the execution time of the control algorithms. The time measurements were done using `clock_gettime` with the `CLOCK_MONOTONIC` clock. On the hardware used, the smallest measurable time difference was, on average, 20 ns.

LQR. LQR is the local controller, and having a fast fallback controller is essential to ensure stability when a control signal is needed in a short period of time.

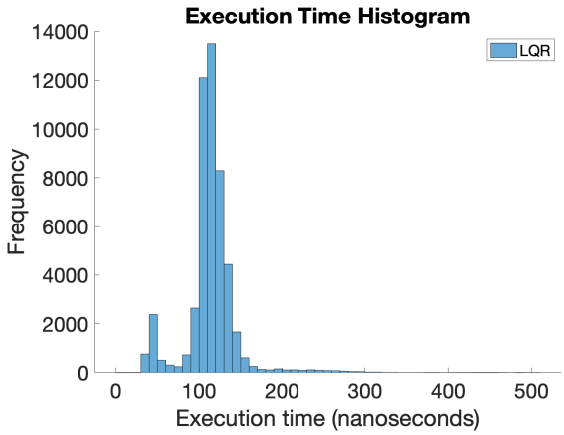

Fig. 7. Execution time histogram for the LQR controller in nanoseconds

As can be seen in Figure 7 the LQR controller is fast, with an average execution time of around 120 ns. This performance is expected, given the simplicity of the controller.

MPC C-code. The CVXGEN MPC controller, written in C, is tested to determine the best-case scenario for the MPC solver's execution time. The average execution time is 1.3 ms, i.e., much longer than for the LQR controller. This difference can be attributed to the fact that the MPC controller must perform an optimization to determine the control signal every time step while LQR is given by a constant control law. The length of the horizon plays a crucial role for the execution time. In this work a horizon of 60 samples was used, which is the number of steps the MPC formulation looks into the future and optimize the states and the control signal subject to the constrains. If the horizon were shorter the execution time would decrease. Although some experiments were conducted with controllers with shorter horizons, they proved to be unreliable and no further experiments were done.

MPC Wasm. The execution time of the WASM CVXGEN solver in interpreted form is notably slow, averaging 219 ms, as presented in Figure 8. While it might be possible to stabilize the pendulum by using a better model and/or a shorter horizon, it would be challenging. For context, the longest period found where it was possible to stabilize the Furuta pendulum was 50 ms using LQR while for MPC the longest period was 25 ms.

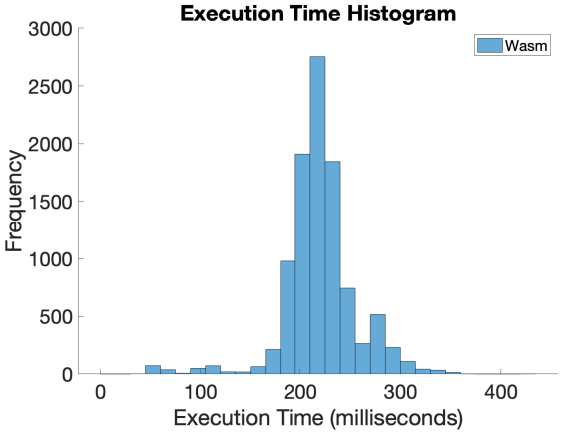

Fig. 8. Execution time histogram for MPC controller executed as Wasm in miliseconds

MPC AOT Wasm. Interpreting Wasm is too slow to stabilise the pendulum. Therefore, Ahead-of-Time (AOT) compilation of Wasm was tested. The data presented in Figure 9 illustrates a notable speed improvement compared to interpreted Wasm.

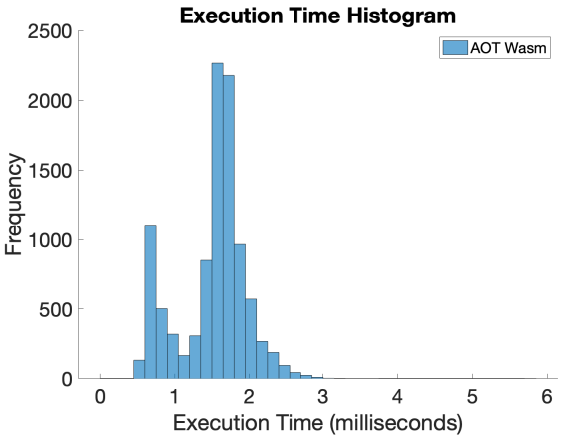

Fig. 9. Execution time histogram for MPC controller executed as AOT Wasm in miliseconds

This efficiency gain comes at the cost of increased file size. The AOT code gets integrated into the Wasm file, allowing it to be executed either as interpreted Wasm or AOT-compiled code. The initial size of the Wasm solver is 666 KB, while the AOT Wasm file has a size of 1.3 MB, approximately twice as large. This have two implications. 1) If the local device has limited memory the solver might be too large to be stored locally, and 2) It will take longer time to send the code to the remote solver. It takes on average 1.8s sending the AOT-compiled Wasm solver to the edge node using the 5G network.

2) Kubernetes Control: The edge node is a bare-metal Kubernetes cluster connected to the network. The hardware of the edge node used in this work is presented in Table I. To deploy the remote solver the container had to be run inside of the Kubernetes cluster, hence Kubernetes control.

Since the only remote solver was the edge node, different communication technologies where tested: 5G, WiFi, and wired Ethernet. The connection to the edge was routed through the Archer router for both the WiFi communication and the wired Ethernet communication and through the 5G antennas and the core network, i.e., with no direct connection between the local device and the edge. Another detail to mention is that we did not specify on which server in the edge node that the remote solver was deployed. It was assumed that all servers where equally fast. The collected data using the edge as the remote solver and testing the different communication technologies is presented in Figure 10 together with the deadline of 14 ms mentioned in Section III-E.

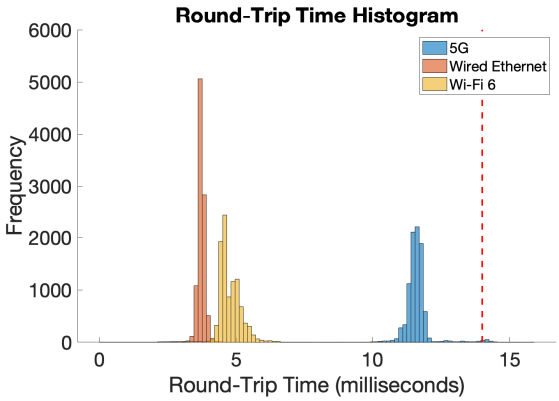

Fig. 10. RTT delay time histogram for 5G, WiFi and wired Ethernet deployment inside of a Kubernetes cluster with the predefined deadline of 14 ms.

5G Communication. As shown in Figure 10, the majority of the RTT delay time falls within the deadline defined, making it possible for the local device to get back the majority of sent offloading requests in a timely fashion. On average the RTT delay for the 5G network was 11.6 ms. In this work standard 5G was used, rather than the URLLC 5G. This is because we did not not have access to URLLC. This work, however, demonstrates that a regular 5G network could be leveraged for offloading.

WiFi Communication In this work WiFi 6 with a bandwidth of 1Gbps was also investigated for offloading, since WiFi is prevalent and accessible. In the experimental setup, WiFi 6 had lower latency compared to 5G with an average of 4.8 ms.

Wired Ethernet Communication While wired Ethernet offers high data transfer rates of 1Gbps, the trade-off is a limitation on the flexibility of the local device and the process due to the constraints imposed by the physical cable. Wired Ethernet is the fastest communication technology with the edge node with an average delay of 3.7 ms.

3) Overview: The different configurations presented above are summarized in Table II.

Table III presents the average and worst-case execution times for the different configurations. As can be seen the worst-case time of the LQR execution is not visible in Figure

TABLE II
TECHNOLOGY USED IN THE DIFFERENT CONFIGURATIONS.

Name	Remote Solver	Com. Technology	Controller
LQR	none	–	LQR
MPC C-code	none	–	MPC C-code
Wasm	none	–	Wasm MPC
AOT Wasm	none	–	AOT Wasm MPC
5G	Edge node (K8s)	5G network	AOT Wasm MPC
Wifi	Edge node (K8s)	Wi-Fi 6	AOT Wasm MPC
Wired Ethernet	Edge node (K8s)	Wired Ethernet	AOT Wasm MPC

7, this is because it was excluded since it was an outlier which made the time resolution bad.

TABLE III
AVERAGE AND WORST-CASE EXECUTION OR RTT DELAY TIMES

Scenario	Average Time (ms)	Worst-case Time (ms)
LQR	0.00012	0.032
MPC C-code	1.3	6.4
Wasm	219	424
AOT Wasm	1.5	5.7
5G	11.6	15.8
WiFi	4.8	14.2
Ethernet	3.7	8.7

B. Timing of the File Sending

As mentioned in Section III-B, the solver file is sent to the remote solver to achieve flexibility in choosing which control application to use when offloading. In this implementation the control application is sent once when the system is started and does not change throughout the experiments duration, we also assumed that the edge node has enough storage capabilities to store the control application. No assumption about the network stability and bandwidth availability where made. The implementation does not preform any offloading until the control application is sent to the remote solver and the remote solver verifies receiving the file and parameters.

In Figure 11, the distribution on how long it took to send the solver file to the remote solver are shown without the outlier points. For each plotbox in the Figure we collected around 10,000 data points. An exception was made for the 5G network where only 1300 data points where collected. It is evident from the plot that sending the file to the edge using the 5G network is the slowest. This has to do with the fact that the network was configured to provide faster communication time where some of the URLLC setting where enabled at the expense of the throughput. The average and worst-case times in milliseconds are presented in Table IV.

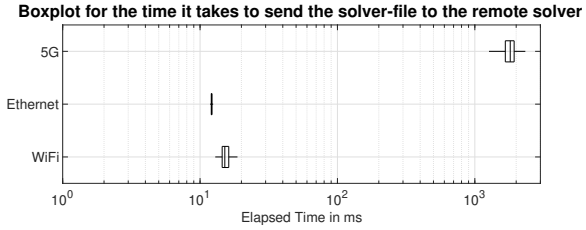

Fig. 11. The time it took to send the the AOT-Wasm file to the remote solver in milliseconds for the different configurations, the Worst-case points were excluded in this figure for all configurations.

In this implementation, the solver was sent using default TCP sockets (the `TCP_NODELAY` flag was activated), no encryption was added to have a secure data transmission. This is sub-optimal since protocols for sending files securely and reliably already exist.

TABLE IV
AVERAGE AND WORST-CASE TIMES OF SENDING THE AOT-WASM FILE FOR THE DIFFERENT CONFIGURATIONS

Scenario	Average Time (ms)	Worst-case Time (ms)
5G	1791	2336
WiFi	16	94
Wired Ethernet	12	265

C. Control Quality

Table V presents the control quality of the different configurations and the fraction of times the remote solver controls the process, following the same naming as in Table II. For each average value, we have collected 10,000 data points.

One can see from Table V that the control quality measure seems to fluctuate even if the fraction of offloading is the same and the same controller is deployed in the remote solver, indicating that the control quality measure used is too simplistic. Two possible reasons for that are the equal weighting of the states and the exclusion of the control signal from the control quality measure presented in Equation 1. The equal weighting might be seen as a crude simplification since the deviation of θ is more severe and should result in a worse control quality compared to the deviation of ϕ. Using the same weights as in the MPC and the LQR would probably provide a better control quality measure. Including the control signal in the cost would also result in a better control quality measure since a large control signal usually which affects the control quality.

D. Demonstration

In Figure 12, we illustrate the pendulum angle, θ, during the dynamic offloading of the pendulum, using 5G to communicate with the edge node. Notably, the pendulum assumes the upright position, as indicated by θ being close to zero. From the figure in the middle, it can be seen which controller is used. Initially, the LQR controller controls the pendulum while the solver file is being sent to the edge. The third graph shows the control quality measure over time. The red dashed

TABLE V
AVERAGE VALUES OF THE CUSTOM COST FUNCTION FOR THE DIFFERENT CONFIGURATIONS

Scenario	Avg. Control Quality Measure	Frac. offloaded
LQR	4.3	0%
5G	2.67	91.05%
WiFi	6.41	97.95%
Wired Ethernet	2.03	97.55%

line represents the threshold used to decide if the remote or the local LQR controls the pendulum. This threshold was determined through experimentation. It can be observed that the threshold is violated sometimes, causing the LQR to take over the control. It is also interesting to note two things. First, the control quality is more oscillatory with more spikes when the LQR controller is controlling the pendulum, indicating that we achieve better control when offloading, compared to controlling it locally. Second, it is evident from the pendulum angle, θ, that the pendulum fluctuates more when controlled using the local LQR compared to the remote MPC which means that the system is controlled better using the edge node compared to the local control.

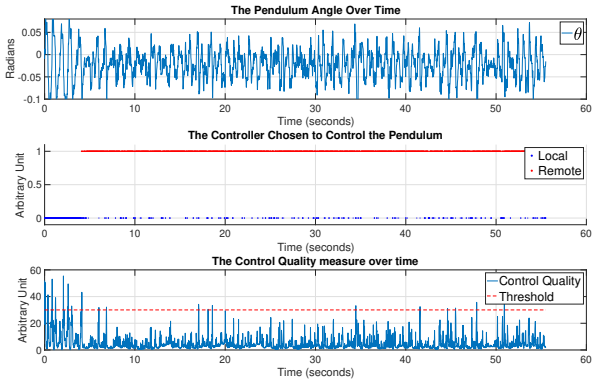

Fig. 12. The first graph shows the pendulum angle, θ, the second graph shows which controller is used to control the pendulum, and the third graph shows the control quality and the threshold used.

V. CONCLUSION

As can be seen from the results, the offloading implementation did work for a different number of configurations.

It was determined that WebAssembly in interpreted form was too slow to control a process with fast dynamics such as the Furuta pendulum, but that AOT-compiled Wasm made it more suitable for this use-case. The paper also found that, although WiFi and Wired Ethernet were faster, the delays of offloading to an edge node using a regular 5G network were deemed short enough for offloading, as it could control the pendulum. Finally, it was also noted that the control quality measure used in this work could be improved by introducing weights to the states and by including the control signal in the control quality measure.

Notably the computational power of the host is sufficient for both the MPC controller and the simple LQR controller. The aim of this work was, however, not to control a system that cannot be controlled by the host computer but to implement an offloading strategy, develop a control quality measure, and to demonstrate its capabilities by controlling a process with fast dynamics. The strategy was tested on an older E51-80 Lenovo Laptop, which could not run the MPC controller locally at a frequency of 66.67 Hz. It was found that offloading worked effectively.

A. Future Work

The offloading strategy presented in this work can be leveraged to explore additional offloading topics and future work.

- In this paper, we assume that the resources at the edge node are always sufficient for executing the MPC solver. However, this is not always the case. The edge node may be a shared resource used by several applications, raising the issue of dynamic resource allocation.
- Another future topic to explore is the impact of network latency when multiple devices are using the network. It would also be valuable to experimentally test the resilience of various communication technologies under conditions where many devices are using the network simultaneously.
- In this work, the offloading strategy was tested on a lab process, and it proved to work well. To further demonstrate the flexibility and usefulness of the implementation, it would be beneficial to control a more practical process using offloaded controllers.

B. Acknowledgements

I would like to thank and acknowledge the AORTA project group, financed by Vinnova, which comprises representatives from Cognibotics AB, Ericsson AB, Lund University, and Mälardalen University for their invaluable support and assistance. I am also grateful to be a part of ELLIIT, which facilitates collaboration among highly talented researchers.

Additionally, I extend my gratitude to everyone who contributed to this work, including Johan Eker, Anders Blomdell, Emil Sundström, Dhiaa Al Bayati, Sundes Al Bayati, and many others at the Department of Automatic Control in Lund University.

REFERENCES

[1] ME Anagnostou, Arto Juhola, and ED Sykas. “Context Aware services as a step to pervasive computing”. In: *Lobster workshop on location based services for accelerating the European-wide deployment of services for the mobile user and worker*. 2002, pp. 4–5.

[2] Anders Blomdell. *Moberg GitLab Repository*. https://gitlab.control.lth.se/anders_blomdell/moberg. 2019.

[3] Xu Chen. “Decentralized computation offloading game for mobile cloud computing”. In: *IEEE Transactions on Parallel and Distributed Systems* 26.4 (2014), pp. 974–983.

[4] Byung-Gon Chun et al. “Clonecloud: elastic execution between mobile device and cloud”. In: *Proceedings of the sixth conference on Computer systems*. 2011, pp. 301–314.

[5] Eduardo Cuervo et al. “Maui: making smartphones last longer with code offload”. In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. 2010, pp. 49–62.

[6] Emscripten. *About Emscripten*. Accessed 2024-02-10. Emscripten, 2024. URL: https://emscripten.org/docs/introducing_emscripten/about_emscripten.html.

[7] Ericsson. *Ericsson - 5G*. Accessed: 2023-12-18. 2023. URL: https://www.ericsson.com/en/5g.

[8] Ericsson. *Ericsson - Edge Computing*. Accessed: 2023-12-18. 2023. URL: https://www.ericsson.com/en/edge-computing.

[9] K Furuta, M Yamakita, and S Kobayashi. “Swing-up Control of Inverted Pendulum Using Pseudo-State Feedback”. In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 206.4 (1992), pp. 263–269. DOI: 10.1243/PIME_PROC_1992_206_341_02.

[10] Magnus Gäfvert. *Modelling the Furuta Pendulum*. Tech. rep. Department of Automatic Control, Lund Institute of Technology, Apr. 1998. ISRN: LUTFD2/TFRT–7574–SE.

[11] Xiaohui Gu et al. “Adaptive offloading inference for delivering applications in pervasive computing environments”. In: *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.(PerCom 2003).* IEEE. 2003, pp. 107–114.

[12] Karim Habak et al. “Femto Clouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge”. In: *2015 IEEE 8th International Conference on Cloud Computing*. IEEE. 2015, pp. 9–16.

[13] Gustav Hansson. *Computation offloading of 5G devices at the Edge using WebAssembly*. 2021.

[14] Wenlu Hu et al. “Quantifying the Impact of Edge Computing on Mobile Applications”. In: *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*. APSys ’16. Hong Kong, Hong Kong: Association for Computing Machinery, 2016. ISBN: 9781450342650. DOI: 10.1145/2967360.2967369. URL: https://doi.org/10.1145/2967360.2967369.

[15] IBM. *IBM Containers*. Accessed: 2023-12-15. 2023. URL: https://www.ibm.com/topics/containers.

[16] Inseego. *What is URLLC?* https://inseego.com/resources/5g-glossary/what-is-urllc/. Accessed on February 2, 2024. 2023.

[17] Kubernetes. *Kubernetes Documentation - Concepts Overview*. 2023. URL: https://kubernetes.io/docs/concepts/overview/.

[18] Kubernetes. *Kubernetes GitHub Repository*. Accessed: 2023-12-18. 2023. URL: https://github.com/kubernetes/kubernetes.

[19] Karthik Kumar et al. “A survey of computation offloading for mobile systems”. In: *Mobile networks and Applications* 18 (2012), pp. 129–140.

[20] Huibert Kwakernaak and Raphael Sivan. *Linear Optimal Control Systems*. 1972. Chap. 6.4.

[21] Borui Li, Wei Dong, and Yi Gao. “Wiprog: A webassembly-based approach to integrated iot programming”. In: *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE. 2021, pp. 1–10.

[22] MathWorks. *dlqr Function Documentation*. Accessed:2023-12-28. 2023. URL: https://se.mathworks.com/help/control/ref/dlqr.html.

[23] Jacob Mattingley and Stephen Boyd. “CVXGEN: a code generator for embedded convex optimization”. In: *Optimization and Engineering* 13.1 (Mar. 2012), pp. 1–27.

[24] MDN. *WebAssembly*. https://developer.mozilla.org/en-US/docs/WebAssembly. Accessed: 11 17, 2023. 2023.

[25] Pangun Park et al. “Wireless network design for control systems: A survey”. In: *IEEE Communications Surveys & Tutorials* 20.2 (2017), pp. 978–1013.

[26] Haorui Peng. “Tamin Cloud Integrated Systems in the Wild”. PhD thesis. Department of Electrical and Information Technology, Lund University, 2023.

[27] Harry Pigot. *Furuta Pendulum 2019 README*. https://gitlab.control.lth.se/processes/furutapendulum2019/-/blob/master/README.md. Accessed on 2024-02-10. 2021.

[28] James B. Rawlings, David Q. Mayne, and Moritz M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. 2nd. Nob Hill Publishing, 2017.

[29] Sebastian Schlor and Frank Allgöwer. “Bootstrapping Guarantees: Stability and Performance Analysis for Dynamic Encrypted Control”. In: *arXiv preprint arXiv:2403.18571* (2024).

[30] W. C. Schultz and V. C. Rideout. “Control system performance measures: Past, present, and future”. In: *IRE Transactions on Automatic Control* AC-6.1 (1961), pp. 22–35. DOI: 10.1109/TAC.1961.6429306.

[31] Per Skarin. “Control over the Cloud: Offloading, Elastic Computing, and Predictive Control”. PhD thesis. Department of Automatic Control, Lund University, 2021.

[32] David Umsonst and Fernando S. Barbosa. “Remote Tube-based MPC for Tracking over Lossy Networks”. In: *CDC2024 Conference*. IEEE. 2024.

[33] WasmEdge. *WasmEdge Documentation*. Accessed on 2024-03-26. 2023. URL: https://wasmedge.org/docs/develop/deploy/intro (visited on 03/26/2024).

[34] WasmEdge. *WasmEdge Documentation The AoT Compiler*. Accessed: 2023-12-15. 2023. URL: https://wasmedge.org/docs/start/build-and-run/aot/.

[35] *WasmEdge Documentation Overview*. Accessed: 2023-12-15. 2023. URL: https://wasmedge.org/docs/start/overview.

[36] WebAssembly. *Overview*. https://webassembly.org/. Accessed: 02 15, 2024. 2023.