Cloud-based Ultrasound System for Real-time Advanced Image Formation and Image Settings Autotuning

Citation for published version (APA):

Document status and date:
Published: 15/02/2023

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Real-time Cloud-based Ultrasound System for Advanced Image Formation and Image Settings Autotuning

Supervisors

Prof. Dr. Ir. Massimo Mischi, Eindhoven University of Technology
Dr. Ir. Ruud van Sloun, Eindhoven University of Technology
Dr. Ir. Andre Immink, Philips Engineering Solutions

EngD Candidate
Beatrice Federici

Graduation: February 2023
Thesis Number: 2023/030
Status: Temporarily Confidential (until 31-12-2027)
The design described in this thesis has been carried out in accordance with the TU/e Code of Scientific Conduct.

The project is sponsored by the Dutch Research Council (NWO), within the High Tech Systemen en Materialen (HTSM) Programme (project: SPICE 17144), and carried out in collaboration with Philips Eindhoven and Eindhoven Catharina Ziekenhuis.
# Contents

<table>
<thead>
<tr>
<th>Summary</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Introduction</strong></td>
<td>7</td>
</tr>
<tr>
<td>1 Background &amp; Motivation</td>
<td>7</td>
</tr>
<tr>
<td>1.1 Ultrasound in Image-guided Therapy</td>
<td>7</td>
</tr>
<tr>
<td>1.2 Ultrasound in Catheter Ablation Procedure</td>
<td>8</td>
</tr>
<tr>
<td>2 Objective</td>
<td>8</td>
</tr>
<tr>
<td>2.1 Project Workflow</td>
<td>9</td>
</tr>
<tr>
<td><strong>2 Conceptual Study</strong></td>
<td>11</td>
</tr>
<tr>
<td>1 End Users’ Expectations</td>
<td>11</td>
</tr>
<tr>
<td>2 Design Constraints</td>
<td>12</td>
</tr>
<tr>
<td>2.1 Application</td>
<td>12</td>
</tr>
<tr>
<td>2.2 Regulatory</td>
<td>13</td>
</tr>
<tr>
<td>3 Measure of Effectiveness</td>
<td>14</td>
</tr>
<tr>
<td>4 Analysis of Alternatives</td>
<td>14</td>
</tr>
<tr>
<td>5 Conceptual Design</td>
<td>16</td>
</tr>
<tr>
<td><strong>3 System Design</strong></td>
<td>17</td>
</tr>
<tr>
<td>1 Preliminary Design</td>
<td>17</td>
</tr>
<tr>
<td>1.1 System Requirements</td>
<td>17</td>
</tr>
<tr>
<td>1.2 Pre-defined Interacting Systems</td>
<td>18</td>
</tr>
<tr>
<td>1.3 System Architecture</td>
<td>19</td>
</tr>
<tr>
<td>2 Logical Decomposition</td>
<td>19</td>
</tr>
<tr>
<td>2.1 Architecture Breakdown</td>
<td>20</td>
</tr>
<tr>
<td>2.1.1 Sub-systems Architecture</td>
<td>20</td>
</tr>
<tr>
<td>2.1.2 Timing Diagram</td>
<td>24</td>
</tr>
<tr>
<td>2.2 Requirements Breakdown</td>
<td>28</td>
</tr>
<tr>
<td>2.2.1 Technical Budgeting</td>
<td>28</td>
</tr>
<tr>
<td>2.2.2 Sub-systems Requirements</td>
<td>30</td>
</tr>
<tr>
<td><strong>4 System Verification</strong></td>
<td>35</td>
</tr>
<tr>
<td>1 Sub-systems Verification</td>
<td>35</td>
</tr>
<tr>
<td>2 System Integration</td>
<td>41</td>
</tr>
<tr>
<td>3 System Verification</td>
<td>42</td>
</tr>
<tr>
<td><strong>5 Discussion</strong></td>
<td>44</td>
</tr>
<tr>
<td>1 Key Outcomes</td>
<td>44</td>
</tr>
<tr>
<td>2 Limitations &amp; Potential Improvements</td>
<td>45</td>
</tr>
<tr>
<td>3 Towards Cloud-based Ultrasound</td>
<td>49</td>
</tr>
<tr>
<td><strong>6 Conclusions</strong></td>
<td>51</td>
</tr>
</tbody>
</table>
A Supplementary Data

1 Communication Protocol .......................................................... 52
  1.1 Materials & Methods ............................................................ 52
     1.1.1 Programming .............................................................. 52
     1.1.2 Performance Assessment .................................................. 53
  1.2 Results & Discussion ............................................................ 55
     1.2.1 Network-level .............................................................. 55
     1.2.2 Application-level .......................................................... 55
  1.3 Conclusions ............................................................................ 59

2 Image Settings Optimization during Run-time .................................. 61
  2.1 Materials & Methods ............................................................ 61
     2.1.1 Programming .............................................................. 61
     2.1.2 Performance Assessment .................................................. 61
  2.2 Results & Discussion ............................................................ 62
  2.3 Conclusions ............................................................................ 64

B Supplementary Info ..................................................................... 66

1 Inference Deployment Environments .............................................. 66
  1.1 Inference Performance ............................................................ 66
  1.2 Deployment Environments ...................................................... 66

2 Network ....................................................................................... 69
  2.1 Network Types ....................................................................... 69
  2.2 Reference Model ..................................................................... 69
  2.3 Socket ..................................................................................... 71

3 Equipment .................................................................................... 72
  3.1 Verasonics Vantage Research System ....................................... 72

4 Theoretical Derivation .................................................................. 75
  4.1 Quadrature Sampling .............................................................. 75
     4.1.1 Second-Order Method Approximating Quadrature Sampling .. 77
  4.2 Digital Proportional-Integral-Derivative Controller ..................... 79

C Catheter Ablation Procedure ......................................................... 80

1 Background ................................................................................... 80
  1.1 Cardiac Physiology .................................................................. 80
     1.1.1 Cardiac Heterogeneity ......................................................... 80
  1.2 Cardiac Arrhythmias ............................................................... 81
     1.2.1 Classification .................................................................... 81
     1.2.2 Electrophysiological mechanisms ...................................... 82
     1.2.3 Current Treatments ............................................................ 83
  1.3 Catheter Ablation Therapy ....................................................... 84
     1.3.1 Multimodality Imaging for Guiding Ablation Procedures ....... 85

2 Clinical Needs .............................................................................. 89

References ....................................................................................... 93

Acronyms ....................................................................................... 99
**List of Figures**

1. Project workflow. ................................................................. 10
2. Conceptual diagram. ............................................................... 16
3. Preliminary system architecture. ................................................. 19
4. System architecture. ............................................................... 20
5. Network diagram. ................................................................. 22
6. Adaptive beamforming by deep learning. .................................... 23
7. Basic principles of reinforcement learning. .................................. 24
8. Synchronous vs asynchronous data writing/reading over the network. .... 26
9. System sequence diagram. ....................................................... 27
10. Timing Diagram. ................................................................. 30
11. Sub-system performance: data transfer rate. ................................. 36
12. System performance: image quality. ........................................... 37
13. Run-time parameters update. .................................................. 40
14. System integration. .............................................................. 41
15. System performance: parallel processes frame rate. ....................... 43
16. Reinforcement learning in a delayed environment. ......................... 46
17. Rate-distortion characteristic. ............................................... 47
18. Transmission time-model complexity characteristic. ...................... 48
19. Envisioned cloud-based ultrasound system for image-guided therapy. .... 49
20. Envisioned cloud-based ultrasound system for point of care ultrasound. .. 50
21. Network diagram. .............................................................. 53
22. Round-Trip-Time. ............................................................... 57
23. Application throughput: protocols. .......................................... 57
25. User datagram protocol packet loss. ......................................... 59
26. Elapsed time for run-time parameters update: tuned parameter. ........ 63
27. Elapsed time for run-time parameters update: manual vs automatic. .... 63
28. Elapsed time for run-time parameters update: simultaneous tuning of multiple parameters. ........................................... 64
29. Elapsed time for run-time parameters update: affecting factors. ........ 65
30. Computer network reference model. ......................................... 70
31. Data flow. .................................................................. 72
32. Verasonics research system. ................................................... 72
33. Verasonics timing diagram. .................................................... 74
34. Quadrature sampling. ............................................................ 75
35. In-phase quadrature demodulation. ........................................... 76
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>Second-order method approximating quadrature sampling.</td>
<td>77</td>
</tr>
<tr>
<td>37</td>
<td>Re-entry circuit in scar tissue.</td>
<td>83</td>
</tr>
<tr>
<td>38</td>
<td>Catheter ablation therapy.</td>
<td>84</td>
</tr>
<tr>
<td>39</td>
<td>Myocardial mapping by magnetic resonance imaging.</td>
<td>87</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Clinical needs & envisioned system goals and objectives. ......................... 11
2.2 Application constraints. ................................................................. 12
2.3 Envisioned system regulatory constraints. ............................................ 13
2.4 System functionalities allocation for different alternative solutions. ............ 14
2.5 Analysis of alternatives. ................................................................. 15

3.1 System requirements. ................................................................. 17
3.2 Technical budgeting: display latency. ................................................. 28
3.3 Technical budgeting: inter-frame period. ............................................ 29
3.4 Sub-systems level requirements. ....................................................... 31

4.1 Sub-system performance: data transfer run-time. ................................. 36
4.2 Sub-system performance: image quality. .......................................... 37
4.3 System performance: processing run-time. ...................................... 38
4.4 System performance: display latency. .............................................. 42
4.5 System performance: parallel processes frame rate. ............................ 43

A.1 Round-Trip-Time. ................................................................. 57

B.1 Inference performance factors. ...................................................... 66
B.2 Inference deployment environments. ............................................... 67

C.1 Mechanism of cardiac arrhythmias .................................................. 82
C.2 Clinical needs & envisioned system goals and objectives. ....................... 89
Image-guided therapy is defined as the use of imaging to assist, monitor and control medical treatments. In recent decades, most traditionally open and invasive surgeries have been replaced by less invasive procedures that make use of the information provided by imaging systems. Similarly, many non-critical interventions previously performed without visual guidance are now supported by imaging devices. Image-guided therapy has been shown to reduce peri-procedural complications of critical procedures, along with associated morbidity and mortality, as well as decrease the overall healthcare expenditure, due to, for example, shorter hospitalizations and a lower number of re-intervention. To support this upward trend towards image-guided procedures, imaging systems must provide physicians with increasingly accurate information within a pre-defined minimal latency.

In this context, ultrasound imaging stands out among other imaging modalities, mainly due to its unique real-time nature and harmlessness. However, the widespread adoption of this imaging modality for image-guided therapy is hampered by the extensive training and experience required to acquire good quality ultrasound images, as well as the relatively limited image quality that prevents the extraction of accurate clinical indices through quantitative analysis. Advanced processing methods, e.g., deep learning-based, offer unprecedented opportunities to address these limitations, allowing easier use of ultrasound systems through image settings autotuning and improved image quality, but they drastically increase the computational footprint.

This thesis aims to explore the use of a distributed architecture to support the deployment of compute-intensive processing methods in latency-sensitive applications, such as image-guided therapy, without imposing strict requirements in terms of cost, power consumption and size on the ultrasound imaging device. The proposed solution includes (1) live-streaming of raw, high-bandwidth, ultrasound channel data from the on-site acquisition device to the cloud, where data processing takes place, and (2) a real-time feedback loop for image rendering and control of image settings. To assess the feasibility of this envisioned system, a prototype is designed and developed. The performance assessment demonstrates real-time cloud-based ultrasound image formation with a trained deep learning based image formation method, and dynamic autotuning of the transmit voltage according to the results of a controller deployed on the cloud. Besides immediate use of the developed platform for rapid prototyping of compute-intensive ultrasound processing methods, the prototype provides important insights for the design of future commercial cloud-based ultrasound imaging systems.
Chapter 1

Introduction

1 Background & Motivation

1.1 Ultrasound in Image-guided Therapy

Image-guided therapy (IGT) is defined as the use of imaging to assist, monitor and control treatments. Over the past decade, IGT has replaced several traditionally open surgeries and interventions as it offers the physicians the opportunity to minimize procedure invasiveness, along with morbidity and mortality, as well as the length of hospitalization and healthcare expenditure. This trend is expected to continue in the coming years, as highlighted by the predicted increment in the global IGT systems market size: from USD 5.1 billion in 2021 to around USD 8.9 billion by 2030 [1]. The raise in minimally invasive interventions requires increasingly accurate imaging technologies capable of providing the necessary information with minimal latency to the physicians. In that regard, X-ray fluoroscopy and ultrasound (US) imaging stand out among other modalities for their real-time nature. X-ray is the most common modality used for interventional guidance with about 50% of all imaging exams [2], but its use is being reduced due to the harmful effect of radiation [3]. On the contrary, US is being approved as gold standard imaging modality for an increasing number of procedures thanks to its non-invasiveness, relative low cost, and its compactness that allows comfortable use even during interventional procedures [4, 5].

Despite its ever more prominent role in IGT, the adoption of US as assisting imaging device remains limited by: first and foremost, the long training and in-depth experience required to acquire high-quality US images [6], and second by the limited image quality with real-time processing methods when compared to other imaging modalities. The shallow learning curve is mainly associated with the in-depth knowledge (‘knobology’) of the device functions and examination technique needed to achieve the correct settings adjustments for the best image quality [7]. In that regard, in [8] it is estimated that, for the majority of examination types, about 75 examinations are required to reach excellent interpretation and good image quality. As for the limited image quality, this is especially affected during ultra-fast unfocused transmissions, such as Plane Wave (PW) [9], or Synthetic Aperture [10] imaging. These unfocused transmissions are getting more and more common in US, as they can offer unprecedented opportunities for clinical applications where high temporal resolution is needed (e.g., cardiac imaging)\(^1\). To compensate for the unfocused transmission, complex processing methods (e.g., Minimum Variance, or Capon beamformer [11]) have been proposed, but are characterized by a long reconstruction time which prevents their use in latency-sensitive applications (i.e., real-time use).

\(^1\)In focused sonification strategies narrow scan lines are sequentially fired by applying focusing time delays to the active transducer elements. The increase of the frame rate is achieved through the reduction of the number of lines in transmit, which either reduces the sector size or the line density of the image. Unfocused techniques offer a unique ability to increase the acquisition frame rate without affecting the number of scan lines and the image size.
CHAPTER 1. INTRODUCTION

To overcome these shortcomings and support optimal and safe use of US during interventions, technology and software advances are urgently needed.

1.2 Ultrasound in Catheter Ablation Procedure

An illustrative example of a use case scenario where overcoming the aforementioned limitations would have a huge impact is catheter ablation procedure (for details about catheter ablation treatment and associated clinical needs, please refer to Appendix C).

Catheter ablation therapy represents an essential treatment for cardiac arrhythmias, including supraventricular tachycardia, atrial flutter, atrial fibrillation and ventricular tachycardia [12, 13, 14, 15, 16]. When the arrhythmia has a well-defined electrophysiologic mechanism and the target region is clearly localized, a successful rate around 90 to 95% is reported [17]. However, when the arrhythmia presents a more complex mechanisms, such as in atrial fibrillation, after-treatment recurrence is observed in 50-70% of patients [18, 19] and force repeating the surgical operation. This lower arrhythmia-free survival rate is likely associated to difficulties in mapping the arrhythmogenic substrate, and to the inability to monitor the lesion formation during the procedure. It follows that imaging could have a key role in improving the procedural outcomes. However, despite the impressive advancements observed over the last decades (Sec. 1.3.1 in Appendix C), an effective solution has yet to be found.

Echocardiography is likely the most promising of all imaging modalities for procedural guidance and lesion monitoring thanks to its unique ability to provide real-time information, its non-invasiveness, and its compactness that allows comfortable use even during interventions. Nevertheless, the limitations mentioned in the previous paragraph are still hampering its widespread adoption as a gold-standard tool for image-guided catheter ablation treatment.

2 Objective

This thesis work is part of the SPICE Project, which aims to tackle the aforementioned US imaging shortcomings by proposing:

- a content-adaptive image formation method able to reconstruct high quality images in real-time from high-frame-rate unfocused transmissions.

- an auto-tuning strategy to simplify the use of US systems by optimizing image settings and provide at any time the best trade-off between various imaging properties, disturbing artefacts, and system-level constraints on power and data-rates.

The deployment of these advanced methods, typically characterized by a large computational footprint, requires the development of a novel US imaging system able to allocate these compute-intensive algorithms without imposing strict requirements on the on-site device or compromising the interactive nature needed by latency-sensitive applications, like the hand-eye coordination required in IGT. The design, realization and verification of this deployment platform constitutes the primary focus of this Engineering Doctorate (EngD) thesis [20].

---

2The SPICE project is funded by the Dutch Research Council (NWO), within the High Tech Systemen en Materialen (HTSM) Programme - Grants NWO Domain Applied and Engineering Sciences December 2018. On the industrial side, the project is supported by Philips Research, as part of the Eindhoven MedTech Innovation Center partnership. On the clinical side, the project is supported by the Catharina Hospital in Eindhoven.
2.1 Project Workflow

Fig. 1 summarizes the project workflow and its main phases:

- **Conceptual study.** Conceptual study includes user’s needs exploration and possible design solutions comparison, followed by the definition of a conceptual design.

- **Technology demonstrator development.** The second phase leads to the development of a technology demonstrator, i.e., a prototype able to prove the feasibility of the proposed conceptual design. In this work, the technology demonstrator includes a framework and two data processing methods. Each component is developed in parallel, with the processing algorithms implemented by two colleagues within the SPICE Project and the framework constructed as part of this EngD thesis. The latter alternates design solution proposition, evaluation, and re-design. It ultimately returns the final proposed design solution. When both the framework and the processing methods are completed, the components are integrated in a full-stacked working system and the system requirements are verified.

- **Critical analysis.** Critical analysis refers to an analytic discussion of the technology demonstrator performance. This analysis is conducted in view of the final application, and it aims to expand on the conceptual study initially conducted to refine and detail the envisioned system.
Figure 1: Project workflow.
Chapter 2

Conceptual Study

Conceptual study is the process that clarifies the system that is meant to be designed and developed. It involves users’ needs and constraints analysis followed by the exploration of different alternative solutions, typically compared in terms of cost, risk, and effectiveness. The final outcome of this phase is a conceptual design.

Note that this conceptual study is performed considering the development of an US system for IGT, such as US-assisted catheter ablation procedure.

1 End Users’ Expectations

Tab. 2.1 captures the users’ needs and the derived system goals. Needs are defined to answer the question “What problem are we trying to solve?”; goals address what must be done to meet the needs; and objectives expand on the goals and provide measurable and verifiable expectations.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N1: High quality images to improve image reading and quantitative measurements with unfocused acquisitions.</td>
<td>G1 - 1. Improve the performance of image reconstruction algorithms (beamforming), w.r.t. traditional approaches with unfocused acquisitions (e.g., plane wave imaging), without compromising the interactive nature of ultrasound imaging.</td>
<td>O1 - 1.1 Real-time plane wave image formation with image quality better than traditional DAS beamforming.</td>
<td>Real-time nature of ultrasonography imposes strong constraints on reconstruction speed of the processing algorithms. Traditional methods (e.g., DAS) have required temporal resolution but low image quality, especially with unfocused transmissions needed for most advanced post-processing algorithms. It has been reported that approximately 10% to 15% of routine echocardiograms have poor image quality [21].</td>
</tr>
<tr>
<td>N2: Simplify US system usage.</td>
<td>G2 - 1. Reduce manual settings to obtain optimal image settings.</td>
<td>O2 - 1.1 Adjust automatically transmission scheme to improve image quality.</td>
<td>Artefacts are frequently encountered during interventional procedures. E.g., comet-tail artefacts due to catheters or pacing wires [22]; blooming artefacts due to metal wires; reverberations due to rib cage, lungs or thick fat layer [23].</td>
</tr>
</tbody>
</table>

Table 2.1: Clinical needs & envisioned system goals and objectives.
CHAPTER 2. CONCEPTUAL STUDY

<table>
<thead>
<tr>
<th>O2 - 1.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjust automatically reconstruction parameters to improve image quality.</td>
</tr>
</tbody>
</table>

Receive parameters could be optimized depending on image content.

2 Design Constraints

2.1 Application

Tab. 2.2 lists a few of the application-related constraints that might affect the design solution.

<table>
<thead>
<tr>
<th>Application Constraints</th>
<th>Technological Implications</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA1 Minimal interference of the system with the intervention.</td>
<td>CA1 - 1 Minimize device size (e.g., portable systems).</td>
<td>Compact systems facilitate the integration in the operating room. Typical dimension for echocardiograph 50x50x120 cm.</td>
</tr>
<tr>
<td></td>
<td>CA1 - 2 Minimize cables (e.g., wireless solutions).</td>
<td>Fewer cables are associated with more comfortable exams, especially in the field of interventional imaging. Also, cleaning of cables after procedures is deemed a problem. Ultimately, cables (and connectors) turn out to be the main sources of system failure.</td>
</tr>
<tr>
<td>CA2 Adaptability.</td>
<td>CA2 - 1 Solution with easy-to-scale compute resources to accommodate novel software for image formation or post-processing.</td>
<td>Different models require different compute resources, memory, specifications.</td>
</tr>
<tr>
<td></td>
<td>CA2 - 2 Easy-to-program solution to accommodate novel software for image formation or post-processing.</td>
<td>Processing methods need to be continuously updated (i.e., progress of technologies, increase of data). Update requires programing the processing unit, this might take longer if the software needs to be updated in multiple locations rather than updating it in one single centralized location.</td>
</tr>
<tr>
<td>CA3 Cost-effectiveness.</td>
<td>CA3 - 1 Limit solution cost (initial investment, maintenance, upgrade).</td>
<td>Affordable solutions favour the widespread use of the technology and reduce the overall Healthcare Expenditure. In that regard, the overall cost estimate includes initial cost (e.g., fewer server nodes cost less) and maintenance cost (i.e., energy cost to power and cool the servers throughout their lifecycle).</td>
</tr>
</tbody>
</table>
### CHAPTER 2. CONCEPTUAL STUDY

#### 2.2 Regulatory

Tab. 2.3 summarizes the most relevant regulatory constraints that have to be considered for the envisioned application.

<table>
<thead>
<tr>
<th>Regulatory Constraints</th>
<th>Technological Implications</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CR1 Safety</strong></td>
<td>CR1 - 1. Ensure safe use of transmitted power and scan time.</td>
<td>Acquisition scheme has to be compliant with the ALARA Principle(^1), and within the approved limits for mechanical index (MI) and thermal index (TI)(^2). To date, there is no evidence that diagnostic ultrasound can produce any harm to humans. However, as it involves the deposition of energy in the body, its use requires a good knowledge of the effect of certain settings on potential thermal and mechanical bio-effects.</td>
</tr>
<tr>
<td></td>
<td>CR1 - 2. Ensure electromagnetic compatibility (EMC)(^3).</td>
<td>Electrically powered medical devices and medical devices with electrical or electronic functions have to demonstrate EMC.</td>
</tr>
<tr>
<td></td>
<td>CR1 - 3. Thermal probe temperature below 43(^{\circ})C.</td>
<td>Surface temperature of a device in direct contact with the patient must be kept below 43(^{\circ})C in order to comply with medical safety regulations (IEC 60601-1 [24])</td>
</tr>
<tr>
<td><strong>CR2 Security</strong></td>
<td>CR2 - 1 Preserve privacy of sensitive data.</td>
<td>In the United States, patients’ data must be treated according to the Health Insurance Portability and Accountability Act Privacy Rule (i.e., national standards to protect sensitive patient health information from being disclosed without the patient’s consent or knowledge). In Europe, the General Data Protection Regulation (GDPR) sets the rules for the processing and free movement of personal data and applies to all domains of the public and private sector.</td>
</tr>
<tr>
<td></td>
<td>CR2 - 2 Ensure an acceptable level of cybersecurity.</td>
<td>In Europe, the Network and Information Security Directive (NISD) 2016/1148/EU imposes: the implementation of minimum security requirements and the establishment of cybersecurity notifications for both Operators of Essential Services (e.g., hospitals) and Digital Service Providers (e.g., cloud service providers) [25].</td>
</tr>
<tr>
<td><strong>CR3 Reliability</strong></td>
<td>CR3 - 1 Operate also in disconnected environment or with power outages.</td>
<td>Critical devices used during intervention are &quot;always-on&quot; systems.</td>
</tr>
<tr>
<td></td>
<td>CR3 - 2 Assurance of minimum quality standards.</td>
<td>Quality assurance usually focuses on maintaining a quality level based on standards set by an external body and enforced through accreditation or certification combined with regular inspections. e.g., ISO 9000-9002 standards, Joint Commission on Accreditation of Healthcare Organizations-Joint Commission International accreditation(^4).</td>
</tr>
</tbody>
</table>

---

\(^1\) ALARA Principle: fundamental approach to the safe use is to use the lowest output power and the shortest scan time consistent with acquiring the required diagnostic information.

\(^2\) Local temperature should not increase more than 1-3\(^{\circ}\)C, according to the CEI EN 45502-1 norm.

\(^3\) Electromagnetic compatibility is defined as the ability of a medical device to function safely and effectively in its intended electromagnetic environment, including immunity to EM disturbances (i.e., interference), without introducing excessive EM disturbances (i.e., emissions) that might interfere with other equipment.

\(^4\) This has important implications for instance on data transmission (e.g., transfer of medical images) being highly loss intolerant data.
3 Measure of Effectiveness

*Measures Of Effectiveness* (MOE) are the measures of success that are designed to correspond to accomplishment of the system objectives. MOE are stated from the stakeholders’ point of view and represent criteria that are to be met to consider the project successful.

MOE for the proposed system include:

- **Display latency**: time between data acquisition and display shall be minimized to maintain ultrasound interactive nature during interventional procedure.

- **Inter-frame period**: time between two consecutive frames shall be such that the human eyes cannot notice the discontinuities between subsequent frames, or - at least - that this perceived discontinuities don’t affect probe handling.

- **Image quality**: spatial resolution, contrast, artefacts shall be improved w.r.t. traditional approaches (e.g., DAS).

- **Autotuning efficiency**: image quality (e.g., spatial resolution, contrast, artefacts) shall be improved w.r.t. to previous frame when using the self-driving paradigm.

Display latency, inter-frame period, image quality and autotuning efficiency are called *Critical To Quality parameters* (CtQs).

4 Analysis of Alternatives

*Analysis Of Alternatives* (AoA) is the identification of alternative technologies that can obtain the defined system objectives. Three manufacturable alternative solutions are identified for the desired deployment platform. The alternatives differ in terms of: (1) processing methods (e.g., basic vs advanced), and (2) system architecture (i.e., allocation of different image chain functionalities to different system components). Note that (1) and (2) are not independent, as advanced methods include complex algorithms which cannot be implemented on hardware (i.e., embedded) and that typically require large computational power available with architecture like edge computing or cloud computing. In that regard, a detailed description of the landscape of deployment platforms available for advanced methods inference is provided in Appendix B Section B.1.

Tab. 2.4 shows the three considered alternatives and the main system functionalities allocation to architecture components. Main system functionalities include: (1) US wave transmit/echo receive (TX/RX), (2) digitization or sampling, (3) processing like image formation or learning controller, (4) storage, (5) image rendering.

<table>
<thead>
<tr>
<th>Alternatives</th>
<th>TX/RX</th>
<th>Digitization</th>
<th>Processing</th>
<th>Storage</th>
<th>Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Embedded w. basic methods</td>
<td>Probe</td>
<td>Probe</td>
<td>Probe</td>
<td>NA</td>
<td>Display</td>
</tr>
<tr>
<td>2: Edge computing w. advanced methods</td>
<td>Probe</td>
<td>Probe/On-site PC (e.g., console US)</td>
<td>On-site servers</td>
<td>On-site servers</td>
<td>Display</td>
</tr>
<tr>
<td>3: Cloud computing w. advanced methods</td>
<td>Probe</td>
<td>Probe/On-site PC (e.g., console US)</td>
<td>Cloud</td>
<td>Cloud</td>
<td>Display</td>
</tr>
</tbody>
</table>

Alternatives are assessed against screening criteria. The *Pugh Matrix* is reported in Tab. 2.5. Screening criteria include needs-derived MoE and constraints. For each criterion, a vote between 0 and 3 is assigned to each design alternative solution based on literature (i.e., 0 corresponds to very poor solution w.r.t. that specific criterion). The total score is computed considering

---

**Table 2.4: System functionalities allocation for different alternative solutions.**

<table>
<thead>
<tr>
<th>Alternatives</th>
<th>TX/RX</th>
<th>Digitization</th>
<th>Processing</th>
<th>Storage</th>
<th>Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Embedded w. basic methods</td>
<td>Probe</td>
<td>Probe</td>
<td>Probe</td>
<td>NA</td>
<td>Display</td>
</tr>
<tr>
<td>2: Edge computing w. advanced methods</td>
<td>Probe</td>
<td>Probe/On-site PC (e.g., console US)</td>
<td>On-site servers</td>
<td>On-site servers</td>
<td>Display</td>
</tr>
<tr>
<td>3: Cloud computing w. advanced methods</td>
<td>Probe</td>
<td>Probe/On-site PC (e.g., console US)</td>
<td>Cloud</td>
<td>Cloud</td>
<td>Display</td>
</tr>
</tbody>
</table>
the given scores and the weight defined for each screening criterion. Weights are assigned according to the degree of importance of each criterion for the envisioned application (i.e., in a trade-off some expectations could be compromised more than others). Where the score requires further clarification, details are indicated within brackets.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>ID</th>
<th>Weight</th>
<th>Embedded</th>
<th>Edge computing</th>
<th>Cloud computing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display latency</td>
<td>N1</td>
<td>1</td>
<td>3</td>
<td>2 (time to transfer to edge servers)</td>
<td>1 (time to transfer to cloud over network)</td>
</tr>
<tr>
<td>Inter-frame period</td>
<td>N1</td>
<td>1</td>
<td>3</td>
<td>1 (advanced methods long run-time)</td>
<td>2 (advanced methods long run-time but on-demand cloud compute power)</td>
</tr>
<tr>
<td>Image quality</td>
<td>N1</td>
<td>1</td>
<td>1</td>
<td>3 (advance methods capacity)</td>
<td>3 (advance methods capacity)</td>
</tr>
<tr>
<td>Autotuning efficiency</td>
<td>N2</td>
<td>1</td>
<td>1</td>
<td>3 (advance methods capacity)</td>
<td>3 (advance methods capacity)</td>
</tr>
<tr>
<td>Minimal interference</td>
<td>CA1</td>
<td>0.7</td>
<td>3</td>
<td>1 (large size of on-site device)</td>
<td>3</td>
</tr>
<tr>
<td>Adaptability</td>
<td>CA2</td>
<td>0.7</td>
<td>0</td>
<td>2</td>
<td>3 (easy-to-scale or upgrade)</td>
</tr>
<tr>
<td>Cost-effectiveness</td>
<td>CA3</td>
<td>0.9</td>
<td>1</td>
<td>1</td>
<td>3 (on-demand resources)</td>
</tr>
<tr>
<td>Safety</td>
<td>CR1</td>
<td>1</td>
<td>1 (high power dissipated within probe)</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Security</td>
<td>CR2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2 (cyberattacks)</td>
</tr>
<tr>
<td>Reliability</td>
<td>CR2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1 (disconnected environment or with power outages)</td>
</tr>
<tr>
<td>Total Score:</td>
<td></td>
<td></td>
<td></td>
<td>17.1</td>
<td>20.3</td>
</tr>
</tbody>
</table>

Although the scores are pretty close to each other, the analysis shows that cloud computing stands out for its unique cost-effectiveness and adaptability. Limitations of this solution remain: (1) display latency, as transferring high-bandwidth US channel data over the network take longer than processing them in the proximity of data source; and (2) reliability, as the solution could only work in connected environment and might stop to work unexpectedly with power outages. To tackle the latter shortcoming, hybrid solutions could be considered which allocate default advanced processing on the cloud, but keep a basic solution on-site (i.e., either on the probe or on a local PC) for sudden Internet drops. With respect to the downsides related to network bandwidth, the latency is expected to decrease with upcoming communication technologies, like 6G wireless communications technologies, which provide larger bandwidth.
5 Conceptual Design

According to the AoA, the envisioned system is a hybrid system which combines advantages of cloud computing with those of local processing (Fig. 2). The system is expected to:

- acquire US channel data and digitize US channel data locally,
- in default mode: send US channel data over the network to the application server on the cloud and process data through advanced image formation algorithms and image settings optimization strategies,
- in emergency (i.e., Internet drop) mode: process channel data on-site through basic processing methods,
- render images to an on-site or remote display,
- enable manual control of run-time acquisition parameters from the user interface,
- enable automatic control of run-time acquisition parameters from the cloud-based auto-tuning strategy.

On-site basic processing is already common in clinics, either using a standard console device or with probe-based beamforming. On the contrary, cloud-based raw data processing, and real-time feedback have so far only been little explored\(^5\). As a consequence, the technology demonstrator developed in this work focuses exclusively on the default mode of the envisioned system presented in Fig. 2.

---

\(^5\)Current commercial cloud-based US solutions support only image (and not raw channel data) sharing and secure archiving to enable collaboration among physicians and remote reviewing [26].
Chapter 3

System Design

System design involves the definition of high level system requirements and a preliminary design (i.e., high level architecture), followed by an iterative logical decomposition (i.e., requirements and architecture breakdown) through design solutions proposition, development, evaluation, and refinement.

1 Preliminary Design

1.1 System Requirements

System-level requirements follow system objectives (functional requirements) defined in Chapter 2 Section 2.1 and CtQ parameters (performance requirements) listed in Section 2.3.

By definition, requirements should be verifiable, hence the generic expectations listed during the Conceptual Study are here detailed and refined. For instance, PR3 translates the more generic “image quality” into image resolution and contrast-to-noise ratio, used here as standard image quality indicators. Other quality metrics might be considered in the future. Remarkably, PR4 remains generic (i.e., to be defined) due to the low readiness technology level (RTL) of the autotuning strategy which prevent the refinement of the MoE: the image quality metric to be used to evaluate the improvement given by the self-driving paradigm is strongly dependent on the parameters that the autotuning strategy learns to control.

Table 3.1: System requirements. ID: All requirements have a unique ID. N/C reference: refer to the unique ID of the users' need listed in Tab. 2.1. Verification method: indicate the planned verification approach, can be inspection, analysis, demonstration, or test.

<table>
<thead>
<tr>
<th>ID</th>
<th>N/C Ref.</th>
<th>Requirement</th>
<th>Rationale</th>
<th>Verification method</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR1</td>
<td>N1</td>
<td>The system shall reconstruct images with advanced beamforming methods.</td>
<td>Functional requirement following system objectives defined in Chapter 2 Section 2.1.</td>
<td>Demonstration.</td>
</tr>
<tr>
<td>FR2</td>
<td>N2</td>
<td>The system shall autotune image settings</td>
<td>Functional requirement following system objectives defined in Chapter 2 Section 2.1.</td>
<td>Demonstration.</td>
</tr>
<tr>
<td>PR1</td>
<td>N1</td>
<td>The system shall display frames with a maximum display latency of 100 ms.</td>
<td>Performance requirement. Maximum latency accepted in surgical intervention.</td>
<td>Test.</td>
</tr>
<tr>
<td>PR2</td>
<td>N1</td>
<td>The system shall display frames with a maximum inter-frame period of 0.03 s (30 fps).</td>
<td>Performance requirement. The human eye can distinguish about 30 frames per second. A lower display frame rate results into perceived discontinuity and might affect surgical intervention.</td>
<td>Test.</td>
</tr>
</tbody>
</table>
### 1.2 Pre-defined Interacting Systems

When moving to effective development, additional design constraints are imposed by the equipment available for implementation. Accordingly, a list of the pre-defined interacting systems defined together with the principal stakeholders is reported below:

- **Ultrasound Research Platform.** A Vantage (Verasonics, WA, USA) open platform is considered for US data acquisition given its programmability and its widespread use in the US community. The Verasonics Vantage Research System includes a Verasonics data acquisition hardware and a host controller computer. The hardware contains: electronic modules for multi-channel transmit waveform generation, analog receive signal amplification and filtering, digital signal processing and scan sequencing. The host controller computer, instead, presents: (1) a hardware abstraction layer which allows the communication between software and hardware; (2) a non-editable function written in the C programming language, named `runAcq`, which handles all operations during run-time, and (3) a Matlab programming environment that allows users to program the desired sequences of events. The host computer presents one Central Processing Unit (CPU) (Intel Xeon Gold 6136 CPU @ 3.00GHz, 2.99GHz, 2 Processors) and it is connected to the TU/e Internet through an ethernet cable. Data transfer to host computer via 8 lanes PCIe 3.0 enables a data transfer rate up to 6.6 GB/s. Verasonics data acquisition hardware has a maximum storage capacity of 2 GB. A more comprehensive description of Verasonics platform is provided in Appendix B Section 3.1.

Design constraints include: (1) the software modules defined in Matlab are the only component of the platform that the user can edit (e.g., the Verasonics data acquisition hardware and the `runAcq` cannot be accessed or modified directly, but only through the Matlab environment), (2) the acquired signal is necessarily sent from hardware to the host computer RAM and only the `runAcq.c` function can drive the raw data from the RAM to the Matlab environment for personalized processing, (3) any data processing step performed on the host computer need to be compliant with the computational power offered by the Verasonics platform.

- **Deep Learning Methods.** The advanced processing methods envisioned for this technology demonstrator are implemented in TensorFlow within a Python environment. As these methods are developed in parallel to the framework, the required compute resources cannot be estimated precisely.

Design constraints include: (1) obliged communication between the Matlab environment (default environment in the Verasonics platform) and the Python environment.
The use of pre-defined interacting systems determines that certain functionalities of the envisioned system can be allocated directly to the already existing systems and do not need to be developed from scratch. This strategy is typically convenient within a time-constrained project, as it reduces the workload. On the other hand, it is worth stressing that using a pre-defined system could limit the performance of the technology demonstrator to those of the used device.

1.3 System Architecture

Considering the pre-defined interacting systems, the principal system functionalities are allocated according to the AoA winning solutions (Tab 2.4): TX/RX to the probe; digitization to the Verasonics data acquisition hardware which represents the on-site device or the analog front-end implemented on a digital probe; processing is allocated to a remote application server which corresponds to the cloud; and rendering either to a third-party connected device or to the Verasonics host computer which offers already a display functionality. This allocation is visualized in Fig. 3. As described in the Conceptual Study, the processing methods include: image formation, or beamforming, and post-processing for high quality image reconstruction (N1 in Tab. 2.1); and learned algorithm (e.g. quality-based) for image settings update (N2 in Tab. 2.1).

The red contour indicates the part of the system which should be design and developed within the project time frame. As anticipated, certain functionalities (e.g. digitization) are fulfilled by the Verasonics Vantage System, and as such should not be designed and developed. Importantly, the border highlights also that Verasonics US research platform allows the users to access data only at the Matlab environment level.

Figure 3: Preliminary system architecture. Red contour indicates what is effectively designed and developed within project scope, the rest is used as provided by the pre-defined interacting systems. Note that connections are missing where there are no info a priori (e.g., it is known that processing should occur in the cloud according to the conceptual diagram, but how data are transmitted from the acquisition module to the cloud is defined throughout the design process).

2 Logical Decomposition

Logical decomposition is the process for creating the detailed list of sub-system requirements to be satisfied to meet the stakeholders’ expectations. It receives as inputs the high-level architecture and top-level requirements defined in the preliminary phase. It includes system archi-
architecture breakdown down to the lowest desired level of the decomposition (e.g., in a project, typically at the level where the responsibilities are divided between different engineers/project teams) and requirements allocation. The requirements allocation is preceded by technical budgeting to translate critical performance requirements (MoE) to sub-system performance requirements explicitly.

Importantly, when agile-like methods are used, architecture breakdown and requirements allocation are performed in an iterative manner. This iterative refinement constitutes the core of the design process: the design engineer proposes a certain system architecture by allocating different functionalities to different components, tests the proposed solution and verifies that the requirements are satisfied. If not, a new design is proposed. The following paragraphs show the final outcome of this iterative design process.

2.1 Architecture Breakdown

2.1.1 Sub-systems Architecture

Fig. 4 shows the proposed system architecture. Verasonics acquisition module is responsible for TX/RX and digitization. Each frame is transferred to the host computer RAM through Direct Memory Access (DMA). On the Verasonics system side, runAcq.c handles all operations during run-time and interacts directly with the hardware to drive data acquisition. US channel data and updates transfer between the on-site device and the remote server is achieved with Verasonics built-in external function processing. This functionality enables personalized events, defined by the user in Matlab, which can be executed directly by the software sequencer runAcq.c during run-time. On the application server side, instead, all tasks are executed within a micro-framework web. Server functionalities are mainly associated with data processing and post-processing. The developed web app is accessed by a third-party connected device and enables streaming of reconstructed frames and graphical user interface (GUI) for manual run-time
Acquisition scheme. A linear probe of 128 elements (L11-4v, Verasonics) and λ spacing (0.2464 mm) is adopted for PW imaging with a varying odd number of steered PWs regularly spaced between ±16°. The pulse repetition frequency (PRF) is set to 6.25 kHz and the acquisition frame rate to 100 fps. Assuming a narrowband signal, the signal is sampled using a second-order method approximating quadrature-sampling (refer to Appendix B Section 4.1 for additional details). For B-mode data, a percent bandwidth of 100% is considered, which implies a real and an imaginary sample per wavelength. The image depth is set to 4.6 cm which corresponds to 576 samples per channel. Verasonics allows a unique bit depth equal to 16-bit per sample.

**BOX 1 - raw US channel data size**

For 1 single acquisition:

\[
\text{Size} = \text{sampling rate} \times \text{max travelling distance} \times \text{velocity} \times \text{no. channels} \times \text{bit length for each sample}
\]

\[
\approx 576 \text{ samples/channel} \times 2 \text{ channels} \times 128 \text{ elements} \times 2 \text{ bytes/sample} \approx 295K \text{Bytes} \approx 2.36M\text{bits}
\]

For 11 acquisitions:

\[
\approx 3.24M\text{Bytes} \approx 26M\text{bits}
\]

Acquired data is transferred through a PCI Express cable from hardware to Verasonics host computer RAM. If multiple acquisitions are executed per frame, data is transferred to the host computer RAM only when all PW acquisitions are received. Data is accessed in real-time using Verasonics built-in external function processing (externalData.m). Function specify the source: host computer RAM (srcbuffer = receive), and the frame number: last frame (srcframenum = -1). Since only the last frame is accessed, acquisition and customized processing can execute asynchronously. Retrieved data is transferred over the network as int16 data type.

Application server. An on-premises workstation is used as server end point. The facility presents one CPU AMD Ryzen 7 3800X (8 Cores, 32MB of L3 cache, base clock 3.9 GHz), and one GPU NVIDIA GeForce RTX 3080 Ti (Gigabyte) with VRAM of 12GB. The GPU provides a processing power of 35.01 TFLOP/s and a I/O bandwidth of 912 GB/s. CPU/GPU memory has distinctly separate entities. Application server is chosen considering memory size to accommodate the models (each model accounts for about 300 MB), compute power (i.e., number of multiple cores) which is strictly related to execution time, and I/O bandwidth.

Web app. The web application is developed using Flask in Python and provides a GUI for remote manual control. Flask provides native support for streaming responses through the use of generator functions. The application has two routes: (1) the / route serves the main page, which is defined in the index.html template. The video_feed route is called directly by the HTML template:

\[
\text{\text少少} <\text{\text少少}\text{img src="\{ url_for('get_frame') \}""\text{\text少少}}>;
\]

(2) the video_feed route returns the streaming response. The generator function used in the video_feed route is called gen(). This function enters a loop where it continuously returns frames. Specifically, gen() retrieves the last reconstructed image from the image S&H storage and yields it to the web page.
Client-Server interaction. A socket is programmed to enable a client-server interaction (see Appendix B Section B.2 for additional details regarding network architecture and socket programming). Socket programming requires a software structure that acts as a communication connection point (i.e., end-point) at each connection extremity for sending and receiving data. A Transmission Control Protocol (TCP or TCP/IP) is used as it provides reliable, ordered, and error-checked delivery of a stream of bytes between the two hosts communicating via an Internet Protocol (IP) network. Note that TCP/IP protocol is chosen over User Datagram Protocol (UDP/IP) according to the analysis performed in Appendix A Section A.1. The results show that UDP provides a lower round-trip time, hence faster data transmission, at the cost of no control on packet loss resulting in an unreliable protocol. Packet loss might be perfectly tolerable in certain applications, but it is considered unacceptable for this application (remember that this system should be used in combination with advanced methods to optimize image quality, hence data loss is counter-productive). When control strategies are implemented from scratch and added to UDP to counteract the packet loss, UDP throughput deteriorates and TCP/IP protocol is preferable.

TCP socket is implemented in Python with `socket.socket` and in Matlab with `tcpclient` function.

Route hops are displayed in Fig. 5 and include:

- Verasonics Vantage 256 Host Computer with a gigabit ethernet card.
- Application server with a gigabit ethernet card.

Both hosts are connected through a Cat6a Ethernet cable (last generation cable) with up to 10 Gbps transmission speed. The theoretical maximum bandwidth is equal to 1 Gbps.

Software. Once the connection between the two hosts is established, `get_frame.py` function starts the execution of two concurrent threads: `read_and_update.py`, `processing.py`. The former, `read_and_update.py`, is responsible for: (1) reading the raw data and saving the last received frame in a sample & hold (S&H) storage, (2) computing the novel values for tunable run-time parameters (either using the autotuning strategy or from user input posted through
the GUI), and (3) sending the updated parameters to the client. The second concurrent thread, processing.py, is responsible for: (1) reading the last received frame from the S&H and processing the data with advanced beamforming, (2) storing the reconstructed image into another S&H storage. During run-time, get_frame.py accesses the stored reconstructed images and yields them to the streaming window of the web app.

**Beamforming.** Inspired by the algorithmic structure of static DAS beamformer, which first applies predetermined delays and weights to the channel signals and then sum up the individual contributions to yield a beamformed signal, the proposed reconstruction algorithm applies: (1) time-of-flight correction; (2) optimal apodization weights estimated though an adaptive strategy; (3) average per pixel contribution. Contrary to other adaptive methods, which typically present complex algorithms with significantly longer reconstruction time, this approach takes advantage of deep learning (DL) methods to adaptively compute a set of optimal image reconstruction parameters given the received US channel data (Fig. 6). This solution achieves performance comparable with adaptive beamforming, but at a fraction of computational cost [27, 28].

![Figure 6: Adaptive beamforming by deep learning](image)

**Figure 6:** Adaptive beamforming by deep learning. Flow charts of [a] standard delay-and-sum beamforming using fixed apodization weights and [b] adaptive beamforming by deep learning. Reproduced from [27, 28]

**Autotuning strategy.** The self-driving US paradigm is implemented by adding a reinforcement learning (RL) framework to the previous signal processing pipeline. The basic principle of RL is to evaluate the quality of the reconstructed images and decide which parameters, if any, in the driving scheme and in the reconstruction chain should be changed to improve the overall system performance. RL scheme is visualized in Fig. 7. During training: (1) an agent performs an action in the environment, (2) an inspector observes the state (i.e. the image obtained) and computes a reward function (i.e., a measure of similarity with a target high quality image), (3) the agent receives the state and the reward and checks if the performed action improved or not the reward. By repeating this process and exploring several actions, the agent learns the action to perform to maximize the reward given a certain state. During inference, the agent receives only the state (i.e., the current reconstructed image), and proposes an action (i.e., updated image settings so to optimize image quality). The agent is typically implemented by a trainable Neural Network (NN).
CHAPTER 3. SYSTEM DESIGN

Figure 7: Basic principles of reinforcement learning.

Run-time parameters update. Updates are sent to the Verasonics platform from the application server (read_and_update.py) function. This function computes the updated parameters, either using the autotuning strategy or according to the values received by the GUI (and changed manually by the user). The values are read by externalUpdate.m function as double data type. Here, the attribute structures in Matlab workspace are called with eval('base', 'var1'), modified according to the received updates, and re-assigned after modification in the Matlab base environment with assignin('base', 'var1', novelValue). Importantly, acquisition settings change only when the structures are re-loaded into the software sequencer and the hardware, which means the next time runAcq calls the VSX function and returns control to Matlab. VSX.m checks if the workspace attributes are modified, if so the software re-computes the dependent structures and re-load every changed setting into the software and hardware sequencer. It follows that the effective update frequency depends on the frequency at which control is returned to Matlab during run-time (i.e., returnToMatlab command in the acquisition sequence).

2.1.2 Timing Diagram

A schematic representation of how operations are carried out in time is provided in Fig. 9. This representation considers the system at steady state and neglects the initialization phase.

Verasonics host computer RAM gets data from the local memory of the Verasonics data acquisition hardware at a constant interval. Each arrow input to the RAM corresponds to one frame, potentially composed by multiple PWs.

Asynchronously, runAcq.c handles all the processing during run-time. First, it calls externalUpdate.m function which reads from the server if any run-time parameters must be updated and, if this is the case, update its value in the Matlab workspace (i.e., settings). Second, the software sequencer runAcq.c calls the externalData.m function, which reads the last acquired frame stored in the Verasonics host computer RAM and transmit it to the server. As soon as this function completes, the control returns to the runAcq.c function which calls the VSX.m, the VSX.m checks whether the run-time parameters in the workspace have changed. If there are attributes which must be updated ([if updates] condition in the alt insert in Fig. 9), the novel imaging settings are reloaded in the data acquisition hardware and software sequencer.

Concurrently, the read_and_update.py function runs on the application server. The function communicates the updates to the host computer over the network. To compute these updates the read_and_update.py function can either check the Python workspace (i.e., settings) which receives input from the web app GUI, or read the last reconstructed image and use it to estimate the novel run-time parameters by the means of the autotuning strategy. The former case implies that no autotuning strategy is used, but updates are only due to manual tuning through the GUI. For the second case the autotuning frequency can be set a priori or de-
fined according to the autotuning strategy outcomes (e.g., is the benefit in terms of image quality worth updating the settings?). Afterwards, the function reads the raw channel data from the network input buffer and save them into a S&H-like memory. Note that data transfer cannot execute till the updates communication is completed, hence `externalUpdate.m`, `externalData.m` execute synchronously (i.e., one has to finish before executing the next one).

Asynchronously, the `processing.py` function reads the last frame from the raw data S&H-like storage, beamform the channel data, and post processes the reconstructed image. Once completed, the final image is stored in a different S&H-like memory, which is accessed by the `get_frame.py` function to retrieve the last reconstructed image and display it through the web app.

**Considerations**  Given the three different concurrent processes, three rates can be identified:

\[
\frac{1}{FR_{\text{acquisition}}} = IFP_{\text{acq}} \geq TX/RX + DMA \tag{3.1}
\]

\[
\frac{1}{FR_{\text{transferring}}} = IFP_{\text{tran}} \geq CK + UT + DT \tag{3.2}
\]

\[
\frac{1}{FR_{\text{processing}}} = IFP_{\text{proc}} \geq BMF + PP \tag{3.3}
\]

with \(IFP_{\text{acq}}\): acquisition inter-frame period (set by the user \(a \text{ priori}\)); \(TX/RX\): transmit/receive, DMA: direct memory access, DT: data transferring; UT: updates transferring; CK: check if updates, BMF: beamforming, PP: post-processing. The time needed to reload structures and attributes into the hardware and software sequencer (RELOAD) is included as it is typically equal to zero (i.e., the re-loading step is not always present). When tuning occurs, RELOAD time is added to the events at the data acquisition hardware side. Additionally, at the server side, an additional time lag (AUT) is added to retrieve the updated acquisition attributes from the workspace (in case of manual tuning through the user interface) or to compute the novel parameters (in case autotuning is set).

Asynchronicity between acquisition, data transfer and processing allows each process to run at its own rate and be bottlenecks only by previous processes: so transferring frame rate cannot be larger than acquisition frame rate, and, likewise, processing frame rate cannot be larger than transferring processing rate (otherwise it would mean that the same frame is processed multiple times). This programming scheme enables the display inter-frame period (i.e., inverse of the display frame rate) to be equal to the longest process among the three parallel ones (acquisition, transferring, processing).

On the other side, data writing and reading over the network are made synchronous by the means of an acknowledgment message. For an efficient implementation, the acknowledgement message is composed by the run-time parameters updates that need to be transferred from the server to the host computer (and then to the hardware) for image settings tuning. In other terms, updates act as an acknowledgment message sent by the server to the host computer to ensure that the server is ready to read the US channel data. On the contrary, if the two processes were asynchronous, Matlab would continuously write channel data to the server network input buffer overrunning Python reading: due to First-In-First-Out (FIFO) reading, Python would need to read a previously sent frame even though a more recent one is already available at input buffer (Fig. 8).
Figure 8: Synchronous vs asynchronous data writing/reading over the network. [a] Asynchronous writing and reading: server reads the buffer from the last received data till the most recent one (read green packets from green frame even though grey packets belonging to grey frame are already available at the end of the buffer queue). [b] Synchronous writing and reading using an acknowledgment message: server always reads the most recent frame.
Figure 9: System sequence diagram. Initialization is not displayed. The alt insert is indicated only once for simplicity, but it should be repeated every time runACq.c calls VSX.m to check for update.
2.2 Requirements Breakdown

2.2.1 Technical Budgeting

Technical budgeting is performed during architecture breakdown to allocate different resources to different system components. Among the CtQ parameters specified in Section 1.1, image quality and autotuning efficiency (PR3, PR4, PR5) are attributable exclusively to the beamforming and the learned algorithm respectively. On the contrary, display latency and inter-frame period (PR1, PR2) involve multiple processes, hence the CtQ parameters need to be transferred to sub-systems (components/elements) and made explicit. Tab. 3.2 and Tab. 3.3 subdivide the display latency time and inter-frame period between the contributing sub-processes (taking into account the sequence diagram displayed in Fig. 9). This performance allocation is the result of a progressive refinement based on testing results. The resources allocated to the software are discussed with the two colleagues responsible for the algorithm implementation.

Note that processes within project scope depend on the design and their run-time can be in principle improved, while steps outside project scope (e.g. DMA) are determined by the pre-defined interacting systems and most likely cannot be modified.

With respect to display latency, the timing is given by the sum of all processes from acquisition to rendering. Importantly, the asynchronicity among data acquisition, data transferring, and data processing introduce a variable latency. For instance, data is transferred from hardware to the host computer RAM and the last frame is retrieved by the `externalData.m` as soon as it finishes the previous task. The time interval between the instant at which the packet is saved in RAM and when it is read by the function in Matlab varies depending on data transfer and the lag introduced by settings re-loading (see stars in Fig. 10). Likewise, the time elapsed between the instant at which channel data is stored in the last frame S&H in Python and when processing starts depends on the processing time of the previous frame and network throughput (see stars in Fig. 10). The technical budgeting (Tab. 3.2) should take into account these added variable lags to ensure that the final latency is always below the CtQ parameter. In that regard:

- time lag A can vary between zero and \(1 / FR_{\text{acquisition}} + \text{RELOAD}\) (with no tuning, time lag A remains below \(1 / FR_{\text{acquisition}}\)). This corresponds to the maximum time lag between the instant at which the data is stored in the host computer RAM and the time at which it is retrieved by the `externalData.m` function and sent over the network. After this time, a novel frame is stored in the RAM and the count re-starts.

- time lag B can vary between zero and \(1 / FR_{\text{transferring}} + \text{RELOAD}\) (with no tuning, time lag A remains below \(1 / FR_{\text{transferring}}\)). This corresponds to the maximum time lag between the instant at which the data is written to the last frame S&H in Python and the time at which it starts to be processed.

Importantly, the resources allocated in Tab. 3.2 should be consistent with the ones defined in Tab. 3.3, as both tables refer to the same processes.

Table 3.2: Technical budgeting: display latency. The given run-time is independent of the number of firing angles. TX/RX: transmit/receive, DMA: direct memory access, DT: data transfer, RELOAD: time to reload structures and attributes into the hardware and software sequencer, BMF: beamforming, PP: post-processing. Values are defined to be consistent with Tab. 3.3.

<table>
<thead>
<tr>
<th>Process</th>
<th>Max run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX/RX</td>
<td>3 ms</td>
</tr>
<tr>
<td>DMA <code>DMA()</code></td>
<td>3 ms</td>
</tr>
<tr>
<td>Extra Time Lag A = (1 / FR_{\text{acquisition}} + \text{RELOAD})</td>
<td>15 ms</td>
</tr>
<tr>
<td>DT <code>writeLastFrame()</code></td>
<td>15 ms</td>
</tr>
<tr>
<td>Extra Time Lag B = (1 / FR_{\text{transferring}} + \text{RELOAD})</td>
<td>30 ms</td>
</tr>
<tr>
<td>BMF <code>beamforming()</code></td>
<td>25 ms</td>
</tr>
</tbody>
</table>
Regarding PR2, the inter-frame period is determined by the slowest concurrent process:

$$\text{IFP}_{\text{display}} = \max\left(\frac{1}{\text{FR}_{\text{acquisition}}}, \frac{1}{\text{FR}_{\text{transferring}}}, \frac{1}{\text{FR}_{\text{processing}}}\right) = \max(\text{IFP}_{\text{acq}}, \text{IFP}_{\text{transf}}, \text{IFP}_{\text{proc}})$$  \hfill (3.4)

with $\text{IFP}_{\text{display}}$: display inter-frame period. The technical budgeting should ensure that in all conditions the CtQ parameter is reached.

Note that for the events occurring at the data acquisition hardware side, RELOAD is added to the standard frame rate to ensure that the requirement is satisfied even when acquisition attributes need to be re-loaded.

**Table 3.3: Technical budgeting: inter-frame period.** The given run-time is independent of the number of firing angles. TX/RX: transmit/receive, DMA: direct memory access, DT: data transfer, CK: check if updates, AUT: compute updates (if autotuning is set), UT: updates transfer, RELOAD: time to reload structures and attributes into the hardware and software sequencer, BMF: beamforming, PP: post-processing. Values are defined to be consistent with Tab. 3.2.

<table>
<thead>
<tr>
<th>Process</th>
<th>Max run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELOAD (reloadSettings())</td>
<td>5 ms</td>
</tr>
<tr>
<td>TX/RX</td>
<td>3 ms</td>
</tr>
<tr>
<td>DMA (DMA())</td>
<td>3 ms</td>
</tr>
<tr>
<td>Inter-frame period $= 1/\text{FR}_{\text{acquisition}} + \text{RELOAD}$</td>
<td>$\leq 30$ ms</td>
</tr>
<tr>
<td>UT (writeUpdates())</td>
<td>5 ms</td>
</tr>
<tr>
<td>CK (checkForUpdate())</td>
<td>5 ms</td>
</tr>
<tr>
<td>RELOAD (reloadSettings())</td>
<td>5 ms</td>
</tr>
<tr>
<td>DT (writeLastFrame())</td>
<td>15 ms</td>
</tr>
<tr>
<td>Inter-frame period $= 1/\text{FR}_{\text{transferring}} + \text{RELOAD}$</td>
<td>$\leq 30$ ms</td>
</tr>
<tr>
<td>DT (writeLastFrame())</td>
<td>15 ms</td>
</tr>
<tr>
<td>AUT (computeUpdates())</td>
<td>5 ms</td>
</tr>
<tr>
<td>UT (writeUpdates())</td>
<td>5 ms</td>
</tr>
<tr>
<td>Inter-frame period $= 1/\text{FR}_{\text{transferring}} + \text{AUT}$</td>
<td>$\leq 30$ ms</td>
</tr>
<tr>
<td>BMF (beamforming())</td>
<td>25 ms</td>
</tr>
<tr>
<td>PP (post-processing())</td>
<td>5 ms</td>
</tr>
<tr>
<td>Inter-frame period $= 1/\text{FR}_{\text{processing}}$</td>
<td>$\leq 30$ ms</td>
</tr>
</tbody>
</table>
CHAPTER 3. SYSTEM DESIGN

Figure 10: Timing Diagram. VERASONICS: (1) transmits waveforms, receives and digitizes ultrasound channel data (TX/RX); (2) transfers data to host computer through direct memory access (DMA). Verasonics host computer: (1) reads updates sent by the application server (UPD. READ.); (2) sends raw channel data to the application server (DATA WRITING); (3) verifies if image settings need to be updated (CHECK IF); (4) if needed, reprogram the hardware with the updated image settings (SETTINGS LOADING). APPLICATION SERVER: (1) reads raw channel data sent by the Verasonics host computer (DATA READING); (2) if autotuning is set, compute updates, otherwise simply retrieve them from the workspace which receives the user inputs from the user interface (COMP. UDP.); (3) sends the feedback message with the updated image settings values or null values if no tuning is requested (UPD. WRIT.).

2.2.2 Sub-systems Requirements

The requirements defined in Tab. 3.1 are refined and allocated to different sub-systems (Tab. 3.4). The rows in gray indicate the requirements dependent on the advanced methods developed by the two colleagues within the SPICE project.

Note that TX/RX, DMA, CK, RELOAD timing are not reported in Tab. 3.4 as these operations depend on the implementation provided by the pre-defined interacting system (i.e., Verasonics Vantage system) and not on the proposed design. Nevertheless, the timing requirements listed for BMF, DT, UT, AUT are derived through technical budgeting considering also the contribution of TX/RX, DMA, CK, RELOAD, thereby the time resources allocated to these processes should also be verified.
Table 3.4: Sub-systems level requirements. ID: All requirements have a unique ID. N/C reference: refer to the unique ID of the users’ needs listed in Tab. 2.1. Requirement reference: refer to the unique ID of the system level requirements listed in Tab. 3.1 (implicit traceability matrix). Verification method: indicate the planned verification approach, can be inspection, analysis, demonstration, or test. Rows in gray indicate the requirements dependent on the advanced methods developed by the two colleagues within the SPICE project.

<table>
<thead>
<tr>
<th>ID</th>
<th>N/C Ref.</th>
<th>Req. Ref.</th>
<th>Requirement</th>
<th>Rationale</th>
<th>Verification method</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW10</td>
<td>N2</td>
<td>FR2</td>
<td>The software sequencer shall call externalUpdate.m function at the beginning of each software loop during run-time.</td>
<td>The software sequencer, runAcq.c, handles all operations during run-time, hence it has to give control to another program if wanted so.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>SW20</td>
<td>N2</td>
<td>FR2</td>
<td>The software sequencer shall call VSX.m function at the end of each software loop during run-time.</td>
<td>The software sequencer, runAcq.c, handles all operations during run-time. By default, Verasonics VSX.m checks whether the attribute structures in Matlab workspace are updated: if so, VSX.m recomputes the parameters dependent on the modified structures and loads the changed settings into both the data acquisition hardware and the software sequencer (i.e., runAcq.c).</td>
<td>Demonstration</td>
</tr>
<tr>
<td>SW30</td>
<td>N1</td>
<td>FR1</td>
<td>The software sequencer shall call externalData.m after externalUpdate.m during run-time.</td>
<td>The software sequencer, runAcq.c, handles all operations during run-time. externalData.m is called after externalUpdate.m because the latter receive data from the server and as soon as it finishes reading the server is ready to read new data.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>EU10-inp</td>
<td>N2</td>
<td>FR2</td>
<td>The externalUpdate.m function shall read the novel values for tunable run-time parameters as binary data sent by the application server over the network.</td>
<td>The novel values for run-time parameters are estimated on the application server, either according to input received by the Graphical User Interface or computed with the autotuning method.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>EU10-out</td>
<td>N2</td>
<td>FR2</td>
<td>The externalUpdate.m function shall update the image settings in Matlab workspace according to the received novel values for run-time parameters.</td>
<td>The externalUpdate.m function cannot directly reload the image settings into the hardware and software sequencer. The externalUpdate.m can modify the attributes in the Matlab workspace, afterwards the VSX.m function checks the workspace if structures are modified and, if so, it reloads the image settings into hardware and software.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>ED10</td>
<td>N1</td>
<td>FR1</td>
<td>The externalData.m function shall flatten the last acquired frame before serialization.</td>
<td>Data are transmitted in line.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>ED20</td>
<td>N1</td>
<td>FR1</td>
<td>The externalData.m function shall serialize the flattened last acquired frame before transmitting it to the server.</td>
<td>Data are transmitted over the network as binary data.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>ID</td>
<td>FR1</td>
<td>N1</td>
<td>FR2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ED10-inp</td>
<td>The externalData.m function shall retrieve the last acquired frame from the host computer RAM (RcvBuffer) where acquired data is stored.</td>
<td>By default, Verasonics data acquisition hardware acquires data, digitize data, transmit data to the host computer RAM through Direct Memory Access.</td>
<td>Demonstration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ED10-out</td>
<td>The externalData.m function shall send the serialized last acquired frame over the network to the application server.</td>
<td>Data are processed on the application server.</td>
<td>Demonstration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU10</td>
<td>The read_and_update.py function running on the application server shall estimate the novel values for tunable run-time parameters based on the last reconstructed image when autotuning mode is selected.</td>
<td>Novel values for tunable run-time parameters are estimated with advanced autotuning strategy on the application server.</td>
<td>Demonstration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU20</td>
<td>The read_and_update.py function running on the application server shall serialize the values of tunable run-time transmit parameters, either estimated by the autotuning strategy or according to the manual adjustment through the GUI.</td>
<td>Data are transmitted over the network as binary data.</td>
<td>Demonstration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU30</td>
<td>The read_and_update.py function running on the application server shall deserialize the received binary data.</td>
<td>Data are transmitted over the network as binary data.</td>
<td>Demonstration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU40</td>
<td>The read_and_update.py function running on the application server shall reshape the received deserialized data to return the US channel data with the shape required by the processing method.</td>
<td>Data are transmitted in line.</td>
<td>Demonstration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU10-inp</td>
<td>The read_and_update.py function running on the application server shall retrieve the last reconstructed image from the image S&amp;H storage when autotuning mode is selected.</td>
<td>Autotuning strategy is based on the previous reconstructed image to select the run-time parameters which provide better quality.</td>
<td>Demonstration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU20-inp</td>
<td>The read_and_update.py function running on the application server shall check if tunable run-time parameters are updated in the Python workspace according to GUI inputs when autotuning mode is not selected.</td>
<td>Settings posted through the GUI are received by the application server, which sends them to the Verasonics host computer to reload the attributes into hardware and software sequencer.</td>
<td>Demonstration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU30-inp</td>
<td>The read_and_update.py function running on the application server shall receive the US channel data as binary data.</td>
<td>Channel data is sent to the server for processing.</td>
<td>Demonstration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU10-out</td>
<td>The read_and_update.py function running on the application server shall store the received US channel data in a data S&amp;H storage.</td>
<td>Data is stored in a S&amp;H to decouple data transmission and processing.</td>
<td>Demonstration</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>RU20-out</td>
<td>N2</td>
<td>FR2</td>
<td>The read_and_update.py function running on the application server shall transmit to the Verasonics host computer the serialized values of tunable run-time transmit parameters.</td>
<td>Updates are transmitted to the Verasonics machine to change the transmit settings.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>P10</td>
<td>N2</td>
<td>FR1</td>
<td>The processing.py function running on the application server shall beamform the last acquired frame to return a reconstructed image.</td>
<td>Channel data is processed with advanced methods on the application server.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>P20</td>
<td>N2</td>
<td>FR1</td>
<td>The processing.py function running on the application server shall post process the last reconstructed image to improve appearance before storing it.</td>
<td>Reconstructed image might need to be post processed (e.g., log scale) before visualization.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>P10-inp</td>
<td>N2</td>
<td>FR1</td>
<td>The processing.py function running on the application server shall retrieve the last acquired frame from the frame S&amp;H storage as soon as it finishes processing the previous one.</td>
<td>Processing executes asynchronously to data transfer, thereby only the last acquired frame - stored in the S&amp;H is processed.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>P10-out</td>
<td>N2</td>
<td>FR1</td>
<td>The processing.py function running on the application server shall store the last reconstructed image after post-processing in the image S&amp;H.</td>
<td>Rendering executes asynchronously to data processing, thereby the last reconstructed image is stored.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>GF10-inp</td>
<td>N1</td>
<td>FR1</td>
<td>The get_frame.py function running on the application server shall retrieve the last reconstructed image from the image S&amp;H storage as soon as it finishes yielding the previous one to the web app.</td>
<td>After reconstruction, the images are stored in a Python S&amp;H.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>GF10-out</td>
<td>N1</td>
<td>FR1</td>
<td>The get_frame.py function running on the application server shall stream reconstructed images to a web app running on a third-party device connected to the Internet.</td>
<td>Images are visualized through a web app.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>WA10</td>
<td>N1</td>
<td>FR1</td>
<td>The web app shall display reconstructed images.</td>
<td>Reconstructed images are visualized using a web app.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>WA20</td>
<td>N2</td>
<td>FR2</td>
<td>The web app shall enable manual tuning of transmit voltage, autotuning mode.</td>
<td>Even though the system allows autotuning, manual tuning is still available through a web app GUI.</td>
<td>Demonstration</td>
</tr>
<tr>
<td>BMF10</td>
<td>N1</td>
<td>PR3</td>
<td>The beamforming method within processing.py function shall improve image contrast w.r.t. DAS beamforming of at least 5%.</td>
<td>Rationale to move to advanced computationally-intensive methods. Refer to high-level performance requirements.</td>
<td>Test. Offline, measure the image contrast of images reconstructed with DAS and advanced method, using the same raw channel data.</td>
</tr>
<tr>
<td>BMF20</td>
<td>N1</td>
<td>PR4</td>
<td>The beamforming method within processing.py function shall improve image resolution w.r.t. DAS beamforming of at least 5%.</td>
<td>Rationale to move to advanced computationally-intensive methods. Refer to high-level performance requirements.</td>
<td>Test. Offline, measure the image resolution of images reconstructed with DAS and advanced method, using the same raw channel data.</td>
</tr>
<tr>
<td>BMF30</td>
<td>N1</td>
<td>PRI</td>
<td>The beamforming method within processing.py function shall last less than 25 ms.</td>
<td>Refer to high-level performance requirements and technical budgeting.</td>
<td>Test. Measure execution time.</td>
</tr>
<tr>
<td>PPI0</td>
<td>N1</td>
<td>PRI</td>
<td>The post-processing within processing.py function shall last less than 5 ms.</td>
<td>Refer to high-level performance requirements and technical budgeting.</td>
<td>Test. Measure execution time.</td>
</tr>
<tr>
<td>AUT10</td>
<td>N2</td>
<td>PR5</td>
<td>The autotuning strategy used to compute the novel values for to be defined run-time parameters within read_and_update.py function shall improve the to be defined quality metric w.r.t. previous frame of at least to be defined.</td>
<td>Refer to high-level performance requirements and technical budgeting.</td>
<td>Test. Measure execution time.</td>
</tr>
<tr>
<td>AUT20</td>
<td>N2</td>
<td>PR2</td>
<td>The autotuning strategy used to compute the novel values for to be defined run-time parameters within read_and_update.py function shall last less than 5 ms.</td>
<td>Refer to high-level performance requirements and technical budgeting.</td>
<td>Test. Measure execution time.</td>
</tr>
<tr>
<td>DT10</td>
<td>N1</td>
<td>PRI</td>
<td>The channel data transfer from Verasonics machine to the application server shall last less than 15 ms.</td>
<td>Refer to high-level performance requirements and technical budgeting.</td>
<td>Test. Measure half the time needed to echo a message of size equal to frame size.</td>
</tr>
<tr>
<td>UT10</td>
<td>N2</td>
<td>PR2</td>
<td>The updates transfer from the application server to Verasonics machine shall last less than 5 ms.</td>
<td>Refer to high-level performance requirements and technical budgeting.</td>
<td>Test. Measure half the time needed to echo a message of size equal to updates.</td>
</tr>
</tbody>
</table>
Chapter 4

System Verification

System verification is the process used to evaluate the proposed solution. It includes sub-systems verification, where system components are assessed to verify that the requirements allocated are satisfied, system integration and system verification. Typically, system integration and verification are completed only when the sub-systems requirements are met.

1 Sub-systems Verification

Sub-systems functional requirements are assessed by the means of demonstration. As for the performance requirements, the sub-systems are evaluated with well-defined tests. As mentioned above, in addition to the defined requirements for the sub-systems developed within project scope, also the resources dependent on the interacting systems (e.g. DMA) should be verified (see Chapter 3.2 Section 2.2.1).

The following paragraphs analyse each sub-system (process) and present the results of the sub-system verification.

TX/RX  TX/RX time is constant (i.e., user defines the inter frame period before starting the acquisition) and determined by the used acquisition protocol. As described in Section 2.1.1, US channel data are acquired with unfocused transmission with a variable number of PWs. Time between subsequent acquisitions (i.e., PWs) is set equal to 160 $\mu$s. Hence, acquisition time (TX/RX) varies between 160 $\mu$s and $11 \times 160 \, \mu s = 1.8 \, ms$, when moving from 1 to 11 PWs. Acquisition time remains below the time resources allocated with technical budgeting.

DMA  DMA time depends on Verasonics research platform design. According to the manual, the DMA speed is 6.00 GB/s. It follows that: DMA duration varies from 0.05 ms for a single PW (i.e., 0.295 MB) to 0.54 ms for 11 PWs (i.e., 3.24 MB)$^1$. Values meet the time resources allocated with technical budgeting.

DT & UT  DT10 & UT10 are associated to data transfer speed over the client-server link. Since the message is sent from one workstation to another, there is no straightforward way to estimate transferring time (i.e., DT, UT). Two approaches are explored: (1) measure half the time needed to echo a message of a specific size and derive application-level throughput; (2) time to send a message of desired size from the client to the server and receive an acknowledgment back (i.e., acknowledgment is intended as a very short message, e.g., 2 bytes).

Regarding the first method, the application-level throughput is measured as the ratio between message size and transferring time, where the latter is estimated as half the time needed to echo a certain message of a given size. Fig. 11 shows the achieved results. Notably, data

$^1$Estimates are also verified through Verasonics EventAnalysisTool.
rate converges to a maximum speed which remains below the theoretical limit imposed by the Ethernet bandwidth, i.e., 1 Gbps. The box plot shows that for large message size the throughput is dependent on data type. This difference might be attributed to the different number of int16 and double characters needed to reach the same number of bytes of a uint8-type message (uint8 character: 1 byte, int16 character: 2 bytes, double character: 8 bytes). In this regard, the application (e.g., Matlab) might introduce some additional character-dependent processing time before reaching network level (see OSI in Appendix B). According to the throughput values shown in Fig. 11 and considering that single PW frame size is of the order of 2.4 Mbits and type int16, mean throughput for DT is estimated to be around 220 Mbps. Throughput increases to 247 Mbps and 290 Mbps for 5PWs and 11PWs respectively. It follows that, according to this approach, DT should be equal to: 10.9 ms, 48.6 ms and 91 ms for PW=1, PWs=5, and PWs=11, respectively.

On the other side, UT message consists of an array of few characters (e.g., 2 tunable parameters) of double type. According to the measured throughput, this message size reaches a data rate of 3 Kbps, hence UT is equal to less than 1 ms.

With respect to the second approach, Tab. 4.1 reports the time measured to send a message of size equal to a frame of N PWs from Matlab running on Verasonics to Python on the application server and receive back a message of 2 characters of type double. This time is likely to approximate the sum of DT and UT. The values measured with this second approach are close to the DT computed according to the measured throughput, even though the DT computed using the throughput is slightly larger than the time measured as sum of DT and UT. This discrepancy could be explained considering the asymmetry of the client-server connection (Matlab vs Python). As this measure matches well the results observed in terms of transferring frame rate (Tab. 4.5), this method is used for the following latency analysis.

<table>
<thead>
<tr>
<th>PW = 1</th>
<th>PWs = 5</th>
<th>PWs = 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT + UT (s)</td>
<td>0.012 ± 0.004</td>
<td>0.033 ± 0.003</td>
</tr>
</tbody>
</table>

Table 4.1: Sub-system performance: data transfer run-time. PWs: number of plane waves. DT: data transfer.

According to these results:
• DT10 is met for single PW: DT < 15 ms
• UT10 is met: UT < 5 ms

BMF & PP  BMF10 and BMF20 are considered to be exclusively determined by the digital beamforming. Their verification is conducted as part of the work presented in [28]. The authors reconstruct images with different beamforming approaches starting from the same set of raw US channel data and evaluate the performance in terms of spatial resolution, or full width half maximum (FWHM), and contrast to noise ratio (CNR). Tab 4.2 summarizes the main results, and Fig. 12 provides a visual representation of the achieved results.

<table>
<thead>
<tr>
<th></th>
<th>DAS</th>
<th>ABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FWHM_{lat}$ (mm)</td>
<td>0.846 ± 0.100</td>
<td>0.704 ± 0.160</td>
</tr>
<tr>
<td>$FWHM_{lat}$ (mm)</td>
<td>0.431 ± 0.006</td>
<td>0.218 ± 0.069</td>
</tr>
<tr>
<td>CNR (dB)</td>
<td>11.35 ± 1.45</td>
<td>11.91 ± 1.03</td>
</tr>
</tbody>
</table>

Table 4.2: Sub-system performance: image quality. Resolution and contrast metrics achieved by the proposed method and by other reconstruction algorithms [28]. DAS: Delay-and-sum beamforming with Hanning apodization; ABLE: Adaptive beamforming by deep learning. FWHM: Full width at half maximum; CNR: Contrast-to-Noise ratio.

Figure 12: System performance: image quality. Single-plane-wave images using: a) Delay-and-sum (DAS) beamforming with Hanning apodization, b) iMAP2 beamforming c) Adaptive beamforming by deep learning (ABLE), and d) Eigen-Based Minimum Variance (EBMV) beamforming. Images are logarithmically compressed with a dynamic range of 60dB. From top to bottom: simulated point scatters, carotid artery cross-section, and a carotid artery longitudinal cross-section. Reproduced from [28].
From Tab 4.2, it can be noticed that:

- **BMF10** is met: 5% of DAS $FWHM_{lat} = 0.0423$ mm, and $DAS_{FWHM} - 0.0423$ mm $> ABLE_{FWHM}$

- **BMF10** is met: 5% of DAS $FWHM_{ax} = 0.02155$ mm, and $DAS_{FWHM} - 0.02155$ mm $> ABLE_{FWHM}$

- **BMF20** is met: 5% of DAS $CNR = 0.5675$ dB, and $DAS_{CNR} + 0.5675$ dB $\approx ABLE_{CNR}$

BMF30, PP10 are related to processing time. Tab. 4.3 shows the mean run-time measured with timer function in Python during normal use of the setup at steady state (i.e., discard the first frame due to initialization). Reported run-times are obtained with an image grid of 128x206 pixels. The grid size is tuned to minimize run-time without affecting spatial resolution. BMF and PP run-times are measured together, as both steps are implemented within a pre-trained model.

<table>
<thead>
<tr>
<th>PW = 1</th>
<th>PWs = 5</th>
<th>PWs = 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMF w. ABLE + PP (s)</td>
<td>0.037 ± 0.003</td>
<td>0.101 ± 0.003</td>
</tr>
</tbody>
</table>

**Table 4.3: System performance: processing run-time.** Mean value ± standard deviation, $n = 100$ events or frames. PWs: number of plane waves. BMF w. ABLE: beamforming with ABLE, PP: post-processing log compression.

From a detailed analysis of the beamforming model run-time, it turns out that: about 50% of the processing time is dedicated to "kernel launching" and the remaining 50% is the actual "computation" time. This explains the different behaviour observed for a single PW as opposed to multiple steering angles: with N PWs, time of flight (TOF) correction and weights estimate are calculated N times, but DL model’s kernels have to be initialized only one time.

Overall, Tab. 4.3 shows that:

- **BMF30, PP10** are not met: BMF + PP = 37 ms for PW = 1, and increases with number of PWs

Importantly, the fact that these requirements are not met do not prevent to proceed with the system integration. Additionally, processing time is expected to be improved with an optimized version of the image reconstruction model.

**AUT** Performance requirements related to the RL autotuning strategy (i.e., AUT10, AUT20) cannot be verified as the RL method is not ready for deployment.

A proportional–integral–derivative (PID) controller is developed as substitute autotuning strategy to test the platform feedback loop (i.e., mitigation from Risk Analysis). When autotuning is set, the PID controller continuously calculates an error value as the difference between a desired set point - specified by the user - and a measured process variable and applies a correction based on proportional, integral, and derivative terms. For details, please refer to Appendix B Section 4.2.

**CK** CK time is measured with a timer in VSX.m. The check process varies between 2 ms, when there is no update, and 5 ms, when workspace attributes have been modified and related structures need to be re-computed. Results are in agreement with technical budgeting.
**RELOAD** Appendix A Section A.2 provides an extensive systematic analysis of RELOAD time; the most relevant results are reported in Fig. 13. According to this analysis, update time can vary between a few milliseconds (e.g., transmit power\(^2\): \(0.0032 \pm 0.0146\) s) and hundreds of milliseconds (e.g., transmit apodization: \(0.2298 \pm 0.0017\) s; transmit frequency: \(0.2222 \pm 0.0021\)). The second group of parameters is used to calculate other dependent attributes (e.g., depth is expressed as a function of frequency due to frequency-dependent attenuation), hence when these parameters are updated additional calculations are performed to re-calculate the dependent attributes (i.e., this justifies their longer update time). For these apex parameters, the update time is also dependent on the sequence complexity (e.g., increases along with the number of compounding angles) and on the update frequency (see Fig. 13c, the update time is over twofold larger when the apex parameters are updated continuously, i.e., at every single frame, rather than when parameters optimization occurs every 5 or more frames). It follows that, if apex parameters are updated, the autotuning strategy should preferably tune them intermittently rather than in a continuous manner. Interestingly, when tuning more than one parameter at a time, the time required to re-program the hardware is equal to the maximum among the tuned acquisition parameters update times and update times do not sum up.

Since the RL framework is not ready for deployment, only transmit power among all the explored parameters is selected for the substitute PID controller. Updating transmit voltage takes less than 5 ms to be updated, thus the requirements set by the technical budgeting are satisfied.

\(^2\)Negative values are explained considering that update time is measured as the difference between the time it takes to acquire a new frame with updated parameters and the average time required for a normal frame acquisition.
Figure 13: Run-time parameters update. TPC: Transmit Power Controller. TGC: Time Gain Control waveform. TX.Apod: Transmit Apodization. Trans.frequency: Transmit Frequency. TW.parameters: Transmit Frequency. TX.waveforms: Transmit Waveform. Event.tx: Transmit Event. [a] Dependency on tuned parameters: update each parameter every 50 frames; 1 plane wave per frame. [b] Dependency on sequence complexity: update TW.parameters every 50 frames; variable of plane waves. [c] Dependency on update frequency: update TW.parameters 1 plane wave per frame; variable update frequency (i.e., of frames between subsequent update events). Update time for each condition is estimated over $N_{updates} = 200$. 

[Graph a] Update Time [s] vs. Parameters

[Graph b] Update Time [s] vs. # Plane Waves

[Graph c] Update Time [s] vs. # Frames between subsequent update events
2 System Integration

Fig 14 shows the complete setup: the Verasonics Vantage research US system streaming channel data to an on-premises server, which performs the beamforming in the GPU and renders the reconstructed images to a web app open on a smartphone. Note that the processing server is displayed through remote desktop on a laptop.

The web app, which acts as a GUI, displays the reconstructed and post processed images and enables the manual update of:

- beamforming method: alternate between traditional method, DAS, and advanced methods, such as ABLE.
- number of PWs: compound a different number of PWs for image formation.
- transmit voltage: manually adjust transmit voltage to modify image intensity.
- autotuning toggle: activate automatic optimization of transmit voltage using the PID controller.
- fps toggle: display the achieved display frame rate.

To initiate acquisition, the user starts the processing server and the web app. As soon as the server is ready for connection, the user can initiate the Matlab script on the Verasonics host computer. The client connects to the server and sends all the initialization parameters. Afterwards, acquisition starts. Data is sent over the network to the application server, processed and visualized through the web app. When Matlab execution stops, the server does not receive further data. After a specific time out, the connection closes and the server listens for a new connection.

If autotuning is set through the web app GUI, a PID controller computes a novel value for the transmit power to reach the desired image intensity. The novel value is transmitted to the Verasonics system, which updates the image settings in the acquisition and software modules.

Figure 14: System integration. Complete setup: Verasonics data acquisition hardware, processing server visualized through remote desktop, user interface open on a smartphone.
CHAPTER 4. SYSTEM VERIFICATION

3 System Verification

Functional requirements are assessed by demonstration. System performance requirements are assessed using the complete setup (Fig. 14) in a condition that simulates a real use case scenario.

PR1 assessment is done by summing up the contribution of all the processes involved, similar to what is done with technical budgeting (but this time with effective measured run-times). The results of this analysis are reported in Tab. 4.4. Both minimal latency with extra time lag A and B equal to zero and maximum latency with max time lags are provided.

A more precise estimate of total latency would require the use of a latency measurement tool\(^3\). However, latency measurement tools are relatively complex and require their own design, so this rough estimate is considered sufficiently accurate at this development stage.

Note that Tab. 4.4 compares also the BMF+PP run-time of DAS and ABLE: DAS beamforming takes slightly less than advanced methods. This difference is attributable to the additional adaptive weights computation in ABLE. The remaining part common to both methods is time of flight (TOF) correction, which appears to be the dominant factor. Notably, even if at sub-system level the BMF30 and PP10 are not satisfied, here latency requirement (PR1) is met for both DAS and ABLE with a single PW. This is related to the technical budgeting which has allocated more resources than needed to other processes which contribute to the total latency.

<table>
<thead>
<tr>
<th></th>
<th>PW = 1</th>
<th>PWs = 5</th>
<th>PWs = 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX/RX (s)</td>
<td>0.000160</td>
<td>0.0008</td>
<td>0.00176</td>
</tr>
<tr>
<td>DMA (s)</td>
<td>0.00006</td>
<td>0.00028</td>
<td>0.00054</td>
</tr>
<tr>
<td>Extra time lag A (s)(^a)</td>
<td>0.0133</td>
<td>0.0133</td>
<td>0.0133</td>
</tr>
<tr>
<td>DT (s)</td>
<td>0.011</td>
<td>0.032</td>
<td>0.065</td>
</tr>
<tr>
<td>Extra time lag B (s)(^b)</td>
<td>0.01564</td>
<td>0.0433</td>
<td>0.08022</td>
</tr>
<tr>
<td></td>
<td>DAS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMF + PP (s)</td>
<td>0.028</td>
<td>0.088</td>
<td>0.175</td>
</tr>
<tr>
<td>Latency (s)</td>
<td>0.03922 ÷ 0.06816</td>
<td>0.12108 ÷ 0.17768</td>
<td>0.2423 ÷ 0.33582</td>
</tr>
<tr>
<td></td>
<td>ABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMF + PP (s)</td>
<td>0.037</td>
<td>0.101</td>
<td>0.202</td>
</tr>
<tr>
<td>Latency (s)</td>
<td>0.04822 ÷ 0.07716</td>
<td>0.13408 ÷ 0.19068</td>
<td>0.2693 ÷ 0.36282</td>
</tr>
</tbody>
</table>

Table 4.4: System performance: display latency. DT and BMF+PP times are indicated as mean value over \(n = 100\) events or frames. PWs: number of plane waves. DT: data transfer, UT: update transfer, BMF: beamforming, PP: post-processing log compression.

\(^a\)Extra time lag A = \(\frac{1}{FR_{acquisition}} + \text{RELOAD}\)

\(^b\)Extra time lag B = \(\frac{1}{FR_{transferring}} + \text{RELOAD}\)

PR2 is verified saving the date-time at which each frame is displayed and measuring the elapsed time between two consecutive frames. As expected, display frame rate turns out to match the frame rate of the slowest concurrent process among acquisition, transferring and processing. The frame rates for each concurrent process are shown in Tab. 4.5 and displayed in Fig. 15. Remind that: (1) acquisition frame rate is user-defined in original settings as the period between two consecutive acquisitions, (2) transferring frame rate depends mainly on DT and UT, (3) processing frame rate is determined by BMF and PP run-times. As aforementioned, acquisition can run faster than transferring thanks to the programmed asynchronous behaviour:

\(^3\)For example, a latency measurement tool is available at Philips to measure the total system latency of an interventional X-Ray system (Philips Azurion Allura), between X-Ray generation and image rendering on the screen.
the external function always reads the last acquired frame. Likewise, transferring frame rate can be higher than processing frame rate, as the beamformer processes always the last frame received by the application server. Tab. 4.5 shows that data processing is the bottleneck for display frame rate. If acquisition or transferring would take more than processing, the display frame rate would be acquisition-constrained or transferring-constrained, respectively.

According to these results, the proposed framework integrated with ABLE does not achieve the required display frame rate (PR2) neither with single PW (27 fps) nor with multiple PWs. Importantly, this is limited by the processing time and not by the framework itself. If the system was framework-limited, instead of processing-limited, single PW imaging would allow a display frame rate of 80 fps, and about 24 fps would be possible also with 5 PWs.

**PR3 and PR4** (i.e., image quality requirements) are satisfied given that BMF10 and BMF20 are met and that no compression is introduced in the image chain.

**PR5** cannot be verified due to the missing advanced autotuning strategy.

---

**Figure 15:** System performance: parallel processes frame rate. Beamforming: ABLE.

<table>
<thead>
<tr>
<th></th>
<th>PW = 1</th>
<th>PWs = 5</th>
<th>PWs = 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquisition (fps)</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Transferring (fps)</td>
<td>80.514</td>
<td>24.658</td>
<td>12.780</td>
</tr>
<tr>
<td>Processing (fps)</td>
<td>26.711</td>
<td>9.888</td>
<td>4.928</td>
</tr>
</tbody>
</table>

**Table 4.5:** System performance: parallel processes frame rate. Processing is intended using advanced beamforming, ABLE.
Chapter 5

Discussion

This work explores the use of cloud computing for the deployment of advanced US processing methods in latency-sensitive applications, such as procedural guidance. In particular, this thesis proves the deployment of two processing methods: (1) a cloud-based advanced digital beamforming using DL, and (2) a cloud-based learning controller for image settings autotuning. To this end, the proposed system sends US channel data to an on-premises processing server over Ethernet connection, displays remotely reconstructed images on a web application open on a third-party device, and enables remote control of image settings either manually from the web app or automatically according to the outcomes of the cloud-based learning controller. The platform is versatile and can be adapted to different imaging schemes and probes. Additionally, the setup can allocate several processing and post-processing methods, given the large compute resources available on the processing server (e.g., load several different digital beamformers and select the optimal one during run-time).

1 Key Outcomes

Performance assessment shows that live streaming of US channel data to the cloud is possible with a frame size up to 10.5 Mbits, which corresponds to a frame of 128 elements, 4.6 cm-depth, and 5 compounding angles. Above this size, the longer transferring time bounds the frame rate below 25 fps, thus restricting the system use to content with slow-moving objects (e.g., thyroid) or where real-time guidance is not required. Although upgradable, this result is already valuable since frame rate around 25 fps are typically needed for navigation, or hand-eye coordination, where a reduced image quality is acceptable (e.g., image quality achieved with only 5 PWs). On the contrary, when the probe stops at the location of interest, the frame rate can decrease, as no probe handling is needed, in favour of a higher image quality needed to support, e.g., quantitative analysis.

With respect to the DL-based image reconstruction algorithm, as expected, the method proves to outperform traditional DAS in terms of spatial resolution (axial and lateral FWHM improve > 5%, Tab. 4.2) and contrast (CNR improve of 5%, Tab. 4.2). This further support the benefit of such a system in clinical context where image quality is key to proper diagnosis and procedure outcome. Despite the increased image quality, the digital beamformer represents the limiting factor for the display frame rate and compromises real-time image rendering. Specifically, the display frame rate is bounded to a maximum of 26 fps with single PW, and even to lower frame rate with multiple firing angles (Tab. 4.4). Also, the processing represents more than 50% of the measured total latency, which means that the long processing time is considerably hampering the use of the setup in latency-sensitive applications which requires eye-hand coordination.

When autotuning is set, the PID controller estimates the desired transmit power making use of the last reconstructed image. The update of this acquisition parameters is fast enough (< 5 ms) to have a negligible effect on the display frame rate and latency. Of note, as explained in Sec-
tion 4.1, other transmit parameters require larger reloading time which compromise extensively the display continuity.

2 Limitations & Potential Improvements

Setup limitations include:

- Long image reconstruction time, especially for large frame size.
- Long update time for certain transmit parameters (e.g., frequency).
- Aperiodic sampling of displayed frames due to the asynchronous behaviour between acquisition and processing (see extra lag time A and B in Chapter 3.2 Section 2.2.1).
- Variable system performance in terms of latency and display frame rate due to the dependency on network traffic.
- Long transmit time for large frame size

With respect to the processing time of the adopted advanced beamforming, since the long run-time is mainly determined by the TOF correction (i.e., equally present in ABLE and DAS) and not by the DL-based stage, potential improvements include: (1) pre-compute the beamforming delays, or (2) use one compute shader thread for every pixel in the image to compute on-the-fly the delays as shown in [29]. As for the long update time, the limit is expected to be mainly associated to the Verasonics Vantage system, hence it is likely to be improved when moving to dedicated hardware.

The last three limitations require a more extensive analysis, and thereby are described in-depth in the following paragraphs.

Aperiodic sampling of displayed frames. Aperiodic sampling refers to time-varying sampling intervals. In the proposed solution, the processed and displayed frames are not acquired at instants equally spaced in time due to the asynchronous programming, which introduces varying extra time lag between concurrent processes. This could affect the system use depending on the considered application. For example, in traditional IGT the aperiodic sampling should not compromise excessively the device use provided that the latency and the display frame rate always meet the real-time requirements. On the contrary, if reconstructed images are meant to be used for shear wave imaging, knowing the exact frame sampling interval is essential to estimate correctly the shear wave velocity and, as a consequence, the tissue elastic properties. To enable the use of this system with post-processing methods, like shear wave imaging, a timestamp could be added to the raw channel data and sent to the processing server as side information.

Importantly, the autotuning strategy could also be affected by an aperiodic sampling, as shown by the extensive research performed in several areas of Control Theory to study the stability of systems with arbitrary time-varying sampling intervals [30]. With particular attention to the proposed autotuning approach, the presence of observation and action delays in dynamical systems is well-known to degrade the performance of the RL agents, especially when these delays cannot be estimated accurately [31, 32, 33]. Fig. 16 shows the agent-environment interaction when observation and action delays are present. In this case, these delays vary over time. Possible solutions to revise RL framework when dealing with stochastic delays are proposed in [32, 33].
Figure 16: Reinforcement learning in a delayed environment. $S$: state; $a$: action. $S_{t+n}$ corresponds to the instant at which the action $a_t$ suggested by the agent according to $S_t$ takes place. Timeline shaded in yellow corresponds to settings re-loading (no state is collected as acquisition stops when re-loading settings).

Variable system performance. In the proposed solution, as in all foreseeable cloud-based systems, the performance is dependent on the network traffic: when the network is congested, data transfer can take longer than usual, ultimately increasing the total latency. If the latency increases over a maximum acceptable delay, the eye-hand coordination is compromised and the system cannot be used for procedural guidance.

Rate control is typically used in interactive applications to ensure that the message is received at the decoder before the pre-defined maximum end-to-end delay [34]. The basic principle of rate control methods is to reduce data size to ensure that the transmission of the compressed message, with the actual network conditions, introduces a delay below the maximum allowable. Importantly, this compression typically introduces data distortion, hence the design of the optimal strategy requires solving a trade-off between source fidelity and data rate. In rate control, this rate-distortion problem is expressed as a delay-constrained allocation problem. Solving such a problem means finding the encoder’s parameters that return a data rate compatible with the actual network bandwidth while minimizing the distortion. The two most common rate control strategies are: (1) adapt the encoder’s quantization parameters to satisfy the delay constraint (quantization typically follows a linear or non-linear transform which improves the performance of the quantization step) [35]; (2) develop several linear or non-linear transforms to target different bandwidth, and during inference select the transform which was developed to target a bandwidth closer to the actual network condition (communicate to the decoder the index of the selected transform as side info) [36].

Given a specific encoder’s structure (e.g., discrete cosine transform followed by quantization), tuning the encoder’s parameters leads to different rate-distortion trade-offs. These working points are identified by prioritizing rate over distortion and vice versa (Fig. 17a). The classical solution to identify different trade-off points is based on the discrete version of Lagrangian optimization [34]. In brief, a Lagrangian multiplier $\lambda \geq 0$ is introduced to define the cost of the rate-distortion (R-D) problem as $J = R + \lambda D$. For each $\lambda$, the encoder’s parameters are adjusted to find the optimal solution to the problem with $J(\lambda)$. This iterative process results into several points which define the so-called rate-distortion curve (Fig. 17b). Importantly, different coding strategies (i.e., encoder’s structure) return a different R-D characteristic. In Fig. 17b, the model that returns the blue R-D curve outperforms the one that returns the red curve because for an equal rate ($R^*$) less distortion is introduced ($D_1 < D_2$). Typically, coding efficiency (i.e., a better R-D curve) increases along with model complexity, hence more sophisticated encoding strategies return better trade-offs. However, when dealing with latency-constrained applications complexity cannot increase indefinitely, as complex models typically result into long encoding and decoding time, which ultimately compromise the end-to-end delay. As schematized in Fig. 18, increasing model complexity leads to an initial improvement in the end-to-end delay, $T$, (equal to encoding time + transmission time + decoding time) due to the reduced data rate. Nevertheless, for very large complex models, $T$ increases again due to the long encoding and
decoding time, which together overcome the benefit achieved by the data rate reduction. Additionally, very complex models require large compute resources, which might be unavailable at the local site or in embedded systems (e.g., digital US probe). The constraints in terms of end-to-end delay \(T_{\text{max}}\) and maximum resources \(C_{\text{max}}\) limit the bunch of feasible models, hence constraining the maximum achievable rate-distortion performance.

If the best of these manufacturable encoders does not offer a working point which satisfies at the same time the image quality and the latency system requirements, the designer should prioritize one of the two requirements over the other based on the use case scenario. In image-guided interventions, for instance, latency and frame rate are likely to be more important than image quality during navigation phase. On the contrary, if the imaging system is used for tissue characterization (e.g., during catheter ablation to characterize the lesion), image quality should be maximized at the cost of a longer latency.

**Long transmit time for large frame size.** Above a certain frame size, the proposed solution prevents live streaming to the processing server (<100 ms latency, >25-30 fps), hence compromising the system use in real-time applications which require very high image quality (e.g., multiple PWs). Improving this outcome is not straightforward, as this is likely the most challenging limiting factor towards the introduction of cloud-based US devices on the market. See, for instance, the work of Meir and Rubinsky, in 2009, where the feasibility of cloud-based 3D image reconstruction was first demonstrated, but without the possibility of real-time feedback [37].

The very first aspect to consider when designing a live communication between two systems is whether to directly stream data or buffering data followed by streaming. The main difference between these two approaches is that direct streaming leads to peak data rate (i.e., instants where the link should transmit data at very high speed, followed by period with no data transmission - e.g., due to inter-frame period); whereas buffering allows distributing the data over a longer period of time. Hence, even though direct streaming reduces overall time to transmit data, it requires much larger network bandwidth to support these peaks in data rate.

So far, only buffer followed by streaming has been considered for US channel data transfer, as acquisition peak data rate typically exceeds what standard communication links can offer. To give an example, with a sequence scheme similar to the one described in Chapter 3.2, the required rate would be \(R = Nch \times Fs \times bpp = 128 \times 2 \times f \times 16\text{bit} = 26.624\text{Gbps}\), which already overcomes the latest generation of Ethernet IEEE 802.3ae (10 Gbps) or Wi-Fi 802.11ax (9.6 Gbps). These peaks in data rate might be supported by next generation 6G Internet connection,
which is expected to offer a bandwidth up to 1 Tbps [38]. However, it is important to stress that, even so, the absence of a local memory would prevent any form of rate control (i.e., data compression) which instead is likely to be valuable to deal with the varying network traffic (as described in the previous paragraph).

On the other hand, the presence of a buffer on the local device decouples the acquisition rate from the network achievable throughput, reducing the needed transmission rate to \( R = S \times PRF \), with \( S \): data size, \( PRF \): pulse repetition frequency. In this case, \( S \) can be either a single acquisition \( (S = Nch \times Nax \times bpp = Nch \times \frac{2}{1} \times Fs \times bpp) \) with \( Nax \): axial samples, \( D \): depth expressed in number of wavelengths, \( f \): transmit frequency), or the whole frame when multiple acquisitions are transmitted together \( (S = Nch \times Nax \times bpp \times Npws) \) with \( Npws \): number of acquisitions per frame). In this latter case, the \( PRF \) is intended as the acquisition frame rate. Note that this reduction in datastream is given by the fact that between two consecutive acquisitions or two consecutive frames, there is a period where no data is acquired (i.e., \( 1/PRF > Nax/Fs \)). For comparison with the aforementioned example, this architecture would reduce the previous rate to \( R = Nch \times Nax \times bpp \times PRF = 128 \times 1024 \times 16bit \times 6.25KHz = 13.981Gbps \) for single acquisition transmission, and to \( R = Nch \times Nax \times bpp \times Npws \times FR = 128 \times 1024 \times 16bit \times 11 \times 100Hz = 2.307Gbps \) in case of frame transmission with 11 PWs per frame. As mentioned, the reduced rate 2.307 Gbps is achieved only because 11 PWs is not the maximum number of PWs which could be acquired in the inter-frame period. If one set the number of steered angles equal to the maximum number which can fit in the inter-frame period, the data rate goes up again to 13.981 Gpbs, as each acquisition needs to be sent before a new acquisition completes.

The presence of a buffer, not only reduces the needed transmission rate, but offers also the possibility to program the transmission asynchronous to the acquisition. This means that, even if data is written to the buffer at a certain data rate (i.e., \( R = S \times PRF \)), the transmission can go at its own rate depending on the available bandwidth (i.e., sending only the most recently acquired frame). This asynchronicity could be advantageous if one wants to store all the acquired data on the local device for offline processing.

Importantly, the choice between direct streaming or buffering is not the only thing to consider when designing a client-server communication. Indeed, even when the bandwidth is high enough to support the data rate, it might occur that the application throughput does not reach the maximum theoretical bandwidth. This could be due to the time required to encode and decode data in a serial transmission, which introduces small delays between subsequent data packets sent over the network. If this is the case, multi-threaded send and receive methods [39] could increase the maximum achievable throughput. With PW imaging, for instance, multiple acquisitions can be transmitted concurrently using multi-threading programming. In this approach, both client and server have a number of parallel threads equal to the number of PWs.
3 Towards Cloud-based Ultrasound

This section details the conceptual diagram proposed in Fig. 2 according to the insights gathered with the technology demonstrator and previous considerations. The objective of this final section is to imagine a future manufacturable cloud-based US device without being constrained by other pre-defined interacting systems (e.g., Verasonics). Importantly, the proposed architecture is strongly dependent on the intended use: US-assisted therapy. As aforementioned (see constraint CR3 in Tab. 2.3), this application requires always-on reliable systems, hence the solution is meant to offer both a default cloud-based beamforming and post-processing and an alternative local image formation to use in emergency situation in disconnected environment or when network is highly congested.

A schematic representation of the envisioned framework is presented in Fig. 19. Ultrasonic echoes are transmitted and received with a US probe, which is connected through a coaxial cable to a local console. At the console level, data is digitized and stored in a memory. This buffering step avoids peak data rate associated with direct streaming and allows rate control. In default mode, a processing unit is used to encode the raw US channel data according to a rate control algorithm and to send it over the network to the cloud where advanced processing occurs. To this end, a multi-threaded data transfer is preferred to exploit all the available network resources. The processor is responsible also for receiving from the cloud the reconstructed images and clinical indices, as well as the image settings optimized according to the cloud-based autotuning strategy. The reconstructed images and the quantitative indices are rendered through the display, whereas the updates are used to re-program the transmit beamformer and the waveform generator. In case of Internet drop or when the available bandwidth requires to compress data (i.e., rate control to satisfy the end-to-end maximum latency) so much that the advanced method no longer offers any benefit over the traditional scheme, the local basic processing is used. In this emergency mode, digital beamforming and post-processing execute on the console CPU and reconstructed images are directly rendered through the connected screen without the need of any wireless transmission. This emergency mode lasts until the network connection is stable again or the available bandwidth increases to the minimally acceptable data rate.

With respect to the user interface, Fig. 19 shows a unique local user interface (i.e., at the place where acquisition occurs), however in default mode the cloud-based reconstructed images and clinical outputs can be transmitted to a remote user interface for real-time surgical consultation and remote manual image settings optimization.

Note that in the proposed solution, the digitization is performed on a console rather than on

---

Figure 19: Envisioned cloud-based ultrasound system for image-guided therapy. On-site device includes ultrasound probe and a console. TXBF: TX-beamformer which generates pulse waveforms; Pulser: high-voltage pulser which converts waveforms into high-voltage signals used to excite the transducer elements; T/R: TX/RX-switch; AFE: Analog receive Front End which include amplifiers and analog-to-digital converter; CPU: Central Processing Unit; RAM: Random-Access Memory; NIC: Network Interface Card.
the digital probe (i.e., front end and mid-end embedded on the probe). This follows the fact that in life-threatening applications, such as US-guided therapy, the probe always needs to be connected to a display to ensure that images can be rendered even in case of Internet drop. This wired connection compromises the major benefit of digital probes, which is the possibility to be completely wireless, hence there is no significant advantage in moving the digitization to the probe. On the contrary, console device could avoid thermal power constraints typically associated with on-probe digitization and offer large storage for offline post-processing.

For comparison, Fig. 20 shows the envisioned architecture for a non-critical application, such as cloud-based point-of-care US or US-guided simple procedures. Importantly, the system in this case is not a critical system (as opposed to a device for US-guided surgical treatment), hence the reliability constraint can be relaxed. Specifically, the wired connection for emergency mode can be removed. This means that the solution could benefit from moving the analog and digital front-ends to the probe. This avoids the need for a console and makes the system more compact and manoeuvrable. In this portable configuration, the probe is also responsible to temporarily store the US channel data (i.e., small memory due to size constraints) and transfer it wirelessly to the cloud, where processing takes place. The outcomes of the processing pipeline are returned to a third-party connected device for display. If Internet drops, the system either stops working or - if basic data processing is available on the probe - the probe can be connected through a cable (e.g., USB) to a display (e.g., a smartphone).

Figure 20: Envisioned cloud-based ultrasound system for point of care ultrasound. [a] Framework. [b] Probe details. TXBF: TX-beamformer which generates pulse waveforms; Pulser: high-voltage pulser which converts waveforms into high-voltage signals used to excite the transducer elements; T/R: TX/RX-switch; AFE: Analog receive Front End which include amplifiers and analog-to-digital converter; CPU: Central Processing Unit; RAM: Random-Access Memory. USB port: Universal Serial Bus, which can be added to make the system working also in disconnected environment or when network is highly congested.

1 Remember that digital probes are thermally limited devices which must keep the surface of the device in contact with the skin below 43 °C in order to comply with medical safety regulations (IEC 60601-1). This constraint is typically more stringent when using invasive probe, e.g., intra-cardiac echo.

2 Recent volume growth in ultrasound-guided procedures is likely the result of the addition of ultrasound guidance to procedures previously performed without it [5].

3 A wireless network interface controller (WNIC) is a network interface controller which connects to a wireless network, such as Wi-Fi. A WNIC works on the layers 1 and 2 of the OSI model and uses an antenna to communicate via radio waves.
Chapter 6

Conclusions

This work proposes a cloud-based US imaging system to support the deployment of compute-intensive US processing methods in critical and latency-sensitive applications, such as IGT. The envisioned system is intended to be used in particular for advanced image formation and automatic image settings optimization through, e.g., quality-based, autotuning strategies. The proposed framework includes live streaming of raw, high-bandwidth, US channel data to the cloud for remote processing and real-time feedback either for image rendering or control of run-time parameters.

The performance of this distributed architecture is assessed with a technology demonstrator. The experimental prototype includes the communication of US in-phase and quadrature channel data from a programmable US platform to an on-premises processing server. Advanced processing executes in Python on the processing server, and the reconstructed images are rendered through a web app open on a third-party connected device (e.g., smartphone). Both manual tuning from the app interface and autotuning from a server-based controller are allowed during run-time. System verification demonstrates (1) near-real-time cloud-based US image formation with a trained DL method, and (2) transmit power autotuning with a basic controller. Current limitations include long processing time and slow hardware re-programming for certain transmit parameters update. Significant improvements are expected when moving to optimized algorithm implementation and dedicated hardware.

Besides being of support for the real-time inference of other US image formation or post-processing methods, the developed technology demonstrator provides important insights for the design of future cloud-based US imaging systems.
Appendix A

Supplementary Data

1 Communication Protocol

This section presents a comparison between different communication protocols over a network which resembles the one used in the final technology demonstrator. The analysis aims to identify the best protocol for the target application.

1.1 Materials & Methods

1.1.1 Programming

Network hosts:

- Verasonics Vantage 256 Host Computer
- Windows server: 2 CPUs (Intel Xeon CPU E5-2640 v4 @ 2.40GHz, 2401 MHz, 10 Cores, 20 Logical Processors), and 4 GPUs (4095 MB NVIDIA TITAN X (Pascal))

The two hosts are connected to the Internet via Ethernet, as shown in Fig. 21. The path of a packet of data as it travels from the client to the server is assessed with Traceroute, a network diagnostic tool.

Route Hops:

- Verasonics Vantage 256 Host Computer has a gigabit Ethernet card, and it is connected through a Cat6a Ethernet cable (last generation cable with up to 10 Gbps transmission speed).
- Fast Ethernet Office Switch (Intellinet 5-port, model: 523301) supports devices at 10 or 100 Mbps over Ethernet cables;
- Router Flux Workspace 131.155.124.1 has a gigabit Ethernet card;
- Server 131.155.34.167 has a gigabit Ethernet card.

Verasonics Host Computer and the switch represent the bottleneck of the network and limit the theoretical maximum bandwidth to 100 Mbps.

Two different communication interfaces are investigated:

- Transmission Control Protocol (TCP or TCP/IP) Interface: communication using the TCP/IP protocol
- User Datagram Protocol (UDP or UDP/IP) Interface: communication using the UDP/IP protocol
Both interface types are implemented with socket module. Note that a socket is defined at the client and another at the server node, where the client uses Matlab environment, while the server uses Python environment.

TCP socket is implemented in Matlab with `tcpclient` function, and in Python with `socket.socket(socket.AF_INET, socket.SOCK_STREAM)`. UDP socket is implemented in Matlab with `udpport` function, and in Python with `socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`.

### 1.1.2 Performance Assessment

The performance of the two communication interfaces is evaluated in terms of:

- **Round-trip-time (RTT)**: duration from when a network endpoint sends a request to when it receives a response from the other network endpoint. Typically, the RTT trend is also evaluated to assess buffer filling (e.g., RTT increase means longer queuing delays because the buffer is filling up). RTT provides a good estimate of network latency. It can be estimated at the network level, or at the application level.

- **Throughput**: the quantity of data being sent/received (all data flowing through a link) by unit of time. It can be estimated at the network level (all data, including headers), or at the application level (only useful data).

These performance metrics are assessed at the network level and at the application level.

**Network-level.** At the network level, the interface performance is evaluated with standard diagnostic tools and has the principal scope of tuning the protocols parameters and define a baseline for the expected performances.
• **Netsh** tool (https://docs.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-contexts) from Microsoft is used to verify Maximum Transmission Unit (MTU) and global parameters.

• **Ping** tool (https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/ping) and **PsPing** tool (https://docs.microsoft.com/en-us/sysinternals/downloads/psping) from Microsoft are used to assess the latency (in terms of RTT) between the Verasonics Host Computer (i.e., Windows machine) and the server over TCP or UDP connection. This is done computing the ping time.

• **iPerf** tool (https://iperf.fr/) is used to measure the maximum achievable throughput on IP networks and support tuning of various parameters related to timing, buffers and protocols (TCP, UDP, with IPv4 and IPv6).

**Application-level.** At the application level, the performance of the software-server interface is estimated by programming a socket module in Matlab and in Python. UDP and TCP protocols are implemented in two different scripts.

- Latency is assessed by computing the RTT: time required to send a packet from Matlab on the Verasonics machine to Python on the server, plus the time required to send the same packet from Python to Matlab. Note that the latency is not always equal to half of the application-level RTT, because the latency may be asymmetric between any two given endpoints. Additionally, application-level RTT includes the processing delay at the application level of the echo endpoint. The packet size for RTT tests is set to 32 bytes (much below the MTU). This allows to prove the minimum time achievable and avoid confounding effects that congestion in the operating system or network can induce. The elapsed time to complete the echo (i.e., send data from client to server and receive it back) is computed with the tic and toc functions in Matlab. Note that Matlab tic/toc has an accuracy of about 0.000001 seconds.

- Application-level throughput is assessed by computing the ratio between the amount of transmitted application-specific data and time required for transmission. The latter is estimated assuming a symmetric connection, hence computing half the overall time that is required to perform an echo transmission: (1) transmit a certain message from Matlab to Python, (2) read it in Python and decode it into the expected datatype, (3) encode it again in bytes and send it back from Python environment to Matlab, (4) read the message in Matlab and verify that it corresponds to the initial payload. Note that this value does not consider packet headers but only useful data, for this reason is sometimes called "goodput". The elapsed time is computed with tic and toc functions at the extremities of the socket functions write and read in Matlab. As mentioned in the previous paragraph, that Matlab tic/toc has an accuracy of about 0.000001 seconds.

\[
Application\;Throughput = \frac{DataLength}{(Elapsed\;Time_{Echo}/2)}
\]

Importantly, this is a rough estimate of the throughput, as the effective time required to transmit the data from one endpoint of the network to the other is not exactly equal to half of the elapsed time to complete the echo transmission. This discrepancy is mainly due to asymmetries between the two endpoints. Even so, this is considered to be the most appropriate estimate of the time required by the final application to communicate data between the link endpoints.

The assessed network performance can also be used to compute:

- **Data Transmission Time**: elapsed time to transmit raw US channel data from Matlab environment on the Verasonics machine to the Python environment on the server.
• **Image Transmission Time**: elapsed time to transmit reconstructed and post-processed frames from the Python environment on the server to the Matlab environment on the Verasonics machine.

### 1.2 Results & Discussion

#### 1.2.1 Network-level

**IP Interface**  *Netsh* tool shows an MTU of 1500 bytes, in agreement with what expected due to Ethernet connection. Additionally, global parameters assessment confirms an enabled receive-side scaling rate for TCP protocol, which means that the receive window can increase up to 1 GB.

**TCP/IP Interface** Both *Ping* and *PsPing* tool estimate a round-trip-time for TCP/IP Interface below 1 ms with packets of 1000 bytes. 1005 iterations (warmup 5) sending 8 bytes TCP latency test: 100 %

TCP round-trip latency statistics (post warmup):
- Minimum = 0.23 ms
- Maximum = 0.34 ms
- Average = 0.26 ms.

*iPerf* returns a throughput of about 94 Mbps when receive window is set above 15 KB; for lower window size the throughput decreases slightly (sent 113 MBytes in 10 s, estimated bandwidth: 94.4 Mbps). The throughput value estimated with this diagnostic tool matches well the maximum theoretical value of 100 Mbps estimated considering the bottleneck over the end-to-end connection, as mentioned above. Interestingly, also the minimum value of receive window size is in good agreement with the theoretical value expected computing the Bandwidth-Delay Product (BDP). Indeed, BDP is defined as the maximum amount of data on the network circuit at any given time (i.e., data transmitted but not yet acknowledged) and it is typically regarded to as the minimum value for the window size (if the receiver is not able to process the data as fast as it arrives, the receive buffer will gradually fill).

\[
data \text{ links capacity } \times \text{ round-trip-time} = 100 \text{Mbps} \times 1\text{ms} = 100\text{Kb} = 12.5\text{KB}
\]

**UDP/IP Interface**  *iPerf* creates a constant bit rate UDP stream (continuously send UDP datagram from client to server) and evaluates how many bytes are sent in a specified time interval. UDP datagram cannot be set larger than 65507 bytes, this is in agreement with the fact that UDP relies on IP protocol for data fragmentation in MTU-long packets (as opposed to TCP which adjusts segment size to be smaller than MTU) and that IP has an upper bound limit in datagram length of 65507 bytes (ref. to Appendix B for a more comprehensive explanation).

This observation is key also to program the socket module properly: if the data that needs to be transmitted is larger than maximum length, it must be divided into chunks before being written to the UDP socket. A related remark is about the datagram loss: as said, a UDP datagram larger than 1470 bytes is fragmented by the IP protocol into multiple IP packets. Therefore, the more the fragments, the higher the risk to lose a single IP packet and thereby to not being able to retrieve the original datagram.

Similar to what observed with TCP, the maximum throughput achievable according to *iPerf* UDP test is \(\sim 95\text{Mbps}\).

#### 1.2.2 Application-level

**TCP/IP Protocol** Two RTT patterns are observed depending on the day and time. Fig. 22 depicts the trend and histogram of RTT values over 10000 iterations in the two different conditions. The statistics are reported in Tab. A.1. The median value and the variability (i.e., IQR) change considerably between the two patterns (i.e., the median value is multiplied by a factor of 6). The differences observed in the RTT statistics depending on the day and time might be explained...
considering the numerous factors that influence this estimate. Some of these factors are constant (e.g., client-server distance, transmission medium, number of network hops), while others can vary over time (e.g., network traffic level, server response time). On this regard, it is worth to underline that the used server is accessible to multiple users, thus its resources might be limited differently depending on the overall use. Remarkably, the minimum value remains almost stable between the two patterns. This observation is further supported by previous works that show that the minimum RTT is less affected by network conditions and thereby a better representative of the effective connection latency [40]. Interestingly, this minimum RTT value is also closer to the latency estimate obtained with the ping tests. The slightly larger value can be explained considering that the ping tests are usually performed within a transport protocol, while application-level RTT includes additional processing delays caused by higher-level protocols and applications.

TCP write function (both in Matlab and in Python) can write any data length. Indeed, TCP/IP has intrinsic protocols to ensure data fragmentation and queuing, as explained in Appendix B. In that regard, independently of data length, the messages are always delivered in good order and without missing packets (0%). This behavior is due to the reliability of the TCP/IP protocol, which automatically ensures the correct transmission through acknowledgement.

As shown in Fig. 23, the maximum TCP application-level throughput increases with larger data length until a certain value around 70-80 Mbps where it saturates. The initial increment could be related to the fact that small data size does not allow exploiting the bandwidth at its maximum potential. On the contrary, the asymptotic behavior observed for high data dimension is due to the limited resources: when the maximum data rate is achieved, additional increase in payload dimension does not result into an improvement in the overall throughput. The fact that the maximum throughput is slightly different from the theoretical maximum bandwidth might be related to TCP headers and re-sending of loss packets.

As presented in Fig. 24, the different data types show no difference for low data size (i.e., the network is distant from the congestion) but diverge for large load. This might be due to the fact that the binarization process is dependent on the number of elements, and not only the total amount of bytes (e.g., character-related encoding steps).
Figure 22: Round-Trip-Time. Message size: 32 bytes. Number of iterations: 10000. (a) Pattern 1 Trend. (b) Pattern 2 Trend. (c) Pattern 1 Histogram. (d) Pattern 2 Histogram.


<table>
<thead>
<tr>
<th>Protocol</th>
<th>Min [ms]</th>
<th>Max [ms]</th>
<th>Median [ms]</th>
<th>IQR [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>1.8061</td>
<td>20.195</td>
<td>5.9688</td>
<td>5.9602</td>
</tr>
<tr>
<td>UDP</td>
<td>0.99885</td>
<td>11.146</td>
<td>3.123</td>
<td>4.0731</td>
</tr>
<tr>
<td>Pattern 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>1.9406</td>
<td>78.157</td>
<td>31.253</td>
<td>0.0404</td>
</tr>
<tr>
<td>UDP</td>
<td>1.0557</td>
<td>22.398</td>
<td>15.622</td>
<td>0.11498</td>
</tr>
</tbody>
</table>

Figure 23: Application throughput: protocols. (1) Transmission Control Protocol (TCP or TCP/IP); (2) User Datagram Protocol (UDP or UDP/IP). Number of iterations: 100. To avoid packet loss with UDP, a pause is added between chunks of 65507 during transmission: pause of 0.003 s in Python when transmitting 1e6 bytes; pause of 0.02 s in Python when transmitting 1e7 bytes; pause of 0.1 s in Matlab and of 0.03 s in Python when transmitting 1e8 bytes.
UDP/IP Protocol  Similar to TCP, two RTT patterns are observed for the UDP/IP interface depending on the day and time (Fig. 22). The statistics shown in Tab. A.1 point out an increase in the median value for the second pattern. Also in this case, the minimum value for UDP RTT seems to represent a more robust statistic, as it remains almost constant between the two patterns (∼1 ms).

As opposed to TCP, UDP relies on the IP protocol to fragment data into MTU-length packets, and IP protocol can only receive datagram of size less than 65507 bytes. As a consequence, when programming UDP socket module, the data must be divided in chunks of maximum 65507 bytes before being written to the socket for transmission. Matlab does not allow setting `u.OutputDatagramSize` above 65507 bytes, and even if the user does not personally divide in chunks of 65507, the socket module does it automatically. In python, the socket cannot send (socket.sendall) messages larger than 65507 bytes. As a consequence, the user is required to divide the message in chunks before writing them to the UDP socket. Even so, if the sender transmits data over the UDP at a speed exceeding the processing speed of the receiver, the receive buffer will gradually fill and once the buffer is full, new packets will just be discarded without re-transmission (contrary to TCP). The principal consequence of this overrunning buffer process is packet loss. Fig. 25 indicates the percentage of packets loss when transmitting data over a UDP socket module. The figure shows that loss percentage increases when moving to higher message size. This is reasonable as larger message means more packets, hence more chance to miss one. To counteract packet loss, a pause can be added between sent data chunks. This could leave more time to the receiver buffer to process data and avoid congestion. Note that this strategy does not ensure that all data is properly sent (i.e., adding a pause during transmission reduces the risk of packet loss, but does not ensure a secure transmission). If data loss cannot be tolerated, control strategies and re-transmission mechanisms need to be implemented from scratch. This typically causes UDP to lose all its advantages w.r.t. TCP in terms of simplicity and transmission speed. A strategy to check the number of missing packets and the order of transmitted packets is implemented. Importantly, the control methods require dividing the message into chunks of 1470 to be consistent with the Ethernet MTU and does not rely on IP for fragmentation. This limits impressively the maximum allowable application throughput (data not shown).
**APPENDIX A. SUPPLEMENTARY DATA**

Figure 25: *User datagram protocol packet loss.* Data is divided into chunks of 65507 bytes and a pause of different values is added between two consecutive chunks. Note the different abscissas between the two conditions and the different pause values.

Fig. 23 shows the application-level UDP Protocol throughput achieved when dividing the message into 65507-bytes-chunks and adding pauses between transmitted datagrams to avoid congestion. Pauses are selected according to Fig. 25: depending on the size, the smallest pause that in Fig. 25 returns a packet loss equal to 0%. Remember that this is a strategy to try to avoid congestion and complete the transmission without packet loss, but it does not ensure successful transmission. Between 10 KB and 10 MB, the UDP protocol outperforms TCP in terms of application-level throughput. This is probably related to the fact that, even though pauses are added between subsequent chunks, the UDP protocol has fewer controls to perform as compared to TCP (i.e., lower latency) and a smaller header (i.e., lower overhead) which contribute to increase the amount of useful data transmitted. Nevertheless, when increasing the data size to 100 MB, also the Python input buffer begins to be congested (for size ≤ 10 MB, only Matlab input buffer is congested and thereby pauses are added only in Python to slow down transmission), requiring additional pauses between chunks sent in Matlab. Given the large values of pauses needed, a dramatic drop in the throughput is observed.

1.3 Conclusions

- UDP/IP Protocol has a lower end-to-end latency than TCP/IP Protocol: both the UDP RTT minimum value as well as the median value are about half the values observed for TCP protocol, independently of the pattern. This is in good agreement with the characteristics of the two protocols (refer to Appendix B).
- With TCP, all the packets sent are transmitted and received in good order.
- With UDP, the larger the size of the message sent, the higher the probability of overrunning the buffer. When this occurs (e.g., for data of size > 1 GB), UDP has no strategies to re-transmit loss packets. If packet loss is not tolerable, developers need to implement a re-transmission strategy from scratch, finally compromising the advantages of UDP in terms of simplicity and speed. Adding a pause between subsequent sent packets can reduce the risk of congestion, but it does not ensure a successful transmission.
• TCP/IP maximum attainable application-level throughput remains around 70 Mbps (theoretical maximum transfer speed for the presented network: 100 Mbps).
2 Image Settings Optimization during Run-time

This section analyses the time needed to update image settings during run-time. This systematic analysis aims to characterize the expected behaviour of the technology demonstrator when an autotuning strategy is used to optimize run-time parameters.

2.1 Materials & Methods

2.1.1 Programming

At the level of the external function, the attribute structures in Matlab workspace are called with eval(base, var1), modified, and re-assigned after modification in the Matlab base environment with assigning(base, var1, novelValue). Of note, also when using this automatized update, the modifications have effect only when the structures are re-loaded into the software sequencer and the hardware, which means the next time runAcq calls VSX.

2.1.2 Performance Assessment

The performance of the software-acquisition hardware interface is estimated by programming a sequence for S5-1 phased array, using divergent wave transmits. Sampling rate is set to 4 times the transmit frequency (mode: NS200BW). The performance is assessed in terms of:

- **Parameter Updating Time**: time added to the normal operation to update the modified structures. It is estimated computing the difference between the time it takes to acquire a new frame with updated parameters (e.g., higher frequency) and the average time required for a normal frame acquisition (without parameter tuning).

The update time is evaluated for different acquisition parameters to detect possible differences in performance. The investigated parameters are listed and briefly described here:

- **Time Gain Control waveform**: modified changing the TGC structure. The TGC structure defines the time gain control curve\(^1\) to be used with the VDAS hardware. The curve is defined by eight control points; the first point always starts at 0, and for the others the values range from 0 to 1023 (max. gain).

- **Transmit Power Controller**: modified changing the TPC structure. TPC provides a high voltage supply for the transmitters on the VDAS modules. Its range is 1 - 96 volts, but typically probes have their own maxima voltage level (around 30-50 volts).

- **Transmit Apodization Weights**: modified changing the TX.Apod structure. TX.Apod specifies an apodization weighting function across the transmit aperture. TX.Apod values range from -1.0 to 1.0, with negative values inverting the waveform polarity. When using the VDAS hardware, the TX.Apod array is used both as an on/off indicator, with 0 values turning off a transmitter, and non-zero values enabling it, and for transmit apodization using transmit waveform duty cycle control.

- **Transmit Frequency**: modified changing either Trans.Frequency, or TW.parameters, or TX.waveform, or Event.tx. All attributes can be used to change the transmit frequency, but they differ in the SetUp Script definition. Trans.Frequency is the center frequency of the transducer, and it is computed when initializing the Trans structure according to the used probe. It is used by the system to determine the scaling of all system variables using wavelength units The first element in the TW.parameters array indicates the transmit

\(^1\)During an ultrasound send-receive cycle, the magnitude of reflected signal depends on the depth of penetration due to attenuation. The purpose of the Time Gain Control is to normalize the signal amplitude with time; compensating for depth.
APPENDIX A. SUPPLEMENTARY DATA

The frequency of the transmitted burst, and it is typically set equal to the Trans.Frequency, but not necessarily. The TX.waveform attribute is the index into an array of TW objects that define the transmit waveforms to be generated by each of the active transmitters, so one can define multiple TW.parameters and then change every time which one is preferred. Note that this scheme allows to tune the frequency only among discrete values. Likewise, one can create multiple TX structures, each one using a different TW.parameters array, and then modify the transmit frequency by changing the Event.tx attribute, which indicates the TX structure array to use for transmit.

For testing purpose, the novel value is set randomly within a prescribed range (note that in the final application, the value is returned by the DL learning controller). Except for one experiment where the influence of this factor on update time is evaluated, adding or subtracting a specific quantity to the initial value.

Structures update is not performed every time a frame is acquired, but with a certain update frequency (i.e., number of frames interleaved between two consecutive updates). Most of the tests are performed with structures update every 50 frames. Except for one experiment where the influence of this factor on update time is evaluated, varying the update frequency.

The update time is compared also with the traditional Verasonics tuning strategy through the GUI: manual tuning.

2.2 Results & Discussion

Fig. 26 depicts the update time computed when tuning different parameters. The box-chart highlights the presence of two distinct groups of parameters: a first group with a short update time where TGC and TPC are located, and a second group characterized by longer update time where TX.Apod structure as well as all the transmit-frequency-modifying structures are identified (TGC: 0.0034 ± 0.0144; TPC: 0.0032 ± 0.0146; TX.APod: 0.2298 ± 0.0017; Trans.Frequency: 0.2222 ± 0.0021; TW.parameters: 0.2250 ± 0.0040; TX.waveform: 0.2250 ± 0.0040; Event.tx: 0.2251 ± 0.0021).

This different behavior might be related to the fact that the second group presents structures that have added VDAS attributes for programming the hardware, or that anyhow determines the update of these structures (e.g., Trans.Frequency has not VDAS attributes, but - when changing it - also TW is changed as the transmit frequency indicated in TW is defined as Trans.Frequency). These VDAS components are typically set by VSX and runAcq using the attributes defined in these structures (e.g., TW and TX). As a consequence, when updating the attributes of these objects, also the VDAS attributes need to be updated. This is done by calling an update function ‘update.m’ when a change to the structure that determines the added VDAS components is detected.

For sake of clarity, note that the negative values observed for TPC and TGC are attributable to the method used to compute the Update Time. Since the Update Time is computed as the difference between a single observed time loop computed when a parameter is updated and a mean value computed when no parameters are updated, the single time loop might remain below the average, returning a negative difference.

Observe also that TPC and TGC update times have a larger variability than update time for the second group of parameters. This might be explained by assuming that the update of the parameters belonging to the second group triggers a well-defined process, i.e., FPGA programming.

When comparing the implemented update strategy with the traditional one through the GUI, the update time is comparable (Fig. 27). Interestingly, also with the manual tuning, the time required to update the parameters that have added VDAS attributes is longer than the time required when modifying TGC and TPC. As mentioned above, TGC and TPC with automatic update have larger variability than TW.parameters and TX.Apod. Remarkably, in this case, manual
update has also a much larger variability than automatic update for both groups of parameters.

Fig. 28 shows the update time required when tuning one or more parameters. Interestingly, when modifying the attributes of multiple structures, the update time remains equal to the update time of the parameter that requires the longer time. This observation suggests that there are two possible update time: one for those parameters that does not require to call VsUpdate.m (e.g., TGC, TPC) and one for those parameters that needs to recalculate the values of certain attributes and re-program the structures in the hardware (e.g., TW.parameters). Once this second update process is triggered, the process takes the same amount of time regardless of how many structures have been modified.

The role of the tuning effect (i.e., variation imposed to the parameters w.r.t. previous value), sequence complexity and update frequency are also investigated (Fig. 29).

Considering one single structure, TW.parameters, Fig. 29a depicts the effect of increasing or decreasing the previous value of 0.25, 0.5, 1 or 2. It can be noticed that the average update time remains constant among conditions, confirming that the variation amount is not affecting the update time.
APPENDIX A. SUPPLEMENTARY DATA

Figure 28: Elapsed time for run-time parameters update: simultaneous tuning of multiple parameters. Update every 50 frames; 1 firing angle per frame. $N_{\text{updates}} = 200$.

The subsequent test tries to assess the effect of a longer, and hence more complex, event list on the update time. Fig. 29b presents the results in terms of update time obtained by changing the structure TW parameters when a frame is acquired using 1 single firing angle (shortest event list), 5 firing angles, etc. till 20 firing angles. As expected, the test demonstrates that larger event structures require more time to be updated than smaller ones (even though the parameter is always one). This is probably due to the fact the larger the number of firing angles, the longer it takes to program the waveform generation field-programmable gate array (FPGA) logic in the hardware system to synthesize the actual transmit output. This hypothesis is confirmed by the results attained for the same test when updating TGC. Indeed, in this case, the update time does not increase significantly when increasing the number of firing angles per frame.

The effect of the update frequency is analysed in Fig. 29c. Apart from the results observed when update frequency is set to 1 (i.e., the parameters are changed for every single frame), the other update frequency are all comparable. This demonstrates that this factor (provided that $>1$) is not affecting the update time.

2.3 Conclusions

- Update time varies depending on tuned parameters and event sequence complexity.
- Structures that have added VDAS attributes TW take more time to be updated than other structures (e.g. TPC).
- When using a simple sequence (1 acquisition per frame), the update time for updating TGC and TPC takes about 0.003 s.
- When using a simple sequence (1 acquisition per frame), the update time for apodization weights and transmit frequency takes about 0.2 s.

---

2 A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence the term field-programmable.
Figure 29: Elapsed time for run-time parameters update: affecting factors. Tuned parameter: TW.parameters. [a] Update TW.parameters every 50 frames; 1 firing angle per frame. The graph shows the effect in terms of update time with different values of constant decrement or constant increment applied to the tuned parameter. [b] Update TW.parameters every 50 frames; variable increment or decrement of the tuned parameter. The plot shows the effect in terms of update time with different firing angles per frame. [c] 1 firing angle per frame; variable increment or decrement of the tuned parameter. The box-chart shows the effect in terms of elapsed time with different update frequency (number of frames between two consecutive update).
Appendix B

Supplementary Info

1 Inference Deployment Environments

1.1 Inference Performance

Systems for Artificial Intelligence (AI) methods inference require fewer computing resources than the ones used for training. Even so, determining the optimal inference deployment configuration is not straightforward, and the choice is influenced by the model being run, and the performance required by the specific application. Typical performance factors are listed in Tab. B.1, together with application-specific constraints about data management and protection (e.g., Health Insurance Portability and Accountability Act).

<table>
<thead>
<tr>
<th>Performance Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>Latency refers to how much time elapses from an input being presented to the AI model to output being available. In some applications, low latency is a critical safety requirement.</td>
</tr>
<tr>
<td>Throughput</td>
<td>Throughput refers to how many inferences can be completed in a fixed unit of time. This also includes how many different models can be served simultaneously from the same servers.</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Level of accuracy ensures that AI model delivers the requisite results. It is key that inference preserves the level of accuracy achieved with training.</td>
</tr>
<tr>
<td>Continuity</td>
<td>Always-on systems need to operate also in disconnected environments or in power outage conditions.</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Flexibility refers to the possibility of easily customizing the compute resources to run desired systems (e.g., legacy systems).</td>
</tr>
<tr>
<td>Programmability</td>
<td>Programmability is intended as the capability to simply upgrade the software to support newer versions.</td>
</tr>
<tr>
<td>Scale-out capability</td>
<td>Scalability refers to the possibility to add compute resources easily when needed.</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Solution efficiency refers to the economies it can deliver around initial servers cost, their maintenance and upgrade, and the energy cost to power and cool these servers throughout their lifecycle.</td>
</tr>
</tbody>
</table>

1.2 Deployment Environments

Deployment environments can span from private or public data centres, to edge or embedded solutions. For each approach, the major pros and cons are summarized in Tab. B.2.

- **Public Cloud.** Deploying AI models in the public cloud implies access to compute resources through the Internet. In this framework, the devices at the user site have limited or no data processing stages and their primary function is to transfer raw data to a central
processing facility (cloud) where several servers are located. These public servers provide on-demand advanced computational service for data processing to an unlimited number of remote users connected through public Internet (i.e., other customers of the same cloud provider).

- **Private Data Center.** Similarly to the public cloud, private data centres refers to the use of remote servers for running the computationally-expensive processing methods. However, in this case, servers are located at the organization or in places where the organization has physical control. This allows higher visibility, data privacy, and security compared to public cloud.

- **Edge.** When inference is performed at the edge, AI methods are run in locations physically close to where data is collected or processed. As a consequence, the environment is often constrained in terms of space and power. Edge computing is mainly driven by the requirement for low-latency results.

- **Embedded.** Embedded inference refers to the deployment of AI methods on a processor integrated within the acquiring system. Like edge computing, this solution is optimal in real-time applications, but imposes several limitations on the processor dimension. Often times, a system-on-chip is considered for implementing AI applications on an embedded system.

<table>
<thead>
<tr>
<th>Deployment Environments</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>
| Public Cloud            | • Cost-effective;  
                          | • Easy scalability;  
                          | • Easy software upgrade;  
                          | • Cloud service providers' access control mechanisms | • No customization;  
                          | • High Latency;  
                          | • Network dependency;  
                          | • Risk of power outages;  
                          | • Risk of distributed denial-of-service attack;  
                          | • Legal implications for data location (e.g. data of EU citizens shall remain in the EU);  
                          | • New business model required (e.g. SaaS, Software as a service). |
| Data Center             | • Easy customization;  
                          | • Personalized access control mechanisms (centralized management). | • Expensive (initial investment, configure, manage, maintain, secure);  
                          | • Time-consuming software upgrade;  
                          | • Expensive and time-consuming scalability;  
                          | • Network dependency;  
                          | • Medium/High Latency;  
                          | • Risk of power outages. |
### APPENDIX B. SUPPLEMENTARY INFO

<table>
<thead>
<tr>
<th>Edge</th>
<th></th>
<th>Embedded</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Low latency;</td>
<td>• Usage environments and conditions constrained in terms of space and power;</td>
<td>• Usage environments and conditions constrained in terms of space and power;</td>
</tr>
<tr>
<td>• High-throughput;</td>
<td>• Difficult access control mechanisms;</td>
<td>• Difficult access control mechanisms;</td>
</tr>
<tr>
<td>• Always-on (also in disconnected);</td>
<td>• Deployment locations not physically secured;</td>
<td>• Time-consuming software upgrade;</td>
</tr>
<tr>
<td></td>
<td>• Time-consuming software upgrade;</td>
<td>• Expensive and time-consuming scalability;</td>
</tr>
<tr>
<td></td>
<td>• Expensive and time-consuming scalability;</td>
<td>• Limited scalability;</td>
</tr>
<tr>
<td></td>
<td>• Limited scalability.</td>
<td>• Limited size.</td>
</tr>
</tbody>
</table>
2 Network

2.1 Network Types

A network is defined as a group of units connected with each other through wires, optical fibres or optical links so that the various devices can interact with each other. The aim of a computer network is the sharing of resources among various devices. Networks types:

- Local Area Network (LAN)
- Personal Area Network (PAN)
- Metropolitan Area Network (MAN)
- Wide Area Network (WAN)

The most common communication paradigm in networked systems is the client-server model. Clients typically communicate with one server at a time, while servers typically provide services to multiple clients. The communication is enabled by a network protocol.

Client-server models can be categorized into two scenarios:

1. client and server on the same local network (LAN)
2. client and the server in different LANs, with both LANs connected to a WAN by means of routers. The largest WAN is the Public Internet, but enterprises may have their own WANs.

2.2 Reference Model

Network functionality is described by standardized reference models. These conceptual models use layered architectural models to summarize the distinct tasks involved in the links process and the different levels of communication. Each lower layer of the architectural model is intended to add its services to the higher layer to provide a full set of services to manage communications and run the applications. It provides modularity and clear interfaces, i.e., provides interaction between subsystems.

The Open Systems Interconnection (OSI) model is the first standard model proposed for network communications\(^1\), even if, nowadays, modern networks are typically based on the simpler TCP/IP model. The TCP/IP model consists of four layers: the application layer, transport layer, network (or Internet) layer, network access layer (Fig. 30). The network access layer defines how the data should be sent physically through the network. The Internet layer, also known as the network layer, ensures that packets are sent from any network and that they arrive at the destination irrespective of the route they take. The transport layer is responsible for the reliability, flow control, and correction of data that is being sent over the network. The application layer is the topmost layer in the TCP/IP model, and it provides the interface to the communication system; major responsibilities include presenting data (e.g. syntax of data), managing sessions, and handling high-level protocols.

These four layers work together to provide a communications system. The communication occurs when a protocol on one system, which is located at a specific layer of the model, communicates with its corresponding layer on another system (Fig. 31). This logical communication does not involve a physical connection between each pair of corresponding layers, but rather it is based on encapsulating the relevant information in a structured way such that a protocol at a specific layer on another device knows which information it has to interpret. As an outgoing

\(^1\) OSI was introduced in 1983 by representatives of the major computer and telecom companies and was adopted by ISO as an international standard in 1984.
transmission passes down through the source layered architecture, each layer adds relevant information, called a header, to the actual data. These headers are used by the devices along the communication line to gather the information needed to get the data to the destination. Each device along the way reads only the information necessary to it (e.g. switches operate at data link layer). This means that a protocol de-encapsulates the received packet data and reads the information sent by the corresponding layer on the source system.

### Network Access Layer Protocol

In a network, information is typically transmitted using data packets. Therefore, it is essential to determine the packet size to ensure packet transmission efficiency.

The Maximum Transmission Unit (MTU) limits the length of a data frame at the data link layer. The default MTU value varies according to the link medium type. For instance, Ethernet default MTU value is 1500 bytes.

### Network Layer Protocol

The Internet Protocol (IP) is the network layer communications protocol in the Internet protocol suite for transmitting datagrams across the network. The first major version of IP, Internet Protocol Version 4 (IPv4), is the dominant protocol of the Internet. Its successor is Internet Protocol Version 6 (IPv6), which has been in increasing deployment on the public Internet since c. 2006.

IP has the task of delivering packets from the source host to the destination host based on the IP addresses in the packet headers. In addition to the destination IP address, the IP header includes the source IP address and other metadata needed to route and deliver the datagram. Among this metadata, the header has a 16 bits long "Total Length" field to indicate the size of the packet in bytes. Note that the field size sets a theoretical limit of 65535 bytes for a datagram.

Notwithstanding this upper bound, when the packet to be sent to the data link layer exceeds the size that the layer below can transport (i.e., MTU), the protocol can either return error or fragment the message. In that regard, IPv4 can automatically fragment a datagram into smaller units for transmission when the link MTU is exceeded. Each fragment is targeted to the destination with its own IP header; it is then the task of the destination IP layer to collect all fragments and re-build the full packet out of them before passing the received data to the next higher layer. Note that an IPv6 network does not perform fragmentation in network elements, but requires end hosts and higher-layer protocols to avoid exceeding the path MTU.

---

<table>
<thead>
<tr>
<th>OSI Model</th>
<th>TCP/IP Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Layer</td>
<td>Application Layer</td>
</tr>
<tr>
<td>Presentation Layer</td>
<td>Transport Layer</td>
</tr>
<tr>
<td>Session Layer</td>
<td>Internet Layer</td>
</tr>
<tr>
<td>Transport Layer</td>
<td>Link Layer</td>
</tr>
<tr>
<td>Network Layer</td>
<td></td>
</tr>
<tr>
<td>Data Link Layer</td>
<td></td>
</tr>
<tr>
<td>Physical Layer</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 30: Computer network reference model.*
Transport Layer protocol  The Transmission Control Protocol (TCP or TCP/IP) and User Datagram Protocol (UDP or UDP/IP) are transport protocols layered on top of the IP. These interfaces are used for reading and writing both binary data and ASCII data to servers, computers, instruments.

TCP is described in RFC 793, RFC 1323, RFC 2581 and RFC 3390. It provides a connection-oriented service, since it is based on connections between clients and servers. The protocol is characterized for its reliability and management logic. When a TCP client sends data to the server, it requires an acknowledgment in return. If an acknowledgment is not received, TCP automatically re-transmits the data and waits for a longer period. This ensures that the complete message is sent to the destination. With respect to management logic, TCP uses "Window Size" (WS) field in the TCP header to indicate the space available in the Receive Buffer. This drives the speed of data transferring (i.e., throughput $T = \frac{WS}{RTT}$, with RTT: Round Trip Time) and enable flow control. Note that, when setting the protocol, TCP attempts to adapt the Receive Buffer size (i.e., initial WS) to match the capacity of the network. In other terms, in order to obtain a throughput comparable to the bandwidth (B), the Receive Buffer size should be set equal to $B \times RTT$. In practice, this upper bound limit is not always reached as many standard TCP allows a maximum window size (MWS) - typically 65535 bytes - which further constrains the throughput: $T = \frac{\min(B \times RTT, MWS)}{RTT} \leq B$. For instance, for 'long fat network', where the bandwidth-delay product (BDP) is greater than 65535 bytes\(^2\), the maximum window size prevents a complete use of the theoretical maximum speed ($MWS < B \times RTT$). To overcome this bottleneck, a window scale option is introduced to define a multiplying factor (range 1-14) which increases the MWS value up to c. 1 GB ($65535 \times 2^{14}$). In addition to flow control, a congestion window is used to apply congestion control and determines the number of bytes that can be sent out at any time. It is maintained by the sender and adjusted over time according to the output of the congestion algorithm (which considers for instance a loss event as a sign of network congestion). Slow start, defined by RFC 5681, is part of the congestion control strategy used by TCP in conjunction with other algorithms to avoid sending more data than the network is capable of forwarding, that is, to avoid causing network congestion. In that regard, TCP is an example of a protocol that adjusts its segment size to be smaller than the MTU.

UDP is described in RFC 768. It provides a connection-less service, as no connection is established between the client and the server. It has no management logic of its own, which makes it very flexible, fast, and simple. The pitfall of this light protocol is that there is no guarantee of that a UDP datagram will reach the destination (i.e. packets loss), that the order of the datagrams will be preserved across the network, or that datagrams arrive only once. In this respect, UDP disregards MTU size, thereby forcing IP to fragment oversized datagrams.

2.3 Socket

Sockets are commonly used for client and server interaction. A socket is a software structure that acts as a communication connection point (endpoint) for sending and receiving data among processes on a local system, or between a local system and a remote system. The structure and properties are defined by an Application Programming Interface (API) for the networking architecture.

Socket programming implies the definition of the network and transport layer protocol to use. Once implemented, the application writes a message to the socket (e.g., TCP socket). The message is then encapsulated in a datagram and passed to the network layer that further encapsulates it into an IP datagram which is sent to the destination through the link layer.

\(^2\)Bandwidth-delay product is defined as the maximum amount of data on the network circuit at any given time (i.e, data transmitted but not yet acknowledged). It is computed by multiplying data links capacity [bits per second] and round-trip-time [s].
3 Equipment

3.1 Verasonics Vantage Research System

The Verasonics Vantage Research System consists of the Verasonics Data Acquisition Hardware connected to a Host Controller Computer (Fig. 32).

The Acquisition Hardware contains electronic modules for multi-channel transmit waveform generation, analog receive signal amplification and filtering, digital signal processing and scan sequencing. The computer, instead, contains a Hardware Abstraction Layer, a Matlab programming environment, and a Matlab External function written in the C programming language, named runAcq (runAcq is defined as a MEX File\(^3\)).

\(^3\)A MEX File is a type of computer file that provides an interface between Matlab or Octave and functions written in C, C++ or Fortran.
The sequence of events is programmed by writing a Matlab script, referred to as Setup script that generates a collection of objects bundled into a .mat file format. This .mat file can be loaded into the system by a loader Matlab program, named VSX. Among other tasks, VSX checks for missing required parameters and programs some default attributes before loading the Sequence Object in the Matlab environment. Additionally, if acquisition hardware is detected, the VSX application is responsible for opening and owning the hardware during the acquisition. The core of the application is an execution loop that drives the sequence of events that bring to data acquisition. In this execution loop, the MEX function runAcq is called to perform all the processing for the sequence events. The function interacts directly with the hardware during run time and handles all processing and display functions. runAcq starts at a specified event in the sequence (Resource.Parameters.startEvent) and continues to execute events until the end of the sequence, or until a specific condition is met (e.g., returnToMatlab). When these conditions verify, runAcq returns the control to the Matlab environment. Afterward, typically, VSX calls again runAcq to repeatedly execute the acquisition events.

When the control is returned to the Matlab environment, the user interface functions or user programs can be executed. If run-time parameters have been modified by the user through a graphical user interface (GUI) object, the sequence structures are changed, but these modifications will not have any effect until reloaded into both the runAcq processing function and the hardware. Before reloading, the GUI control sets up a Control structure in the Matlab environment, which will be passed to runAcq on the next call from VSX, specifying the structures or attributes to be updated. When there are no actions to be performed in Matlab, the runAcq processing function is called again, and the sequence repeats from the start or continues from where it returned to Matlab.

A common processing task is displaying a reconstructed image. While the output of the reconstruction processing can be pixel intensity data, which in itself defines an image that could be viewed on the display, there is typically additional processing that needs to be performed before rendering the image data to the display. The reconstruction process returns the frames in an image buffer, which is subsequently accessed by the MEX function runAcq. The rendering process consists of processing the source data in a specified image buffer (call the vsDisplay function in the Utilities directory), according to the attributes defined in the Image structure through the SetUp Script. The resulting pixel data is written to a DisplayData array to be copied to the CData of the DisplayWindow.

During the execution of the sequence events, it is also possible to have the system call a user provided Matlab function (external function) at specific points in the sequence. This is useful when the users want to provide their processing routines, as the function can access acquired and possibly processed data to perform some unique function/processing, create their data displays, or return the processed data to the processing software. To this end, an event should be defined in the Event Sequence that identifies the source (e.g., buffer name, frame number, section number) and the destination where to send the processed data (e.g., buffer name, frame number, section number).

A representative sequence diagram for Verasonics asynchronous operation is reported in Fig. 33a. As mentioned, acquisition proceeds at its own rate to acquire and store data in the Receive Buffer (for further offline post-processing), while the online processing frame rate depends on the computational power of the Host Computer and the kind of processing. Importantly, note that a new frame is processed only when the previous processing is completed. For completeness, Fig. 33b shows the system behaviour when manual run-time parameters tuning is performed by the user through the GUI.
Figure 33: **Verasonics sequence diagram.** Glow lines indicate control, which alternates between the software sequencer runAcq.c and VSX.m. Arrows represent data transfer between modules, while the dashed arrow stands for user input through the Graphical User Interface. White dots depict latency due to data transfer from one module to another. White triangles represent the checking for run-time parameters update executed in Matlab when runAcq.c calls VSX.m. White square indicates VsUpdate.m called to re-compute attributes on tuned parameters. [a] Timing diagram including initialization. [b] Timing diagram including run-time parameters tuning.
4 Theoretical Derivation

4.1 Quadrature Sampling

For a band-limited signal, the Nyquist sampling\textsuperscript{4} theorem states that a sampling frequency above twice the frequency of the highest frequency component of the signal is required to guarantee a perfect signal reconstruction. However, it can be demonstrated that Nyquist sampling frequency is typically unnecessary to avoid aliasing and that a smarter sampling scheme can be used instead.

Under quadrature sampling (Fig. 34), the signal can be sampled at the Nyquist cut-off of the complex envelope (i.e. at the signal bandwidth) rather than at twice the frequency of the highest frequency component of the signal, yielding an appreciable reduction in sampling rate for narrowband signals.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure34}
\caption{Quadrature sampling. (a) Conceptual diagram. (b) Implementation diagram.}
\end{figure}

Complex demodulation (or complex baseband modulation technique) is the standard approach used to obtain quadrature samples and consists of 3 main steps (Fig. 35):

- **Down-mixing**
- **Low-pass filtering**
- **Decimation**

Consider a continuous input bandpass signal of bandwidth $B$, with the carrier frequency equals to $f_c$ Hz (i.e., the bandpass signal is centred at $f_c$ Hz). Complex down-conversion (i.e., frequency spectrum translation) is performed by multiplying (i.e., mixing) the time signal with a complex sinusoid signal of frequency $f_{demod}$ Hz (Fig. 34a). After the down-mixing, the resulting signal is complex, and the frequency spectrum is no longer symmetric around zero. Note that, given the relationship between complex exponential functions and sine and cosine functions, the complex sinusoid signal can be thought of as mixing with two sinusoid signals with 90° phase difference.

After down-mixing, the complex signal is low-pass filtered to remove the negative frequency spectrum and noise outside the desired bandwidth. This is implemented by applying the filter to the real and imaginary parts separately (Fig. 34b). Removing the frequencies stemming from the negative spectrum of the original real signal, the filter halves the energy in the signal, and thereby the complex signal should be multiplied by the square root of 2 to preserve the total energy.

\textsuperscript{4}Sampling is a process of converting a signal (for example, a function of continuous time or space) into a sequence of values (a function of discrete time or space).
Figure 35: In-phase quadrature demodulation. (a) Band pass signal from an ultrasound transducer. (b) Down-mixing. (c) Low pass filtering. (d) Decimation.

The filter cut-off frequency is lower than the highest frequency component of the original signal. This implies that the complex signal can be sampled at a sampling frequency which is twice this cut-off frequency, rather than twice the original highest frequency component. As the complex signal does not have an ambiguity between positive and negative frequencies, the bandwidth of the signal equals the complex sampling rate. In practice, the smallest integer fraction of the original sampling rate which is larger than twice the filter cut-off frequency is selected as sampling rate for the complex envelope. This value is used to compute the decimation factor. For instance, if sampling rate is reduced by a factor 6 (i.e., 6 is the ratio between original sampling frequency and complex envelope frequency), quadrature sampling is obtained by keeping every 6th sample and throwing away the rest. Note that typically a lower decimation factor is observed. This value follows the fact that the data in this example was originally sampled at 4 times the centre frequency $f_c$ ($4 \times 2.5 \text{MHz} = 10 \text{MHz}$).

The reconstruction of the original real signal from In-phase Quadrature (IQ) data is a reversal of the complex demodulation process. The decimation step is reversed by interpolation; the low-pass filter cannot be reversed, but should be chosen without loss of information in the first place; the down-mixing is reversed by up mixing (with a complex sinusoid signal of frequency $f_{\text{demod}}$ Hz). At last, the real signal is found by taking the real-value of the complex up-mixed signal.
4.1.1 Second-Order Method Approximating Quadrature Sampling

When highly narrowband signals are used, a second-order approximation can be used to perform quadrature sampling. The basic principle of this second-order technique is to use a demodulation frequency that allows to perform the down-mixing process by merely changing the sign of alternating samples [42]. This method can be much simpler to implement, as the analog multipliers and filters typically required to obtain IQ data are not needed.

Consider a real bandpass signal sampled at frequency \( f_s = 4f_c \), down-mixed with a complex signal of demodulation frequency \( f_{dem} = f_c = \frac{1}{4}f_s \).

In-phase mixing component is obtained multiplying by:
\[
\cos[2\pi f_{dem}n] = \cos[2\pi \frac{f_c}{4}n \frac{1}{4}] = \cos[\pi \frac{n}{2}] = 1, 0, -1, 0, 1, 0, \ldots
\]

Quadrature mixing component is obtained multiplying by:
\[
\sin[2\pi f_{dem}n] = \sin[2\pi \frac{f_c}{4}n \frac{1}{4}] = \sin[\pi \frac{n}{2}] = 0, 1, 0, -1, 0, 1, 0, \ldots
\]

It follows that samples can be multiplied by \( \pm 1 \) or 0 to get the real (in-phase) and imaginary (quadrature) components of the IQ data. Since the following step in quadrature sampling is decimation\(^5\), only few cosine and sine samples are effectively used. The number of samples retained and skipped is computed considering that the complex demodulation reduces the minimum sampling frequency to the signal bandwidth (i.e., Nyquist cut-off of the complex envelope).

Percentage bandwidth is defined as:
\[
BW\% = \frac{B}{f_c} \times 100
\]

Minimum sampling frequency of IQ data after demodulation:
\[
f_{sIQ} = 2f_c l \geq B
\]

The undersampling factor \((l)\):
\[
l_{\text{max}} = \frac{2f_c}{f_c} = 2
\]

As a consequence:

- if \( BW\% = 200\% \rightarrow \frac{B}{f_c} = 2 \rightarrow l_{\text{max}} = 1 \rightarrow f_{sIQ} = f_c
- if \( BW\% = 100\% \rightarrow \frac{B}{f_c} = 1 \rightarrow l_{\text{max}} = 2 \rightarrow f_{sIQ} = f_c
- if \( BW\% = 50\% \rightarrow \frac{B}{f_c} = \frac{1}{2} \rightarrow l_{\text{max}} = 4 \rightarrow f_{sIQ} = f_c

A conceptual diagram for the second-order approximating technique is shown in Fig. 36.

![Conceptual Diagram of Second-Order Approximating Quadrature Sampling](image)

**Figure 36: Second-order method approximating quadrature sampling.**

Real signal \( x(t) \) is sampled with a frequency \( f_{sIQ} = 2f_c l \). In particular, in-phase samples are sampled at period \( \Delta = \frac{1}{2f_c} \), with \( l \) the undersampling factor (i.e., an integer that must be selected such that \( \Delta \) remains less than \( \frac{1}{B} \) to avoid aliasing). Quadrature samples are taken after the signal is delayed by \( \alpha \), where \( \alpha = \frac{f_c}{4} + K\Delta \) with \( K \) as integer. Typically, \( K \) is set to zero, which means that the quadrature samples are taken after a quarter-period of the carrier frequency relative to the in-phase samples. In other terms, the first sample of a signal sampled

---

\(^5\)Low pass filter is assumed to be applied to the real original signal before sampling.
at $f_s = 4f_c$ is multiplied by the in-phase mixing component and returns the in-phase sample, while the following one is multiplied by the quadrature mixing component and returns the quadrature sample.

The results of this sampling process are multiplied for $(-1)^n$, this factor accounts for alternate sign of cosine and sine when sampling rate is $2f_c$.

If $BW_{\%} = 200\% \to 2$ samples over $T_c$ for each component
→ multiplies for 1, -1, 1, ... to get in-phase mixing component
→ multiplies for 1, -1, 1, ... to get quadrature mixing component with a delay of $\alpha$
indeed for $l = 1$: $(-1)^n = 1, -1, 1, ...$

If $BW_{\%} = 100\% \to 1$ sample over $T_c$ for each component
→ multiplies by 1, 1, 1, ... to get in-phase mixing component
→ multiplies by 1, 1, 1, ... to get quadrature mixing component with a delay of $\alpha$
indeed for $l = 2$: $(-1)^n = 1, 1, 1, ...$

If $BW_{\%} = 50\% \to 1$ sample over $2 \times T_c$ for each component
→ multiplies by 1, 1, 1, ... to get in-phase mixing component
→ multiplies by 1, 1, 1, ... to get quadrature mixing component with a delay of $\alpha$
indeed for $l = 4$: $(-1)^n = 1, 1, 1, ...$

At last, the quadrature component is multiplied by $-1$ to compensate for spectral inversion caused by the down-conversion by $e^{j\omega t}$ and low pass filtering. Indeed, multiplying the real input with the complex $e^{j\omega t}$ (cosine and sine with positive sign) shifts the spectrum to the right, as opposed to a condition where the sign of the quadrature component is inverted ($e^{-j\omega t}$) which causes a shift of the spectrum to the left and avoids spectral inversion.

The approximation implied in the second-order technique is that the in-phase and quadrature samples are simultaneous, which is not realized as the components are by construction mutually separated by the interval $\alpha$.

$IQ[n] = i[n] + jq[n + \alpha]$, with $\alpha = \frac{1}{4f_c}$
As shown below, this approximation is valid only with narrowband signals:
$q[n + \alpha] = q[n(1 + \frac{\alpha}{B})] = q[n(1 + \frac{1}{2B})] \to$ with $l = l_{max} = \frac{2f_c}{B}$, $q[n(1 + \frac{B}{4f_c})]$
which approximates $q[n]$ when $B$ is small.
4.2 Digital Proportional-Integral-Derivative Controller

The PID controller equation is typically presented in the continuous Laplace domain. To implement the PID controller in software, the continuous Laplace domain equation must be transformed into the difference equation. Once transformed into a difference equation, this equation can be implemented to develop a digital PID controller to control any closed-loop system.

Continuous PID controller equation:

\[ u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \]  \hspace{1cm} (B.1)

with \( u(t) \): updated value for the tunable parameter; \( K_p, K_i, K_d \): proportional, integrative and derivative constants; and \( e(t) \): difference between actual value at time \( t \) and reference value.

Discretize:

\[ u[n] = K_p e[n] + K_i i[n] + K_d d[n] \]  \hspace{1cm} (B.2)

where, using trapezoidal rule approximation:

\[ i[n] = i[n-1] + T \frac{e[n] + e[n-1]}{2} \]  \hspace{1cm} (B.3)

and, using backward Euler approximation:

\[ d[n] = \frac{e[n] - e[n-1]}{T} \]  \hspace{1cm} (B.4)

Compute \( u[n] - u[n-1] \):

\[ u[n] - u[n-1] = K_p e[n] + K_i i[n-1] + T \frac{e[n] + e[n-1]}{2} + K_d \frac{e[n] - e[n-1]}{T} + \]  \[ -[K_p e[n-1] + K_i i[n-1] + K_d \frac{e[n-1] - e[n-2]}{T}] \]  \hspace{1cm} (B.5)

\[ u[n] = u[n-1] + \left( K_p + \frac{K_i T}{2} + \frac{K_d}{T} \right) e[n] + \left( \frac{K_i T}{2} - K_p - \frac{2K_d}{T} \right) e[n-1] + \frac{K_d}{T} e[n-2] \]  \hspace{1cm} (B.6)

Set \( \kappa_P = K_p; \kappa_I = \frac{K_i T}{2}; \kappa_D = \frac{K_d}{T} \)

\[ u[n] = u[n-1] + (\kappa_P + \kappa_I + \kappa_D) e[n] + (\kappa_I - \kappa_P - 2\kappa_D) e[n-1] + \kappa_D e[n-2] \]  \hspace{1cm} (B.7)
Appendix C

Catheter Ablation Procedure

1 Background

1.1 Cardiac Physiology

The heart is a muscular organ primarily responsible for pumping blood and distributing oxygen and nutrients throughout the body. In mammals and birds, the heart is a four-chambered organ, divided into a right atrium and ventricle (commonly referred to as the right heart) separated from a left ventricle and atrium (the left heart). The left heart is pumping oxygenated blood to the body, while the right heart is pumping the deoxygenated blood to the lungs [43]. The fulfillment of this blood-pumping function is provided by a coordinated contraction of the heart muscular tissue, the myocardium [43]. Interestingly, the mechanical response of the cardiac muscle follows an electrical activation of its cells, according to the well-studied excitation-contraction coupling mechanism [44]. Basically, when an electrical stimulus (e.g. a positive current) arrives in proximity of a cardiac cell, a cascade of events is triggered. If the amplitude of the stimulus overcomes a certain threshold, the cascade of events results into cellular contraction.

In a healthy heart, this electrical stimulus is generated spontaneously and periodically at a small area of electrically active cells situated in the right atrium. From here, it is routed through the anatomic conduction system of the heart to the rest of the myocardium tissue, where it can activate the cardiac cells and induce their contraction. The sequential activation and contraction of adjacent cardiac cells enables the heart to behave as a functional syncytium.

1.1.1 Cardiac Heterogeneity

The maintenance of this coordinated contraction is strictly related to the peculiar cytoarchitecture of the myocardium, and requires a precise myocardial heterogeneity at the cellular level [45]. The most important distinction is between the cells specialized in the conduction of the electrical signal, the cardiac conduction system, and the cells involved in the myocardium contraction, the working myocardium. The main differences between these two cell types can be observed in their electrical properties and in their arrangement. Noteworthy, any alteration of this organized structure can substantially compromise the heart functionality, leading to life-threatening complications.

Cardiac conduction system. The proper conduction of the electrical signal and the consequent synchronized contraction is controlled and triggered by the cardiac conduction system. The main components of this conductive system are the sinoatrial (SA) node, the atrioventricular (AV) node, the bundle of His, the bundle branches, and the Purkinje fibers. The triggering stimulus originates in the SA, a cluster of cells in the right atrium that show autorhythmicity properties, i.e., they spontaneously depolarize and initiate an electrical impulse in the absence of an external electrical stimulation [43]. From this localized region, the depolarization wave spread
through the atria, causing atrial contraction. After atria depolarization, the impulse stimulates
specialized conduction tissues in the AV, located in the interatrial septum. This AV junction
slows down the impulse propagation and allows completing the atrial contraction before pro-
ceeding towards the ventricles. The signal is then conducted through the His-Purkinje system,
which rapidly transmit the activation wave to the right and left ventricular myocardium. The
main left bundle bifurcates into two primary subdivisions, a left anterior fascicle and a left pos-
terior fascicle. The depolarization wavefronts then spread through the ventricular wall, from
endocardium to epicardium, triggering ventricular contraction and enabling blood-pumping.

**Working myocardium.** As opposed to the cardiac conduction system which enables fast con-
duction of the impulse, other cardiac cells are mainly responsible for mechanical contraction
and are therefore called *working myocardium*. Once the electrical signal reaches these regions,
cardiac cells are excited and the contraction process is activated. The coordinated contraction
of the myocardium is the result of a rapid propagation of the electrical signals through the
working myocardium and the strong cohesion between adjacent cells that allows a proper force
transmission. These properties are both possible by the electro-mechanical coupling between
adjacent cardiac cells and the fiber-like arrangement of myocardium.

### 1.2 Cardiac Arrhythmias

Diseases or injuries can alter the physiological electrical activation of the heart, leading to ab-
normal heart rhythms, named *arrhythmias*. Abnormalities of cardiac rhythm are prevalent and
affect more than 2% of middle-aged and older adults, with a rate of about 0.5% per year [46].
Risk factors for arrhythmias include older age, male sex, traditional cardiac risk factors, chronic
kidney disease and heart failure.

#### 1.2.1 Classification

Cardiac arrhythmias refer to any abnormality or disturbance in the normal activation sequence
of the myocardium and can be classified by rate (tachycardia, bradycardia), mechanism (auto-
maticity, triggered activity, reentry), duration (isolated premature beats, couplets, runs), or site
of origin (supraventricular, ventricular).

According to the site of origin classification, cardiac arrhythmias can be classified as follows:

**Supraventricular Arrhythmias.** Supraventricular arrhythmias is a broad term that includes
many forms of arrhythmia originating in the atria or at the AV node.

Supraventricular bradycardia, or simply bradycardia, arises when the heart rate is slower
than normal. The most dangerous consequence is that when the heart rate is too slow, an insuf-
ficient amount of blood reaches the brain. Bradycardia can be broadly classified into 2 general
categories: (1) sinus node dysfunction and (2) atrioventricular block [47].

Supraventricular tachycardia (SVT), instead, occurs when faulty electrical connections in the
heart or abnormal areas of electrical activity trigger and sustain an abnormal rhythm. When a
tachycardia arises, the heart rate accelerates too quickly and prevent the heart to fill completely
before contracting. As a consequence, the stroke volume might be insufficient to properly oxy-
genate body organs. There are four main types: atrial fibrillation (AF), paroxysmal supraven-
tricular tachycardia (PSVT), atrial flutter (AFL), and Wolff–Parkinson–White syndrome [48].

**Ventricular Arrhythmias.** Ventricular arrhythmias include a broad spectrum that ranges from
premature ventricular complex to ventricular fibrillation (VF), with a different clinical presenta-
tion that ranges from a total lack of symptoms to cardiac arrest. Most life-threatening ventricular
arrhythmias are associated with ischemic heart disease, particularly in older patients. The risks
of ventricular arrhythmias and sudden cardiac death (SCD) vary in specific populations with
different underlying cardiac conditions, and with specific family history and genetic variants, and this variation has important implications for studying and applying therapies [49].

1.2.2 Electrophysiological mechanisms

The mechanisms responsible for cardiac arrhythmias may be divided into (1) disorders of impulse formation, (2) disorders of impulse conduction, or (3) a combination of both. Among disorders of impulse formation, arrhythmias can originate from abnormal automaticity of the SA node or of regions which are not ordinarily responsible for impulse formation. Additionally, impulse can be pathologically initiated by abnormal depolarizations of cardiac myocytes, named early or delayed after-depolarizations, or generally speaking triggered activity. On the other hand, disorders of impulse conduction usually result in reentrant excitation, or reentry [43, 50].

<table>
<thead>
<tr>
<th>Disorders of Impulse Formation</th>
<th>Disorders of Impulse Conduction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Automaticity</strong></td>
<td><strong>Reentry</strong></td>
</tr>
<tr>
<td>Altered normal automaticity</td>
<td>Anatomic reentry</td>
</tr>
<tr>
<td>Abnormal automaticity</td>
<td>Functional reentry</td>
</tr>
<tr>
<td><strong>Triggered activity</strong></td>
<td></td>
</tr>
<tr>
<td>Delayed afterdepolarization</td>
<td></td>
</tr>
<tr>
<td>Early afterdepolarization</td>
<td></td>
</tr>
</tbody>
</table>

Table C.1: Mechanism of cardiac arrhythmias

**Abnormal automaticity.** Arrhythmic mechanism linked to automaticity can be classified as altered automaticity of the SA node or abnormal automaticity [50]. The former indicates the enhanced or suppressed automaticity of SA pacemaker cells, which causes an increase (sinus tachycardia) or a decrease in the heart rate (sinus bradycardia, or sinus arrest when no impulse is generated). Abnormal automaticity, instead, occurs when atrial or ventricular non-pacemaker myocardial cells, which in the normal heart typically do not exhibit spontaneous activity, may exhibit automaticity properties. This can happen under conditions that drive the maximum diastolic potential towards the threshold potential. When the depolarization of these abnormal cells is able to initiate an activation wavefront, cardiac arrhythmias can arise.

**Triggered Activity** Cardiac arrhythmias can also be generated by early or delayed afterdepolarizations1 of cardiac cells. When the amplitude of these abnormal afterdepolarization suffices to bring the membrane to its threshold potential, a spontaneous action potential is generated. These triggered events give rise to extrasystoles, which can precipitate tachyarrhythmias [50].

**Reentry** During normal electrical activity, the impulse elicited in the SA node can propagate only in one direction, as cardiac cells become temporary refractory after excitation. In this way, when all cardiac fibers have been depolarized the activation wave dies out, and only when cardiac cells have restored their excitability - which usually occurs in less than 250 ms - a new cardiac cycle begins. However, if a group of isolated fibers is not activated during the initial wave of depolarization or if a region can restore its excitability in time, an excitable region might serve to the depolarizing wave to re-enter the circuit and re-excite previously depolarized fibers [43].

---

1Depolarizations that attend or follow the cardiac action potential and depend on preceding transmembrane activity for their manifestation are referred to as afterdepolarizations. Early afterdepolarization interrupts or retards repolarization, ultimately increasing action potential duration, whereas delayed afterdepolarization occurs after full repolarization.
Such a process is commonly denoted as \textit{reentry}, and it represents the most common type of supraventricular and ventricular arrhythmias that requires treatment [51]. These arrhythmias can be classified as anatomic or functional, depending on whether the core of the reentry circuit is a non-exciteable anatomic obstacle or an electrically silent region due to functional causes. An example of anatomical reentry caused by the presence of isolated surviving fibers within scars is shown in Fig. 37.

1.2.3 Current Treatments

Usually, only those cardiac arrhythmias that cause significant symptoms or present high risk of complications are treated. In these circumstances, the most common cardiac rhythm management strategies are antiarrhythmic drugs therapy, catheter ablation, implantable electronic devices and medical procedures or surgery. The choice of the treatment is defined according to the specific mechanism generating the arrhythmia and the patient’s health condition.

A brief description of the current treatments is given below.

\textbf{Antiarrhythmic Drugs.} Antiarrhythmic drugs are typically used to slow a fast heart rate (e.g. beta blockers or calcium channel blockers), while no drug are currently available to reliably speed up a slow heart rate. Traditionally used as the first treatment options for abnormal heart rhythm, antiarrhythmic drugs have been recently relegated to an ancillary role in the treatment of most cardiac arrhythmias. This is mainly due to the complex interaction of antiarrhythmic drugs with cardiac tissues and the resulting - sometimes adverse - electrophysiologic changes [43].

\textbf{Catheter Ablation.} Catheter ablation therapy aims at the destruction of a critical anatomic region of impulse generation or propagation required for the initiation and maintenance of the arrhythmia. By scarring this region - and thus making it silent to the electrical signals - the arrhythmia can be eliminated and the normal beat restored. Importantly, this procedure is a treatment option only for those kinds of arrhythmias that are caused by a certain tissue region.

\textbf{Cardiovascular Implanted Electronic Device.} Some arrhythmias, that cannot be successfully treated by medications and that do not respond well to cardiac ablation, require the implantation of cardiac pacemakers or implantable cardioverter defibrillators [52]. The former uses short electrical pulses to prompt the heart to beat at a normal rate. It can speed up a slow heart rhythm, control a fast heart rhythm, and coordinate the chambers of the heart. The latter, instead, is of extreme relevance in controlling life-threatening arrhythmias (i.e., patients at high risk of sudden cardiac death), as it can recognize dangerous rhythms and delivers electrical shocks to restore the heartbeat.
Surgery or medical procedures. Some arrhythmias can be treated with surgery, especially if the operation is already being done for another reason. An example of surgery that was traditionally performed to treat atrial fibrillation (AF) was maze surgery. During this procedure, small cuts or burns were created in the atria, such that multiple scars became available at atria level to prevent the spread of disorganized electrical signals. Despite the traditional use of this approach, with the advent of minimally invasive surgery this kind of procedures have been drastically reduced. Other simpler medical procedures involves the use of particular maneuvers, like holding the breath and straining, dunking the face in ice water, or coughing. These maneuvers are generally called vagal maneuvers, as they affect the vagus nerve, which in turn helps in controlling the heart rate.

1.3 Catheter Ablation Therapy

A previously mentioned, catheter ablation aims at destructing a certain area of the heart which is responsible for initiating or maintaining a certain cardiac arrhythmia. The site affected either by heating or freezing is destroyed, and the viable myocardium is substituted with scar tissue. As scars are electrically silent (i.e., cannot conduct electricity), ablation knocks out a critical spot or acts as a fence around an arrhythmogenic area.

Energy Source. The first catheter ablation using a direct current (DC) energy source was performed in the early 1980s by Scheinman and colleagues. However, the high incidence of complications associated with the use of DC ablation prevented its widespread use. Almost ten years later, radiofrequency (RF) energy, already used in other biomedical applications, had been adapted for use in catheter-based ablation in the heart [43]. The ability of RF energy to produce a controlled focal tissue ablation, as compared with the more extensive damage caused by DC fulguration, facilitated its adoption in the medical practice. Today, RF energy is the gold standard technology for catheter ablation. Energy is generally delivered from the tip of a catheter to the endocardial surface, even if in selected cases it may be applied to the epicardial surface via a pericardial approach. The principle at the basis of this technology is that energy in the RF frequency band is able to heat the tissue by a resistive mechanism. When a proper amount of energy is delivered, the temperature raises enough to cause tissue necrosis, ultimately destroying the target region and restoring the normal heart beat. Although RF energy is vastly superior to DC energy, efforts continue to identify alternative sources for catheter ablation. Cryothermal ablation, for instance, uses extreme cold to destroy tissue, and it is showing distinct clinical advantages over RF energy [53, 54]. An illustrative representation of RF and cryothermal ablation is shown in Fig. 38.

![Figure 38: Catheter ablation therapy. Schematic drawing showing catheter ablation of atrial fibrillation using either (a) radiofrequency energy or (b) cryoballoon ablation. (a) Ablation lesions are delivered in a figure of eight pattern around the left and right pulmonary veins. A multielectrode circular mapping catheter is positioned in the left inferior pulmonary vein. (b) Ablation lesions have been created surrounding the right pulmonary veins, and the cryoballoon ablation catheter is positioned in the left superior pulmonary vein. Illustration: Tim Phelps - 2017 Johns Hopkins University, AAM. Reproduced from [55]](image-url)
**Procedure.** Catheter ablation usually takes 2 to 4 hours, and it is carried out in a special hospital room called an electrophysiology (EP) lab or a cardiac catheterization (cath) lab. During this procedure, the tip of a catheter is guided to the area of heart tissue that is producing abnormal electrical signals. Vascular access is typically obtained through the femoral vein, though the right internal jugular vein is sometimes used as an alternative. In both circumstances, the use of fluoroscopic and/or US guidance is strongly suggested to achieve a safe femoral puncture. Once the access has been created, a guidewire is advanced. The needle is removed, and a sheath introducer is inserted and then flushed. At this point, the guidewire can be advanced under fluoroscopic guidance from the femoral vein to the superior vena cava. Transesophageal echocardiography (TEE) may be used to confirm that the guidewire has reached the position. Depending on the region that needs to be ablated, a trans-septal puncture might be necessary to enable catheter access to the left heart. If this is the case, a needle is advanced gently under fluoroscopic guidance. When the needle is *in situ*, its tip is moved until it falls into the fossa ovalis. When needle and sheath fall into the fossa, the needle is advanced and the sheath is kept in place. After puncture, both the transeptal sheath and the needle are advanced in the left atrium. From here, the needle can be removed and the sheath is positioned at the region of interest. Once the sheath is properly located, the catheter can easily reach the target site. At this point, the area of interest can be ablated either using a mild, painless, radiofrequency energy or by freezing it.

1.3.1 Multimodality Imaging for Guiding Ablation Procedures

Cardiac imaging is a prerequisite before, during and after ablation procedures. Focusing on pre-procedure and intra-procedure cardiac tissue characterization, a description of the diagnostic modalities currently adopted is presented below.

**Pre-procedure Evaluation** The pre-procedure characterization of the cardiac tissue focuses on the identification of the sites at which radiofrequency or cryothermal ablation will be successful at curing the arrhythmia. To this end, electrophysiological and anatomical information are usually integrated and used to plan the treatment. A better pre-procedural assessment allows for a more comprehensive understanding of the mechanisms involved and has the potential to reduce fluoroscopy time during procedures and to increase safety.

- **Cardiac Mapping.**
  
  Cardiac mapping refers to the movement of a mapping catheter in the area of interest, probing for the site of origin or a critical site of conduction for an arrhythmia. The process identifies the temporal and spatial distributions of electrical potentials generated by the myocardium during normal and abnormal rhythms, and allows description of the spread of the activation wavefront within a region of interest. Several distinct mapping systems have been developed over the last decades. The simplest form of mapping is achieved by moving a single electrode sequentially to various points of interest on the endocardium or mapping simultaneously from as many sites. This *endocardial mapping* approach allows to measure local activation, but it is inappropriate for the detection of polymorphic or monomorphic ventricular tachycardia (VT). Additionally, when the critical portion of the arrhythmia circuit is located epicardially the site of origin of the tachycardia cannot be reached from the endocardial mapping catheter and requires *epicardial mapping* techniques. A slightly different strategy is *noncontact endocardial mapping*, which includes the use of multielectrode array mounted on a balloon tipped catheter. This technique has been deployed in all four cardiac chambers using a transvenous, transeptal, or retrograde transaortic approach. Despite the high number of available techniques, *electroanatomical mapping* remains the most common approach in many EP laboratories. This mapping system is based upon the use of a special catheter with a locatable sensor tip, connected to a mapping and navigation system. The system can generate isochrones of electrical
activity as color-coded static maps or animated dynamic maps of activation wavefront. The widespread use of this technique might be related to the ability of these systems to be integrated with other imaging modalities. The two most common 3D electroanatomical mapping systems are the Carto system (Biosense Webster, Diamond Bar, California), which is based on a magnetic sensor, and the NavX Ensite (St. Jude Medical, St. Paul, Minnesota), an impedance-based 3D mapping system [56]. Of note, a cardiac ablation treatment is not always preceded by a detailed mapping of the heart electrical activity. **Anatomic approaches** to ablation are used when the arrhythmia has a known anatomic course. For instance, the isolation of the pulmonary vein ostia from the body of the left atrium is typically performed as treatment in AF, and it is entirely anatomically based without any pre-procedural cardiac mapping. In this circumstances, however, usually a second procedure is required as anatomic approaches might fail to ablate completely the arrhythmogenic substrate.

- **Anatomical Information.**
A part from cardiac mapping, precise anatomical information are also required when performing catheter ablation. This is due to high inter-individual variability which might affect the final outcomes. **Echocardiography** provides useful information on anatomy and possible presence of underlying diseases or abnormalities, such as hypertrophic cardiomyopathy. Trasthoracic echocardiography (TTE) is often performed routinely preceding the ablation procedure, since it has the advantages of no emission of radiation, low cost, ready availability, and rapidity. Many centres employ also TEE, especially when left atrial ablation has to be performed. The transesophageal probe could help revealing the presence of thrombus or other abnormalities, but carries additional costs as well as a not negligible risk [57]. **Magnetic resonance** (MR) can be an alternative to echocardiography in cases of low echogenicity. Images are produced in 3D with high spatial resolution and reproducibility [58]. Despite the excellent performance, this technique is rarely adopted for the routine workup of patients, due to the relatively high costs of acquisition and the need for dedicated personnel. Another alternative to echocardiography and MR is computer tomography (CT). Like MR, the major advantages of this technique over medical US are the high spatial resolution and the optimal reproducibility of extracardiac structures. However, the modality is disadvantaged by the low functional information, the lack of temporal resolution and, as MR, the need for a dedicated hardware, software, and personnel [58].

- **Diffuse Fibrosis & Scarring.**
In addition to cardiac anatomy and cardiac mapping, the information about the possible presence of myocardial fibrosis\(^2\) and focal scars\(^3\) is also used to plan the ablation procedure. Indeed, the presence of electrically silent scar tissue or low conduction diffuse fibrosis affects cardiac wave propagation, eventually determining cardiac arrhythmias onset or maintenance [59]. During catheter ablation, the contour of these scars is electrically isolated to avoid fragmented electrical signals that might affect cardiac conduction. For this reason, a comprehensive understanding of scars location, extend and transmurality is required to define properly the critical regions that must be ablated and the therapeutic approach that need to be adopted (e.g. endocardial or epicardial ablation).

---

\(^2\)Fibrosis is a pathological feature of distinct cardiac diseases that results in increased wall stiffness, cardiac remodeling, and heart failure

\(^3\)Scars are regions of non-excitable fibrous tissue that replaces normal tissue after an injury. In several cardiac diseases (e.g., myocardial infarction, congestive heart failure, hypertension), this biological process of wound repair is altered and leads to the formation of large focal scars that deteriorate the peculiar myocardial structure and its functionality.
The clinical gold standard approach for detecting myocardial scar tissue is late gadolinium enhancement (LGE) by cardiac MR (CMR). The technique requires the use of gadolinium based contrast agents which modify the magnetic properties of tissues in which they accumulate, providing a high-intensity signal on MR images. As these large molecule agents distribute differently into healthy and diseased myocardium, the scar tissue can be distinguished from the viable myocardium. LGE has been validated for mapping fibrosis in the ventricles while it remains at an investigational level for atrial fibrosis [58]. Despite the excellent resolution that characterize this technique, the dimension of certain structures critical for reentrant arrhythmias (e.g., bundles of surviving myofibers embedded in scar tissue) is still beyond the imaging resolution of current clinical CMRs [56]. Moreover, although powerful in detecting focal scars, LGE imaging is limited in detecting diffuse myocardial fibrosis, as it requires regions of presumed normal myocardium to provide the necessary contrast between affected and unaffected tissue. In this context, other quantitative CMR techniques have been developed to assess certain parameters that cannot be studied with LGE. A summary is reported in Fig. 39.

Other imaging modalities for myocardial tissue characterization includes: SPECT, PET, CT and echocardiography [60]. In particular, echocardiography has been largely explored over the last years, due to the undeniable advantages of this diagnostic tool as compared to others. Major benefits includes no emission of radiation, low cost, ready availability, and rapidity. US elasticity imaging, for instance, has been explored for detecting local myocardial stiffness variations [61, 62, 63, 64, 65]. Tissue doppler imaging and speckle tracking echocardiography have also been investigated to measure myocardial strain. Although substantial research has been already conducted in the field with promising results, the technique is not yet available for clinical testing using commercial US systems. Further studies are needed to support the effective translation of these approaches from bench to bedside.

**Intra-procedure Monitoring** In addition to pre-procedure assessment of cardiac tissue for planning the procedure, intra-procedural information could also be of extreme value for the cardiac electrophysiologists. Intra-procedure monitoring can support clinicians in guiding the ablation catheter to the target region, detecting peri-procedural complications, and ensuring a
complete and permanent elimination of the critical substrate without producing collateral injury.

- **Catheter Navigation.**
  
  **Fluoroscopy** is the most common approach currently used to guide the catheter during an ablation procedure. Despite its widespread use, this imaging technique uses X-rays to obtain real-time moving images of heart structures and catheters, exposing the patients and especially the physicians to high levels of radiation for prolonged times. For this reason, extensive research has been conducted over the last decades to identify alternatives that minimize the use of X-rays in catheter ablation and in other interventional procedures. **Non-fluoroscopic navigational maps** are obtained integrating electroanatomical maps, generated during the procedure, with 3D images of the heart obtained pre-procedurally with MR or CT. More recently, images of the anatomical structures collected in real-time with imaging systems, like **intracardiac echocardiography** (ICE), are integrated with the electroanatomical maps or used independently to guide the catheter. This is of high interest, not only for the visualization of the catheter tip with respect to the myocardium, but also for imaging mobile structures (e.g., the esophagus).

- **Peri-procedural Complications.**
  
  Despite continuous efforts to improve safety of catheter ablation therapy, the incidence of major peri-procedural complications (i.e., complication that result in permanent injury or death) is still relatively high. This can be associated with the complexity of the procedure and with the lack of a standardized technique for the early detection of adverse events. In this regard, the use of **echocardiography** has become more and more frequent intra-procedurally, not only because it assists the navigation of the catheter, but also because it allows a early detection of adverse events and a prompt intervention. Specifically, ICE has proved its role in recognizing complications, like perforations, cardiac tamponade, and thrombi formation [66].

- **Lesion Formation.**
  
  Permanent destruction of abnormal cardiac tissue is critical to the success of catheter ablation therapy and long term arrhythmia-free survival in patients. Ineffective lesion formation is commonly related to remaining arrhythmogenic substrate or conduction recovery (i.e., the ablated tissue recovers its electrical excitability) [67]. The former can occur when there is an inappropriate cardiac mapping and an incomplete understanding of the driving mechanisms of the treated arrhythmia. Conduction recovery, instead, is likely due to the delivery of insufficient thermal dose which prevent the formation of contiguous and transmural lesions without viable gaps.

  In the current ablation procedures, the ablation is inferred based on numerous indirect parameters or by identifying conduction block via changes in cardiac activation sequence or arrhythmia non-inducibility [68]. Affecting factors that have been investigated include: power delivered, temperature at the catheter tip, ablation circuit impedance, catheter-tissue contact force, electrode size and myocardial perfusion. Unfortunately, these indirect measures neglect completely the inter-individual variability and thereby cannot directly determine whether an ablation lesion, though intended, has actually formed [69]. For this reason, novel experimental tools able to characterize RF lesions in situ and map the lesion transmural extent in real-time are urgently needed.

  Noteworthy, even though also in this case the ultimate objective is the characterization of the myocardial tissue, lesion monitoring differs from myocardial fibrosis mapping, as it
requires an imaging modality able to provide information in real-time. As a consequence, most of the approaches described for the pre-procedural assessment of myocardial fibrosis cannot be transferred in this setting.

**Echocardiography** is a good candidate in this context, because of its superior capability to perform real-time image reconstruction. **Acoustic radiation force impulse** (ARFI) imaging, for instance, has attracted particular attention for its ability to measure tissue elasticity using standard clinical US imaging systems [70, 71, 72]. The technique evaluates the amplitude of the tissue displacement induced by an ultrasonic pulse focused at a target region. As stiff tissue displaces less than soft tissue, the ARFI signal is able to assess relative tissue compliance. Similar to ARFI, also **shear wave imaging** has shown its potentialities in lesion formation assessment [73, 74, 75, 76, 77]. Also in this case, an US-based imaging method is used to estimate the local mechanical compliance of the myocardium, but - contrary to ARFI - the tissue is imaged away from the excitation point to track the propagation of transverse waves generated by the ultrasonic pulse. Both ARFI and SWI have been implemented mostly using ICE catheter probes, but these techniques could be ideally implemented with all cardiac US systems.

Although not readily available, **real-time MR** should also be mentioned among the emerging approaches for intra-procedural assessment of the completeness of ablation lesions [78, 79]. The use of MR to acutely identify ablation lesions has been traditionally hampered by the fact that this imaging modality requires time-consuming image acquisition scan, making real-time images possible only with low image quality or low temporal resolution. However, the recent development of new iterative reconstruction algorithms has improved impressively the temporal resolution without compromising excessively the in-plane resolution [80]. Despite the new promises offered by these cutting-edge technologies, it is worth to mention that the widespread use of intra-procedural MR imaging might be continue to be limited by the high-cost, the need for dedicated personnel, the need for MR-compatible ablation catheter, and the additional restrictions imposed in the case of patients with implantable devices.

## 2 Clinical Needs

Following the previous literature analysis, Tab. C.2 provides an overview of the unmet clinical needs in the context of catheter ablation procedure for cardiac arrhythmias treatment.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N1: Gold standard technique for pre-procedure assessment of myocardial fibrosis at ventricles and atria level.</td>
<td>G1 - 1. Visualize the relevant anatomical structures in real-time.</td>
<td>O1 - 1.1 Display anatomical images with a spatial resolution down to 1 mm, and a frame rate of at least 30 fps.</td>
<td>The visualization of anatomical structures helps to identify the critical regions. Structures can be very small. E.g., ridge separating the LPV from the LAA: 3-4 mm at its narrowest point, fossa ovalis: down to 4 mm in diameter. Human eye can see between 30 and 60 fps.</td>
</tr>
</tbody>
</table>
O1 - 1.2 Visualize up to 20-24 cm depth, when imaging the atria from the chest surface. Visualize up to 12 cm depth, when imaging the ventricles from the chest surface. Visualize up to 12-15 cm, when imaging with an intracardiac probe or a transesophageal. Depending on the location of the probe (e.g., transthoracic probe from chest surface, or intracardiac close to the tissue to be investigated) and the cardiac tissue that has to be characterized (e.g., supraventricular arrhythmia or ventricular arrhythmia) the requirements on the penetration depth can change.

O1 - 1.3 Be robust against cardiac structures motion. Beating heart and breathing can alter the visualization of cardiac structures.

O1 - 1.4 Provide 3D information. 3D information can facilitate anatomical structures recognition.

G1 - 2. Distinguish fibrosis from health myocardium.

O1 - 2.1 Assess tissue stiffness with spatial resolution down to 0.2 mm and a maximum estimation error of 1.8 kPa. Fibrosis increases tissue stiffness [81, 62]. E.g., mean myocardial stiffness of hypertrophic cardiomyopathy populations vs healthy volunteers: 12.68 ± 2.91 kPa vs. 4.47 ± 1.68 kPa [62]. Bundles of surviving myocytes in focal scars can have a diameter of < 100-200 µm [82].

O1 - 2.2 Assess tissue shear viscosity with spatial resolution down to 0.2 mm and a maximum estimate error of 1 Pa·s. Fibrosis increases tissue viscosity [83, 84]. E.g., average values of viscous coefficient at systole in normal vs reperfused myocardium: 5.0 ± 3.6 Pa·s vs 14.9 ± 4 Pa·s [84]. Bundles of surviving myocytes can have a diameter of < 100-200 µm [82].

O1 - 2.3 Assess coronary artery blood flow magnitude in a range from 5-50 cm/s. Myocardial infarction, typically associated with coronary artery disease, results into tissue death and ultimately degenerates in cardiac fibrosis [85]. Coronary flow velocity can increase up to 50 cm/s in patients with severe stenosis [86].

O1 - 2.4 Assess coronary microvasculature. Irreversible perfusion defects are indirect markers of fibrosis [87]. Intramyocardial vessels diameter down to 100 µm [88].

O1 - 2.5 Provide 3D information. 2D images hamper the assessment of scar volume, scar transmurality and absolute coronary flow estimation due to angular dependence of axial Doppler estimate [88].

N2: Gold standard technique for intra-procedure lesion formation monitoring at ventricles and atria level.

O2 - 1.1 Display anatomical images with a spatial resolution down to 1 mm, and a frame rate of at least 30 fps. Visualize critical regions in real-time may improve the safety and efficacy of catheter ablation at these sites [89]. Structures can be very small. E.g., ridge separating the LPV from the LAA: 3-4 mm at its narrowest point, fossa ovalis: down to 4 mm in diameter. Human eye can see between 30 and 60 fps.
### APPENDIX C. CATHETER ABLATION PROCEDURE

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>O2 - 1.2</strong></td>
<td>Visualize up to 20-24 cm depth, when imaging the atria from the chest surface. Visualize up to 12 cm depth, when imaging the ventricles from the chest surface. Visualize up to 12-15 cm, when imaging with an intracardiac probe or a transesophageal.</td>
</tr>
<tr>
<td><strong>O2 - 1.3</strong></td>
<td>Be robust against cardiac structures motion.</td>
</tr>
<tr>
<td><strong>O2 - 1.4</strong></td>
<td>Provide 3D information.</td>
</tr>
<tr>
<td><strong>G2 - 2.</strong></td>
<td>Visualize the intracardiac surgical instruments (e.g., catheter, needle) in real-time.</td>
</tr>
<tr>
<td><strong>O2 - 2.1</strong></td>
<td>Display anatomical images with a spatial resolution down to 0.8 mm, and a frame rate of at least 30 fps.</td>
</tr>
<tr>
<td><strong>O2 - 2.2</strong></td>
<td>Track catheter over time.</td>
</tr>
<tr>
<td><strong>G2 - 3.</strong></td>
<td>Distinguish ablation lesion from health myocardium in real-time.</td>
</tr>
<tr>
<td><strong>O2 - 3.1</strong></td>
<td>Assess tissue viscoelasticity with a maximum estimation error of 10 kPa.</td>
</tr>
<tr>
<td><strong>O2 - 3.2</strong></td>
<td>Display viscoelasticity maps with a frame rate of at least 30 fps.</td>
</tr>
<tr>
<td><strong>O2 - 3.3</strong></td>
<td>Provide 3D information.</td>
</tr>
<tr>
<td><strong>G2 - 4.</strong></td>
<td>Assess completeness of ablation lesion (i.e., durable lesion).</td>
</tr>
<tr>
<td><strong>O2 - 4.1</strong></td>
<td>Assess tissue viscoelasticity. Minimum performance to be defined.</td>
</tr>
</tbody>
</table>

Depending on the location of the probe (e.g., transthoracic probe from chest surface, or intracardiac close to the tissue to be investigated) and the cardiac tissue that has to be characterized (e.g., supraventricular arrhythmia or ventricular arrhythmia) the requirements on the penetration depth can change. Beating heart and breathing can alter the visualization of cardiac structures. 3D information can facilitate spatial orientation, and anatomical structures recognition.

Ablation Catheter: 8-9 Fr diameter (1 Fr = 0.33 mm). Needle for transeptal procedure: 0.8 mm diameter. Human eye can see between 30 and 60 fps.

Catheter tracking is critical to properly localize the ablation tip at the level of the arrhythmogenic region. This can be done manually by the user, or automatically.

Tissue ablation causes local variations in the viscoelastic properties [90]. E.g., ablation stiffness at the outer transition zone boundary vs at the condensation boundary: 3.1 ± 1.0 kPa vs 36.2 ± 9.1 kPa [90].

Human eye can see between 30 and 60 fps. The border of the lesion should be continuously assessed to avoid collateral tissue injury (e.g., esophagus damage during pulmonary vein isolation).

2D images hamper the assessment of lesion volume, lesion transmurality.

Important limitations exist when defining "complete lesion". The gold standard of transmural tissue necrosis is histopathology, which is not applicable clinically. So far, the effectiveness of any new technology in human studies is primarily assessed by acute and long-term clinical outcomes. Among other objectives, SPICE aims to validate intra-procedure parameters for assessing acute lesion efficacy that reliably translate to long-term procedural success. Once identified the parameters, it would be possible to define the required performance to properly distinguish complete lesion from incomplete.
### O2 - 4.2 Assess coronary microvasculature.
Regional blood flow can have a cooling effect that alter the heat transfer to tissue during cardiac ablation [67, 91].

### O2 - 4.3 Assess ablation tip-tissue contact.
Catheter tip-tissue contact can directly affect lesion size, depth and volume [67].

### O2 - 4.4 Display viscoelasticity maps, coronary microvasculature maps and catheter tip-tissue contact force with a frame rate of at least 30 fps.
Human eye can see between 30 and 60 fps. The information about lesion viscoelasticity and affecting factors (e.g., cooling effect from microvasculature) should be continuously assessed to assess ablation lesion completeness.

### N3: Simple high quality image acquisition

#### G3 - 1. Improve the performance of image reconstruction algorithms w.r.t. traditional approaches.
Real-time nature of ultrasonography imposes strong constraints on reconstruction speed of the processing algorithms. Traditional methods (e.g. Delay-and-Sum) have required temporal resolution at the expense of a lower image quality. It has been reported that approximately 10% to 15% of routine echocardiograms have poor image quality [21].

#### G3 - 2. Reduce manual settings to obtain optimal image settings by non-expert users.
Artifacts are frequently encountered during echocardiographic examinations. E.g., comet-tail artifacts due to catheters or pacing wires [22]. Blooming artifacts due to metal wires. Reverberations due to rib cage, lungs or thick fat layer [23].

#### O3 - 1.1 Increase spatial resolution, contrast-to-noise ratio or other image quality metrics w.r.t traditional Delay-and-Sum (DAS) beamforming, without compromising the near real-time processing.
Once the ROI has been identified (e.g., a hyperechoic region which corresponds to a focal scar), the system can automatically track it. This could be relevant especially when using ICE as it is more complex to operate. Update frequency in this case could be derived considering the speed of the ROI moves in the FOV.

#### O3 - 1.2 Suppress image artifacts.
Transmit parameters could be optimized depending on image content. Update frequency has been defined considering the variation between cardiac cycle phases. As heart rate can reach up to 100-300 bpm in young adults with arrhythmias (i.e., heart cycle can last 0.6-0.2 s), parameters update should be performed every 0.3-0.1 s to properly follow the environmental changes associated with heart cycle.

#### O3 - 2.1 Adjust automatically transmission scheme every 0.1 s to improve image quality.
Receive parameters could be optimized depending on image content. Update frequency has been defined considering the variation between cardiac cycle phases (cfr. O3 - 2.1).

#### O3 - 2.2 Adjust automatically reconstruction parameters every 0.1 s to improve image quality.
Once the ROI has been identified (e.g., a hyperechoic region which corresponds to a focal scar), the system can automatically track it. This could be relevant especially when using ICE as it is more complex to operate. Update frequency in this case could be derived considering the speed of the ROI moves in the FOV.

#### O3 - 2.3 Adjust automatically the steering angle every second to maintain the region of interest (ROI) in the field of view (FOV).
References


REFERENCES


REFERENCES


[37] A. Meir and B. Rubinsky. “Distributed network, wireless and cloud computing enabled 3-D ultrasound; a new medical technology paradigm”. In: *PloS one* 4.11 (2009), e7974.


REFERENCES


[65] M. Strachinaru et al. “Local myocardial stiffness variations identified by high frame rate shear wave echocardiography”. In: Cardiovascular Ultrasound (2020).


REFERENCES


[78] R. Ranjan et al. “Identification and acute targeting of gaps in atrial ablation lesion sets using a real-time magnetic resonance imaging system”. In: Circulation: Arrhythmia and Electrophysiology 5.6 (2012), pp. 1130–1135.


**Acronyms**

AI  Artificial Intelligence.
ALARA  As Low As Reasonably Achievable.
AoA  Analysis of Alternatives.
API  Application Programming Interface.
ASIC  Application Specific Integrated Circuit.
BDP  Bandwidth-Delay Product.
CEI  Italian Electrotechnical Commission.
CPU  Central Processing Unit.
CtQs  Critical to Quality Parameters.
DAS  Delay And Sum.
DL  Deep Learning.
EMC  ElectroMagnetic Compatibility.
EN  Hamonized Rules.
FPGA  Field Programmable Gate Array.
GDPR  General Data protection Regulation.
GPU  Graphical Processing Unit.
GUI  Graphical User Interface.
IEC  International Electrotechnical Commission.
IGT  Image-Guided Therapy.
IP  Internet Protocol.
IQ  In-phase Quadrature.
IQR  Interquartile range.
JCAHO  Joint Commission on Accreditation of Healthcare Organizations.
JCI  Joint Commission International.
LAN  Local Area Network.
MAN  Metropolitan Area Network.
MEX  Matlab EXternal.
MI   Mechanical Index.
MoE  Measure of Effectiveness.
MTU  Maximum Transmission Unit.
NWO  Dutch Research Council.
OSI  Open Systems Interconnection.
PAN  Personal Area Network.
PID  Proportional Integral Derivative.
PW   Plane Wave.
RF   Radio Frequency.
RTT  Round-trip-time.
SH   Sample Hold.
SOI  System Of Interest.
TCP  Transmission Control Protocol.
TI   Thermal Index.
TOF  Time Of Flight.
UDP  User Datagram Protocol.
US   Ultrasound.
VPN  Virtual Private Network.
WAN  Wide Area Network.