POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# Simulating Aeration at Birth: building an Open-Source Newborn Lung Model

Tesi di Laurea Magistrale in
Biomedical Engineering - Ingegneria Biomedica

**Luca Andriotto, 928454**

---

**Advisor:**
Prof. Raffaele Dellacà

**Co-advisors:**
Prof.ssa Chiara Veneroni

**Academic year:**
2023-2024

**Abstract:** Here goes the Abstract in English of your thesis (in article format) followed by a list of keywords. The Abstract is a concise summary of the content of the thesis (single page of text) and a guide to the most important contributions included in your thesis. The Abstract is the very last thing you write. It should be a self-contained text and should be clear to someone who hasn't (yet) read the whole manuscript. The Abstract should contain the answers to the main research questions that have been addressed in your thesis. It needs to summarize the motivations and the adopted approach as well as the findings of your work and their relevance and impact. The Abstract is the part appearing in the record of your thesis inside POLITesi, the Digital Archive of PhD and Master Theses (Laurea Magistrale) of Politecnico di Milano. The Abstract will be followed by a list of four to six keywords. Keywords are a tool to help indexers and search engines to find relevant documents. To be relevant and effective, keywords must be chosen carefully. They should represent the content of your work and be specific to your field or sub-field. Keywords may be a single word or two to four words.

**Key-words:** here, the keywords, of your thesis

## 1. Introduction

WIP.

## 2. Methods

Figure 1 provides a high-level overview of the main operations performed. Each of the two classes of methods correspondss to one of the two models:

1. *Chaste* library is utilized in the generation of *morphometric model*.
2. *Julia Programming Language* is employed to describe and instantiate the *mechanical model*, as well as to perform *simulations*.

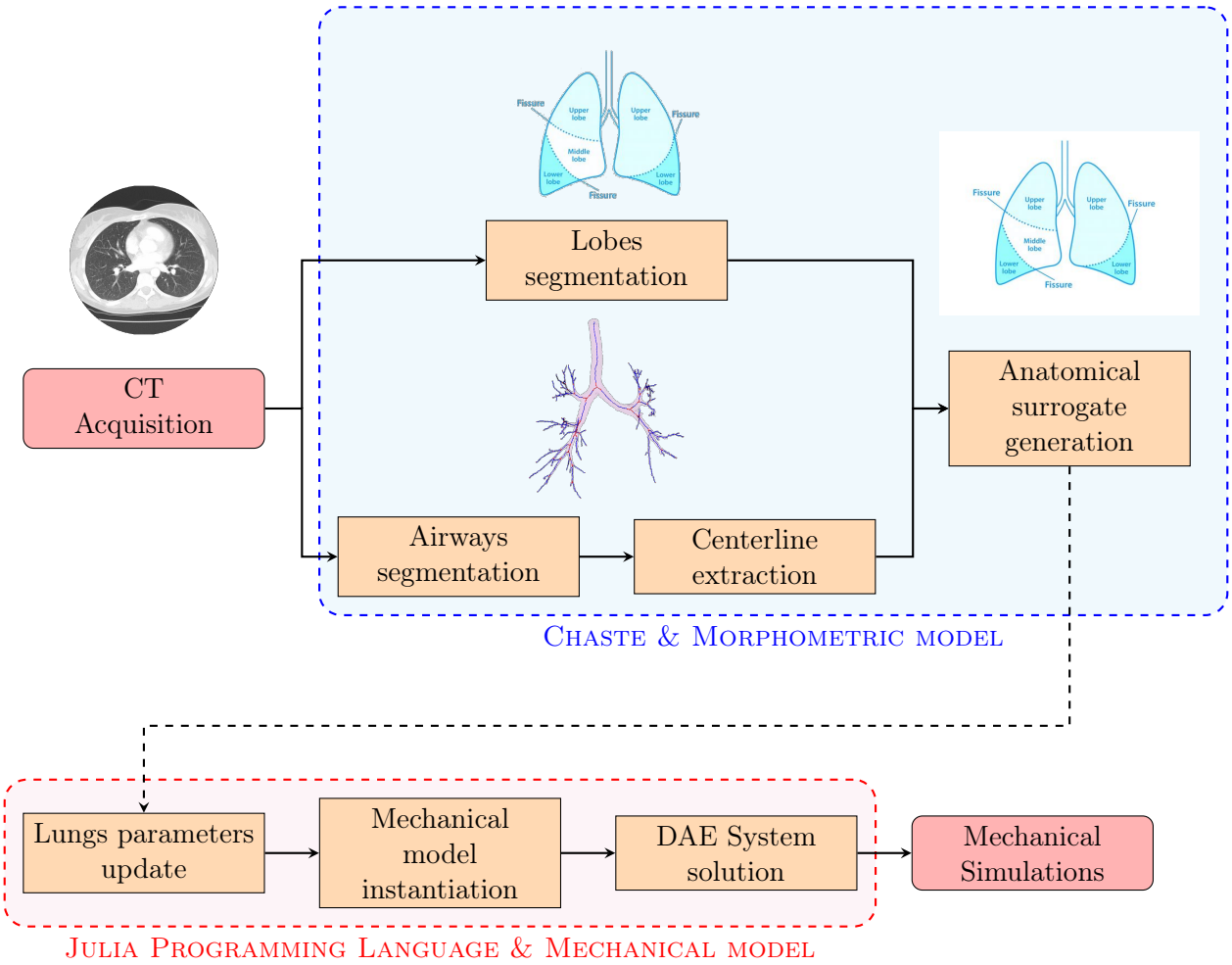


Figure 1: Data pipeline. The process begins with a *patient-specific image* (i.e. CT) of a premature newborn. The extracted data, comprising *two segmentations*, are then processed to obtain an anatomical surrogate of the airway tree. This is necessary due to scanner resolution not allowing for the discrimination and localization of small branches. From the resulting morphometric model, the *mechanical parameters* can be derived, which are essential for generating an accurate simulation model. Finally, a numerical solver for differential equations provides the final output.

## 2.1. Chaste

Chaste is a C++ open-source (BSD licensed) library developed by Oxford University. It has multiple use cases across various biomedical fields, with an emphasis on cardiac electrophysiology and cancer development[4]. It can be integrated into a C++ program or used via «User Project» (i.e. `ctest`). Specifically, the "AirwayGenerationTutorial" is considered as a first codebase and properly adapted to match newborn parameters[7].
The required input consists of two pieces of information:

- A *mesh of centerline points*. This mesh is provided in TetGen format, comprising "airways.node" and "airways.edge" files. The first file lists centerline points coordinates, respective sampled airways radius and a boolean value to indicate if the point is generative. The second one contains all the connections between pairs of points.
- Four (or five) *lobes segmentations* in STL format. These segmentations are necessary as they physically impose a limit on the growth algorithm.

## 2.2. Morphometric Model

### 2.2.1 CT Image Processing: Lung Segmentation, Centerline and Radii Extraction

«3D Slicer» is an open-source software used for CT image processing. Two extensions are installed:

«*Chest_ imaging_ platform*»: This extension enables semi-automatic segmentation of major airways from a single fiducial point. It can also extract adult lobes using three fiducial points per lung fissure. However, in our case, the fissures are not visible, necessitating manual intervention.

«*SlicerVMTK*»: This extension is used for extracting centerline points.

### 2.2.2 Generation of the Statistical Portion

The Chaste User Project reads the input files (see Section 2.1), and begins growing the anatomical surrogate from the points labeled as «generative». The algorithm operating under the hood is a modified version of the one described in [6]. The generated output is available in various formats:

- vtu: Unstructured Grid (base64 encoded) format used by VTK library. It can be displayed by ParaView, an open-source viewer.
- node and edge: TetGen format. Such files are better suited for further processing.

## 2.3. Julia Programming Language

Julia is a free, open-source (MIT licensed), fast, scientific and numerical computing-oriented programming language. Its computational efficiency is comparable to that of statically-typed languages like C or Fortran. Moreover, its high-level code expressivity rivals that of languages like Python, R and MATLAB[1].
Two key features, inspired by the *Lisp Language*, are highlighted here.

*Metaprogramming*: Code is treated as any other Julia data structure, thus can be dynamically generated and manipulated at runtime.

*Macros*: They help instantiate the generated code in the body of a program.

Their importance is closely tied to the concept of Domain-Specific Languages (aka DSLs). These dialects are composed by abstractions that can be properly exploited to solve particular problems (e.g. modeling complex systems, solving differential equations).
Julia REPL has a built-in package manager (i.e. «`Pkg.jl`») used for managing project dependencies and ensuring the *repeatability* of computational setups. This is achieved by saving the required package names and commits into 'Project.toml' and 'Manifest.toml' files.

### 2.3.1 «`ModelingToolkit.jl`» handles Model Complexity

This Julia package encompasses all the tools necessary for model design. `ModelingToolkit.jl`» is equation-driven, requiring each system to be described by Differential-Algebraic Equations (i.e. DAEs) for subsequent solving[3]. Its built-in DSL optimizes every stage of modeling, from prototyping components to instantiating the complete system.
An acausal paradigm can be adopted, allowing users to reason in terms of *components*[5]. This modularity facilitates system extensibility compared to the causal approach, where the entire system of Differential-Algebraic Equations must be considered and manually simplified[2].
In particular, the usage of `@mtkmodel` macro enables hierarchical generation of building blocks recurring in the highest-order model (i.e. «Lungs»). Here is how information is structured within `@mtkmodel` macro.

Listing 1: `@mtkmodel`: a macro for systems prototyping.

```
@mtkmodel <name_of_model> begin
    @parameters begin
        # (Optional) Some constant (e.g. Resistance, Capacitance) ...
    end
    @components begin
        # (Optional) Some dependency system (e.g. Resistor, Capacitor) ...
    end
    @variables begin
        # (Optional) Internal variables ...
    end
    @equations begin
        # Differential Algebraic Equations describing the model's behavior.
    end
    @continuous_events begin
        # (Optional) Some callback function ...
    end
end
```

Replicating the behavior of electrical components using this language is straightforward, once you are familiar with the syntax and understand the Differential-Algebraic Equations that represent their characteristics. Each generated system can then be composed into more complex ones, using the internal `@components` macro, thereby implementing the hierarchical structure mentioned earlier.
After describing the highest-order system, Julia compiler requires its instantiation before any simulation can be performed. This is accomplished using `@mtkbuild` macro, which minimizes the number of equations that need to be solved.

### 2.3.2 Callbacks' Role in State Variables Discontinuity Handling

Not all characteristics of electrical components can be defined solely by DAEs. Voltages or currents may suddenly change, triggered by a circuit event. In such cases, *continuous callback functions* can be employed to appropriately alter the value of state variables. These callbacks consist of two functions:

- `condition`: Specifies the event to be tested.
- `affect`: Defines how the state variable(s) should be changed.

The component-based approach allows for the definition of callbacks directly within the (sub)system being modeled.

## 2.4. Mechanical Model

Code modularity is directly reflected in the electrical equivalent circuit. Specifically, by encapsulating systems with the internal `@components` macro, it becomes possible to generate models of increasing complexity. This approach enables a clear separation between components belonging to different hierarchical levels and facilitate compartmentalization during the model design phase.

### 2.4.1 Blocks Description

The following blocks are listed in a bottom-up order (from lowest to highest).

1. **Electrical components**. The simplest blocks are derived from «`ModelingToolkitStandardLibrary`», while integral-dependent ones rely on a modified mathematical block to manage both current integration and its timing correctly. Their behavior varies based on the neonatal pulmonary fluid interface.
   - *Current Integral-Dependent Inductor*: $L(t) = L_a + L_b \cdot \left(1 - \frac{\int idt}{V_{\text{FRC}}}\right)$
   - *Current Integral-Dependent Resistor*: $R(t) = R_a + R_b \cdot \left(1 - \frac{\int idt}{V_{\text{FRC}}}\right)$
   - *Diode*
   - *Inductor*
   - *Resistor*
2. **Modules**. Obtained by connecting the aforementioned components together into functional models representing a physiological structure.
   - *Alveolus*
   - *Airway*. It has a similar behavior with respect to a transmission line.

3. **Lungs**. Highest order model as it is a combination of alveoli and airways.

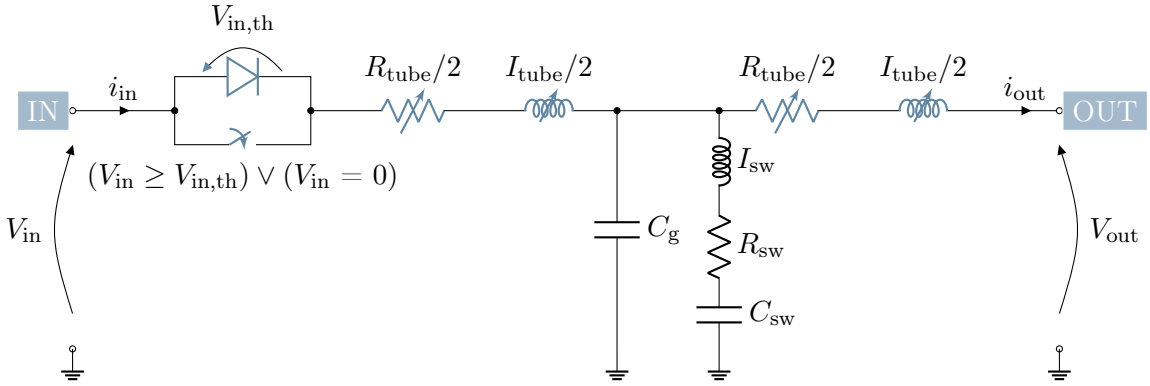


Figure 2: Airway equivalent circuit. In blue: all current integral-dependent components.

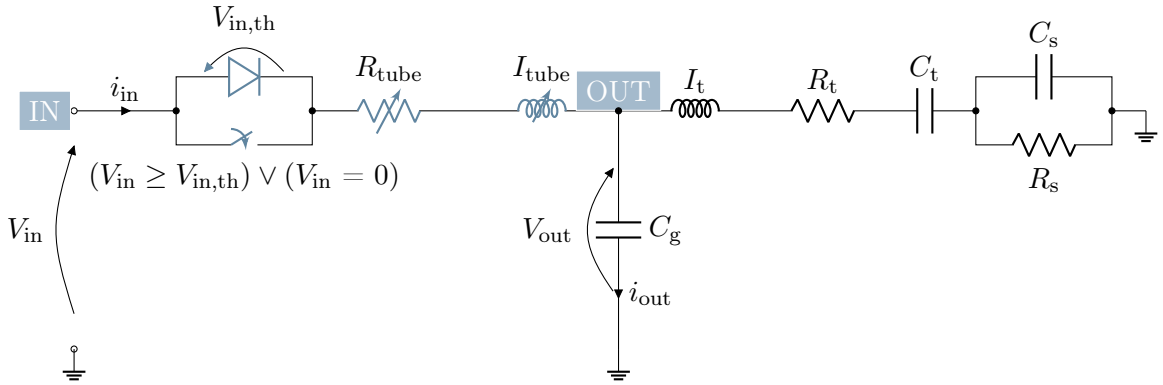


Figure 3: Alveolus equivalent circuit. In blue: all current integral-dependent components.

### 2.4.2 Model Testing on A Subtree

Simulations are executed starting from a subnet, as the full circuit (comprising over 50k modules) requires more memory space than typically available on a common laptop.

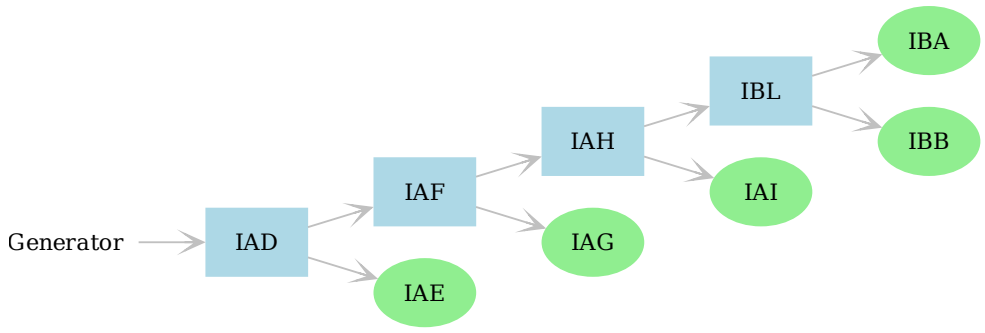


Figure 4: The simulated subtree. Airways are represented in light blue, alveoli in light green.

## References

[1] *Julia Documentation*. URL: `https://docs.julialang.org/`.

[2] Yingbo Ma. *Scaling Equation-based Modeling to Large Systems*. 2024. URL: https://www.youtube.com/watch?v=c-bZ2v1uF14.

[3] Yingbo Ma et al. *ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling*. 2021. arXiv: 2103.05244 [cs.MS].

[4] Gary R. Mirams et al. "Chaste: An Open Source C++ Library for Computational Physiology and Biology". In: *PLOS Computational Biology* 9.3 (Mar. 2013), pp. 1–8. DOI: 10.1371/journal.pcbi.1002970. URL: https://doi.org/10.1371/journal.pcbi.1002970.

[5] *ModelingToolkit.jl Documentation*. URL: https://docs.sciml.ai/ModelingToolkit/stable/.

[6] M. Howatson Tawhai, A. J. Pullan, and P. J. Hunter. "Generation of an Anatomically Based Three-Dimensional Model of the Conducting Airways". In: *Annals of Biomedical Engineering* 28.7 (July 2000), pp. 793–802. ISSN: 1573-9686. DOI: 10.1114/1.1289457. URL: https://doi.org/10.1114/1.1289457.

[7] *TestAirwayGeneration (Chaste Tutorial)*. URL: https://chaste.github.io/docs/user-tutorials/airwaygeneration/.

## Abstract in lingua italiana

Qui va l'Abstract in lingua italiana della tesi seguito dalla lista di parole chiave.

**Parole chiave:** qui, le parole chiave, della tesi, in italiano

## Acknowledgements

Here you might want to acknowledge someone.