Universidade do Minho
Escola de Engenharia

João Paulo Oliveira Vidal

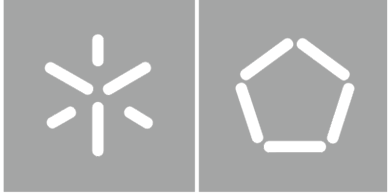
Improved simulation methods to control the temperature in thermoplastic extrusion

Improved simulation methods to control the temperature in thermoplastic extrusion

João Paulo Oliveira Vidal

Uminho | 2023

October 2023



Universidade do Minho
Escola de Engenharia

João Paulo Oliveira Vidal

Improved simulation methods to control the temperature in thermoplastic extrusion

Dissertação de Mestrado

Mestrado em Engenharia do Produto

Trabalho efetuado sob a orientação do

Professor Doutor João Miguel Nóbrega

outubro de 2023

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Acknowlegments

Firstly, I would like to express my profound gratitude to Dr. Miguel Nóbrega for his invaluable guidance and steadfast support throughout this academic journey.

I extend my sincere thanks to Soprefa - Componentes Industriais SA, represented by Dr. Avelino Fonseca and Alberto Sacramento, for their support and collaboration.

To my family, especially my brother and parents, I am deeply thankful for their unwavering support, encouragement, and belief in my abilities.

Last but certainly not least, I want to express my heartfelt gratitude to my wife, Pheak, for her patience, unwavering support, and for being my constant source of inspiration throughout this journey and in life in general.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Universidade do Minho, 31 de outubro de 2023

Resumo

Metodologias avançadas para controlo da temperatura em extrusão de termoplásticos

A extrusão de termoplásticos é um método amplamente utilizado para o processamento de materiais termoplásticos, sendo o controlo de temperatura um fator crítico que afeta a qualidade do produto devido à sensibilidade dos materiais termoplásticos à temperatura. As atuais ferramentas de engenharia assistida por computador (CAE) empregues para modelar o processo de extrusão frequentemente simplificam o processo de cálculo do campo de temperatura e baseando-se em aproximações, como a definição de temperatura apenas nas superfícies dos canais de fluxo. Por outro lado, os processos práticos de extrusão usam sensores em locais específicos para monitorizar e regular dispositivos de aquecimento, criando uma lacuna entre a simulação e os sistemas de controlo do mundo real. As consequências dessa diferença nunca foram avaliadas. Esta dissertação de mestrado aborda essas limitações ao introduzir uma abordagem de modelação com múltiplas regiões que representa fielmente a configuração real e o comportamento dos sistemas de controlo de temperatura de extrusão.

Nesta dissertação de mestrado, é apresentada uma metodologia inovadora, com o objetivo de superar as limitações da modelação numérica atual dos processos de extrusão de perfis. A abordagem implementada considera condições de controlo de temperatura mais realistas. As principais contribuições incluem o desenvolvimento de um sistema de cálculo transiente, incompressível, não-isotérmico e com capacidade de resolver múltiplas regiões, implementado na biblioteca computacional OpenFOAM. Além disso, é introduzida uma condição de fronteira inovadora para replicar o controlo Proporcional-Integral-Diferencial (PID) de resistências de aquecimento que as controla com base em medições de termopar. Os resultados do estudo revelam desvios significativos nos campos de temperatura nas paredes dos canais de fluxo em comparação com as abordagens convencionais, enquanto demonstram efeitos reduzidos no campo de velocidade e uniformidade do fluxo na saída, particularmente durante a operação em regime estacionário.

Os resultados deste projeto de mestrado contribuem significativamente para o avanço da compreensão dos aspectos dos processos de extrusão, fornecendo informações valiosas para otimizar o controlo de temperatura no processo. Os modelos e análises desenvolvidos estabelecem uma base sólida para investigações futuras nesse domínio e abrem caminho para o desenvolvimento de estratégias de controlo de temperatura mais eficientes e precisas na extrusão de termoplásticos.

Palavras-Chave: extrusão de perfis poliméricos, modelação numérica, OpenFOAM®, simulação multi região

Abstract

Improved simulation methods to control the temperature in thermoplastic extrusion

Thermoplastic extrusion is a widely utilized method for processing thermoplastic materials, with temperature control being a critical factor impacting product quality due to the temperature sensitivity of thermoplastic materials. Current computer-aided engineering (CAE) tools for modeling the extrusion process often oversimplify the temperature field calculation, relying on approximations such as performing the temperature calculation just to to the flow channel surfaces. Practical extrusion processes, on the other hand, use sensors at specific locations to monitor the temperature and control the operation of the heating devices, creating a gap between simulation and real-world control systems. The consequences of these differences were never assessed before. This MSc dissertation addresses these shortcomings by introducing a multi-region modeling approach that faithfully represents the actual setup and behavior of extrusion temperature control systems.

Within this master's dissertation, a novel methodology is presented, aimed at overcoming the limitations of current state-of-the-art numerical modeling in profile extrusion transformation processes. The approach focus on achieving more realistic temperature control conditions, departing from the simplifications employed in previous approaches. Key contributions include the development of a transient, incompressible, non-isothermal, and multi-region solver incorporated into the OpenFOAM computational library. Additionally, a specialized boundary condition is introduced to emulate Proportional-Integral-Differential (PID) control of heaters based on real-time thermocouple measurements. The study's findings reveal deviations in temperature fields at flow channel walls compared to conventional assumptions, while demonstrating reduced effects on the velocity field and flow uniformity at the outlet, particularly during steady-state operation conditions.

The outcomes of this MSc project significantly contribute to advancing our comprehension of the thermal aspects in extrusion processes, providing valuable insights for optimizing temperature control within the process. The developed models and analyses establish a strong foundation for future research in this domain and pave the way for the development of more efficient and precise temperature control strategies in thermoplastic extrusion.

Keywords: multi-region simulation, numerical modeling, OpenFOAM®, polymer profile extrusion

CONTENTS

Acknowlegments ..... **iii**

Resumo ..... **v**

Abstract ..... **vi**

List of Figures ..... **ix**

List of Tables ..... **xi**

List of Abbreviations and Acronyms ..... **xii**

Nomenclature ..... **xiii**

1. Introduction 1

1.1. The polymer extrusion process 1

1.2. Computational simulation codes 3

1.3. State of the art 4

1.4. Motivation 8

1.5. Objectives 8

1.6. Dissertation Structure 9

2. Numerical Developments 10

2.1. Multi region solver 10

2.1.1. Methodology 11

2.1.2. Solver Implementation 13

2.2. Heater control boundary condition 18

2.2.1. Methodology 18

2.2.2. Implementation 18

3. Code assessment 22

3.1. Rheology model 22

3.2. 2D Case studies 23

3.3. Results and discussion 26

4. Industrial case study 31

4.1. Presentation 31

4.1.1. Geometries and Boundary conditions 33

4.1.2. Conventional approach 33

4.1.3. Multi region approach 34

4.1.4. Mixed approach....36
4.2. Results and Discussion....37
4.2.1. Mesh sensitivity analysis....37
4.2.2. Results and Discussion....39
4.2.3. Comparison Multi-region, conventional and hybrid case studies....41
5. Conclusions and Future work....45
References....46
Appendix 1 – initContinuityErrs.H Code....51
Appendix 2 – continuityErrs.H Code....52
Appendix 3 – Tfluid.H Code....52
Appendix 4 – Tsolid.H Code....53
Appendix 5 – createMesh.H Code....53
Appendix 6 – createFields.H Code....54
Appendix 7 – chtMultiRegionPimpleFoam.H Code....57
Appendix 8 – externalWallHeatFluxTemperaturePIDFvPatchScalarField.C code....60
Appendix 9 – externalWallHeatFluxTemperaturePIDFvPatchScalarField.H code....68

List of Figures

Figure 1: Polymer extrusion profile example....1
Figure 2: Typical extrusion line....1
Figure 3: Typical die heating control elements....2
Figure 4: Cartridge heater (adapted from [5])....2
Figure 5: Band heater (adapted from [6])....3
Figure 6: Extruder Screw....4
Figure 7: Single screw extruder....4
Figure 8: Twin-screw extruder....5
Figure 9: Extrusion die mounted on the extruder....6
Figure 10: Typical openFoam case structure (adapted from [53])....10
Figure 11: Typical chtMultiRegionPimpleFOAM case struture....11
Figure 12: pimpleFoam source code folder contents....12
Figure 13: chtMultiRegionPimpleFoam source code folder contents....12
Figure 14: chtMultiRegionPimpleFoam solution flowchart....14
Figure 15: openFoam implementation o fluid energy conservation equation....15
Figure 16: openFoam implementation o solid energy conservation equation....15
Figure 17: Fluid mesh creation implementation on createMesh.C....16
Figure 18: Solid mesh creation implementation on createMesh.C....16
Figure 19: Temperature field declaration on createFields.C....17
Figure 20: Temperature control flowchart....18
Figure 21: area-averaged temperature implementation....19
Figure 22: PID function implementation....20
Figure 23: Representation of the Robin boundary condition....20
Figure 24: Conditional response and Robin boundary condition implementation....21
Figure 25: Polycarbonate rheology data....22
Figure 26: 2D Case study base geometry and boundary patches....23
Figure 27: 2D case study mesh block division....24
Figure 28: 2D case study mesh 1....26
Figure 29: 2D case study mesh 2....26
Figure 30: 2D case study mesh 3....26
Figure 31: 2D case study mesh refinement study....27
Figure 32: 2D case study walls and sensor temperature behaviour....28

Figure 33: 2D case study effect of sensor distance to flow channel....28
Figure 34: 2D case study effect of sensor distance to flow channel inlet....29
Figure 35: 2D case study temperature field (t=1000s)....30
Figure 36: 2D case study pressure field (t=1000s)....30
Figure 37: 2D case study velocity field(t=1000s)....30
Figure 38: Industrial case study profile cross-section....31
Figure 39: Industrial case study extrusion die zones....31
Figure 40: Industrial case study die heating control elements....32
Figure 41: Industrial case study outlet cross section division....32
Figure 42: Conventional approach geometry and boundary patches....33
Figure 43: Industrial case study multi region approach geometry....34
Figure 44: Industrial case study mesh 1....37
Figure 45: Industrial case study mesh 2....37
Figure 46: Industrial case study mesh 3....38
Figure 47: Industrial case study multi region mesh 1....38
Figure 48: Industrial case study multi region mesh 2....39
Figure 49: Industrial case study multi region temperature evolution in the adapter....40
Figure 50: Industrial case study multi region temperature evolution in the die....40
Figure 51: Industrial case study multi region objective function evolution....41
Figure 52: Industrial case study temperature field , comparison between conventional , multi-region and mixed approaches....42
Figure 53: Industrial case study pressure field , comparison between conventional, multi-region and mixed approaches....42
Figure 54: Industrial case study velocity field at outlet ,comparison between conventional , multi-region and mixed approaches....43
Figure 55: Industrial case study temperature field at the flow channel outlet,comparison between conventional , multi-region and mixed approaches....43
Figure 56: Industrial case study individual objective functions(Fobj,i) plot , comparison between conventional , multi-region and mixed approaches....43
Figure 57: Industrial case study temperature at ES7, comparison between conventional, multi-region and mixed approaches....44

List of Tables

Table 1: Polymer properties.....22
Table 2: Die material properties.....23
Table 3: 2D case study boundary conditions.....24
Table 4:2D case study heater temperature boundary conditon parameters.....24
Table 5: 2D case study mesh size by block and total.....25
Table 6: Industrial case study conventional approach boundary conditions.....34
Table 7: Industrial case study multi region approach boundary conditions.....35
Table 8: Industrial case study, multi region approach adapter heater boundary condition parameters..35
Table 9: Industrial case study, multi region approach die land heater boundary condition parameters. 36
Table 10: Industrial case study mixed approach boundary conditions.....36
Table 11: Industrial case study conventional approach errors in function of cell number.....38
Table 12: Industrial case study multi region approach errors in function of cell number.....39

List of Abbreviations and Acronyms

BEM - Boundary Element Method

CAD - Computer-Aided Design

CAE - Computer-Aided Engineering

CFD - Computational Fluid Dynamics

FEM - Finite Element Method

FVM - Finite Volume Method

GNU - GNU's Not Unix!

LED - Light-Emitting Diode

PIMPLE – Concatetantion of SIMPLE and PISO

PID - Proportional-Integral-Derivative

PISO - Pressure-Implicit with Splitting of Operators

SIMPLE - Semi-Implicit Method for Pressure Linked Equations

SMEs - Small and Medium-sized Enterprises

Nomenclature

Greek symbols

α - Thermal Diffusivity

$\dot{\gamma}$ - Shear Rate

η - Shear Viscosity

η_0 - Viscosity at zero shear rate

η_∞ - Viscosity at infinite shear rate

λ - Relaxation Time

ρ - Fluid Density

$\boldsymbol{\tau}$ - Stress Tensor

k -Thermal Conductivity

Roman symbols

A - Area

C_p - Specific Heat Capacity

$\boldsymbol{D}$ - Strain Tensor

E_T - Energy Flux

h - Heat Transfer Coefficient

K_d - Derivative Gain

K_i - Integral Gain

K_p - Proportional Gain

n - Power-law index

q - Heat Flux

R - Universal Gas Constant

t - Time

T - Temperature

$T_{thermocouple}$ - Target Temperature

T_∞ - Room Temperature

$\boldsymbol{u}$ - Vector Velocity

1. INTRODUCTION

1.1. THE POLYMER EXTRUSION PROCESS

The polymer extrusion process is an important industrial manufacturing technique used to produce thermoplastic profiles, as shown in Figure 1 for a broad range of industrial sectors, from the construction industry to the automotive sector [1].

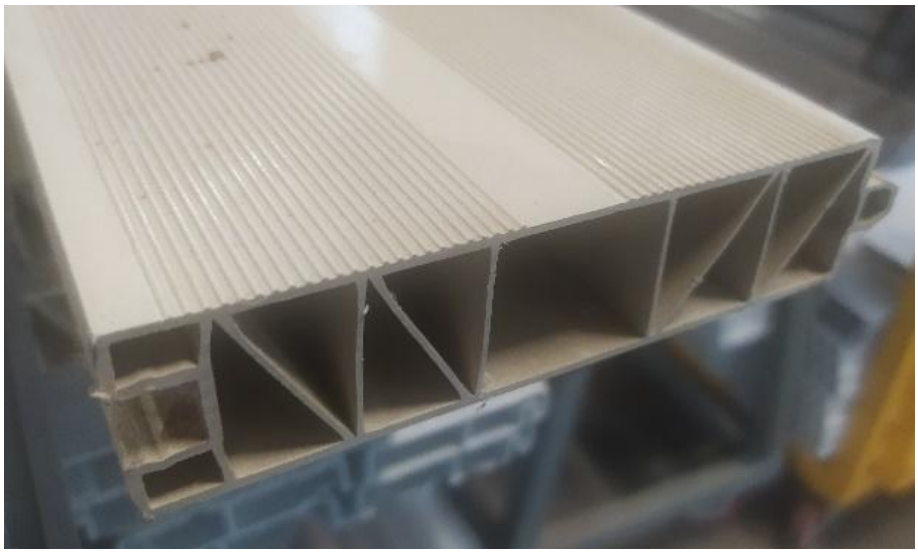


Figure 1: Polymer extrusion profile example

A typical profile extrusion line comprises five main parts: Extruder, Die, Calibration & Cooling, Haul-off, and Cutting, as shown in Figure 2. Each of these components has a well-defined function in the production process.

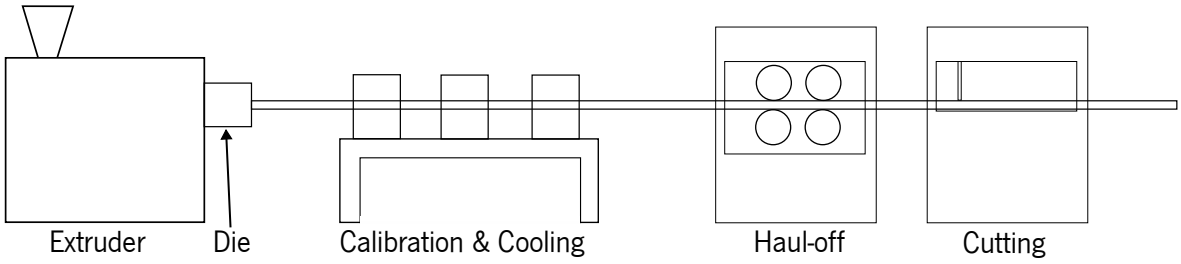


Figure 2: Typical extrusion line

The process begins by introducing the polymer pellets into a hopper, which feeds the barrel of the extruder by gravity. Within the extruder, the pellets are conveyed by one or multiple rotating screws. As the polymer progresses, it is heated to the desired temperature, generating at the polymer melt. At the

outlet of the extruder, the molten polymer material is forced through the die, which shapes it into a specific cross-section. After exiting the die, the polymer profile needs to be cooled and calibrated to reach its final cross-section. This is usually achieved by pulling it through a calibrating/cooling system. The intermediate product is then cut at the cutting unit. In addition to these components, the extrusion line includes a haul-off unit, which pulls the profile at a constant linear velocity [2].

Temperature is one of the most critical variables in the polymer extrusion process, as it plays a fundamental role in achieving high-quality products [4]. To ensure good thermal stability, it is important to have an effective control of the extrusion die heating.The control is based on the temperature values acquired by the thermocouples, which will regulate the state of the heaters.

The typical locations of heaters and thermocouples can be seen at Figure 3. There are two main types of heaters commonly used for this purpose: cartridge heaters (Figure 4) and band heaters (Figure 5) [5,6]. The band heaters are widely utilized and offer ceramic insulation, which helps maintaining the desired temperature in the extrusion process.

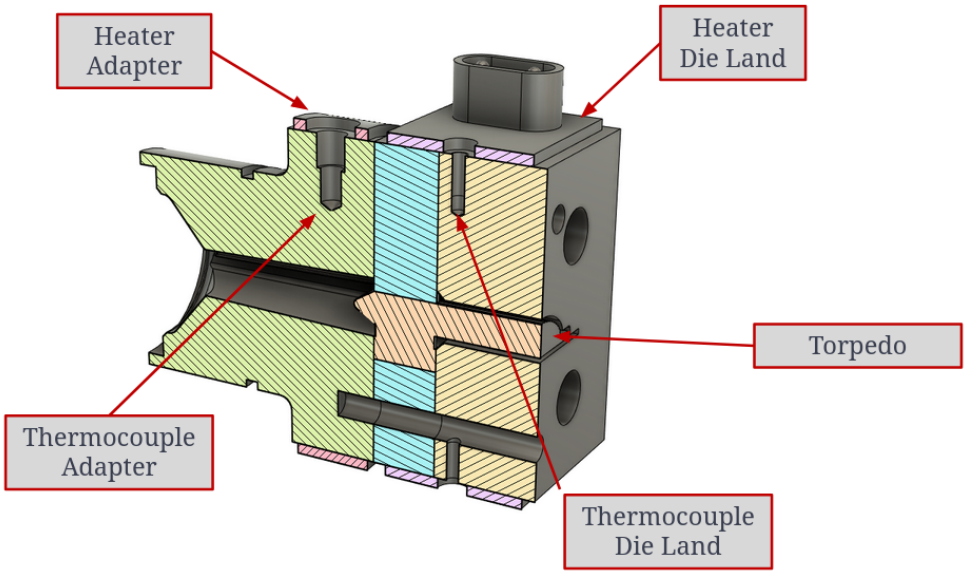


Figure 3: Typical die heating control elements

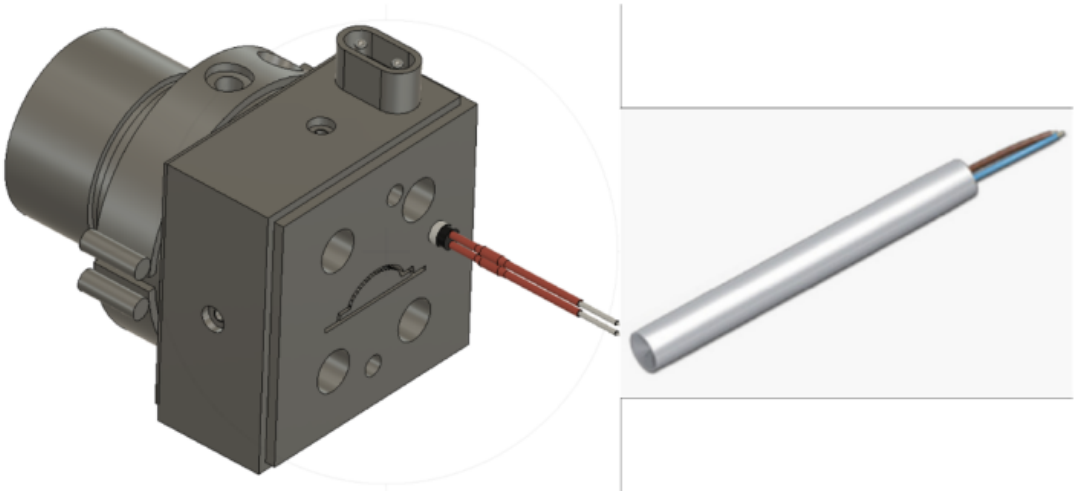


Figure 4: Cartridge heater (adapted from [5])

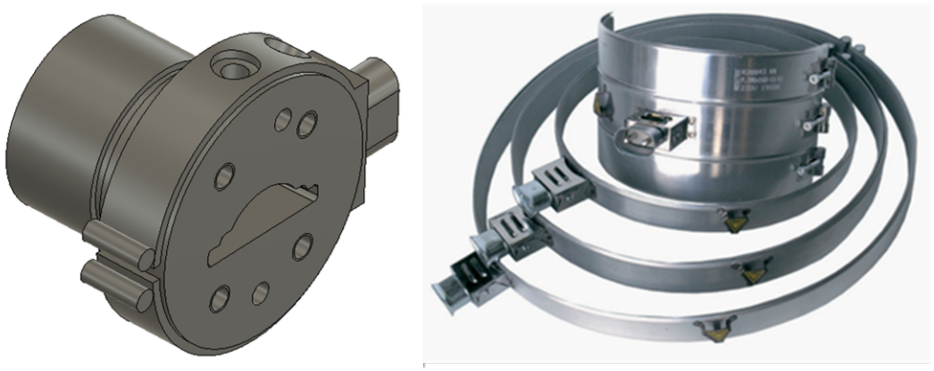


Figure 5: Band heater (adapted from [6])

To control the heater devices and achieve a stable temperature field in the die, several methods have been developed. Starting from the 1970s, researchers proposed Proportional-Integral-Derivative (PID) control algorithms to regulate temperature in polymer extrusion and concluded that the lack of capability of most polymer extrusion process controllers to handle nonlinearities and obtain temperature feedback is a significant limitation [7]. Later, researchers began implementing fuzzy logic algorithms [8].

1.2. COMPUTATIONAL SIMULATION CODES

Computational simulation is an important tool for modern companies because it allows designers and engineers to acquire knowledge on physical processes and data that would be difficult, expensive, or even impossible to obtain experimentally [9].

Nowadays, commercial CFD packages are available [10], however, the costs can be prohibitive for Small and Medium-sized Enterprises(SMEs) and researchers [11]. With the advent of open-source software, companies and individual users have come together in communities to promote the development of software [12]. These software options have the advantage of being available for free, relying on a community of users for service and support, and allowing faster innovation and customization in the field of computational modelling. Some libraries in this domain are Calculix [12], Elmer FEM [13] and OpenFOAM [14].

OpenFOAM® is an open-source computational numerical modelling library written in C++, which makes it a satisfactory solution for solving CFD problems, while enabling users to implement complex physical models easily and reliably [15]. This library is distributed under the GNU license, providing users with the freedom to modify and redistribute the software and a guarantee of permanent free use [4]. OpenFOAM® comprises approximately 200 applications divided into two categories: solvers and utilities. The solvers

are designed to solve specific problems in fluid (or continuum) mechanics, while the utilities are designed to perform tasks involving data manipulation.

1.3. STATE OF THE ART

The development and use of CAE tools have experienced rapid growth over the last 3 decades, enabling the modelling of various stages of the extrusion process.

As mentioned earlier, the polymer extrusion process consists of five main parts, with three being the most important and difficult to design, thus can benefit from computational simulation. These parts are the Extruder, Die, and Calibrator.

Starting with the extruder, the main focus is on modelling the flow in the barrel, which is induced by the screw rotation, as depicted in Figure 6. Extruders can be categorized into two main types: single-screw extruders and twin-screw extruders.

Figure 6: Extruder Screw

For single-screw extruders (as shown in Figure 7) in 1966, Tadmor Z.[14] presented a method for modelling polymer melting, where the temperature was imposed at the walls. Later, Altınkaynak *et al.* [15] assessed the effect of the melting profile on various material properties and processing conditions using a three-dimensional approach based on the Finite Element Method (FEM). In these studies, the temperature was assumed to be imposed at the barrel and screw walls.

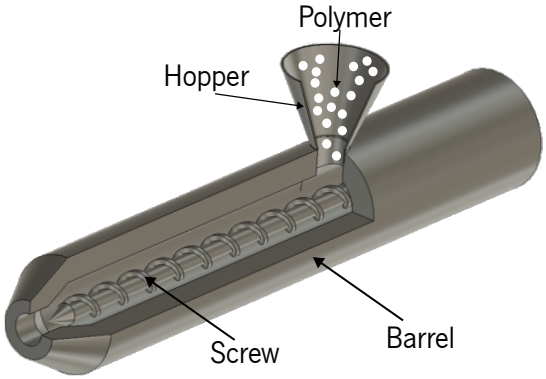


Figure 7: Single screw extruder

For twin-screw extruders (as shown in Figure 8), Bawiskar [16] conducted an assessment in 1998 on the effect of operating conditions on melting. They also considered a constant temperature at the walls of the barrel and screw. Subsequently, Wilczynski and White [17], on the other hand, developed a

model for melting in counter-rotating twin-screw extruders. They incorporated a heat transfer coefficient and utilized the temperature of the barrel and screw materials to calculate the temperature at the walls.

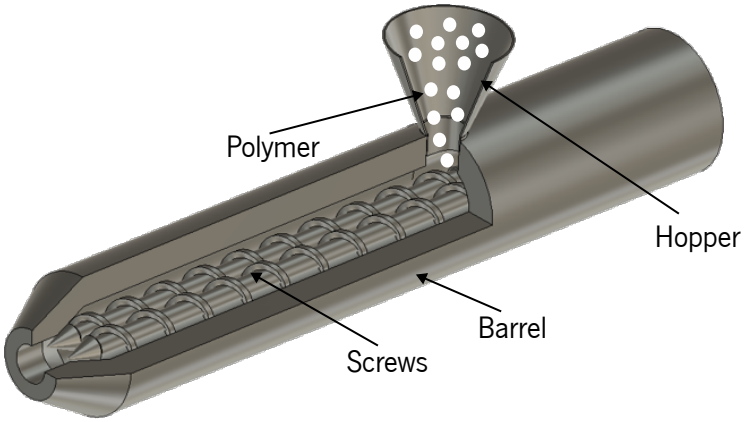


Figure 8: Twin-screw extruder

The extrusion die (as depicted in Figure 9) is the crucial component that shapes the molten material to the desired cross section geometry, making it the most important tool in the extrusion process. Therefore, aiming at guiding its design, it is essential to develop modelling tools that enable the simulation of extrusion die performance. In 1990, Vicek *et al.* [18] developed a model for a small laboratory sheet die and compared it with experimental results. It is worth noting that the energy flux at the flow channel wall was quantified by:

$$\frac{E_T}{A} = h\left(T - T_w\right), \tag{1}$$

where $\frac{E_T}{A}$ is the energy flux divided by the area, T is the temperature at cell center, T_w is the temperature at the wall and h the heat transfer coefficient. This evidences that the wall temperature was assumed to be constant.

In 2013, Nobrega *et al.* [19] introduced a methodology for numerical modelling of polymer flow at the extrusion die. Their approach involved imposing a boundary condition for the temperature at the outer surface of the flow channel, while considering the torpedo (see Figure 3) as insulated. Although there is no clear evidence that this is the appropriate boundary condition for this region. Their study concluded that the flow distribution is primarily influenced by the melt inlet temperature and the temperature of the flow channel wall, particularly in regions with small thickness [19], which evidences the relevance of the accuracy on the temperature field boundary conditions.

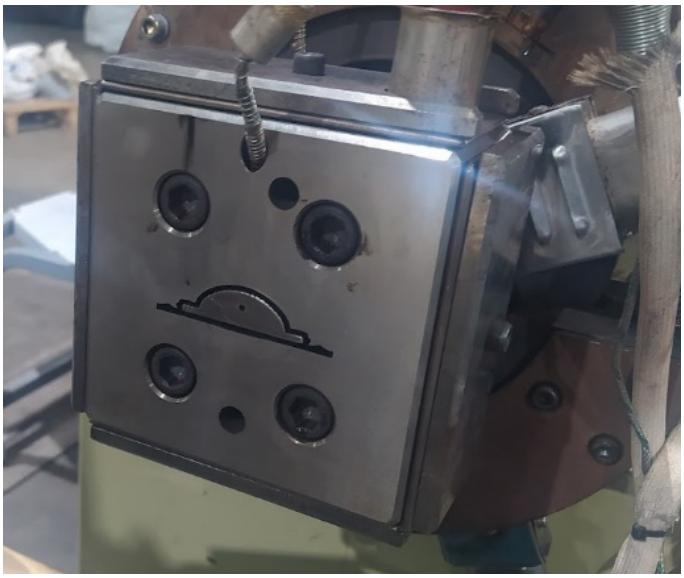


Figure 9: Extrusion die mounted on the extruder

Later, Gonçalves [20] expanded the previously mentioned work to enable the modelling of complex 3D shapes. However, this specific code did not consider temperature effects on the flow.

Extrudate swell is a significant phenomenon in polymer extrusion that has garnered attention from researchers. This rheological phenomenon is characterized by the expansion of the polymer melt at the die outlet, which occurs due to flow redistribution and stress relaxation [21,22]. In 1988, Tran-Cong and Phan-Tien [52] initially introduced an implementation of the Boundary Element Method (BEM) to solve a general three-dimensional viscoelastic flow problem, specifically with the aim of simulating extrudate swell. Later in 2003, Gifford [23] proposed a methodology to compensate for extrudate swell. It is noteworthy to mention that Gifford's work assumed isothermal conditions and Newtonian constitutive models.

The utilization of numerical tools is well-known to enhance the efficiency of tool development [24].

Linked with the ability to model the flow in the extrusion dies, and in response to an increasing market demand for higher product quality and production rates [25,26], several authors have proposed methodologies to improve the design and/or optimize polymer profile extrusion dies. This task is complex, particularly due to the intricate profile cross section, which, among others, usually comprises varying thicknesses that promote different restriction to flow and difficult the achievement of the desired flow balance [1].

Traditionally, companies have relied on trial-and-error approaches, which resort on the expertise of workers with years of experimental knowledge [27], to design profile extrusion dies. However, these approaches often require numerous iterations before achieving a satisfactory result , and the number

of iterations tends to increase with the complexity of the profile [1]. As a result, the cost of profile development rises due to raw material expenditure, machine time, and labour costs [28].

In 2004, Michaeli and Klau [29] proposed a method that combines Finite Element Analysis (FEM) and a Flow Analysis Network to automate die design optimization. In 2006, Sienz *et al.* [30] utilized an isothermal FEM solver to model and optimize slit dies.

In 2016, Sai and Pradeep [33] investigated the effects of various features on flow balance in polymer profile extrusion dies, considering mandrel features and imposing temperature at the walls. In 2019, Lebaal [34] published a study on the optimization of slit extrusion dies, highlighting that even a 5% variation in temperature had significant effects on flow distribution. In this case, temperature was imposed at the flow channel walls.

Nobrega *et al.* [37–39] further contributed to this field by implementing and verifying a 3D non-isothermal code specifically for the calibrator stage. Their work in this area was conducted in the years 2004, 2008, and 2016.

Currently, there are two main commercial numerical modelling software programs used to simulate polymer extrusion die flow channels: PolyXtrue [40] and Polyflow [41]. Analysing the literature that utilizes these software programs, we can observe that in case studies involving mandrel/torpedo features often the temperature in imposed on those surfaces [42–46].

1.4. MOTIVATION

Polymer extrusion, as highlighted in the State of the Art (Section 1.3) and the preceding process description, is a crucial industrial process. However, the current modelling approach employed to simulate flow within the extruder and extrusion die relies on several simplifications. Regarding temperature control, computational modelling assumes that the temperature set for the tool prevails at the flow channel wall [49]. In practical applications, though, temperature control is achieved using thermocouples that measure the temperature within the metallic tool. Furthermore, heaters are typically situated on the tool surface, and their operation is guided by thermocouple readings [50].

Moreover, despite an extensive literature review, no prior validation of the approach commonly adopted in published research, which treats torpedo surfaces as insulated, was performed.

With the advancement of Computer-Aided Engineering (CAE) tools, the opportunity arises to reduce these simplifications and evaluate the errors they introduce. This progress might enable researchers and industrial professionals to expedite the development and assessment of novel temperature control techniques.

1.5. OBJECTIVES

The primary goal of this study was to establish and validate a novel modelling approach for polymer extrusion dies that closely resembles real-world practices. To ensure the widespread applicability of these advancements, numerical implementations were conducted using an open-source computational library.

To achieve this overarching goal, several intermediate objectives must be addressed: (i) a comprehensive exploration of the OpenFOAM framework to determine the most suitable approach. (ii) the development of essential codes for both the calculation tool and boundary conditions. (iii) specification and implementation of the most appropriate methods for geometry and computational mesh generation. (iv) the assessment of these developments through multiple case studies.

Given its widespread use, adaptability, and the author's experience with OpenFOAM®, it has been selected as the computational tool for this research.

1.6. Dissertation Structure

This dissertation is organized as follows. In the present chapter to help to better understand the polymer extrusion process and the works performed a polymer extrusion process description the state of the art, motivation, and the objectives of this work are presented. Chapter 2 covers the implementation of a new boundary condition and a new solver in the OpenFOAM® computational library. The subsequent chapter, Chapter 3, addresses the assessment work done for the new boundary condition and solver. Chapter 4 presents an industrial case analysis where traditional, and the proposed modelling approach are compared. Finally, in Chapter 5 the main conclusions and proposals for future work are provided.

2. NUMERICAL DEVELOPMENTS

This section, details the actions taken to develop a multi-region solver and a heater control boundary condition. The primary objective is to enhance the modeling process by modelling the flow channel and extrusion die.

2.1. MULTI REGION SOLVER

A typical single region case solver in openFoam structure (see Figure 10) is composed by a *case folder*, that comprises a time directory, where the files with boundary conditions and initial conditions for the fields are set, a *constant* folder with a *(x)Properties* file (e.g. transportProperties, thermophysicalProperties, ...) and a *polyMesh* folder, *being the first* usually named *transportProperties* where the physical properties of the material are defined and the last where the mesh data is stored. The last main folder is the system folder where the files *controlDict*, *fvSchemes*, *fvSolutions* and *blockMeshDict* are located. The *controlDict* file is the file that mainly serves to control the timestep, the initial and end time of the simulation, the *fvSchemes* is where the discretization schemes are defined, the fvSolution is the file where the linear solvers are selected and its operation specified, the last file is the *blockMeshDict* although not mandatory, it provides the mesh generation dictionary that generates the mesh in OpenFOAM.

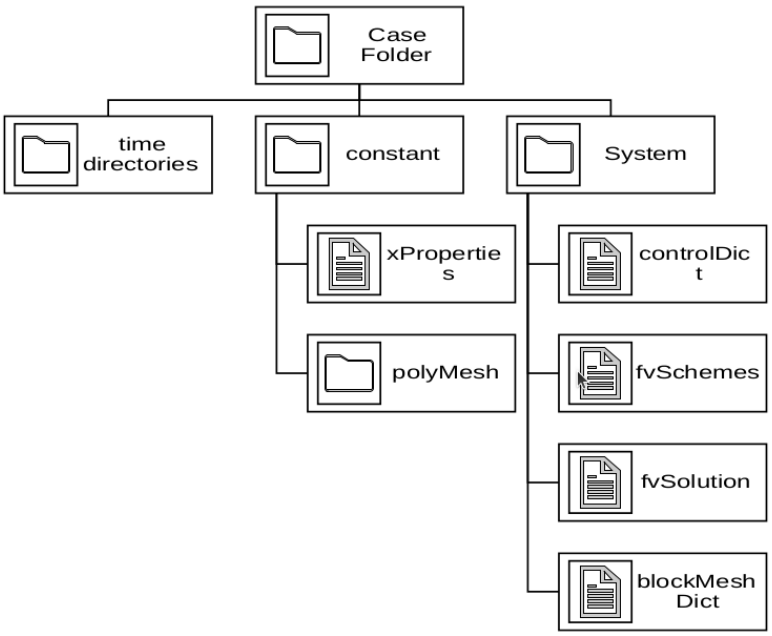


Figure 10: Typical openFoam case structure (adapted from [53])

The implemented multi region solver approach case structure is presented at Figure 11, and mainly adds for each main folder (time directories, constant and system) a new folder for each new region, namely fluid and solid. At folder *0* in the *solid* folder the file *T* is where the initial temperature and boundary conditions are defined for the solid region. In the folder fluid, a file *p*, *T* and *U* are required to define the pressure , temperature and velocity fields boundary and initial conditions for the fluid region. In the *constant* folder, we find three folders, *polyMesh* , *solid* and *fluid*. Inside solid we find *transportProperties* file were the thermal properties (kappa - thermal conductivity and DT - thermal diffusivitty) for the solid region are defined. The fluid folder contains two files, *transportProperties* and *turbulenceProperties*, which are the files to define, respectively, the rheological and the turbulence modelling parameters.

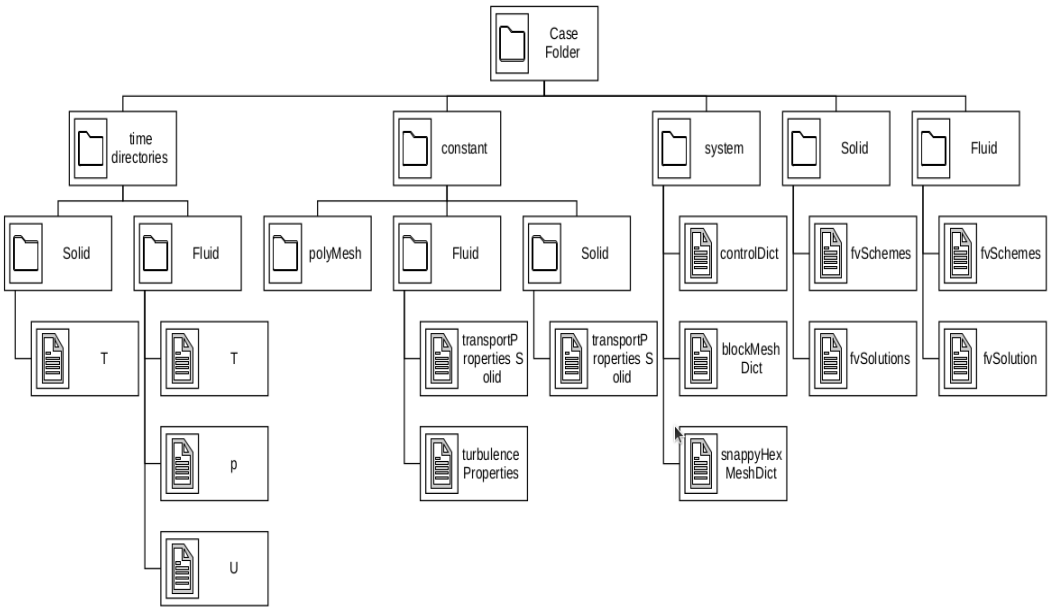


Figure 11: Typical chtMultiRegionPimpleFOAM case struture

2.1.1. METHODOLOGY

To customize the solver according to the requirements, the following steps were performed:

- pimpleFoam source code folder (see Figure 12) was copied, and folder name renamed as chtMultiRegionPimpleFoam. The solver pimpleFoam was selected as it is a large time-step transient solver for incompressible flows.

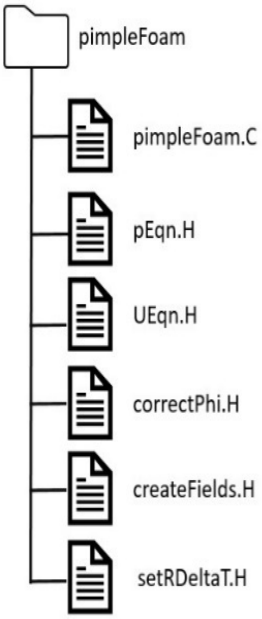


Figure 12: pimpleFoam source code folder contents

- Contains the files initContinuityErrs.H, continuityErrs.H, createMesh.H, courantNo.H, Tfluid.H and Tsolid.H resulting in the structure illustrated in Figure 13.

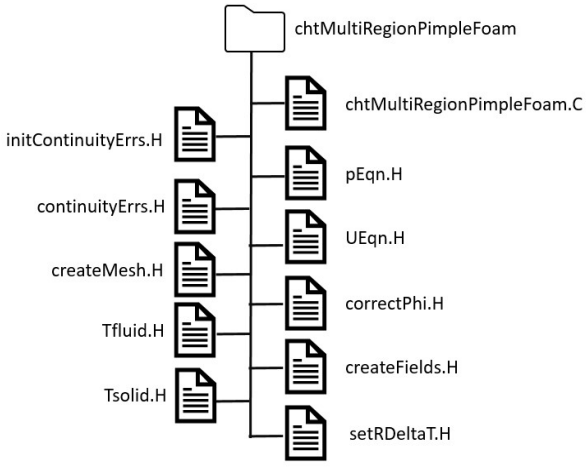


Figure 13: chtMultiRegionPimpleFoam source code folder contents

- the file initContinuityErrs.H from the folder $FOAM_SRC/finiteVolume/cfdTools/incompressible/ was copied and changed to initialise the cumulative continuity error for the fluid region only, as presented in Appendix 1,
- the file continuityErrs.H from the folder $FOAM_SRC/finiteVolume/cfdTools/incompressible/ was copied and changed to calculate and print the continuity errors for the fluid region only, as shown in Appendix 2,
- the "Tfluid.C" and "Tsolid.C" files were created to incorporate the fluid (Appendix 3) and solid (Appendix 4) equations to be solved, respectively,
- To create the meshes for the 2 domains (fluid, and solid) the file createMesh.H was created (Appendix 5),
- the file createFields.H was updated to account for the needed dictionaries and fields. The resulting code is presented at Appendix 6.
- The main solver file was updated to include the new files in the main code (Appendix 7).

2.1.2. Solver Implementation

As stated at Section 2.1.1 the multi region solver implementation was based on pimpleFoam [54], that is a single incompressible phase unsteady solver that uses PIMPLE method to address pressure – velocity coupling, by combining PISO and SIMPLE methods.

The equations solved are the momentum conservation Eq.(1)

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla\cdot(\rho\mathbf{u}\mathbf{u}) = -\nabla p + \nabla\cdot\tau, \quad (1)$$

and the mass conservation Eq.(2)

$$\nabla\cdot\mathbf{u} = 0, \quad (2)$$

To account for the temperature effect, the energy conservation equations for both domains should be considered, both for the fluid (Eq.3) and for the solid (Eq.(4),

$$\frac{\partial T}{\partial t} + \nabla\cdot(\mathbf{u}T) - \nabla\cdot(\alpha\nabla T) = \frac{1}{c_p}\underline{\underline{\tau}}:\nabla\mathbf{u}, \quad (3)$$

$$\frac{\partial T}{\partial t} - \nabla\cdot(\alpha\nabla T) = 0, \quad (4)$$

In the equations presented above, T represents the temperature, ρ the fluid density, $\mathbf{u}$ the velocity vector, p the pressure, c_p the specific heat, and α the thermal difusivity. In Eq.(3), the last term on the right-hand side ($\boldsymbol{\tau}:\nabla\boldsymbol{u}$) accounts for the viscous dissipation contribution, from which $\underline{\underline{\tau}}$ is the deviatoric stress tensor that is calculated as presented in Eq.(5),

$$\tau = 2\eta(\dot{\gamma}, T)\boldsymbol{D}, \tag{5}$$

η is shear viscosity that depends both on temperature (T) and shear rate ($\dot{\gamma}$) and $\mathbf{D}$ is the rate of strain tensor that is given by Eq.(6),

$$\mathbf{D} = \frac{1}{2}([\nabla \mathbf{u}] + [\nabla \mathbf{u}]^T) \tag{6}$$

where $\nabla \mathbf{u}$, is the velocity gradient tensor.

To account for the shear rate and temperature effects on the flow in result of shear viscosity changes the Bird-Carreau model was used coupled with the Arrhenius law, as given by Eq.(7) and Eq.(8):

$$\eta(\dot{\gamma}, T) = a_T \eta_\infty + \frac{a_T(\eta_0 - \eta_\infty)}{\left(1 + (a_T \lambda \dot{\gamma})^2\right)^{\frac{1-n}{2}}} \tag{7}$$

$$a_T = \exp\left(\frac{E}{R}\left(\frac{1}{T} - \frac{1}{T_0}\right)\right). \tag{8}$$

As illustrated in the flowchart presented in Figure 14, the developed transient solver is mainly composed by a loop (PIMPLE loop) that solves the momentum balance equations, mass conservation, and energy conservation equations for the fluid and solid for a pre-defined number of iterations at each time step.

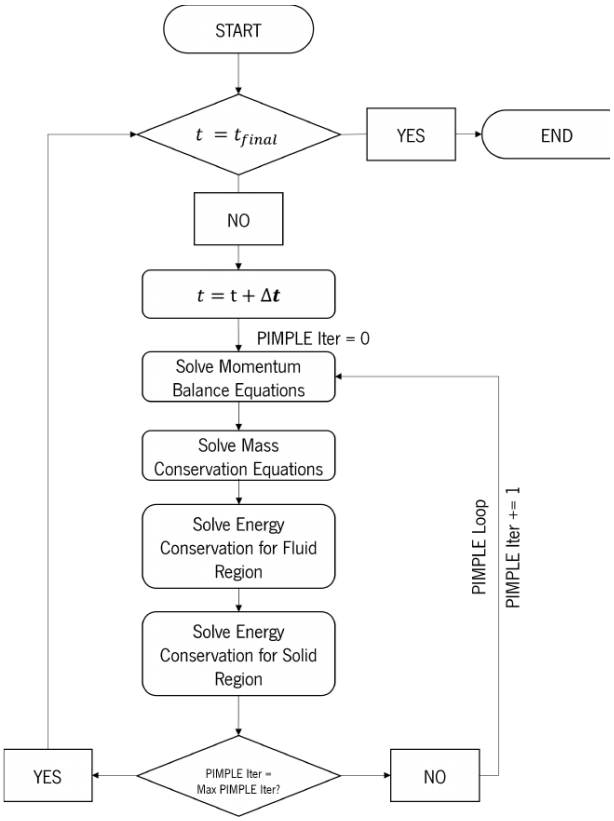


Figure 14: chtMultiRegionPimpleFoam solution flowchart

Since the solver selected already incorporates the essential capability for solving the equations of governing momentum balance and mass conservation, the focus of this presentation will be directed

towards detailing the implementation of the fluid and solid energy conservation equations. Additionally, the tasks performed to create two distinct mesh regions, will present a fundamental requirement for addressing the specific challenges posed by the envisaged problem. The implementation of the energy conservation for the fluid (Eq.(3)) presented in Figure 15 accounts for several terms:

- ddt(Tf) which stands for the $\frac{\partial T}{\partial t}$.
- div(phi,Tf) representing $\nabla \cdot (uT)$
- laplacian(DTf,Tf) which corresponds to the $\nabla \cdot (\alpha \nabla T)$
- the last term represents the viscous dissipation, expressed as (1/c_)*(tau && gradU), which represents $\frac{1}{c_p} \tau : \nabla u$
- the terms preceded by "fvm::" indicate that they are evaluated implicitly
- the terms preceded by "fvc::" indicate that they are evaluated explicitly

```
volTensorField gradU = fvc::grad(U);
volScalarField nu = laminarTransport.nu();
volTensorField tau = nu*(gradU + gradU.T());
fvScalarMatrix fluidTEqn
(
fvm::ddt(Tf)
+ fvm::div(phi,Tf)
- fvm::laplacian(DTf,Tf)
- (1/c_)*(tau && gradU)
);
```

Figure 15: openFoam implementation of fluid energy conservation equation

The implementation of the energy conservation for solids (Eq.(4)) was coded as shown in Figure 16 where

- ddt(Ts) represents the $\frac{\partial T}{\partial t}$.
- laplacian(DTs,Ts) corresponds to $\nabla \cdot (\alpha \nabla T)$

```
fvScalarMatrix solidTEqn
(
fvm::ddt(Ts)
- fvm::laplacian(DTs,Ts)
);
```

Figure 16: openFoam implementation of solid energy conservation equation

The implementation of the multi-region capacity in pimpleFoam solver was achieved by creating and including the file 'createMesh.C' in the main solver file. The code for this implementation is presented in Figure 17 and Figure 18 which are the part of the code where the mesh reading is defined.

```
28  Info << "Create fluid mesh";
29
30  fvMesh fluidMesh
31  (
32  IOobject
33  (
34  "fluid",
35  runTime.timeName(),
36  runTime,
37  IOobject::MUST_READ
38  )
39  );
```

Figure 17: Fluid mesh creation implementation on createMesh.C

```
41  Info << "Create solid mesh";
42  fvMesh solidMesh
43  (
44  IOobject
45  (
46  "solid",
47  runTime.timeName(),
48  runTime,
49  IOobject::MUST_READ
50  )
51  );
```

Figure 18: Solid mesh creation implementation on createMesh.C

To add the needed temperature fields (fluid, Tf, and solid, Ts) to the multi region solver, they must be declared in the file "createFields.C" as shown at Figure 19, linking each variable, Tf and Ts to its mesh domain, "*fluidMesh"* and "*solidMesh",* respectivily.

```
Info<< "Reading field Tfluid\n" << endl;
volScalarField Tf
(
IOobject
(
"T",
runTime.timeName(),
fluidMesh,
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
fluidMesh
);

Info<< "Reading field Tsolid\n" << endl;
volScalarField Ts
(
IOobject
(
"T",
runTime.timeName(),
solidMesh,
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
solidMesh
);
```

Figure 19: Temperature field declaration on createFields.C

The coupling between the two regions was performed by using the already implemented boundary condition named *compressible::turbulentTemperatureRadCoupledMixed.*

2.2. HEATER CONTROL BOUNDARY CONDITION

2.2.1. METHODOLOGY

To create the new boundary condition that will perform the control of the heaters, the following steps were performed:

- A boundary condition similar to the one we need to implement was selected. The boundary condition selected was the externalWallHeatFluxTemperature. This boundary condition already have the option to apply a power to a wall instead of applying temperature.
- The selected boundary code was copied and renamed externalWallHeatFluxTemperaturePID and the files were renamed accordingly
- The files externalWallHeatFluxTemperaturePID.C and externalWallHeatFluxTemperaturePID.H were modified to implement the PID control of the heat flux.

2.2.2. IMPLEMENTATION

The boundary condition code implements a control loop that mimics the polymer extrusion heaters control loop. The input this boundary condition requires is the power density of the heater, the target temperature, the PID control parameters(proportional, integral and derivative gains), the natural convection coefficient between Air and the Heater, the ambient temperature, the die material thermal conductivity and the sensor patch name. As presented in the Figure 20, the control loop starts by reading the temperature from the sensor patch name. This task is performed as shown in Eq.(9),

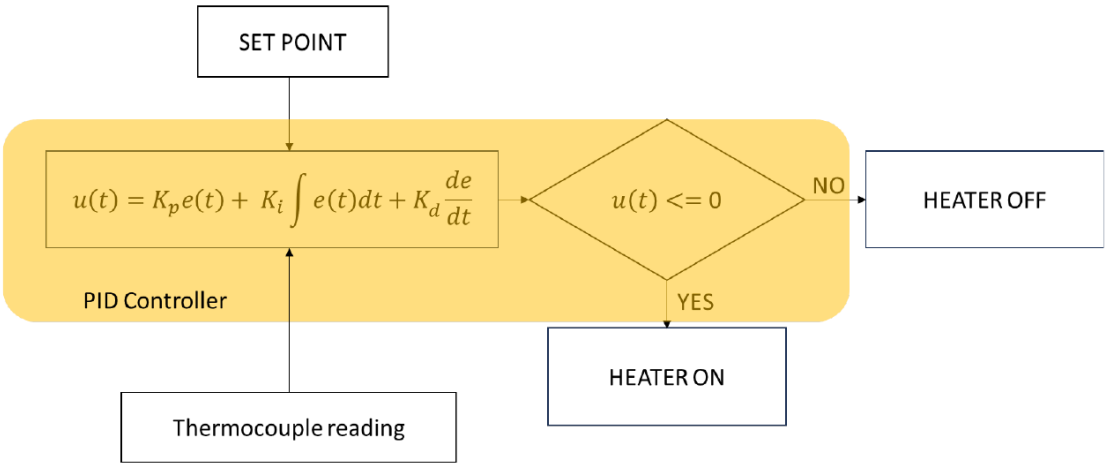


Figure 20: Temperature control flowchart

$$T_{avg} = \frac{\sum_{i=1}^{n} (T_i \cdot A_i)}{\sum_{i=1}^{n} A_i}, \quad (9)$$

where:

- T_{avg} is the area-averaged temperature.
- T_i represents the temperature value on each individual element or cell within the patch.
- A_i represents the area of each individual element or cell within the patch.
- n is the total number of faces within the patch.

This boundary condition was implemented with source code as shown in Figure 21, where T.boundaryField()[sensorPatchID] is the temperature field in the sensor patch and mesh.Sf().boundaryField()[sensorPatchID] is the face area normal vector.

```
380sensorPatchT=mag(gSum(T.boundaryField()
[sensorPatchID]*mesh.Sf().boundaryField()[sensorPatchID]));
381  // get boundary area
382  const scalar sensorArea =
mag(gSum(mesh.Sf().boundaryField()[sensorPatchID]));
383  // get Tave_
384  scalar Tave_ = sensorPatchT / sensorArea;
```

Figure 21: area-averaged temperature implementation

The implementation of the PID control algorithm was based in Eq.(10),

$$u(t) = K_p e(t) + K_i \int e(t)\,dt + K_d \frac{de}{dt}, \quad (10)$$

where

$$e = T_{probe} - T_{obj}, \quad (11)$$

- K_p is the proportional gain
- K_i is the integral gain and
- K_d is the derivative gain.

The implemented code presented in Figure 22 represents the implementation of Eq.(10) and Eq.(11) in OpenFOAM.

```
386  error_ = Tave_ - Tobj_;
387  errorIntegral_ = oldErrorIntegral_ + error_;
388  scalar errorDifferential = -(oldError_ - error_) / deltaT;
389  scalar PIDfunction =P_*error_+I_*errorIntegral_+D_*errorDifferential;
```

Figure 22: PID function implementation

where:

- `line 386 represents` Eq.(10)
- `P_ is` K_p
- `I_ is` K_i
- `D_ is` K_d

Finally, after implementing the PID equation, a conditional operator was added to the boundary condition. If $u(t) < 0$ energy should be supplied by the heater, thus a fixed gradient boundary condition is applied to the temperature field to provide the power supplied by the heater. Conversely, if $u(t) \geq 0$ a Robin boundary condition [56] is applied, as shown in Figure 23 representing the heat loss by natural convection, Eq.(12) and Eq.(13). This implementation in OpenFOAM is described in Figure 24.

$$\begin{cases} \begin{cases} \phi f = 0 \\ \phi ref = 0 \\ \nabla \phi ref = \frac{q+q_r}{k} \end{cases} & : \ u(t) < 0 \\ \begin{cases} \phi f = \frac{T_\infty \times h + q_r}{h} \\ \phi ref = 0 \\ \nabla \phi ref = \frac{k}{k+|d|} \end{cases} & : \ u(t) \geq 0 \end{cases} \tag{12}$$

$$T_{face} = \phi f \times \phi ref + (1 - \phi f) \tag{13}$$

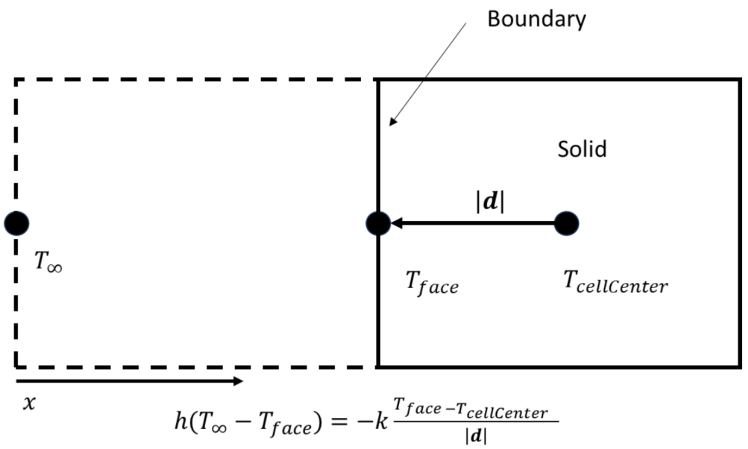


Figure 23: Representation of the Robin boundary condition

```
396  if (PIDfunction < 0)
397  {
...
417  refGrad() = (heatFlux + qr)/kappa(Tp);
418  refValue() = 0;
419  valueFraction() = 0;
...
434  }
435  else
436  {
...
480  refGrad() = 0;
481  forAll(Tp, i)
482  {
483  refValue()[i] = (hpTa[i] + qr[i])/hp[i];
484  valueFraction()[i] = hp[i]/(hp[i] + kappaDeltaCoeffs[i]);
485  }
...
491  mixedFvPatchScalarField::updateCoeffs();
...
501  }
```

Figure 24: Conditional response and Robin boundary condition implementation

3. CODE ASSESSMENT

This section, describes the task undertaken to validate the developed codes, specifically through the description of several representative 2D simulations of the extrusion process.

3.1. RHEOLOGY MODEL

For the rheology model, a polycarbonate material was chosen and the shear viscosity/shear rate curve resultant from the Bird-Carreau coupled with Arrhenius law is presented at Figure 25.

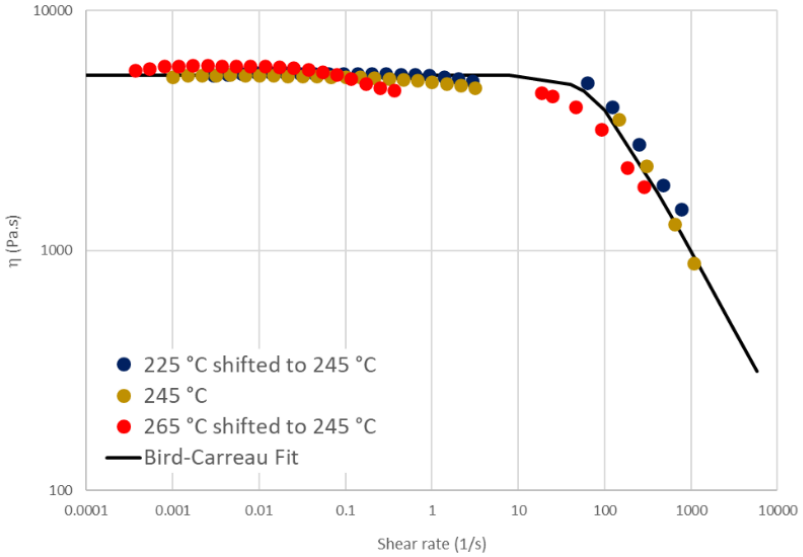


Figure 25: Polycarbonate rheology data

To perform the simulation, the polymer material properties used are presented in Table 1 and the die material properties are presented in Table 2.

Table 1: Polymer properties

Property	Symbol	Value	Units
viscosity at zero shear rate	η_0	5382	Pa.s
viscosity at infinite shear rate	η_∞	0	Pa.s
relaxation time	λ	0.0013	s
power-law index	n	0.35	
activation energy divided by the universal gas constant	$\frac{E}{R}$	13951.59	K
reference temperature	T_0	518.15	K
thermal diffusivity	α	1.458e-7	m^2/s
specific heat capacity	C_p	1200	J/(kg. K)
thermal conductivity	k	0.21	W/(m.K)

Table 2: Die material properties

Property	Symbol	Value	Units
thermal diffusivity	α	3.33e-6;	m^2/s
thermal conductivity	k	16	W/(m.K)

3.2. 2D CASE STUDIES

To assess the code implementations simplified 2D cases were tested with the aim of confirming if the code and the boundary conditions were behaving as expected.

For that a 2D geometry representative of an extrusion die cross section was build, as illustrated in Figure 26 where the Heater boundary represents the heating elements of a extrusion die. The wall represents the extrusion die surfaces exposed to air and the thermocouple patch the surface where the thermocouple touchs the extrusion die material.

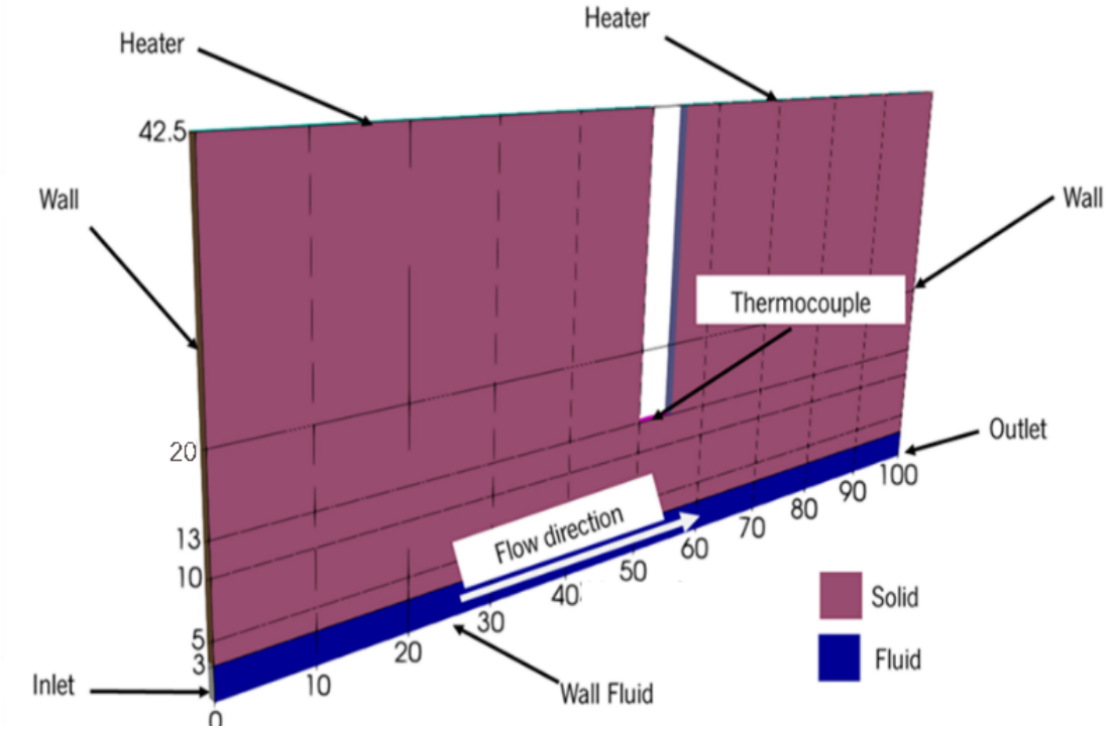


Figure 26: 2D Case study base geometry and boundary patches

The boundary conditions used are presented in Table 3 being the Heater controlled by new Heater control boundary condition "externalWallHeatFluxTemperaturePID".

Table 3: 2D case study boundary conditions

Patch	Pressure	Velocity	Temperature
Inlet	Null Gradient	Fixed Value (0.25 m/min)	Fixed Value (235 °C)
Heater	N/A	N/A	ExternalWallHeatFluxTemperaturePID (see Table 4 for the parameters)
Thermocouple	N/A	N/A	Null Gradient
Wall	N/A	N/A	Null Gradient
Wall Fluid	Null Gradient	No Slip	Null Gradient
Interface	Null Gradient	No Slip	compressible::turbulentTemperatureRadCoupled Mixed
Outlet	Fixed Value (0 MPa)	Null Gradient	Null Gradient

Table 4:2D case study heater temperature boundary conditon parameters

Property	Symbol	Value	Units
Proportional gain	K_p	1	-
Integral gain	K_i	0	-
Derivative gain	K_d	0	-
Room Temperature	T_∞	25	°C
Target Temperature	$T_{thermocouple}$	245	°C
Conductivity	k	16	W/(m.K)
Heat transfer coefficient	h	25	W/(m^{2}K)
Heat flux	q	35000	W/m^2

The meshing was performed using blockMesh and eight blocks were generated as shown in Figure 27, For the purpose of studying mesh independence, three meshes were generated (see Table 5 for mesh size details) as shown in Figure 28, 29 and 30 where is clear that the first mesh is the least refined, while the last mesh is the most refined.

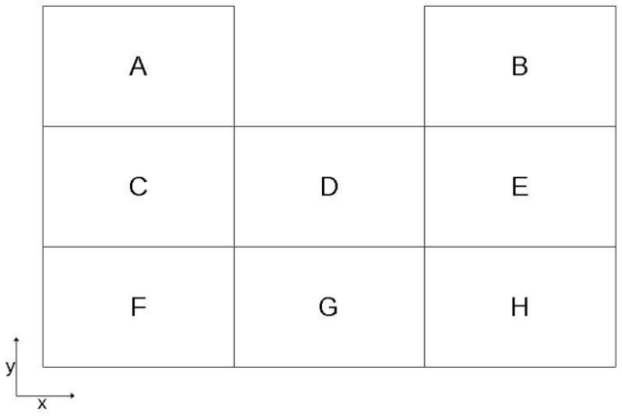


Figure 27: 2D case study mesh block division

Table 5: 2D case study mesh size by block and total

Block	Mesh 1	Mesh 2	Mesh 3
A	14750	59000	236000
B	14750	59000	236000
C	5000	20000	80000
D	500	2000	24000
E	5000	20000	80000
F	1500	6000	24000
G	150	600	800
H	1500	6000	24000
Total Cells	43150	172600	690400

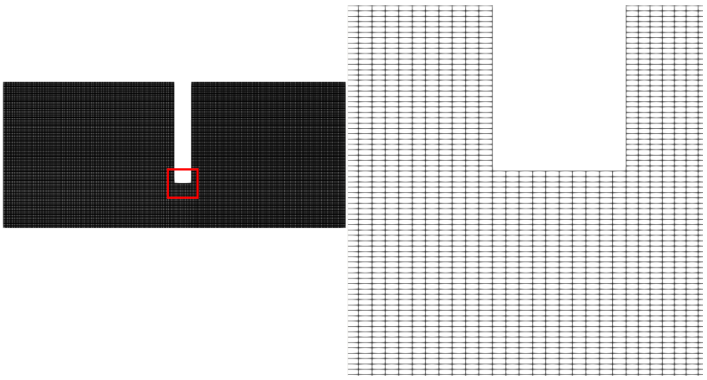


Figure 28: 2D case study mesh 1

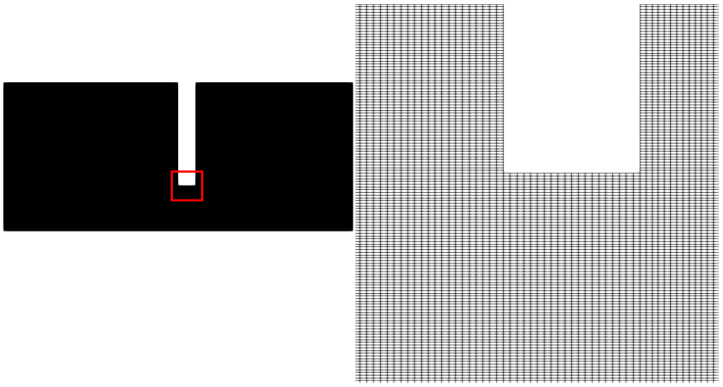


Figure 29: 2D case study mesh 2

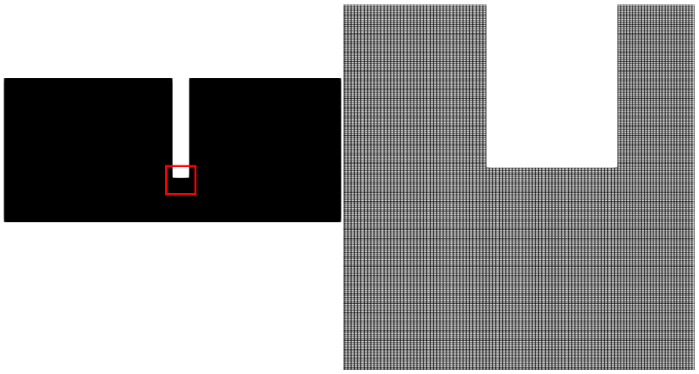


Figure 30: 2D case study mesh 3

3.3. RESULTS AND DISCUSSION

To assess grid independence, the simulations were run for 100 seconds, and the values for pressure at inlet (Pinlet) , average temperature at outlet (Toutlet), average temperature at walls (Twalls), average temperature at sensor (Tsensor) and run time were normalized with the reference value being the reference field for the most refined mesh (See Figure 31). Based on the obtained results, Mesh 2 was selected for subsquent studies.

First, the PID function was assessed to determine if it was working as expected. The behaviour of the temperature at the heater was observed, and it matched the expected pattern, as illustrated in Figure 32. When the temperature at the thermocouple (sensor) was below the target temperature (245°C), the PID function would turn on the heater until the thermocouple reached the desired temperature. The observed temperature peak in the sensor and the time lag between the control location and the measurement location is a common characteristic in control systems . This phenomenon occurs due to several factors: process inertia causes a delay in reaching the desired temperature when increasing heater power, leading to a temporary rise in the sensor's temperature, the spatial separation between the heater and the sensor introduces a time lag as the heat propagates through the system. (The heater's electrical resistance typically responds quickly to the increased power, contributing to an initial temperature spike, while the sensor, such as a thermocouple, may have a slower response due to heat propagation delays). Consequently, the temperature peak observed in the sensor is a natural consequence of the system's dynamics, and proper PID controller tuning is essential to minimize such temperature spikes and ensure precise and stable temperature control.

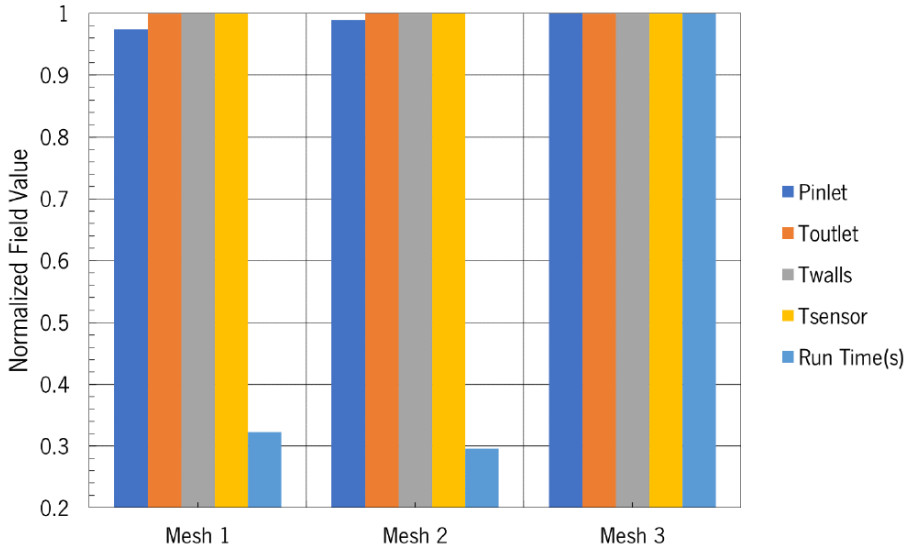


Figure 31: 2D case study mesh refinement study

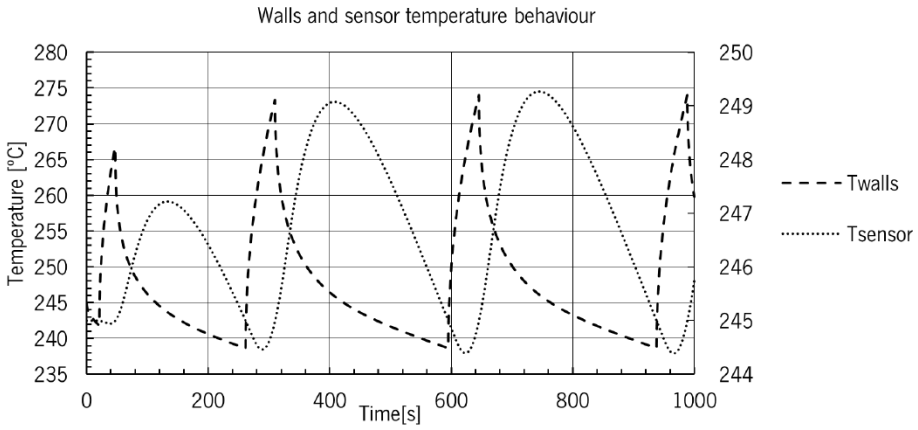


Figure 32: 2D case study walls and sensor temperature behaviour

After assessing the behavior of the heater boundary condition, the effect of the distance from the thermocouple to the flow channel was studied. For this purpose, the distance was varied by +/-10% when compared with the Base Case. The results obtained are plotted in Figure 33. The curve labeled "-10%" represents the farthest location from the flow channel to the thermocouple, while the "+10%" curve represents the closest location. Based on these results, it can be concluded that for this system, when the thermocouple is closer to the flow channel, there are higher temperature fluctuations at the outlet, which likely result from a greater distance between the heater and the thermocouple. This greater distance leads to longer propagations times and thus, higher temperatures at the heater, which propagate through the system, causing increased temperature fluctuations at the outlet. Another effect noted in this test case is the increase in temperature fluctuation amplitude for each cycle. This is probably a result of the system losing more heat through the outer walls than through the fluid outlet, resulting in an increase in the outlet's average temperature every time the heater turns on. However, for longer periods the steadystate oscilation is expected.

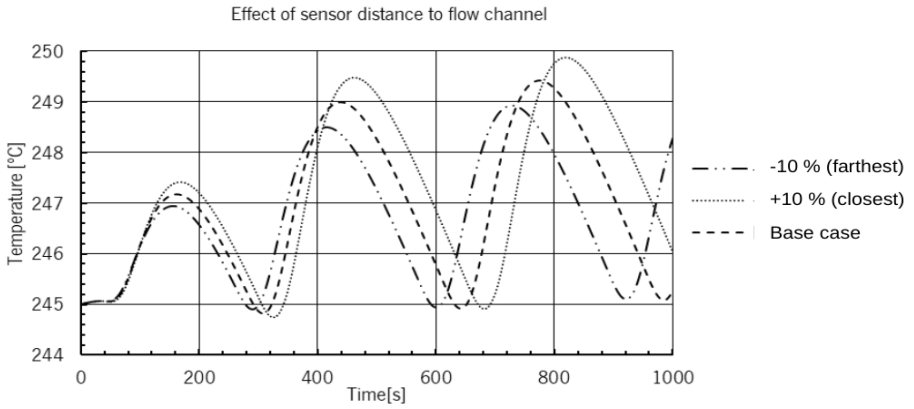


Figure 33: 2D case study effect of sensor distance to flow channel

In the final study of the 2D case, the objective was to assess the impact of varying the distance between the thermocouple and the flow channel inlet, with distances adjusted by +/-10%. The results

are presented in Figure 34, where the "-10%" curve represents the farthest distance from the outlet to the thermocouple, while the "+10%" curve represents the closest distance. Based on these findings, it can be concluded that, for this system, the placement of the thermocouple close to the inlet has a negligible effect on temperature fluctuations. However, when the thermocouple is positioned farther away from the inlet, larger temperature fluctuations are observed. This is likely due to its proximity to the outlet, causing the sensor to detect temperature changes more rapidly, resulting in longer heater operation times and subsequently higher outlet temperatures. Similar to previous observations, an increasing amplitude of temperature fluctuations was noted, indicating that the system loses more heat through its outer walls than through the fluid outlet, consequently leading to a rise in the outlet's average temperature each time the heater activates. However, for longer periods the steadystate oscilation is expected.

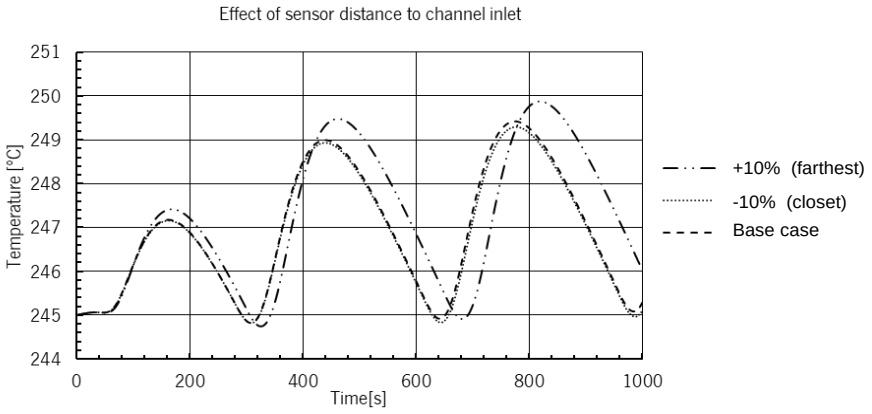


Figure 34: 2D case study effect of sensor distance to flow channel inlet

The results presented in the 2D case study have effectively assessed and validated the approach implemented in OpenFOAM. Furthermore, from a qualitative perspective, the solver has accurately resolved the fields of temperature, pressure, and velocity, as illustrated in Figure 35, where the temperature field exhibits continuity and higher values near the heater. In Figure 36, the pressure field is also continuous, with higher pressure at the inlet compared to the outlet, demonstrating variation along the channel. Finally, Figure 37 displays a velocity field with a parabolic profile, further confirming the correctness of the solver's performance.

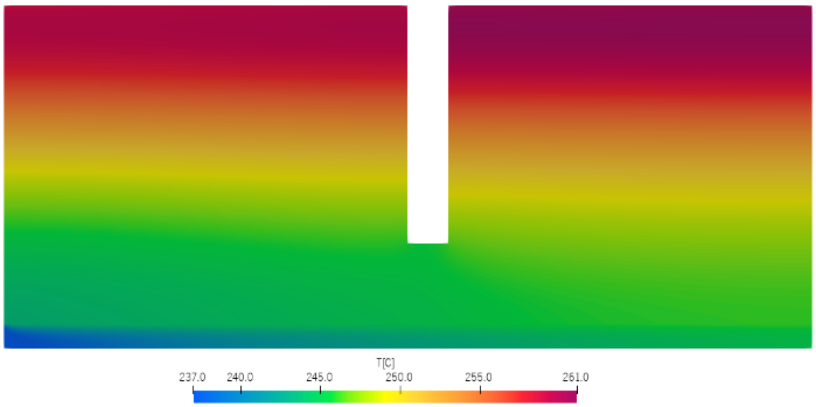


Figure 35: 2D case study temperature field (t=1000s)

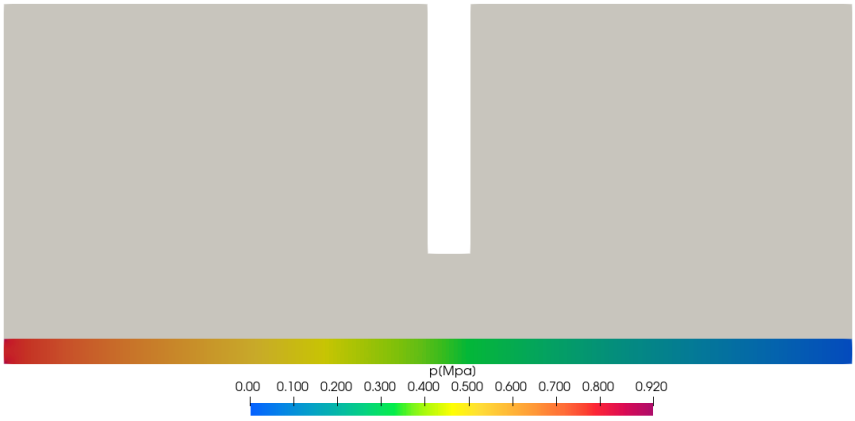


Figure 36: 2D case study pressure field (t=1000s)

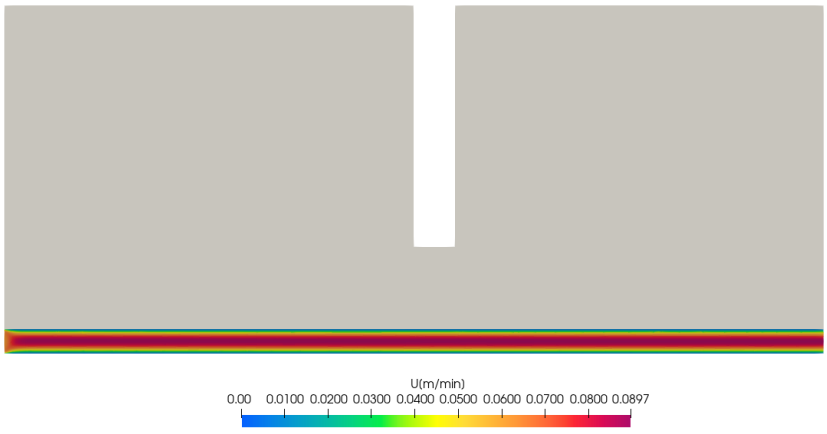


Figure 37: 2D case study velocity field(t=1000s)

4. INDUSTRIAL CASE STUDY

In this section, three distinct approaches for modeling the extrusion process are presented, and a comparative analysis is conducted to assess the influence of the simplifications currently employed in the modeling of the extrusion process.

4.1. PRESENTATION

The industrial case study focuses on the production of a LED encasing profile, as depicted in Figure 38. The extrusion die used in this study comprises two different regions with independent heaters, the adapter and the die land, as illustrated in Figure 39 . Additionally, each heater has its own thermocouple, as shown in Figure 40. To accurately represent the flow channel and die geometry, the CAD software was utilized to create the corresponding models.

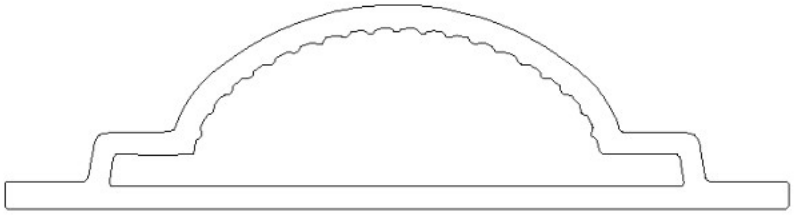


Figure 38: Industrial case study profile cross-section

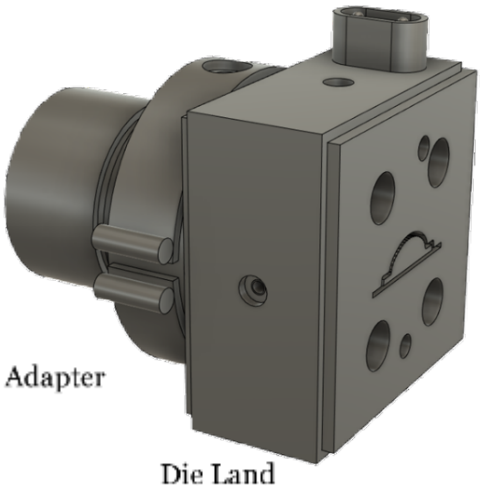


Figure 39: Industrial case study extrusion die zones

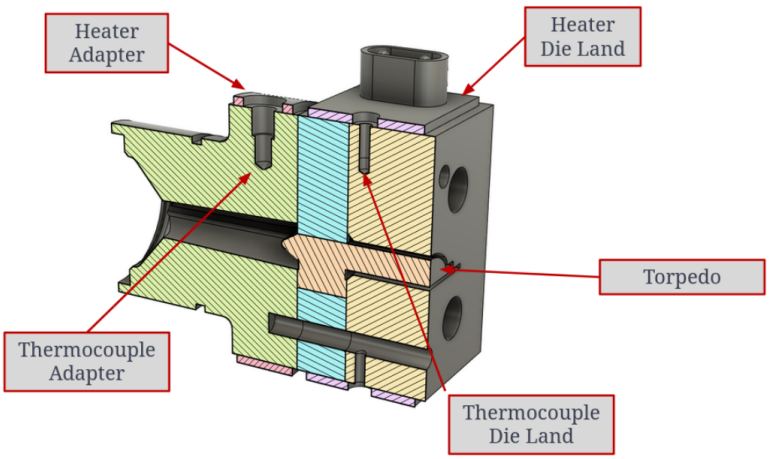


Figure 40: Industrial case study die heating control elements

To quantify the die performance, first the outlet section was divided into Elemental Sections (ESi) and Intersection Sections (ISi) (see Figure 41).

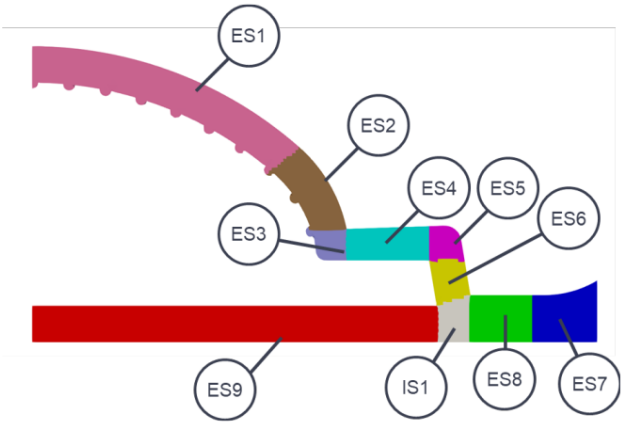


Figure 41: Industrial case study outlet cross section division

Then for each section the flow rate (Q_i) is computed, which is used to calculate the individual section objective function ($F_{obj,i}$) as presented at Eq.(14),

$$F_{obj,i} = \frac{\frac{Q_i}{Q_{\text{target}}} - 1}{max\left(\frac{Q_i}{Q_{\text{target}}}, 1\right)}, \tag{14}$$

where Q_{target} is the objective flow rate for individual section with is computed as shown in Eq.(15)

$$Q_{\text{target}} = U_{target} \times A_{total} \times \frac{A_i}{A_{total}}, \tag{15}$$

U_{target} is the target velocity, A_i is the section cross-section area and A_{total} is the outlet cross-section area.

With the $F_{obj,i}$ the global objective function F_{obj} is computed by area weight average summing the absolute value of all ES and IS, $F_{obj,i}$, as given in Eq.(16)

$$F_{obj} = \frac{\sum_{ES+IS} \left\| F_{obj,i} \right\| A_{\text{target},i}}{A_{\text{target},tot}} \tag{16}$$

4.1.1. Geometries and Boundary conditions

The industrial case study comprises two geometries and three sets of boundary conditions that will be presented below. Initially were assessed the Conventional and the Multi Region approach. The conventional approach only considers the flow channel, while the herein proposed Multi-region approach includes additionally the die in the simulation. It is worth noting that in both approaches a symmetry plane was used to reduce the computational size of the problem. After the conventional and multi region studies, a third study named mixed was prepared using the knowledge acquired by the multi region approach to tune the boundary condition used at the conventional approach. All cases are described in the following subsections.

4.1.2. Conventional approach

To simulate the process using the conventional approach, only the flow channel was considered and it was divided into several sections, as shown in Figure 42. This division of the CAD geometry is necessary for the subsequent application of different boundary conditions, as presented in Table 6.

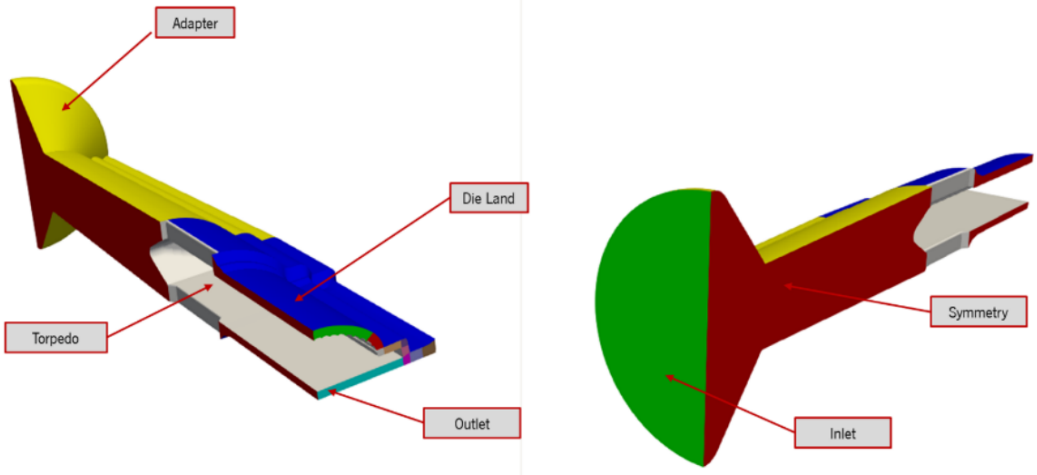


Figure 42: Conventional approach geometry and boundary patches

Table 6: Industrial case study conventional approach boundary conditions

Patch	Pressure	Velocity	Temperature
Inlet	Null Gradient	Fixed Value (0.282 m/min)	Fixed Value (245 °C)
Adapter	Null Gradient	No Slip	Fixed Value (228 °C)
Die Land	Null Gradient	No Slip	Fixed Value (220 °C)
Torpedo	Null Gradient	No Slip	Null Gradient
Symmetry	Symmetry	Symmetry	Symmetry
Outlet	Fixed Value (0 MPa)	Null Gradient	Null Gradient

4.1.3. MULTI REGION APPROACH

The new approach proposed to simulate the flow in the extrusion die requires dividing both the die and flow channel surface into various patches, as shown in Figure 43.

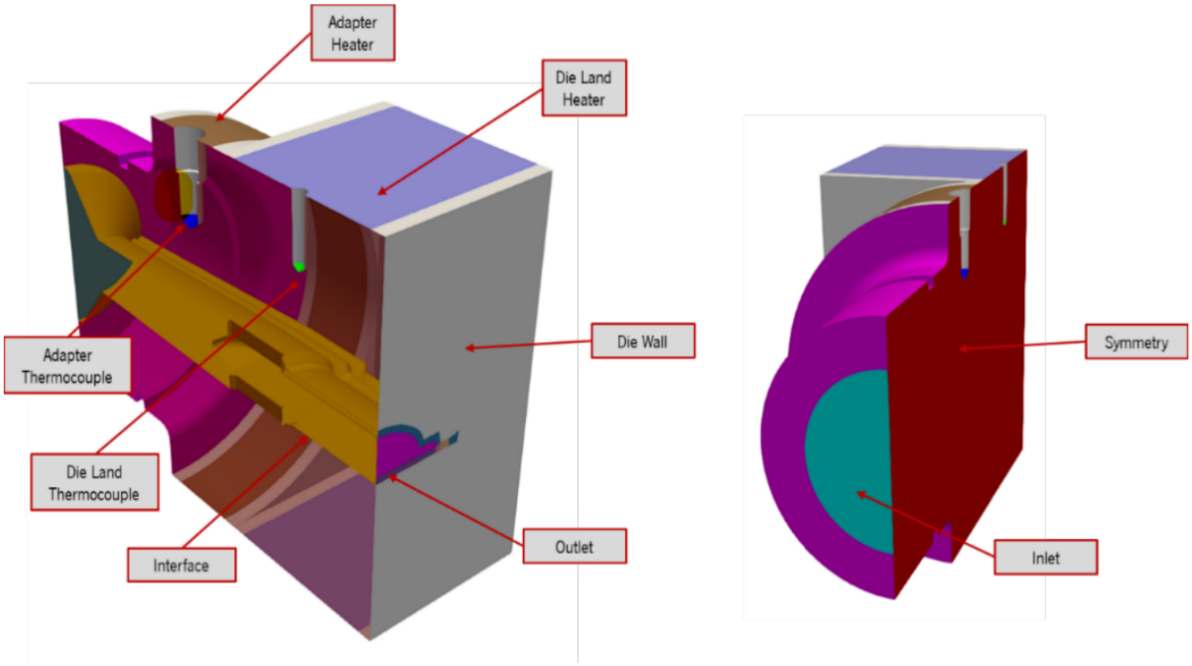


Figure 43: Industrial case study multi region approach geometry

The applied boundary conditions are presented in Table 7 and are the compressible::turbulentTemperatureRadCoupledMixed and *externalWallHeatFluxTemperaturePID* already discussed in the Section 2.1.

Table 7: Industrial case study multi region approach boundary conditions

Patch	Pressure	Velocity	Temperature
Inlet	Null Gradient	Fixed Value (0.282 m/min)	Fixed Value (245 °C)
Adapter Heater	N/A	N/A	ExternalWallHeatFluxTemperaturePID (see Table Table 8)
Die Land Heater	N/A	N/A	ExternalWallHeatFluxTemperaturePID (see Table Table 9)
Adapter Thermocouple	N/A	N/A	Null Gradient
Die Land Thermocouple	N/A	N/A	Null Gradient
Die Wall	N/A	N/A	Natural Convection Convective Heat Transfer Coefficient : 25 W/(m^{2}K)
Adapter Wall	N/A	N/A	Null Gradient
Interface	Null Gradient	No Slip	compressible::turbulentTemperatureRadCoupledMixed
Symmetry	Symmetry	Symmetry	Symmetry
Outlet	Fixed Value (0 MPa)	Null Gradient	Null Gradient

Table 8: Industrial case study, multi region approach adapter heater boundary condition parameters

Property	Symbol	Value	Units
Proportional gain	K_p	1	-
Integral gain	K_i	0	-
Derivative gain	K_d	0	-
Room Temperature	T_∞	25	°C
Target Temperature	$T_{thermocouple}$	220	°C
Thermal Conductivity	k	16	W/(m.K)
Heat transfer coefficient	h	25	W/(m^{2}K)
Heat flux	q	35000	W/m^2

Table 9: Industrial case study, multi region approach die land heater boundary condition parameters

Property	Symbol	Value	Units
Proportional gain	K_p	1	-
Integral gain	K_i	0	-
Derivative gain	K_d	0	-
Room Temperature	T_∞	25	°C
Target Temperature	$T_{thermocouple}$	230	°C
Conductivity	k	16	W/(m.K)
Heat transfer coefficient	h	25	W/(m^{2}K)
Heat flux	q	35000	W/m^2

4.1.4. MIXED APPROACH

The mixed approach was carried out after completing the simulations of the multi-region approach. Since the temperature of the flow channel wall was significantly higher than that imposed in the conventional approach, it was decided to adopt the same setup as used in the conventional approach. However, and contrary to the conventional approach applied before, the temperature boundary conditions were then updated to correspond to the values obtained for the multi-region approach. The new boundary conditions are presented in Table 10.

Table 10: Industrial case study mixed approach boundary conditions

Patch	Pressure	Velocity	Temperature
Inlet	Null Gradient	Fixed Value (0.282 m/min)	Fixed Value (245 °C)
Adapter	Null Gradient	No Slip	Fixed Value (232 °C)
Die Land	Null Gradient	No Slip	Fixed Value (230 °C)
Torpedo	Null Gradient	No Slip	Null Gradient
Symmetry	Symmetry	Symmetry	Symmetry
Outlet	Fixed Value (0 MPa)	Null Gradient	Null Gradient

4.2. Results and Discussion

4.2.1. Mesh sensitivity analysis

Before conducting the numerical studies, a mesh sensitivity study was performed to determine the required level of refinement. The mesh generation process was carried out using snappyHexMesh [55] for all case study approaches. The meshes generated for the conventional approach are presented in Figures 44,45 and 46 From these meshes, Mesh M2 was selected based on its low error, as demonstrated in Table 11.

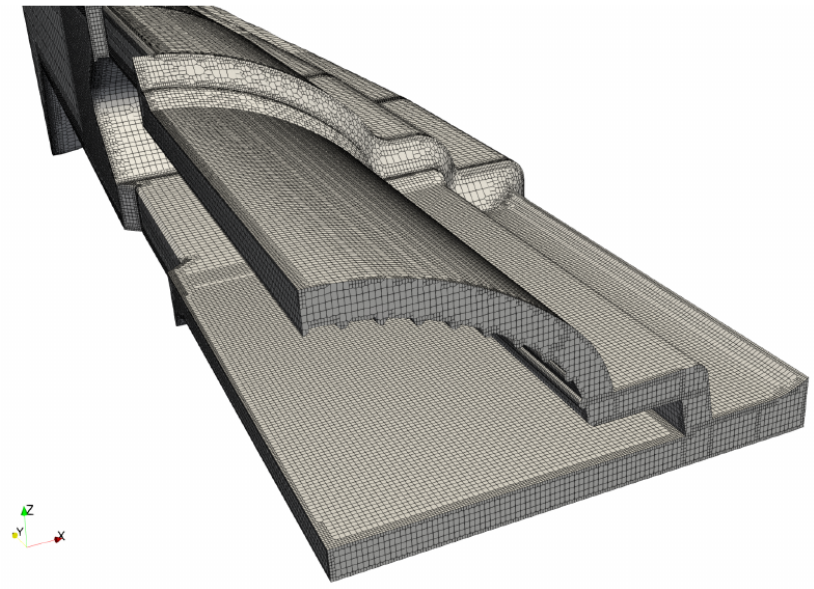


Figure 44: Industrial case study mesh 1

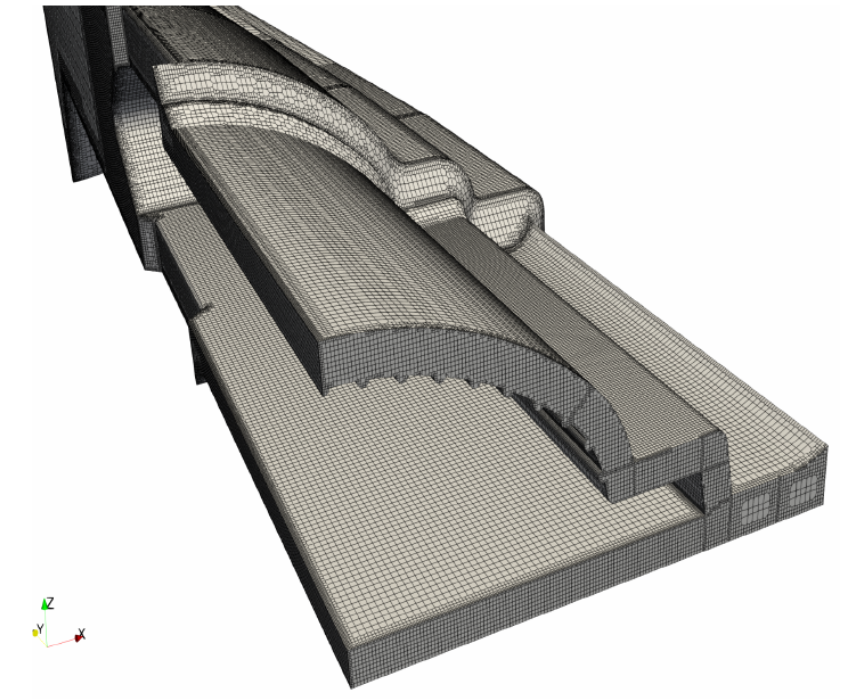


Figure 45: Industrial case study mesh 2

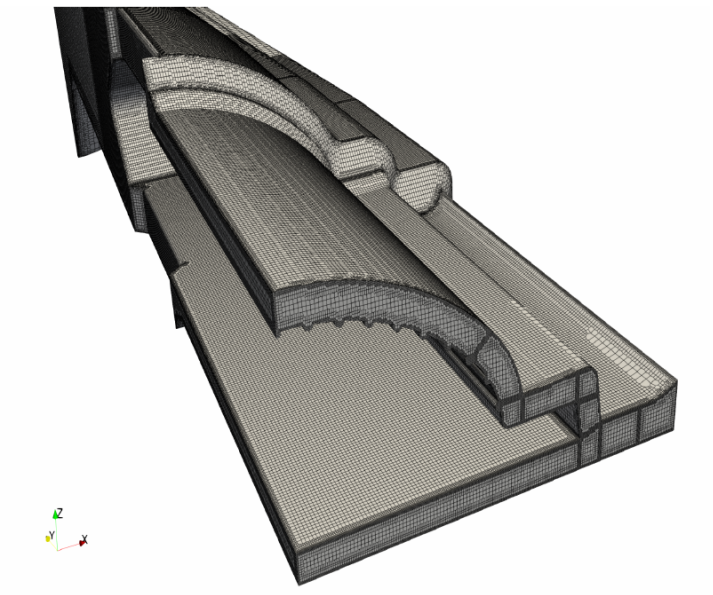


Figure 46: Industrial case study mesh 3

Table 11: Industrial case study conventional approach errors in function of cell number

	N° Cells	P_0/P_{0_M3}	F_{obj}/F_{obj_M3}
M1	339866	1.38%	4.30%
M2	714931	1.06%	2.23%
M3	1657684	0.00%	0.00%

Due to the complexity of the multi-region approach, it was only possible to generate two grids using snappyHexMesh. Higher refinement levels generated meshes with surfaces allocated to the wrong domain and that invalidated the mesh. This issue should be further investigated in the future.

The generated meshes for the multi-region approach are displayed in Figures 47 and 48. Despite the limited refinement, the quantities of interest did not show significant changes, as indicated in Table 12 . As a result of these findings, Mesh 2 was selected to proceed with the studies.

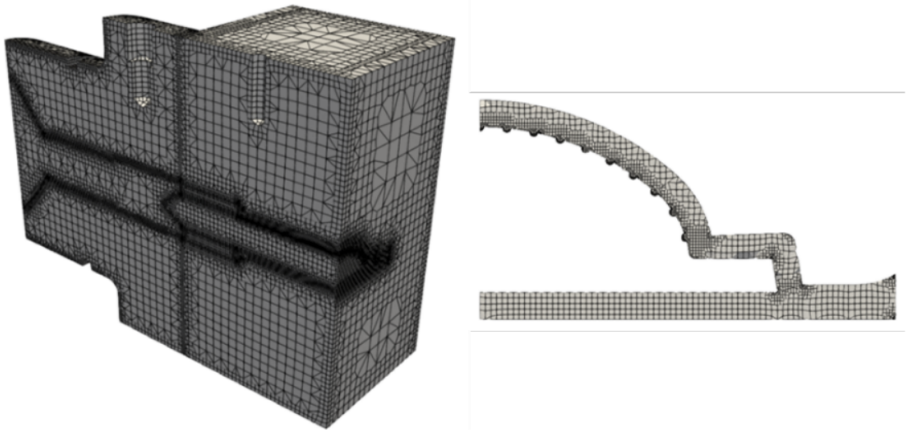


Figure 47: Industrial case study multi region mesh 1

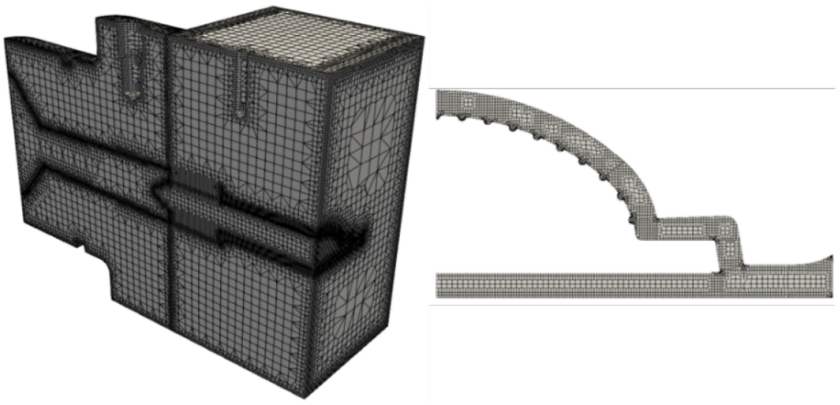


Figure 48: Industrial case study multi region mesh 2

Table 12: Industrial case study multi region approach errors in function of cell number

			Time=1000 s	
	N° Cells	Cell Siz[(mm]	P_0 [MPa]	T Average ES1 [C]
M1	2234129	0.716133799	17.16	253.03
M2	4923331	0.550314017	16.77	253.21

4.2.2. Results and Discussion

In Figures 49 and 50, it is possible to evaluate the impact of the heater operation and the corresponding temperature at the thermocouple (Probe die). It is evident that system inertia causes the heater and the thermocouple to exhibit different temperature profiles. While the heater experiences sharp rises and losses in temperature, the thermocouple displays an almost parabolic profile.

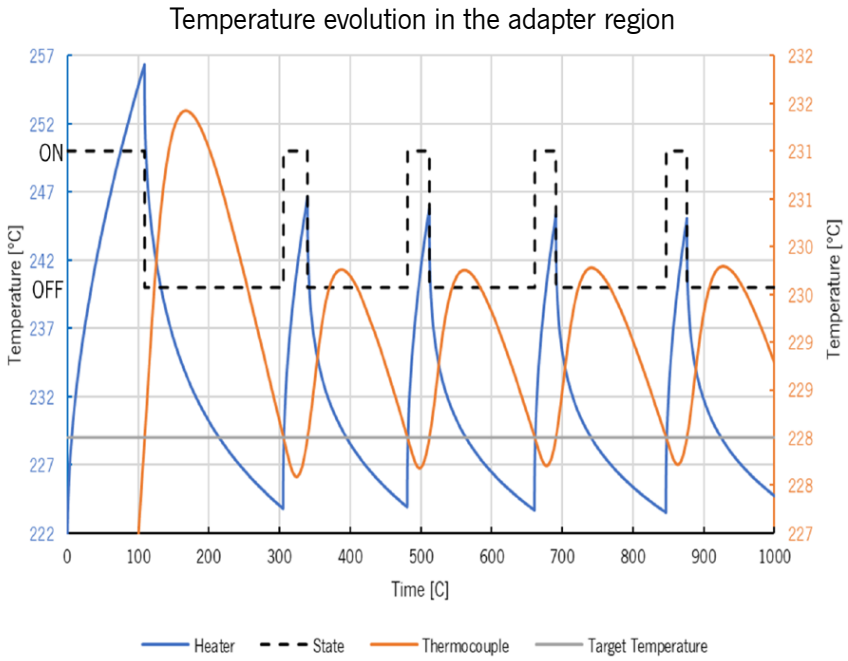


Figure 49: Industrial case study multi region temperature evolution in the adapter

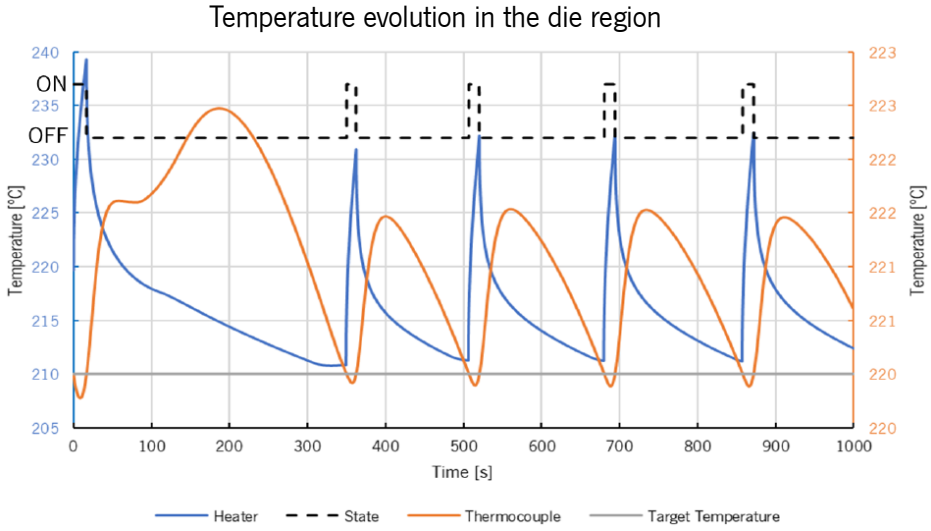


Figure 50: Industrial case study multi region temperature evolution in the die

At t=300s of simulation, the objective function stabilized, and the impact of the heater operation became almost negligible, as demonstrated in Figure 51. This indicates that, in this particular case, steady-state conditions were nearly attained.

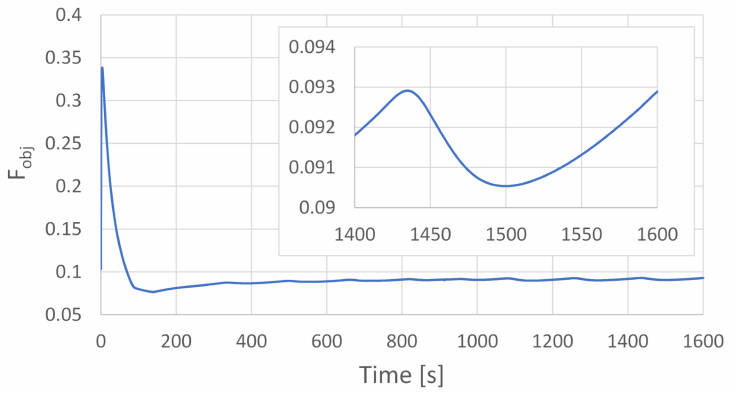


Figure 51: Industrial case study multi region objective function evolution

4.2.3. Comparison Multi-region, conventional and hybrid case studies

In the multi-region case, higher temperature were predicted at the outer wall of the flow channel, as shown in Figure 52. This likely occurs as a result of thermal inertia and temperature overshoot when the controller turns the heaters on and off. Comparing the temperature fields calculated from the different simulation approaches, it is also possible that the traditional approach that uses the torpedo as insulated predicted similar results to the multi-region approach.

Regarding the pressure field, the higher temperatures predicted in the multi-region approach led to a lower pressure drop, as illustrated in Figure 53. At the outlet, the predicted flow fields were similar in all approaches. However, the multi-region approach exhibited a higher velocity peak, as presented in Figure 54. This is also likely due to the higher temperatures that we can see at Figure 52 and Figure 55, which induce a decrease in viscosity and, consequently, higher velocity. This effect is also seen at Figure 56 that presents the objective function results which are very similar in all sections, except ES7 where was predicted more flow in mixed approach than in the multi-region and traditional approachs. Analysing the Figure 57, which presents the temperature field in that specific section, it becomes clear that the higher flow prediction is a consequence of the higher temperatures on this particular section.

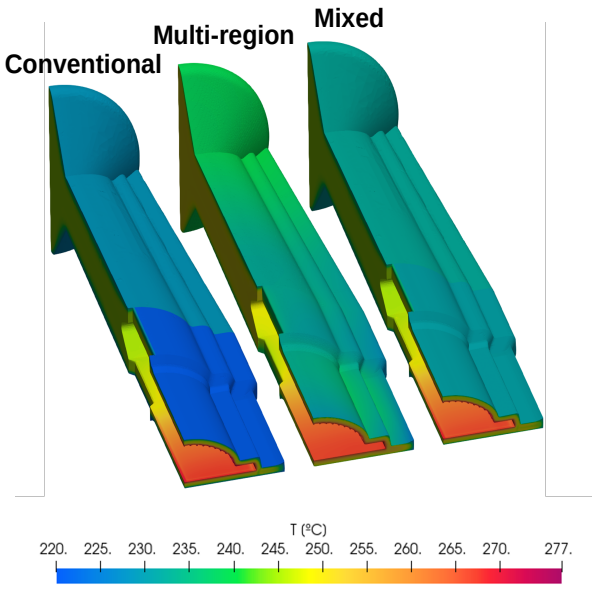


Figure 52: Industrial case study temperature field , comparison between conventional , multi-region and mixed approaches

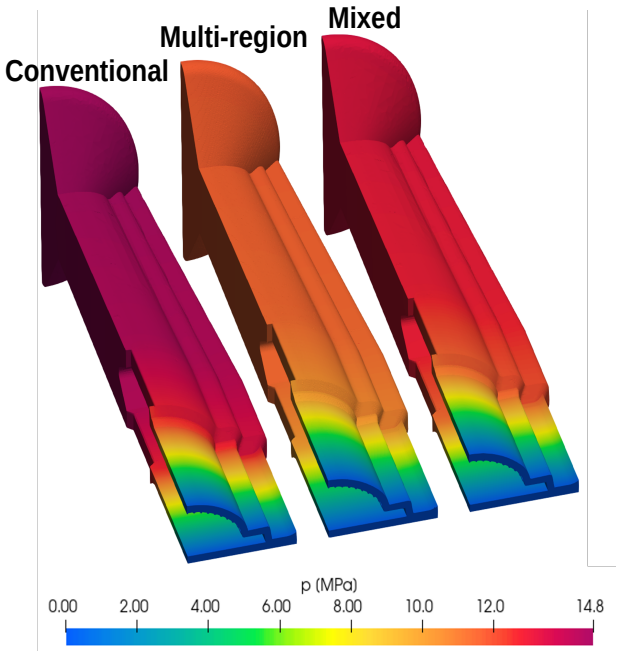


Figure 53: Industrial case study pressure field , comparison between conventional, multi-region and mixed approaches

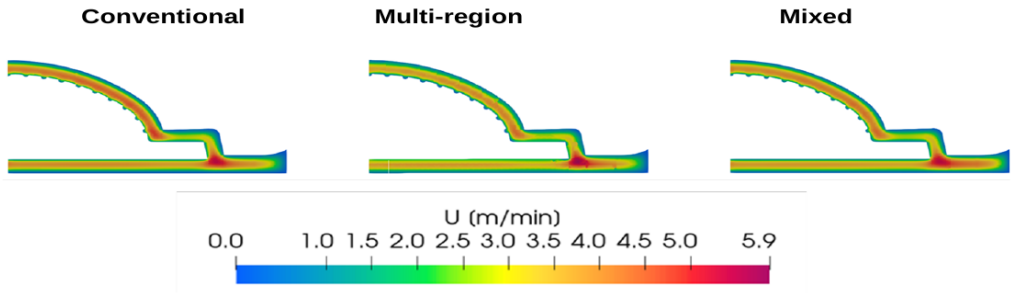


Figure 54: Industrial case study velocity field at outlet ,comparison between conventional , multi-region and mixed approaches

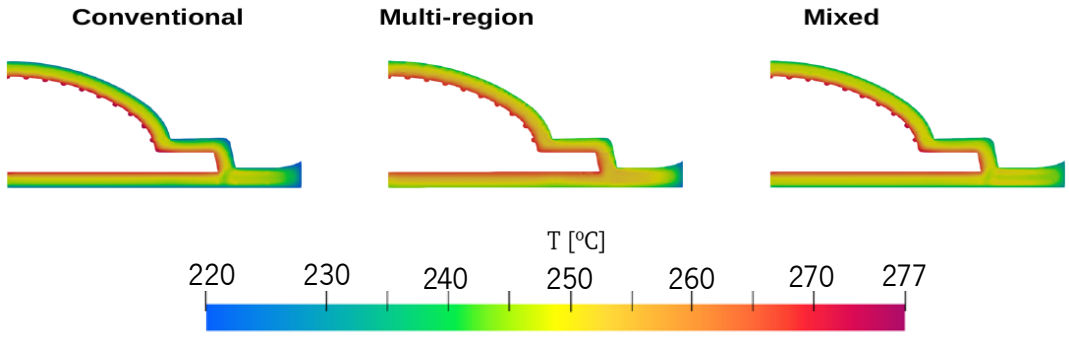


Figure 55: Industrial case study temperature field at the flow channel outlet,comparison between conventional , multi-region and mixed approaches

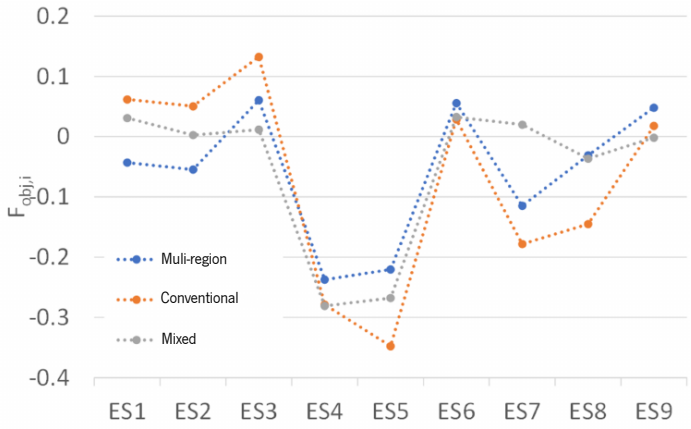


Figure 56: Industrial case study individual objective functions(Fobj,i) plot , comparison between conventional , multi-region and mixed approaches

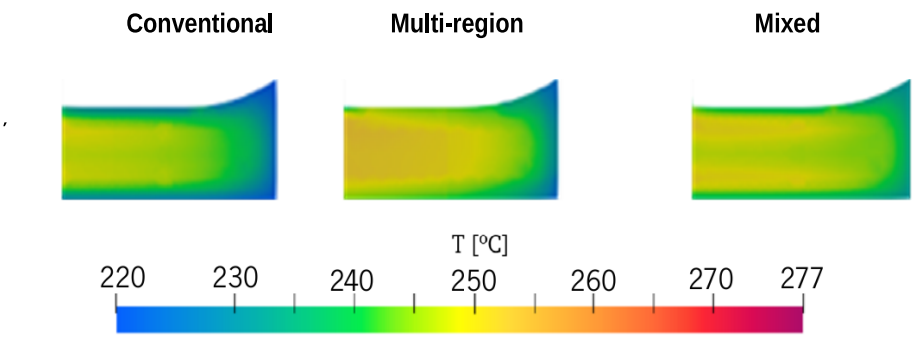


Figure 57: Industrial case study temperature at ES7, comparison between conventional, multi-region and mixed approaches

5. Conclusions and Future work

In this MSc project, a novel methodology for numerical modeling of the profile extrusion die transformation process was implemented and evaluated. This methodology aimed to model the process under more realistic temperature control conditions, in contrast to the simplifications made in the previous state-of-the-art approaches. We also aimed to assess how these changes affected the accuracy of the simulation predictions.

During the implementation, we developed a new transient, incompressible, non-isothermal, and multi-region solver that was implemented on the OpenFOAM computational library. Additionally, we created a new boundary condition to mimic the behavior of heaters controlled by a Proportional-Integral-Differential (PID) function, which takes measurements from a thermocouple located at a specific point within the tool.

This MSc project, assessed three polymer extrusion die modelling approaches, namelly the conventional approach where only the flow channel is modelled, the multi-region approach where both flow channel and extrusion die are modelled and a mixed approach that used the calculated temperatures in the solid-fluid interface from the multi-region approach as boundary condition for the flow channel.

Our findings demonstrate that the pressure drop calculated using the conventional approach was higher than with the multi-region approach. This difference resulted from temperature fields at the flow channel walls, which significantly deviated from the assumptions made in the conventional approaches. These deviations led to increased flow resistance due to lower temperatures. However, in the industrial case we studied, temperature variations had a reduced impact on the velocity field. Furthermore, the results obtained revealed that temperature fluctuations had a negligible effect on flow uniformity at the flow channel outlet once the process reached a steady state. Nevertheless, it is worth noting that in some specific locations, variations in wall-imposed temperatures can substantially modify local flow distributions.

As part of future work, applying the multi-region approach to a wider range of industrial cases and evaluating the effect of PID control parameters on process stability and flow distribution, is advised. This will allow to fully explore the potential of the multi region approach, including the optimization of PID parameters for enhanced control.

References

[1] Michaeli, W. (2003). Extrusion Dies for Plastics and Rubber. Carl Hanser Verlag GmbH & Co. KG. https://doi.org/10.3139/9783446401815

[2] Rauwendaal, C. (2014). Polymer Extrusion. Carl Hanser Verlag GmbH & Co. KG. https://doi.org/10.3139/9781569905395

[3] Industrial Quick Search (IQS®). (n.d.). Plastic Extrusion https://www.iqsdirectory.com/articles/plastic-extrusion.html.Acessed in: 1 jun. 2023.

[4] Abeykoon, C., Martin, P. J., Li, K., & Kelly, A. L. (2014). Dynamic modelling of die melt temperature profile in polymer extrusion: Effects of process settings, screw geometry and material. In Applied Mathematical Modelling (Vol. 38, Issue 4, pp. 1224–1236). Elsevier BV. https://doi.org/10.1016/j.apm.2013.08.004

[5] RESITEC. Catalogo Resistências Cartucho.(n.d.) https://www.resitec.pt/en/produtos/sondas-e-resistencias/resistencias-eletricas-para-diversos-fins/resistencias-de-cartucho. Acessed in: 1 jun. 2023.

[6] RESITEC. Resistências de Banda. (n.d.) https://www.resitec.pt/en/produtos/sondas-e-resistencias/resistencias-eletricas-para-diversos-fins/resistencias-de-banda. Acessed in: 1 jun. 2023.

[7] C. Abeykoon, K. Li, P. J. Martin, M. McAfee, and G. W. Irwin, "Extruder melt tempeature control with fuzzy logic," Proceedings of 18th IFAC World Congress, pp. 8577–8582, 2011.

[8] Chen, G., Xiong, Q., Morris, P. J., Paterson, E. G., Sergeev, A., & Wang, Y.-C. (2014). OpenFOAM for Computational Fluid Dynamics. In Notices of the American Mathematical Society (Vol. 61, Issue 4, p. 354). American Mathematical Society (AMS). https://doi.org/10.1090/noti1095

[9] Codes – CFD-Wiki, the free CFD reference. (n.d.) https://www.cfd-online.com/Wiki/Codes. Acessed in: 7 jul. 2023.

[10] Fuggetta, A. (2003). Open source software––an evaluation. In Journal of Systems and Software (Vol. 66, Issue 1, pp. 77–90). Elsevier BV. https://doi.org/10.1016/s0164-1212(02)00065-1

[11] K. Wittig, 'CalculiX USER'S MANUAL-CalculiX GraphiX, Version 2.20'. 2022. http://www.dhondt.de/cgx_2.20.pdf. Acessed in: 7 jul. 2023.

[12] P. Råback and M. Malinen, 'Overview of Elmer'. (n.d.) http://www.csc.fi/elmer. Acessed in: 7 jul. 2023.

[13] H. Jasak, 'OpenFOAM: Open source CFD in research and industry', International Journal of Naval Architecture and Ocean Engineering, vol. 1, no. 2, pp. 89–94, 2009, doi: https://doi.org/10.2478/IJNAOE-2013-0011.

[14] Z. Tadmor, 'Fundamentals of plasticating extrusion. I. A theoretical model for melting', Polym Eng Sci, vol. 6, no. 3, pp. 185–190, 1966, doi: https://doi.org/10.1002/pen.760060303.

[15] A. Altınkaynak, M. Gupta, M. A. Spalding, and S. L. Crabtree, 'Melting in a Single Screw Extruder: Experiments and 3D Finite Element Simulations', International Polymer Processing, vol. 26, no. 2, pp. 182–196, 2011.

[16] S. Bawiskar and J. L. White, 'Melting model for modular self-wiping co-rotating twin-screw extruders', Polym Eng Sci, vol. 38, no. 5, pp. 727–740, 1998, doi: https://doi.org/10.1002/pen.10238.

[17] K. Wilczynski and J. L. White, 'Melting model for intermeshing counter-rotating twin-screw extruders', Polym Eng Sci, vol. 43, no. 10, pp. 1715–1726, 2003, doi: https://doi.org/10.1002/pen.10145.

[18] J. Vlcek, G. N. Mailvaganam, J. Vlachopoulos, and J. Perdikoulias, 'Computer simulation and experiments of flow distribution in flat sheet dies', Advances in Polymer Technology, vol. 10, no. 4, pp. 309–322, Dec. 1990, doi: 10.1002/ADV.1990.060100407.

[19] J. M. Nóbrega, O. S. Carneiro, P. J. Oliveira, and F. T. Pinho, 'Part I: Automatic Design', International Polymer Processing, vol. 18, no. 3, pp. 298–306, 2003, doi: doi:10.3139/217.1745.

[20] N. D. Gonçalves, O. S. Carneiro, and J. M. Nóbrega, 'Design of complex profile extrusion dies through numerical modeling', J Nonnewton Fluid Mech, vol. 200, pp. 103–110, Oct. 2013, doi: 10.1016/j.jnnfm.2013.02.007.

[21] J. Vlachopoulos, 'Extrudate Swell in Polymers', Reviews on the deformation behavior of materials, vol. 3, pp. 219–248, Jan. 1981.

[22] D. Tang, F. H. Marchesini, L. Cardon, and D. R. D'hooge, 'State of the-Art for Extrudate Swell of Molten Polymers: From Fundamental Understanding at Molecular Scale toward Optimal Die Design at Final Product Scale', Macromol Mater Eng, vol. 305, no. 11, p. 2000340, Nov. 2020, doi: 10.1002/MAME.202000340.

[23] W. A. Gifford, 'Compensating for die swell in the design of profile dies', Polym Eng Sci, vol. 43, no. 10, pp. 1657–1665, Oct. 2003, doi: 10.1002/PEN.10139.

[24] Karadogan, Celalettin, 'Advanced methods in numerical modeling of extrusion processes', 2005, doi: 10.3929/ETHZ-A-004940061.

[25] N. D. F. Gonçalves, 'Computer-aided design of extrusion forming tools for complex geometry profiles', Universidade do Minho (Portugal), 2013.

[26] V. Hristov and J. Vlachopoulos, 'Thermoplastic silicone elastomer lubricant in extrusion of polypropylene wood flour composites', Advances in Polymer Technology, vol. 26, no. 2, pp. 100–108, Jun. 2007, doi: 10.1002/ADV.20090.

[27 Carneiro, O.S. and Nóbrega, J.M. (2012) Design of extrusion forming tools. Shawbury, Shrewsbury: Smithers Rapra Technology Ltd.

[28] Tadmor, Z. and Gogos, C.G. (2006) Principles of polymer processing. Hoboken: Wiley-Interscience.

[29] W. Michaeli and S. Kaul, 'Approach of an automatic extrusion die optimization', Journal of Polymer Engineering, vol. 24, no. 1-3 SPEC. ISS., pp. 123–136, May 2004, doi: 10.1515/POLYENG.2004.24.1-3.123.

[30] J. Sienz, S. J. Bates, and J. F. T. Pittman, 'Flow restrictor design for extrusion slit dies for a range of materials: Simulation and comparison of optimization techniques', Finite Elements in Analysis and Design, vol. 42, no. 5, 2006, doi: 10.1016/j.finel.2005.06.008.

[31] N. Lebaal, F. Schmidt, and S. Puissant, 'Design and optimization of three-dimensional extrusion dies, using constraint optimization algorithm', Finite Elements in Analysis and Design, vol. 45, no. 5, pp. 333–340, Apr. 2009, doi: 10.1016/J.FINEL.2008.10.008.

[32] O. Yilmaz, H. Gunes, and K. Kirkkopru, 'Optimization of a profile extrusion die for flow balance', Fibers and Polymers, vol. 15, no. 4, 2014, doi: 10.1007/s12221-014-0753-3.

[33] A. Sai and E. Pradeep, 'Design features and optimization of profile extrusion dies', 2016, doi:10.37099/mtu.dc.etdr/166

[34] N. Lebaal, 'Robust low-cost meta-modeling optimization algorithm based on meta-heuristic and knowledge databases approach: Application to polymer extrusion die design', Finite Elements in Analysis and Design, vol. 162, pp. 51–66, Sep. 2019, doi: 10.1016/J.FINEL.2019.05.004.

[35] P. Sheehy, P. A. Tanguy, and D. Blouin, 'A finite element model for complex profile calibration', Polym Eng Sci, vol. 34, no. 8, pp. 650–656, Apr. 1994, doi: https://doi.org/10.1002/pen.760340806.

[36] Fradette, L., Tanguy, P.A., Hurez, P. and Blouin, D. (1996), "On the determination of heat transfer coefficient between pvc and steel in vacuum extrusion calibrators", International Journal of Numerical Methods for Heat & Fluid Flow, Vol. 6 No. 1, pp. 3-12. https://doi.org/10.1108/EUM0000000004095

[37] O. S. Carneiro, J. M. Nóbrega, J. Covas, P. Oliveira, and F. Pinho, A study on the thermal performance of calibrators, vol. 1. 2004.

[38] Nóbrega, J. M., Carneiro, O. S., Gaspar-Cunha, A. and Gonçalves, N. D.. "Design of Calibrators for Profile Extrusion – Optimizing Multi-step Systems" International Polymer Processing, vol. 23, no. 3, 2008, pp. 331-338. https://doi.org/10.3139/217.2148

[39] F. Habla et al., 'Development and validation of a model for the temperature distribution in the extrusion calibration stage', Appl Therm Eng, vol. 100, Feb. 2016, doi: 10.1016/j.applthermaleng.2016.01.166.

[40] Dassault Systèmes, 'PolyXtrue | SOLIDWORKS'. (n.d.) https://www.solidworks.com/partner-product/polyxtrue (accessed Jun. 29, 2023).

[41] I. ANSYS, 'Ansys Polyflow | Plastic Extrusion Simulation Software'. (n.d.) https://www.ansys.com/products/fluids/ansys-polyflow (accessed Jun. 29, 2023).

[42] K. Ryckebosh and M. Gupta, 'Optimization of a profile coextrusion die using a three-dimensional flow simulation software', 2015.

[43] M. Gupta and K. Ryckebosch, 'Simulation of the Flow in a Bilayer PVC Window Profile Die with Gradually Changing Calibrator Profiles', Society of Plastics Engineers Annual Technical (ANTEC), vol. 68, 2021.

[44] Pepliń i, K., & Mozer, A. (2012). Comparison of bottle wall thickness distribution obtain in real manufacturing conditions and in ansys polyflow simulation environment. Journal of Polish CIMAC, 7, 231-235.

[45] Gupta, Mahesh. (2012). Effect of Polymer Viscosity on Post-Die Extrudate Shape Change in Coextruded Profiles. Annual Technical Conference - ANTEC, Conference Proceedings. 2.

[46] Zhang, G., Huang, X., Li, S. et al. Improved inverse design method for thin-wall hollow profiled polymer extrusion die based on FEM-CFD simulations. Int J Adv Manuf Technol 106, 2909–2919 (2020). https://doi.org/10.1007/s00170-019-04785-w

[47] Rajkumar, A., Ferrás, L., Fernandes, C., Carneiro, O., Becker, M. & Nóbrega, J. (2017). Design Guidelines to Balance the Flow Distribution in Complex Profile Extrusion Dies. International Polymer Processing, 32(1), 58-71. https://doi.org/10.3139/217.3272

[48] O. S. Carneiro, A. Rajkumar, L. L. Ferrás, C. Fernandes, A. Sacramento, and J. M. Nóbrega, 'Computer-aided die design: A new open-source methodology', in AIP Conference Proceedings, American Institute of Physics Inc., May 2017. doi: 10.1063/1.4982987.

[49] Rajkumar, 'A. Improved methodologies for the design of extrusion forming tools'. Universidade do Minho (Portugal), 2017.

[50] C. Abeykoon, P. J. Martin, A. L. Kelly, and E. C. Brown, 'A review and evaluation of melt temperature sensors for polymer extrusion', Sens Actuators A Phys, vol. 182, pp. 16–27, 2012, doi: 10.1016/j.sna.2012.04.026.

[51] J. Vera-Sorroche et al., 'Thermal optimisation of polymer extrusion using in-process monitoring techniques', Appl Therm Eng, vol. 53, no. 2, pp. 405–413, May 2013, doi: 10.1016/j.applthermaleng.2012.04.013.

[52] Tran-Cong, T., Phan-Thien, N. Three-dimensional study of extrusion processes by Boundary Element Method.. Rheol Acta 27, 639–648 (1988). https://doi.org/10.1007/BF01337460

[53] OPENCFD LTD. File structure of OpenFOAM cases. (n.d.) https://www.openfoam.com/documentation/user-guide/2-openfoam-cases/2.1-file-structure-of-openfoam-cases. Acessed in: 26 set. 2023.

[54} OPENCFD LTD. OpenFOAM: User Guide: pimpleFoam. (n.d.) <https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-incompressible-pimpleFoam.html>. Acessed in: 23 maio. 2023.

[55] OPENCFD LTD. OpenFOAM: User Guide: snappyHexMesh. (n.d.) <https://www.openfoam.com/documentation/guides/latest/doc/guide-meshing-snappyhexmesh.html>. Acessed in: 2 maio. 2023.

[56] SIMSCALE. What Are Boundary Conditions? Numerics Background. (n.d.) <https://www.simscale.com/docs/simwiki/numerics-background/what-are-boundary-conditions/>. Acessed in: 2 set. 2023.

APPENDIX 1 – INITCONTINUITYERRS.H CODE

```
/*---------------------------------------------------------------------------*\
========= |
\\ / F ield | OpenFOAM: The Open Source CFD Toolbox
\\ / O peration |
\\ / A nd | www.openfoam.com
\\/ M anipulation |
-------------------------------------------------------------------------------
Copyright (C) 2011 OpenFOAM Foundation
Copyright (C) 2019 OpenCFD Ltd.
-------------------------------------------------------------------------------
License
This file is part of OpenFOAM.

OpenFOAM is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
for more details.

You should have received a copy of the GNU General Public License
along with OpenFOAM. If not, see <http://www.gnu.org/licenses/>.

Global
cumulativeContErr

Description
Declare and initialise the cumulative continuity error.

\*---------------------------------------------------------------------------*/

#ifndef initContinuityErrs_H
#define initContinuityErrs_H

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

uniformDimensionedScalarField fluidcumulativeContErrIO
(
IOobject
(
"cumulativeContErr",
runTime.timeName(),
"uniform",
fluidMesh,
IOobject::READ_IF_PRESENT,
IOobject::AUTO_WRITE
),
dimensionedScalar(dimless, Zero)
);
scalar& cumulativeContErr = fluidcumulativeContErrIO.value();

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif

// ************************************************************************* //
```

APPENDIX 2 – CONTINUITYERRS.H CODE

```
/*---------------------------------------------------------------------------*\
========= |
\\ / F ield | OpenFOAM: The Open Source CFD Toolbox
\\ / O peration |
\\ / A nd | www.openfoam.com
\\/ M anipulation |
-------------------------------------------------------------------------------
Copyright (C) 2011 OpenFOAM Foundation
-------------------------------------------------------------------------------
License
This file is part of OpenFOAM.

OpenFOAM is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
for more details.

You should have received a copy of the GNU General Public License
along with OpenFOAM. If not, see <http://www.gnu.org/licenses/>.

Global
continuityErrs

Description
Calculates and prints the continuity errors.

\*---------------------------------------------------------------------------*/

{
volScalarField contErr(fvc::div(phi));

scalar sumLocalContErr = runTime.deltaTValue()*
mag(contErr)().weightedAverage(fluidMesh.V()).value();

scalar globalContErr = runTime.deltaTValue()*
contErr.weightedAverage(fluidMesh.V()).value();
cumulativeContErr += globalContErr;

Info<< "time step continuity errors : sum local = " << sumLocalContErr
<< ", global = " << globalContErr
<< ", cumulative = " << cumulativeContErr
<< endl;
}

// ************************************************************************* //
```

APPENDIX 3 – TFLUID.H CODE

```
volTensorField gradU = fvc::grad(U);
volScalarField nu = laminarTransport.nu();
volTensorField tau = nu*(gradU + gradU.T());
fvScalarMatrix fluidTEqn
(
fvm::ddt(Tf)
+ fvm::div(phi,Tf)
```

```
 - fvm::laplacian(DTf,Tf)
 - (1/c_)*(tau && gradU)
 );
```

Appendix 4 – TSolid.H Code

```
fvScalarMatrix solidTEqn
(
fvm::ddt(Ts)
 - fvm::laplacian(DTs,Ts)
);
```

Appendix 5 – createMesh.H Code

```
/*---------------------------------------------------------------------------*\
========= |
\\ / F ield | OpenFOAM: The Open Source CFD Toolbox
\\ / O peration |
\\ / A nd | www.openfoam.com
\\/ M anipulation |
-------------------------------------------------------------------------------
Copyright (C) 2018-2021 OpenCFD Ltd.
-------------------------------------------------------------------------------
License
This file is part of OpenFOAM, distributed under GPL-3.0-or-later.

Description
Create a fvMesh (specified region or defaultRegion) with
additional handling of -dry-run and -dry-run-write options.

Required Variables
- args [argList]
- runTime [Time]

Provided Variables
- regionName [word]
- mesh [fvMesh]
- meshPtr [autoPtr<fvMesh>]

\*---------------------------------------------------------------------------*/

Info << "Create fluid mesh";

fvMesh fluidMesh
(
IOobject
(
"fluid",
runTime.timeName(),
runTime,
IOobject::MUST_READ
)
);

Info << "Create solid mesh";
fvMesh solidMesh
(
IOobject
(
"solid",
runTime.timeName(),
```

```
runTime,
IOobject::MUST_READ
)
);

// ************************************************************************* //
```

APPENDIX 6 – CREATEFIELDS.H CODE

```
#include "createRDeltaT.H"

Info<< "Reading field p\n" << endl;
volScalarField p
(
IOobject
(
"p",
runTime.timeName(),
fluidMesh,
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
fluidMesh
);

Info<< "Reading field U\n" << endl;
volVectorField U
(
IOobject
(
"U",
runTime.timeName(),
fluidMesh,
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
fluidMesh
);
Info<< "Reading field Tfluid\n" << endl;
volScalarField Tf
(
IOobject
(
"T",
runTime.timeName(),
fluidMesh,
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
fluidMesh
);

Info<< "Reading field Tsolid\n" << endl;
volScalarField Ts
(
IOobject
(
"T",
runTime.timeName(),
solidMesh,
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
solidMesh
);
```

```
58
59  IOdictionary solidTransportProperties
60  (
61  IOobject
62  (
63  "transportProperties",
64  runTime.constant(),
65  solidMesh,
66  IOobject::MUST_READ,
67  IOobject::NO_WRITE
68  )
69  );
70  Info<< "\tReading solid thermal diffusivity\n" << endl;
71  volScalarField DTs
72  (
73  IOobject
74  (
75  "DT",
76  runTime.timeName(),
77  solidMesh,
78  IOobject::NO_READ,
79  IOobject::NO_WRITE
80  ),
81  solidMesh,
82  dimensionedScalar
83  (
84  "DT",
85  dimViscosity,
86  solidTransportProperties.lookup("DT")
87  )
88  );
89  Info<< "\tReading solid thermal conductivity\n" << endl;
90  volScalarField kappaS
91  (
92  IOobject
93  (
94  "kappa",
95  runTime.timeName(),
96  solidMesh,
97  IOobject::NO_READ,
98  IOobject::NO_WRITE
99  ),
100 solidMesh,
101 dimensionedScalar
102 (
103 "kappa",
104 dimViscosity,
105 solidTransportProperties.lookup("kappa")
106 )
107 );
108 IOdictionary fluidTransportProperties
109 (
110 IOobject
111 (
112 "transportProperties",
113 runTime.constant(),
114 fluidMesh,
115 IOobject::MUST_READ,
116 IOobject::NO_WRITE
117 )
118 );
119 Info<< "\tReading fluid thermal conductivity\n" << endl;
120 volScalarField kappaF
121 (
122 IOobject
123 (
124 "kappa",
125 runTime.timeName(),
126 fluidMesh,
```

```
127  IOobject::NO_READ,
128  IOobject::NO_WRITE
129  ),
130  fluidMesh,
131  dimensionedScalar
132  (
133  "kappa",
134  dimViscosity,
135  fluidTransportProperties.lookup("kappa")
136  )
137  );
138  Info<< "\tReading fluid thermal diffusivity\n" << endl;
139  volScalarField DTf
140  (
141  IOobject
142  (
143  "DT",
144  runTime.timeName(),
145  fluidMesh,
146  IOobject::NO_READ,
147  IOobject::NO_WRITE
148  ),
149  fluidMesh,
150  dimensionedScalar
151  (
152  "DT",
153  dimViscosity,
154  fluidTransportProperties.lookup("DT")
155  )
156  );
157  volScalarField c_
158  (
159  IOobject
160  (
161  "c",
162  runTime.timeName(),
163  fluidMesh,
164  IOobject::NO_READ,
165  IOobject::NO_WRITE
166  ),
167  fluidMesh,
168  dimensionedScalar
169  (
170  "c",
171  fluidTransportProperties.lookup("c")
172  )
173  );
174  #include "createPhi.H"
175
176  label pRefCell = 0;
177  scalar pRefValue = 0.0;
178  setRefCell(p, pimple.dict(), pRefCell, pRefValue);
179  fluidMesh.setFluxRequired(p.name());
180
181  singlePhaseTransportModel laminarTransport(U, phi);
182
183  autoPtr<incompressible::turbulenceModel> turbulence
184  (
185  incompressible::turbulenceModel::New(U, phi, laminarTransport)
186  );
187
188  #include "createMRF.H"
189  #include "createFvOptions.H"
```

APPENDIX 7 – CHTMULTIREGIONPIMPLEFOAM.H CODE

```
  1  /*---------------------------------------------------------------------------*\
  2  ========= |
  3  \\ / F ield | OpenFOAM: The Open Source CFD Toolbox
  4  \\ / O peration |
  5  \\ / A nd | www.openfoam.com
  6  \\/ M anipulation |
  7  -------------------------------------------------------------------------------
  8  Copyright (C) 2011-2017 OpenFOAM Foundation
  9  Copyright (C) 2019 OpenCFD Ltd.
 10  -------------------------------------------------------------------------------
 11  License
 12  This file is part of OpenFOAM.
 13
 14  OpenFOAM is free software: you can redistribute it and/or modify it
 15  under the terms of the GNU General Public License as published by
 16  the Free Software Foundation, either version 3 of the License, or
 17  (at your option) any later version.
 18
 19  OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
 20  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 21  FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
 22  for more details.
 23
 24  You should have received a copy of the GNU General Public License
 25  along with OpenFOAM. If not, see <http://www.gnu.org/licenses/>.
 26
 27  Application
 28  chtMultiPimpleFoam.C
 29
 30  Group
 31  grpIncompressibleSolvers
 32
 33  Description
 34  Multi-region Transient solver for incompressible, turbulent flow of Newtonian
fluids
 35  on a moving mesh.
 36
 37  \heading Solver details
 38  The solver uses the PIMPLE (merged PISO-SIMPLE) algorithm to solve the
 39  continuity equation:
 40
 41  \f[
 42  \div \vec{U} = 0
 43  \f]
 44
 45  and momentum equation:
 46
 47  \f[
 48  \ddt{\vec{U}} + \div \left( \vec{U} \vec{U} \right) - \div \gvec{R}
 49  = - \grad p + \vec{S}_U
 50  \f]
 51
 52  Where:
 53  \vartable
 54  \vec{U} | Velocity
 55  p | Pressure
 56  \vec{R} | Stress tensor
 57  \vec{S}_U | Momentum source
 58  \endvartable
 59
 60  Sub-models include:
 61  - turbulence modelling, i.e. laminar, RAS or LES
 62  - run-time selectable MRF and finite volume options, e.g. explicit porosity
 63
 64  \heading Required fields
```

```
65  \plaintable
66  U | Velocity [m/s]
67  p | Kinematic pressure, p/rho [m2/s2]
68  \<turbulence fields\> | As required by user selection
69  \endplaintable
70
71  Note
72  The motion frequency of this solver can be influenced by the presence
73  of "updateControl" and "updateInterval" in the dynamicMeshDict.
74
75  \*---------------------------------------------------------------------------*/
76
77  #include "fvCFD.H"
78  #include "dynamicFvMesh.H"
79  #include "singlePhaseTransportModel.H"
80  #include "turbulentTransportModel.H"
81  #include "pimpleControl.H"
82  #include "CorrectPhi.H"
83  #include "fvOptions.H"
84  #include "localEulerDdtScheme.H"
85  #include "fvcSmooth.H"
86
87  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
88
89  int main(int argc, char *argv[])
90  {
91  argList::addNote
92  (
93  "Transient solver for incompressible, turbulent flow"
94  " of Newtonian fluids on a moving mesh."
95  );
96
97  //#include "postProcess.H"
98
99  #include "addCheckCaseOptions.H"
100  #include "setRootCaseLists.H"
101  #include "createTime.H"
102  #include "createDynamicFvMesh.H"
103  #include "createDyMControls.H"
104  #include "createMesh.H"
105  #include "initContinuityErrs.H"
106  #include "createFields.H"
107  #include "createUfIfPresent.H"
108  #include "CourantNo.H"
109  #include "setInitialDeltaT.H"
110
111  turbulence->validate();
112
113  if (!LTS)
114  {
115  #include "CourantNo.H"
116  #include "setInitialDeltaT.H"
117  }
118
119  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
120
121  Info<< "\nStarting time loop\n" << endl;
122
123  while (runTime.run())
124  {
125  #include "readDyMControls.H"
126
127  if (LTS)
128  {
129  #include "setRDeltaT.H"
130  }
131  else
132  {
133  #include "CourantNo.H"
```

```
#include "setDeltaT.H"
}

++runTime;

Info<< "Time = " << runTime.timeName() << nl << endl;

// --- Pressure-velocity PIMPLE corrector loop
while (pimple.loop())
{
if (pimple.firstIter() || moveMeshOuterCorrectors)
{
// Do any mesh changes
mesh.controlledUpdate();

if (mesh.changing())
{
MRF.update();

if (correctPhi)
{
// Calculate absolute flux
// from the mapped surface velocity
phi = mesh.Sf() & Uf();

#include "correctPhi.H"

// Make the flux relative to the mesh motion
fvc::makeRelative(phi, U);
}

if (checkMeshCourantNo)
{
#include "meshCourantNo.H"
}
}
}
#include "UEqn.H"
// --- Pressure corrector loop
while (pimple.correct())
{
#include "pEqn.H"
}
if (pimple.turbCorr())
{
laminarTransport.correct();
turbulence->correct();
}
Info<< "Solving temperature\n" << endl;
#include "Tsolid.H"
scalar TSResidual = solidTEqn.solve().initialResidual();
#include "Tfluid.H"
scalar TFResidual = fluidTEqn.solve().initialResidual();
scalar globalResidual = (TSResidual + TFResidual)/2;
Info<< "Global Temperature Residuals :" << globalResidual << endl;

}

runTime.write();

runTime.printExecutionTime(Info);
}

Info<< "End\n" << endl;

return 0;
}

//************************************************************************* //
```

APPENDIX 8 – EXTERNALWALLHEATFLUXTEMPERATUREPIDFVPATCHSCALARFIELD.C CODE

```
/*---------------------------------------------------------------------------*\
========= |
\\ / F ield | OpenFOAM: The Open Source CFD Toolbox
\\ / O peration |
\\ / A nd | www.openfoam.com
\\/ M anipulation |
-------------------------------------------------------------------------------
Copyright (C) 2011-2017 OpenFOAM Foundation
Copyright (C) 2015-2020 OpenCFD Ltd.
-------------------------------------------------------------------------------
License
This file is part of OpenFOAM.

OpenFOAM is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
for more details.

You should have received a copy of the GNU General Public License
along with OpenFOAM. If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "externalWallHeatFluxTemperaturePIDFvPatchScalarField.H"
#include "addToRunTimeSelectionTable.H"
#include "fvPatchFieldMapper.H"
#include "surfaceFields.H"
#include "volFields.H"
#include "physicoChemicalConstants.H"
#include "OFstream.H"

using Foam::constant::physicoChemical::sigma;

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

const Foam::Enum
<
Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField::operationMode
>
Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField::operationModeNames
({
{ operationMode::fixedHeatFlux, "flux" },
});

// * * * * * * * * * * * * * * * * * * Constructors * * * * * * * * * * * * * * * * * * //

Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField::
externalWallHeatFluxTemperaturePIDFvPatchScalarField
(
const fvPatch& p,
const DimensionedField<scalar, volMesh>& iF
)
:
mixedFvPatchScalarField(p, iF),
```

```
60  temperatureCoupledBase
61  (
62  patch(),
63  "undefined",
64  "undefined",
65  "undefined-K",
66  "undefined-alpha"
67  ),
68  mode_(fixedHeatFlux),
69  //ADDED
70  sensorName_("wall_probe"),
71  Tobj_(650),
72  P_(1),
73  I_(1),
74  D_(1),
75  Tave_(0),
76  error_(0),
77  errorIntegral_(0),
78  oldTave_(0),
79  oldError_(0),
80  oldErrorIntegral_(0),
81  timeIndex_(db().time().timeIndex()),
82  //ADDED
83  Q_(nullptr),
84  q_(nullptr),
85  h_(nullptr),
86  Ta_(nullptr),
87  relaxation_(1),
88  emissivity_(0),
89  qrRelaxation_(1),
90  qrName_("undefined-qr"),
91  thicknessLayers_(),
92  kappaLayers_()
93  {
94  refValue() = 0;
95  refGrad() = 0;
96  valueFraction() = 1;
97  }
98
99  Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField::
100 externalWallHeatFluxTemperaturePIDFvPatchScalarField
101 (
102 const fvPatch& p,
103 const DimensionedField<scalar, volMesh>& iF,
104 const dictionary& dict
105 )
106 :
107 mixedFvPatchScalarField(p, iF),
108 temperatureCoupledBase(patch(), dict),
109 mode_(operationModeNames.get("mode", dict)),
110 //ADDED
111 sensorName_(dict.getOrDefault<word>("sensorName", "None")),
112 Tobj_(dict.getOrDefault<scalar>("Tobj", 650)),
113 P_(dict.getOrDefault<scalar>("P", 1)),
114 I_(dict.getOrDefault<scalar>("I", 1)),
115 D_(dict.getOrDefault<scalar>("D", 1)),
116 Tave_(0),
117 error_(0),
118 errorIntegral_(0),
119 oldTave_(0),
120 oldError_(0),
121 oldErrorIntegral_(0),
122 timeIndex_(db().time().timeIndex()),
123 //ADDED
124 Q_(nullptr),
125 q_(nullptr),
126 h_(nullptr),
127 Ta_(nullptr),
128 relaxation_(dict.getOrDefault<scalar>("relaxation", 1)),
```

```
129  emissivity_(dict.getOrDefault<scalar>("emissivity", 0)),
130  qrRelaxation_(dict.getOrDefault<scalar>("qrRelaxation", 1)),
131  qrName_(dict.getOrDefault<word>("qr", "none")),
132  thicknessLayers_(),
133  kappaLayers_()
134  {
135  switch (mode_)
136  {
137  case fixedHeatFlux:
138  {
139  q_ = PatchFunction1<scalar>::New(patch().patch(), "q", dict);
140  break;
141  }
142  }
143
144  fvPatchScalarField::operator=(scalarField("value", dict, p.size()));
145
146  if (qrName_ != "none")
147  {
148  if (dict.found("qrPrevious"))
149  {
150  qrPrevious_ = scalarField("qrPrevious", dict, p.size());
151  }
152  else
153  {
154  qrPrevious_.resize(p.size(), Zero);
155  }
156  }
157
158  if (dict.found("refValue"))
159  {
160  // Full restart
161  refValue() = scalarField("refValue", dict, p.size());
162  refGrad() = scalarField("refGradient", dict, p.size());
163  valueFraction() = scalarField("valueFraction", dict, p.size());
164  }
165  else
166  {
167  // Start from user entered data. Assume fixedValue.
168  refValue() = *this;
169  refGrad() = 0;
170  valueFraction() = 1;
171  }
172
173  h_ = PatchFunction1<scalar>::New(patch().patch(), "h", dict);
174  Ta_ = Function1<scalar>::New("Ta", dict, &db());
175  }
176
177  Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField::
178  externalWallHeatFluxTemperaturePIDFvPatchScalarField
179  (
180  const externalWallHeatFluxTemperaturePIDFvPatchScalarField& rhs,
181  const fvPatch& p,
182  const DimensionedField<scalar, volMesh>& iF,
183  const fvPatchFieldMapper& mapper
184  )
185  :
186  mixedFvPatchScalarField(rhs, p, iF, mapper),
187  temperatureCoupledBase(patch(), rhs),
188  mode_(rhs.mode_),
189  //ADDED
190  sensorName_(rhs.sensorName_),
191  Tobj_(rhs.Tobj_),
192  P_(rhs.P_),
193  I_(rhs.I_),
194  D_(rhs.D_),
195  Tave_(rhs.Tave_),
196  error_(rhs.error_),
197  errorIntegral_(rhs.errorIntegral_),
```

```
198  oldTave_(rhs.oldTave_),
199  oldError_(rhs.oldError_),
200  oldErrorIntegral_(rhs.oldErrorIntegral_),
201  timeIndex_(rhs.timeIndex_),
202  //ADDED
203  Q_(rhs.Q_.clone()),
204  q_(rhs.q_.clone(patch().patch())),
205  h_(rhs.h_.clone(patch().patch())),
206  Ta_(rhs.Ta_.clone()),
207  relaxation_(rhs.relaxation_),
208  emissivity_(rhs.emissivity_),
209  qrPrevious_(),
210  qrRelaxation_(rhs.qrRelaxation_),
211  qrName_(rhs.qrName_),
212  thicknessLayers_(rhs.thicknessLayers_),
213  kappaLayers_(rhs.kappaLayers_)
214  {
215  if (qrName_ != "none")
216  {
217  qrPrevious_.resize(mapper.size());
218  qrPrevious_.map(rhs.qrPrevious_, mapper);
219  }
220  }
221
222  Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField::
223  externalWallHeatFluxTemperaturePIDFvPatchScalarField
224  (
225  const externalWallHeatFluxTemperaturePIDFvPatchScalarField& rhs
226  )
227  :
228  mixedFvPatchScalarField(rhs),
229  temperatureCoupledBase(rhs),
230  mode_(rhs.mode_),
231  //ADDED
232  sensorName_(rhs.sensorName_),
233  Tobj_(rhs.Tobj_),
234  P_(rhs.P_),
235  I_(rhs.I_),
236  D_(rhs.D_),
237  Tave_(rhs.Tave_),
238  error_(rhs.error_),
239  errorIntegral_(rhs.errorIntegral_),
240  oldTave_(rhs.oldTave_),
241  oldError_(rhs.oldError_),
242  oldErrorIntegral_(rhs.oldErrorIntegral_),
243  timeIndex_(rhs.timeIndex_),
244  //ADDED
245  Q_(rhs.Q_.clone()),
246  q_(rhs.q_.clone(patch().patch())),
247  h_(rhs.h_.clone(patch().patch())),
248  Ta_(rhs.Ta_.clone()),
249  relaxation_(rhs.relaxation_),
250  emissivity_(rhs.emissivity_),
251  qrPrevious_(rhs.qrPrevious_),
252  qrRelaxation_(rhs.qrRelaxation_),
253  qrName_(rhs.qrName_),
254  thicknessLayers_(rhs.thicknessLayers_),
255  kappaLayers_(rhs.kappaLayers_)
256  {}
257
258  Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField::
259  externalWallHeatFluxTemperaturePIDFvPatchScalarField
260  (
261  const externalWallHeatFluxTemperaturePIDFvPatchScalarField& rhs,
262  const DimensionedField<scalar, volMesh>& iF
263  )
264  :
265  mixedFvPatchScalarField(rhs, iF),
266  temperatureCoupledBase(patch(), rhs),
```

```
267  mode_(rhs.mode_),
268  //ADDED
269  sensorName_(rhs.sensorName_),
270  Tobj_(rhs.Tobj_),
271  P_(rhs.P_),
272  I_(rhs.I_),
273  D_(rhs.D_),
274  Tave_(rhs.Tave_),
275  error_(rhs.error_),
276  errorIntegral_(rhs.errorIntegral_),
277  oldTave_(rhs.oldTave_),
278  oldError_(rhs.oldError_),
279  oldErrorIntegral_(rhs.oldErrorIntegral_),
280  timeIndex_(rhs.timeIndex_),
281  //ADDED
282  Q_(rhs.Q_.clone()),
283  q_(rhs.q_.clone(patch().patch())),
284  h_(rhs.h_.clone(patch().patch())),
285  Ta_(rhs.Ta_.clone()),
286  relaxation_(rhs.relaxation_),
287  emissivity_(rhs.emissivity_),
288  qrPrevious_(rhs.qrPrevious_),
289  qrRelaxation_(rhs.qrRelaxation_),
290  qrName_(rhs.qrName_),
291  thicknessLayers_(rhs.thicknessLayers_),
292  kappaLayers_(rhs.kappaLayers_)
293  {}
294
295  // * * * * * * * * * * * * * * * * * * Member Functions * * * * * * * * * * * * * * * //
296
297  void Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField::autoMap
298  (
299  const fvPatchFieldMapper& mapper
300  )
301  {
302  mixedFvPatchScalarField::autoMap(mapper);
303  // temperatureCoupledBase::autoMap(mapper);
304
305  if (q_)
306  {
307  q_->autoMap(mapper);
308  }
309  if (h_)
310  {
311  h_->autoMap(mapper);
312  }
313
314  if (qrName_ != "none")
315  {
316  qrPrevious_.autoMap(mapper);
317  }
318  }
319
320  void Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField::rmap
321  (
322  const fvPatchScalarField& ptf,
323  const labelList& addr
324  )
325  {
326  mixedFvPatchScalarField::rmap(ptf, addr);
327
328  const auto& rhs =
329  refCast<const externalWallHeatFluxTemperaturePIDFvPatchScalarField>(ptf);
330
331  // temperatureCoupledBase::rmap(rhs, addr);
332
333  if (q_)
334  {
335  q_->rmap(rhs.q_(), addr);
```

```
336  }
337
338  if (qrName_ != "none")
339  {
340  qrPrevious_.rmap(rhs.qrPrevious_, addr);
341  }
342  }
343
344  oid Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField::updateCoeffs()
345  {
346  if (updated())
347  {
348  return;
349  }
350  //get time step
351  scalar deltaT =db().time().deltaTValue();
352  // Update the old-time quantities
353  if (timeIndex_ != db().time().timeIndex())
354  {
355  timeIndex_ = db().time().timeIndex();
356  oldTave_ = Tave_;
357  oldError_ = error_;
358  oldErrorIntegral_ = errorIntegral_;
359  }
360  const fvPatch& p = this->patch();
361  //get patch ID
362  const label sensorPatchID =
363  p.patch().boundaryMesh().findPatchID(sensorName_);
364
365  if (sensorPatchID < 0)
366  {
367  FatalErrorInFunction
368  << "Unable to find sensor patch " << sensorName_
369  << abort(FatalError);
370  }
371  //get Patch
372  const fvPatch& sensorPatch = p.boundaryMesh()[sensorPatchID];
373  //get Temperature
374  auto& T =
375  this->db().objectRegistry::template lookupObject<volScalarField>
376  ("T");
377  const fvMesh& mesh = patch().boundaryMesh().mesh();
378  //sum Temperature
379  scalar sensorPatchT = 0;
380sensorPatchT=mag(gSum(T.boundaryField()[sensorPatchID]*mesh.Sf().boundaryField()
[sensorPatchID]));
381  // get boundary area
382  const scalar sensorArea = mag(gSum(mesh.Sf().boundaryField()[sensorPatchID]));
383  // get Tave_
384  scalar Tave_ = sensorPatchT / sensorArea;
385  // Errors
386  error_ = Tave_ - Tobj_;
387  errorIntegral_ = oldErrorIntegral_ + error_;
388  scalar errorDifferential = -(oldError_ - error_) / deltaT;
389  scalar PIDfunction = P_*error_ + I_*errorIntegral_ + D_*errorDifferential;
390  //scalar PIDfunction = P_*error_;
391  //scalar PIDfunction = P_*error_ + I_*errorIntegral_;
392  Info<< nl << " PID function :" << PIDfunction << " s"
393  << nl << " Tave :" << Tave_ << " s"
394  << nl << " Tobj :" << Tobj_ << " s"
395  << nl << endl;
396  if (PIDfunction < 0)
397  {
398  const scalarField& Tp(*this);
399
400  const scalarField valueFraction0(valueFraction());
401  const scalarField refValue0(refValue());
402
403  scalarField qr(Tp.size(), Zero);
```

```
404  if (qrName_ != "none")
405  {
406  qr =
407  qrRelaxation_
408  *patch().lookupPatchField<volScalarField, scalar>(qrName_)
409  + (1 - qrRelaxation_)*qrPrevious_;
410
411  qrPrevious_ = qr;
412  }
413
414  tmp<scalarField> heatFlux =
415  q_->value(this->db().time().timeOutputValue());
416
417  refGrad() = (heatFlux + qr)/kappa(Tp);
418  refValue() = 0;
419  valueFraction() = 0;
420
421  //valueFraction() =
422  // relaxation_*valueFraction() + (1 - relaxation_)*valueFraction0;
423  //refValue() = relaxation_*refValue() + (1 - relaxation_)*refValue0;
424
425  mixedFvPatchScalarField::updateCoeffs();
426  DebugInfo
427  << patch().boundaryMesh().mesh().name() << ':' << patch().name() << ':'
428  << internalField().name() << " :"
429  << " heat transfer rate:" << gSum(kappa(Tp)*patch().magSf()*snGrad())
430  << " wall temperature "
431  << " min:" << gMin(*this)
432  << " max:" << gMax(*this)
433  << " avg:" << gAverage(*this) << nl;
434  }
435  else
436  {
437  const scalarField& Tp(*this);
438
439  const scalarField valueFraction0(valueFraction());
440  const scalarField refValue0(refValue());
441
442  scalarField qr(Tp.size(), Zero);
443  if (qrName_ != "none")
444  {
445  qr =
446  qrRelaxation_
447  *patch().lookupPatchField<volScalarField, scalar>(qrName_)
448  + (1 - qrRelaxation_)*qrPrevious_;
449  qrPrevious_ = qr;
450  }
451  //refGrad() = 0;
452  //refValue() = 0;
453  //valueFraction() = 0;
454
455  tmp<scalarField> thtcCoeff =
456  (
457  h_->value(this->db().time().timeOutputValue()) + VSMALL
458  );
459  const auto& htcCoeff = thtcCoeff();
460  scalar totalSolidRes = 0;
461  if (thicknessLayers_.size())
462  {
463  forAll(thicknessLayers_, iLayer)
464  {
465  const scalar l = thicknessLayers_[iLayer];
466  if (kappaLayers_[iLayer] > 0)
467  {
468  totalSolidRes += l/kappaLayers_[iLayer];
469  }
470  }
471  }
472  scalarField hp(1/(1/htcCoeff + totalSolidRes));
```

```
473  const scalar Ta =
474  Ta_->value(this->db().time().timeOutputValue());
475  scalarField hpTa(hp*Ta);
476  const scalarField kappaDeltaCoeffs
477  (
478  this->kappa(Tp)*patch().deltaCoeffs()
479  );
480  refGrad() = 0;
481  forAll(Tp, i)
482  {
483  refValue()[i] = (hpTa[i] + qr[i])/hp[i];
484  valueFraction()[i] = hp[i]/(hp[i] + kappaDeltaCoeffs[i]);
485  }
486
487  //valueFraction() =
488  // relaxation_*valueFraction() + (1 - relaxation_)*valueFraction0;
489  //refValue() = relaxation_*refValue() + (1 - relaxation_)*refValue0;
490
491  mixedFvPatchScalarField::updateCoeffs();
492  DebugInfo
493  << patch().boundaryMesh().mesh().name() << ':' << patch().name() << ':'
494  << internalField().name() << " :"
495  << " heat transfer rate:" << gSum(kappa(Tp)*patch().magSf()*snGrad())
496  << " wall temperature "
497  << " min:" << gMin(*this)
498  << " max:" << gMax(*this)
499  << " avg:" << gAverage(*this) << nl;
500  }
501  }
502
503  void Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField::write
504  (
505  Ostream& os
506  ) const
507  {
508  fvPatchScalarField::write(os);
509
510  os.writeEntry("mode", operationModeNames[mode_]);
511  temperatureCoupledBase::write(os);
512
513  if (Q_)
514  {
515  Q_->writeData(os);
516  }
517  if (q_)
518  {
519  q_->writeData(os);
520  }
521
522  if (Ta_)
523  {
524  Ta_->writeData(os);
525  }
526
527  os.writeEntry("qr", qrName_);
528
529  if (qrName_ != "none")
530  {
531  os.writeEntry("qrRelaxation", qrRelaxation_);
532
533  qrPrevious_.writeEntry("qrPrevious", os);
534  }
535
536  refValue().writeEntry("refValue", os);
537  refGrad().writeEntry("refGradient", os);
538  valueFraction().writeEntry("valueFraction", os);
539  //ADDED//
540  os.writeEntry("Tobj", Tobj_);
541  os.writeEntry("sensorName", sensorName_);
```

```
os.writeEntry("P", P_);
os.writeEntry("I", I_);
os.writeEntry("D", D_);
os.writeEntry("error", error_);
os.writeEntry("errorIntegral", errorIntegral_);
Info<< nl << " Gradient :" << refValue() << " s"
<< nl << endl;
//ADDED//
writeEntry("value", os);
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{
makePatchTypeField
(
fvPatchScalarField,
externalWallHeatFluxTemperaturePIDFvPatchScalarField
);
}

                                                                        //
************************************************************************* //
```

APPENDIX 9 – EXTERNALWALLHEATFLUXTEMPERATUREPIDFVPATCHSCALARFIELD.H CODE

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2017 OpenFOAM Foundation
    Copyright (C) 2020 OpenCFD Ltd.
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Class
    Foam::externalWallHeatFluxTemperaturePIDFvPatchScalarField

Group
    grpThermoBoundaryConditions grpWallBoundaryConditions

Description
```

```
 34      This boundary condition applies a heat flux condition to temperature
 35      on an external wall in one of three modes:
 36
 37        - fixed power: supply Q
 38        - fixed heat flux: supply q
 39        - fixed heat transfer coefficient: supply h and Ta
 40
 41      where:
 42      \vartable
 43          Q  | Power [W]
 44          q  | Heat flux [W/m^2]
 45          h  | Heat transfer coefficient [W/m^2/K]
 46          Ta | Ambient temperature [K]
 47      \endvartable
 48
 49      For heat transfer coefficient mode optional thin thermal layer resistances
 50      can be specified through thicknessLayers and kappaLayers entries.
 51
 52      The thermal conductivity \c kappa can either be retrieved from various
 53      possible sources, as detailed in the class temperatureCoupledBase.
 54
 55      The ambient temperature Ta is specified as a Foam::Function1 of time but
 56      uniform in space.
 57
 58  Usage
 59      \table
 60       Property     | Description                                    | Required |
Default
 61      mode         | 'power', 'flux' or 'coefficient'         | yes |
 62      Q            | Power [W]                              | for mode 'power' |
 63      q            | Heat flux [W/m^2]                       | for mode 'flux' |
 64      h          | Heat transfer coefficient [W/m^2/K] | for mode 'coefficient' |
 65      Ta           | Ambient temperature [K]           | for mode 'coefficient' |
 66      thicknessLayers | Layer thicknesses [m]                | no  |
 67      kappaLayers | Layer thermal conductivities [W/m/K]     | no  |
 68      relaxation  | Relaxation for the wall temperature      | no  | 1
 69      emissivity  | Surface emissivity for radiative flux to ambient | no | 0
 70      qr           | Name of the radiative field              | no  | none
 71      qrRelaxation | Relaxation factor for radiative field   | no  | 1
 72      kappaMethod | Inherited from temperatureCoupledBase    | inherited |
 73      kappa       | Inherited from temperatureCoupledBase    | inherited |
 74      \endtable
 75
 76      Example of the boundary condition specification:
 77      \verbatim
 78      <patchName>
 79      {
 80          type            externalWallHeatFluxTemperature;
 81
 82          mode            coefficient;
 83
 84          Ta              constant 300.0;
 85          h               constant 10.0;
 86          thicknessLayers (0.1 0.2 0.3 0.4);
 87          kappaLayers     (1 2 3 4);
 88
 89          kappaMethod     fluidThermo;
 90
 91          value           $internalField;
 92      }
 93      \endverbatim
 94
 95  Note
 96      Quantities that are considered "global" (eg, power, ambient temperature)
 97      can be specified as Function1 types.
 98      Quantities that may have local variations (eg, htc, heat-flux)
 99      can be specified as PatchFunction1 types.
100
101  See also
```

```
102     Foam::temperatureCoupledBase
103     Foam::mixedFvPatchScalarField
104
105 SourceFiles
106     externalWallHeatFluxTemperaturePIDFvPatchScalarField.C
107
108 \*---------------------------------------------------------------------------*/
109
110 #ifndef externalWallHeatFluxTemperaturePIDFvPatchScalarField_H
111 #define externalWallHeatFluxTemperaturePIDFvPatchScalarField_H
112
113 #include "mixedFvPatchFields.H"
114 #include "temperatureCoupledBase.H"
115 #include "PatchFunction1.H"
116
117 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
118
119 namespace Foam
120 {
121
122 /*---------------------------------------------------------------------------*\
123       Class externalWallHeatFluxTemperaturePIDFvPatchScalarField Declaration
124 \*---------------------------------------------------------------------------*/
125
126 class externalWallHeatFluxTemperaturePIDFvPatchScalarField
127 :
128     public mixedFvPatchScalarField,
129     public temperatureCoupledBase
130 {
131 public:
132
133     // Public Data
134
135         //- Operation mode enumeration
136         enum operationMode
137         {
138             fixedPower,             //!< Heat power [W]
139             fixedHeatFlux,          //!< Heat flux [W/m2]
140         };
141
142         static const Enum<operationMode> operationModeNames;
143
144
145 private:
146
147     // Private Data
148
149         //- Operation mode
150         operationMode mode_;
151
152         //- Heat power [W]
153         autoPtr<Function1<scalar>> Q_;
154
155         //- Heat flux [W/m2]
156         autoPtr<PatchFunction1<scalar>> q_;
157
158   //- Heat flux coeficient [W/m2K
159         autoPtr<PatchFunction1<scalar>> h_;
160
161         //- Ambient temperature [K]
162         autoPtr<Function1<scalar>> Ta_;
163
164         //- Relaxation for the wall temperature (thermal inertia)
165         scalar relaxation_;
166
167         //- Optional surface emissivity for radiative transfer to ambient
168         scalar emissivity_;
169
170         //- Cache qr for relaxation
```

```
171         scalarField qrPrevious_;
172
173         //- Relaxation for qr
174         scalar qrRelaxation_;
175
176         //- Name of the radiative heat flux
177         const word qrName_;
178
179         //- Thickness of layers
180         scalarList thicknessLayers_;
181
182         //- Conductivity of layers
183         scalarList kappaLayers_;
184         //ADDED for PID
185         //- Name of the sensor patch
186         const word sensorName_;
187
188         //- Desired Temperature
189         const scalar Tobj_;
190
191         //- Proportional gain
192         const scalar P_;
193
194         //- Integral gain
195         const scalar I_;
196
197         //- Derivative gain
198         const scalar D_;
199
200         //- Average Temperature
201         scalar Tave_;
202
203         //- Error
204         scalar error_;
205
206         //- Error integral w.r.t. time
207         scalar errorIntegral_;
208
209         //- Old Average Temperature
210         scalar oldTave_;
211
212         //- Old error
213         scalar oldError_;
214
215         //- Old error integral w.r.t. time
216         scalar oldErrorIntegral_;
217
218         //- Time index of the last update
219         label timeIndex_;
220
221 public:
222
223     //- Runtime type information
224     TypeName("externalWallHeatFluxTemperaturePID");
225
226
227     // Constructors
228
229         //- Construct from patch and internal field
230         externalWallHeatFluxTemperaturePIDFvPatchScalarField
231         (
232             const fvPatch&,
233             const DimensionedField<scalar, volMesh>&
234         );
235
236         //- Construct from patch, internal field and dictionary
237         externalWallHeatFluxTemperaturePIDFvPatchScalarField
238         (
239             const fvPatch&,
```

```
240             const DimensionedField<scalar, volMesh>&,
241             const dictionary&
242         );
243 
244         //- Construct by mapping given
245         //  externalWallHeatFluxTemperaturePIDFvPatchScalarField
246         //  onto a new patch
247         externalWallHeatFluxTemperaturePIDFvPatchScalarField
248         (
249             const externalWallHeatFluxTemperaturePIDFvPatchScalarField&,
250             const fvPatch&,
251             const DimensionedField<scalar, volMesh>&,
252             const fvPatchFieldMapper&
253         );
254 
255         //- Construct as copy
256         externalWallHeatFluxTemperaturePIDFvPatchScalarField
257         (
258             const externalWallHeatFluxTemperaturePIDFvPatchScalarField&
259         );
260 
261         //- Construct and return a clone
262         virtual tmp<fvPatchScalarField> clone() const
263         {
264             return tmp<fvPatchScalarField>
265             (
266               new externalWallHeatFluxTemperaturePIDFvPatchScalarField(*this)
267             );
268         }
269 
270         //- Construct as copy setting internal field reference
271         externalWallHeatFluxTemperaturePIDFvPatchScalarField
272         (
273             const externalWallHeatFluxTemperaturePIDFvPatchScalarField&,
274             const DimensionedField<scalar, volMesh>&
275         );
276 
277         //- Construct and return a clone setting internal field reference
278         virtual tmp<fvPatchScalarField> clone
279         (
280             const DimensionedField<scalar, volMesh>& iF
281         ) const
282         {
283             return tmp<fvPatchScalarField>
284             (
285            new externalWallHeatFluxTemperaturePIDFvPatchScalarField(*this, iF)
286             );
287         }
288 
289 
290     // Member functions
291 
292         // Access
293 
294             //- Allow manipulation of the boundary values
295             virtual bool fixesValue() const
296             {
297                 return false;
298             }
299 
300 
301         // Mapping functions
302 
303             //- Map (and resize as needed) from self given a mapping object
304             virtual void autoMap
305             (
306                 const fvPatchFieldMapper&
307             );
308 
```

```
309             //- Reverse map the given fvPatchField onto this fvPatchField
310             virtual void rmap
311             (
312                 const fvPatchScalarField&,
313                 const labelList&
314             );
315 
316 
317         // Evaluation functions
318 
319             //- Update the coefficients associated with the patch field
320             virtual void updateCoeffs();
321 
322 
323         // I-O
324 
325             //- Write
326             void write(Ostream&) const;
327 };
328 
329 
330 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
331 
332 } // End namespace Foam
333 
334 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
335 
336 #endif
337 
338 // ************************************************************************* //
```