

TABLE OF CONTENTS

1. INTRODUCTION	2
2. THEORY	3
2.1. Accelerating Charged Particles	3
2.1.1. Relation between momentum and acceleration	3
2.1.2. Lorentz Force	5
2.1.3. Relativistic Lorentz Force	5
2.1.4. Acceleration caused by lorentz force	6
2.2. Particle Accelerators	8
2.2.1. Key Concepts	9
2.2.1.1. Resonance Frequency	9
2.2.1.2. Bunch	9
2.2.1.3. Phase Stability	10
2.2.1.4. Phase Lag	10
2.2.1.5. Shunt Impedance	10
2.2.2. Acceleration Cavities	10
2.2.3. Bending Magnets	12
2.2.4. Rhodotron Accelerator	12
2.2.4.1. Cavity of a Rhodotron	13
2.2.4.2. Acceleration cycle of Rhodotron	14
2.3. Numerical Integration Methods	20
2.3.1. Leapfrog	20
2.3.2. Runge Kutta	21
2.4. Simulation	24
2.4.1. Basic Concepts in Programming	24
2.4.1.1. Clock Cycle	24
2.4.1.2. Concurrency	25
2.4.1.3. Object Oriented Programming	26
3. TOOLS	27

3.1. Poisson Superfish	27
3.2. CST Studio Suite	27
3.3. ROOT	27
3.4. gnuplot	28
4. DESIGN	29
4.1. Cavity Design	29
4.2. Magnet Design	34
4.2.1. $n\lambda$ Technique	35
4.2.2. L_{out} Parameter Sweep	37
4.3. Initial Design at KAHVELab	40
5. SIMULATION	47
5.1. Proof of Concept	47
5.2. Intermediate Versions	50
5.2.1. L_{out} Optimization For Single e^-	50
5.2.2. ϕ_{lag} Optimization For Bunches	50
5.2.3. Simulation in 3D	52
5.2.4. Acceleration in Magnetic Field	55
5.2.4.1. RK4-1	57
5.2.4.2. RK4-2	57
5.2.5. Acceleration in Electric Field	60
5.2.6. Multithreading	66
5.3. Graphical User Interface	69
5.3.1. Configuration Frame	71
5.3.2. Simulation Frame	71
5.3.3. Render Frame	71
5.3.4. Analyze Frame	71
5.3.5. Sweep Frame	75
6. PRODUCTION	83
6.1. Final Design	83
6.2. Technical Drawing	87
6.3. Manufacturing	92

7. FUTURE WORK	96
7.1. Cavity	96
7.2. Magnets	96
7.3. Rhodotron Simulation	96
7.4. Conclusion	97
APPENDIX A: Intermediate Codes	98
APPENDIX B: Example Simulation Runs	112
APPENDIX C: Data and Graphs	117
REFERENCES	124

1. INTRODUCTION

2. THEORY

2.1. Accelerating Charged Particles

2.1.1. Relation between momentum and acceleration

In classical mechanics, Newton's second law defines force and states

$$\vec{F} = \frac{\partial \vec{p}}{\partial t} = m \frac{\partial \vec{v}}{\partial t} = m \vec{a} \quad (2.1)$$

In special relativity however, relativistic momentum is defined as $\vec{p} = \gamma m_0 \vec{v}$ where γ is the Lorentz Factor;

$$\gamma = \frac{1}{\sqrt{1 - v^2/c^2}} = \frac{1}{\sqrt{1 - \beta^2}}$$

Considering these two statements, we can find the relation between momentum and acceleration as follows

$$\begin{aligned} \frac{\partial \vec{p}}{\partial t} &= m_0 \frac{\partial(\gamma \vec{v})}{\partial t} = m_0 \left\{ \frac{\partial \gamma}{\partial t} \vec{v} + \gamma \frac{\partial \vec{v}}{\partial t} \right\} \\ \frac{\partial \gamma}{\partial t} &= \gamma^3 \vec{\beta} \cdot \frac{\partial \vec{\beta}}{\partial t} = \frac{\gamma^3}{c} \vec{\beta} \cdot \vec{a} \\ \frac{\partial \vec{p}}{\partial t} &= m_0 \left\{ \frac{\gamma^3}{c} (\vec{\beta} \cdot \vec{a}) \vec{v} + \gamma \frac{\partial \vec{v}}{\partial t} \right\} \end{aligned}$$

$$\vec{F} = \gamma m_0 \left\{ \vec{a} + \gamma^2 (\vec{\beta} \cdot \vec{a}) \vec{\beta} \right\} \quad (2.2)$$

It is clear that acceleration is not necessarily parallel to the force anymore.

To start separating the parallel and perpendicular components relative to $\vec{\beta}$, we

can find $\vec{\mathbf{a}}_{||}$ and $\vec{\mathbf{F}}_{||}$.

$$\vec{\mathbf{a}}_{||} = \frac{(\vec{a} \cdot \vec{\beta})}{\beta^2} \beta \quad \vec{\mathbf{F}}_{||} = \frac{(\vec{F} \cdot \vec{\beta})}{\beta^2} \beta \quad (2.3)$$

$$\begin{aligned} \vec{\mathbf{F}} \cdot \vec{\beta} &= \gamma m_0 \{ \vec{\mathbf{a}} \cdot \vec{\beta} + \gamma^2 (\vec{\beta} \cdot \vec{\mathbf{a}}) \beta^2 \} \\ &= \gamma m_0 (\vec{\mathbf{a}} \cdot \vec{\beta}) \{ \gamma^2 \beta^2 + 1 \} \end{aligned}$$

Using $\gamma^2 \beta^2 + 1 = \gamma^2$ we have,

$$\vec{\mathbf{F}} \cdot \vec{\beta} = m_0 \gamma^3 (\vec{\mathbf{a}} \cdot \vec{\beta})$$

Inserting this

$$\begin{aligned} \vec{\mathbf{F}}_{||} &= \frac{(\vec{\mathbf{F}} \cdot \vec{\beta})}{\beta^2} \vec{\beta} \\ &= m_0 \gamma^3 \frac{(\vec{\mathbf{a}} \cdot \vec{\beta})}{\beta^2} \vec{\beta} \\ &= m_0 \gamma^3 \vec{\mathbf{a}}_{||} \end{aligned} \quad (2.4)$$

Therefore from *equations 2.2 and 2.4*

$$\begin{aligned} \vec{\mathbf{F}} &= m_0 \gamma^3 \vec{\mathbf{a}}_{||} \beta^2 + m_0 \gamma \vec{\mathbf{a}} \\ &= m_0 \gamma^3 \vec{\mathbf{a}}_{||} \beta^2 + m_0 \gamma \{ \vec{\mathbf{a}}_{||} + \vec{\mathbf{a}}_{\perp} \} \\ &= m_0 \vec{\mathbf{a}}_{||} \gamma \{ \gamma^2 \beta^2 + 1 \} + m_0 \gamma \vec{\mathbf{a}}_{\perp} \\ &= m_0 \vec{\mathbf{a}}_{||} \gamma^3 + m_0 \gamma \vec{\mathbf{a}}_{\perp} \\ &= \vec{\mathbf{F}}_{||} + m_0 \gamma \vec{\mathbf{a}}_{\perp} \end{aligned}$$

We finally have two separate equations which are in similar form with *equation 2.1*.

$$\vec{\mathbf{F}}_{||} = \gamma^3 m_0 \vec{\mathbf{a}}_{||} \quad \vec{\mathbf{F}}_{\perp} = \gamma m_0 \vec{\mathbf{a}}_{\perp} \quad (2.5)$$

2.1.2. Lorentz Force

Force acting on a charged particle moving in electromagnetic fields is called Lorentz Force and is given by the formula

$$\frac{\partial \vec{p}}{\partial t} = \vec{F}_L = q(\vec{E} + \vec{v} \times \vec{B}) \quad (2.6)$$

where the q is the charge and \vec{v} is the velocity of the particle.

2.1.3. Relativistic Lorentz Force

Similar to non-relativistic version, relativistic Lorentz Force is given by the following 4-vector equality

$$\frac{\partial p^\mu}{\partial \tau} = q F^{\mu\nu} u_\nu \quad (2.7)$$

Where $\partial\tau = \partial t/\gamma$,

$$p^\mu = \begin{bmatrix} W/c \\ p_x \\ p_y \\ p_z \end{bmatrix} \quad F^{\mu\nu} = \begin{bmatrix} 0 & -E_x/c & -E_y/c & -E_z/c \\ E_x/c & 0 & -B_z & B_y \\ E_y/c & B_z & 0 & -B_x \\ E_z/c & -B_y & B_x & 0 \end{bmatrix} \quad u_\nu = \gamma \begin{bmatrix} c \\ -v_x \\ -v_y \\ -v_z \end{bmatrix} \quad (2.8)$$

Where W is the energy of the particle and γ is the lorentz factor mentioned in the previous section.

For $\mu = 0$, we have the time component of the equation;

$$\frac{\gamma}{c} \frac{\partial W}{\partial t} = \frac{q\gamma \vec{E} \cdot \vec{v}}{c} = \frac{q\gamma}{c} \frac{\vec{E} \cdot \partial \vec{r}}{\partial t} \quad (2.9)$$

$$\frac{\partial W}{\partial t} = q \frac{\vec{E} \cdot \partial \vec{r}}{\partial t} \quad (2.10)$$

This is the definition of work done by an electric field. For $\mu = 1, 2, 3$, we have the spacial components;

$$\frac{\partial \vec{p}}{\partial \tau} = \gamma \frac{\partial \vec{p}}{\partial t} = q\gamma(\vec{E} + \vec{v} \times \vec{B})$$

Which simplifies to non-relativistic Lorentz Force in *equation 2.6*.

2.1.4. Acceleration caused by lorentz force

Due to the nature of the cross product, lorentz force caused by a magnetic field is always perpendicular to the velocity of the particle. Therefore the acceleration of the magnetic field is straightforward

$$\vec{F}_B = \vec{F}_\perp = \gamma m_0 \vec{a}_\perp = \gamma m_0 \vec{a}_B$$

The same thing cannot be said about electric field however. It can create force in any direction with respect to velocity. Therefore, we have the following equality;

$$\vec{a}_{||} = \frac{q}{\gamma^3 m_0} \vec{E}_{||} \quad \vec{a}_\perp = \frac{q}{\gamma m_0} \{ \vec{E}_\perp + \vec{v} \times \vec{B} \} \quad (2.11)$$

Acceleration due to electric field can be simplified as;

$$\begin{aligned} \vec{a}_{\{B=0\}} = \vec{a}_E &= \vec{a}_{||} + \vec{a}_{\perp\{B=0\}} \\ &= \frac{q}{m_0 \gamma} \left\{ \frac{\vec{E}_{||}}{\gamma^2} + \vec{E}_\perp \right\} \\ &= \frac{q}{m_0 \gamma} \{ \{1 - \beta^2\} \vec{E}_{||} + \vec{E}_\perp \} \end{aligned}$$

$$\begin{aligned}
&= \frac{q}{m_0\gamma} \{\vec{\mathbf{E}}_{||} + \vec{\mathbf{E}}_{\perp} - \beta^2 \vec{\mathbf{E}}_{||}\} \\
&= \frac{q}{m_0\gamma} \{\vec{\mathbf{E}} - \beta^2 \vec{\mathbf{E}}_{||}\}
\end{aligned}$$

Using the fact that $\vec{\mathbf{E}}_{||} = \vec{\beta}(\vec{\mathbf{E}} \cdot \vec{\beta})/\beta^2$, we finally have

$$\vec{\mathbf{a}}_E = \frac{q}{\gamma m_0} \left\{ \vec{\mathbf{E}} - \vec{\mathbf{v}} \frac{(\vec{\mathbf{E}} \cdot \vec{\mathbf{v}})}{c^2} \right\} \quad \vec{\mathbf{a}}_B = \frac{q}{\gamma m_0} (\vec{\mathbf{v}} \times \vec{\mathbf{B}}) \quad (2.12)$$

2.2. Particle Accelerators

Particle accelerators are sophisticated scientific instruments designed to accelerate charged particles, such as electrons, protons, or ions, to high speeds and energies. These accelerators play a crucial role in advancing our understanding of the fundamental properties of matter and the universe. They are widely used in various fields of research, including particle physics, nuclear physics, materials science, and medicine.

At their core, particle accelerators utilize electromagnetic fields to impart energy to particles and control their trajectories. These fields are generated by intricate arrangements of magnets and RF (radiofrequency) cavities within the accelerator structure. By precisely controlling these fields, accelerators can propel particles to speeds close to the speed of light, enabling them to acquire high energies.

Accelerators can be categorized into two main types: linear accelerators (linacs) and circular accelerators. Linacs accelerate particles in a straight line, while circular accelerators use magnetic fields to bend the particle trajectory into a circular path.

The acceleration process in accelerators involves multiple stages. Initially, particles are injected into the accelerator at a relatively low energy. As they progress through the accelerator, they are subjected to alternating electric fields that accelerate them, while magnetic fields guide their trajectories. Focusing elements, such as magnetic lenses or quadrupole magnets, ensure the particles remain tightly controlled.

As particles gain energy in the accelerator, they approach relativistic speeds, where relativistic effects become significant. Special relativity governs the increase in mass and energy of the particles as they approach the speed of light, providing insights into the behavior of matter at high energies.

Particle accelerators are essential tools for scientific research. They enable scientists to probe the fundamental constituents of matter, study particle interactions, and

explore the laws of physics. Accelerators have been instrumental in discovering and characterizing fundamental particles, such as quarks, leptons, and the Higgs boson. They also facilitate the production of high-intensity beams for applications in material science, radiotherapy, and industrial processes like particle irradiation and sterilization.

***Image
Coming
Soon***

Figure 2.1: A linear proton accelerator in KAHVELab

2.2.1. Key Concepts

2.2.1.1. Resonance Frequency. Resonance frequency is the frequency in which the electromagnetic fields form standing waves in a cavity. It is determined by the physical dimensions and the speed of light in the cavity's medium. In a rectangular cavity for example, the resonance frequency can be calculated from *equation 2.13*.

$$f_{klm} = \frac{c}{2\pi\sqrt{\mu_r\epsilon_r}} \sqrt{\left(\frac{k}{w}\right)^2 + \left(\frac{l}{u}\right)^2 + \left(\frac{m}{v}\right)^2} \quad (2.13)$$

Where w , u , v are the dimensions of the cavity, μ_r and ϵ_r are relative permability and permittivity of the cavity respectably.

2.2.1.2. Bunch. Bunch refers to a tightly grouped collection of charged particles, such as electrons or protons, that are accelerated and maintained close together within a particle accelerator.

2.2.1.3. Phase Stability. Phase stability refers to the preservation of the timing relationship between particles and fields within an accelerator. It ensures that the phases of various electromagnetic fields or particles remain synchronized, which is crucial for achieving efficient particle acceleration. Maintaining phase stability is essential to prevent particles from becoming out of phase as they travel through accelerator structures, ensuring that they receive the correct energy boosts and interact predictably with detectors. Deviations in phase stability can lead to particle loss and decreased beam quality.

2.2.1.4. Phase Lag. Phase lag refers to the time delay between the oscillations of two interacting waveforms or particles. It describes the difference in phase angles within their respective cycles, between two signals.

2.2.1.5. Shunt Impedance. The shunt impedance of an RF accelerator is a measure of the efficiency at which the accelerator can transform the supplied RF power into acceleration.

2.2.2. Acceleration Cavities

Radiofrequency (RF) cavities, also known as accelerating cavities or resonant cavities, are key components in particle accelerators. These cavities generate strong electromagnetic fields at specific frequencies to accelerate charged particles through clever engineering.

RF cavities are typically hollow metallic structures made of or coated with high-conductivity materials such as copper. They are designed to resonate at a specific frequency, which is determined by the size and shape of the cavity. The cavity is often cylindrical or spherical in shape, and its inner surface is polished to minimize energy losses through resistive heating. The RF cavity is designed to be resonant, meaning that it naturally amplifies the electric fields at its resonant frequency. The resonant

frequency is determined by the cavity's dimensions and the speed of light in the cavity material.

To achieve efficient energy transfer to the particles, the RF cavity is driven by an external RF power source operating at the resonant frequency. The power source supplies radiofrequency energy to the cavity, which causes the electric fields inside the cavity to oscillate at the desired frequency. These oscillating fields then transfer energy to the passing particles, increasing their kinetic energy by pushing and pulling on the charged particles as they pass through the cavity.

In addition to accelerating the particles, RF cavities are often designed to provide focusing forces. By carefully shaping the cavity and adjusting the electromagnetic fields, the particles can experience focusing effects as they pass through the cavity. This helps to maintain a tight and controlled beam. To ensure efficient acceleration, it is essential to maintain phase stability. This means that the particles should experience the strongest electric fields at the correct time during their passage through the cavity. Precise timing and synchronization of the RF power source with the particle beam are crucial to achieve phase stability and maximize energy transfer.

*Image
Coming
Soon*

Figure 2.2: A rf cavity used in KAHVELab

2.2.3. Bending Magnets

Bending magnets, also known as dipole magnets, are fundamental components used in particle accelerators to control the trajectory of charged particles. They utilize the Ampere's Law to exert a magnetic field that interacts with the charged particles in the accelerator.

According to the Lorentz Force Law (*Section 2.1.2*), when a charged particle moves through a magnetic field, it experiences a force perpendicular to both its velocity vector and the magnetic field direction. This force causes the particle's trajectory to curve, resulting in a bending effect.

***Image
Coming
Soon***

Figure 2.3: A bending magnet used in KAHVELab

2.2.4. Rhodotron Accelerator

Rhodotron Accelerator is a type of particle accelerator that was proposed by *Jacques POTTIER* in his publication *A new type of rf electron accelerator: The Rhodotron* in 1989 [1]. First prototype was built at CEA Saclay later in 1992 [2]. Its named after the greek word for rose, *rhodos*, due to the shape of the design [3].

The design of a rhodotron mainly consists of a coaxial cylindrical RF cavity and bending magnets surrounding it. RF cavity is fed by an external RF source, accelerating the electrons entering from an attached electron gun.

2.2.4.1. Cavity of a Rhodotron. Coaxial design of the cavity concentrates the electric field, while the magnetic field diminishes in the middle of the cylinders. Therefore the electrons are injected and accelerated in the plane of zero magnetic field where the electric field is strongest.

With the expectation that the electrons will accelerate to speeds $\geq 0.9c$ after the first pass, a rhodotron cavity is designed so that the length of the path between successive passes is an integer multiple of λ , wavelength of the RF field ($l = p\lambda$). This constraint helps with phase stability and synchronization of the beam.

In the table below, optimized characteristics of a rhodotron cavity can be observed.

Table 1
Optimized characteristics

P	R_2 (m)	R_1/R_2	Z_{se} (MΩ)	Z_{sp} (MΩ)
1	0.27λ	$1/4$	$5.77\lambda^{1/2}$	$4.9\lambda^{1/2}$
2	0.5λ	$1/7$	$10.4\lambda^{1/2}$	$8.83\lambda^{1/2}$

Figure 2.4: Optimized characteristics of a rhodotron cavity [1]

Here, p is the integer multiplier in the equation ($l = p\lambda$) mentioned above, R_1 is the radius of the inner cylinder, R_2 is the radius of the outer cylinder, Z_{se} is effective shunt impedance, Z_{sp} is practical shunt impedance which was taken to be $0.85Z_{se}$.

Energy gain after each pass can be calculated by the following formula [1]

$$W_1 = Z_{sp}^{1/2} P^{1/2} \cos\phi \quad (2.14)$$

Where P is the dissipated power, ϕ is the phase lag, taken as 15° [1].

Considering $Z_{sp} \propto \lambda^{1/2}$, $W_1 \propto \lambda^{1/4}$, $V \propto \lambda^3$, where V is the volume of the cavity, implementing the $p = 1$ design in *figure 2.4* is much more space efficient.

Table 2
Energy W (MeV) for $P = 100$ kW and $f = 130$ MHz

R_2 (m)	2	3	4	5	6	7	8	9	10	11	12
0.62	1.8	2.7	3.6	4.5	5.4	6.3	7.2	8.1	9	9.9	10.8
1.15	2.4	3.6	4.8	6	7.2	8.4	9.6	10.8	12	13.2	14.4

Figure 2.5: Energy of a synchronous electron after each pass for both $p = 1$ and $p = 2$ [1]

Total energy gain after n passes W can then be found by *equation 2.14*, taking $\phi = 15^\circ$, $p = 1$, i.e $R_2 = 0.27\lambda$.

$$W \approx 2.14\lambda^{1/4}P^{1/2}n \text{ keV} \quad (2.15)$$

2.2.4.2. Acceleration cycle of Rhodotron. Electrons undergo four different stages during a single pass. They are accelerated between the cylindrical plates and are shielded from the changing RF field while inside the inner cylinder and outside the cavity.

First acceleration Electrons in the rhodotron cavity are accelerated by the electric field created between two coaxial cylinders, towards the inner cylinder when they are ejected into the cavity. See *figure 2.6*

Inner shielding While inside the inner cylinder, the cylinder acts as a faraday cage and shields the electrons inside while the electric field is being reversed. See *figure 2.7*

Second acceleration Once the electrons leave the inner cylinder, they accelerate towards the outer cylinder by the reversed electric field until they leave the cavity.

See *figure 2.8*

Magnets After leaving the cavity, an electromagnet placed in their path steers the electrons back into the cavity in which time the electric field changes the direction again. See *figure 2.9*

This cycle can be repeated as long as real world constraints such as; placements and dimensions of the electromagnets, power requirements due to increasing magnetic field in order for sharper turns, can be overcomed. After desired amount of pass has been completed, the electrons exit the accelerator.

This process is explained further in the following figures where T is the period of the electric field.

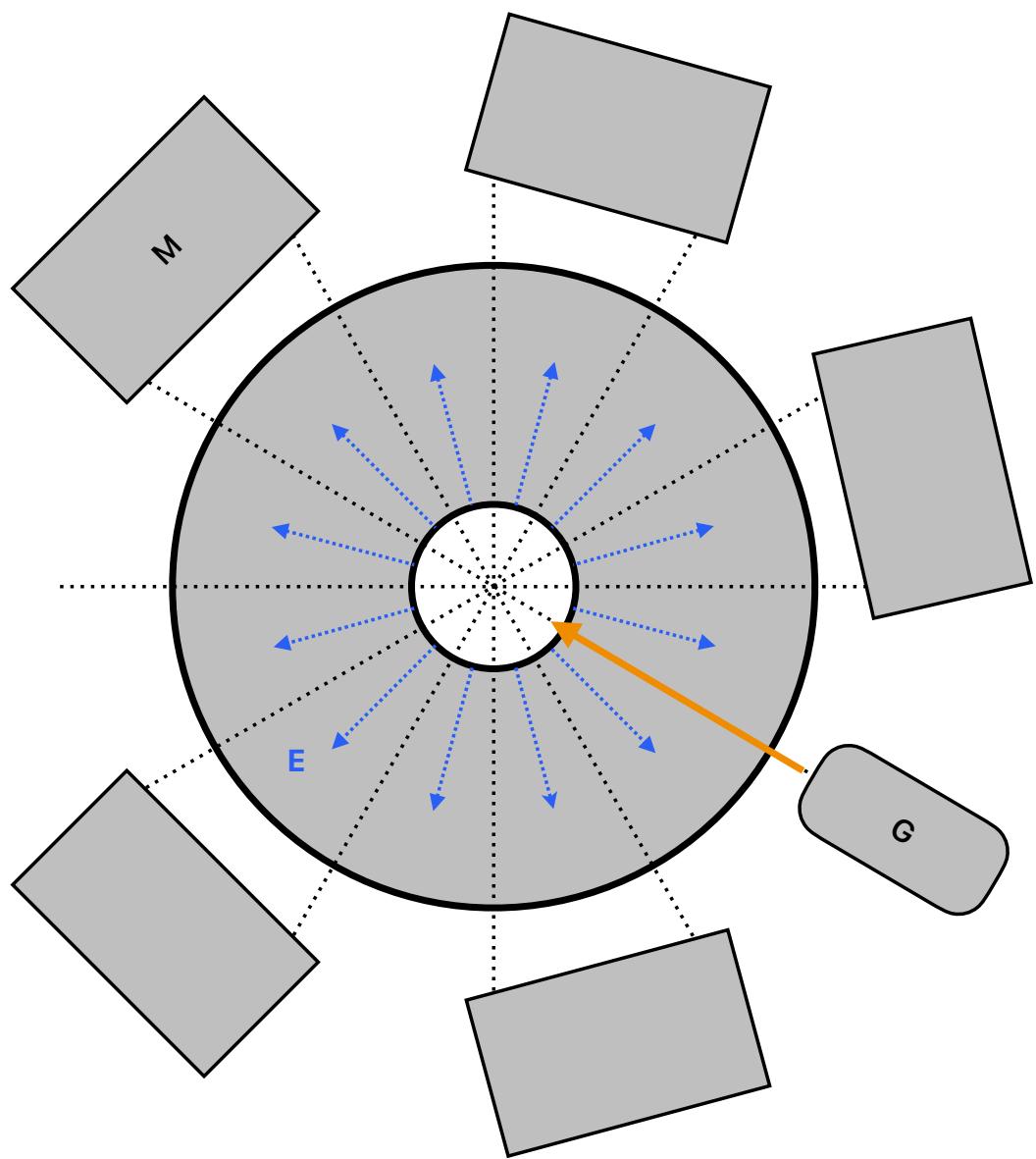


Figure 2.6: $[0, \frac{T}{4}]$ time frame of a rhodotron

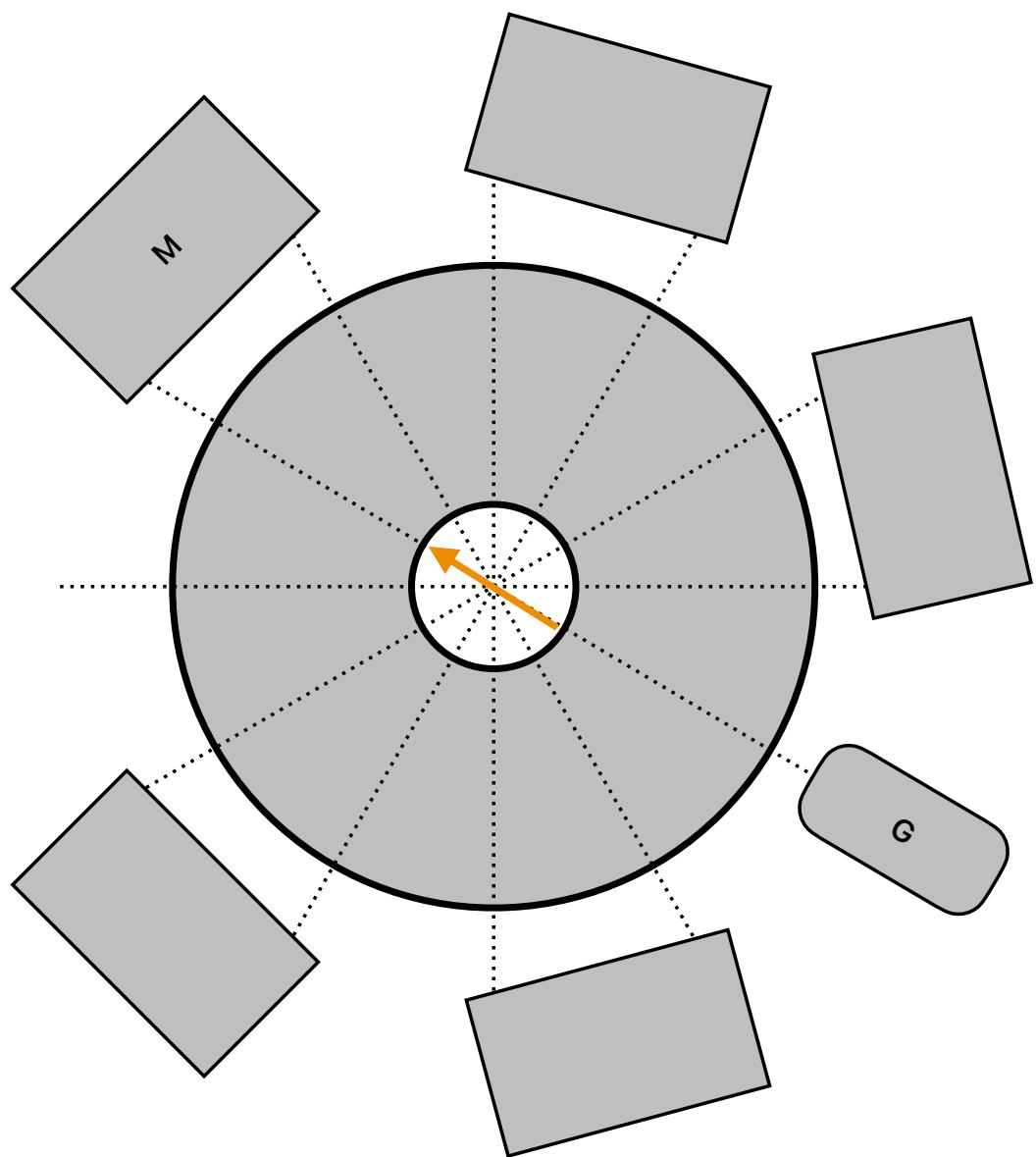


Figure 2.7: $[\frac{T}{4}, \frac{T}{2}]$ time frame of a rhodotron

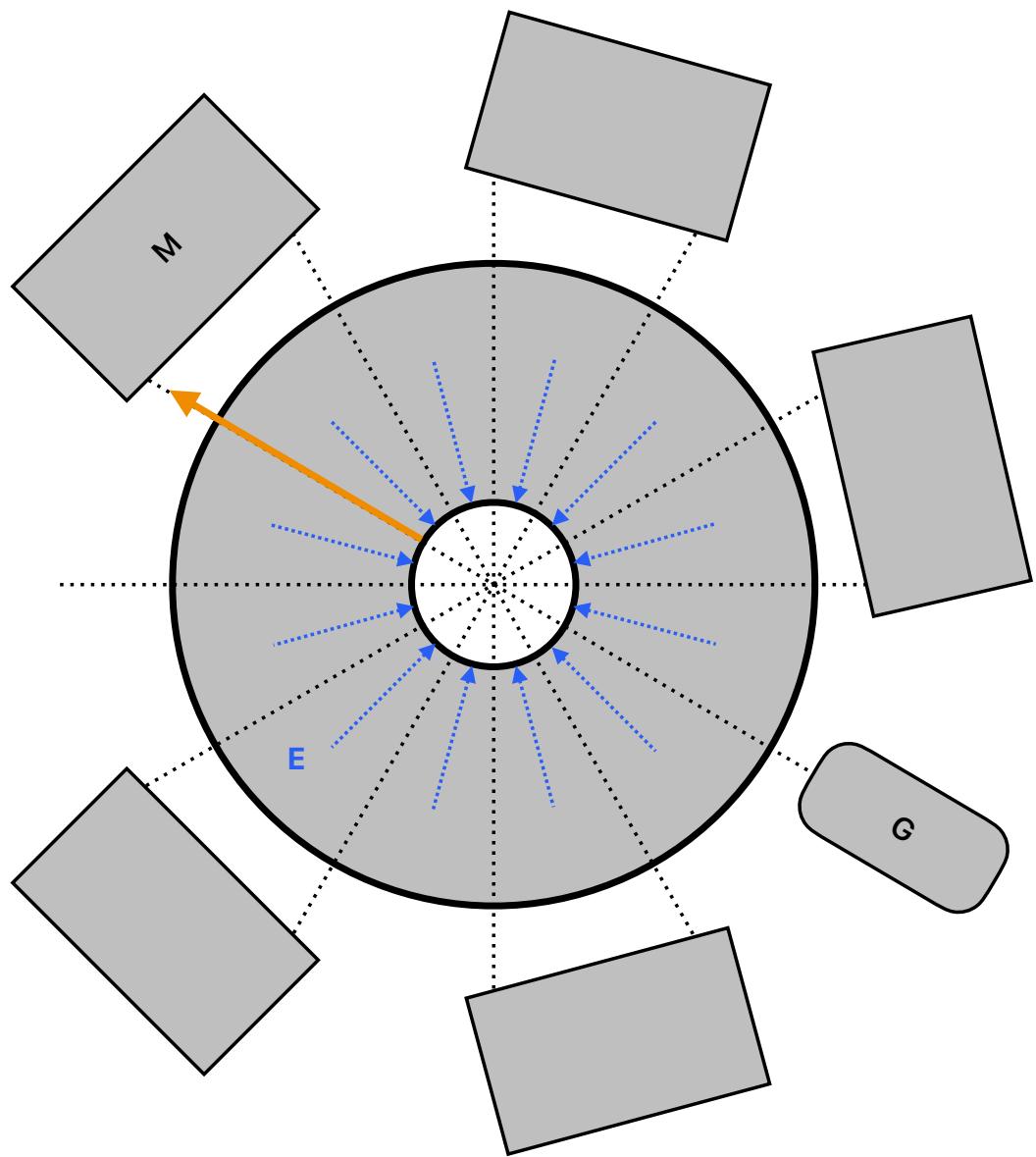


Figure 2.8: $[\frac{T}{2}, \frac{3T}{4}]$ time frame of a rhodotron

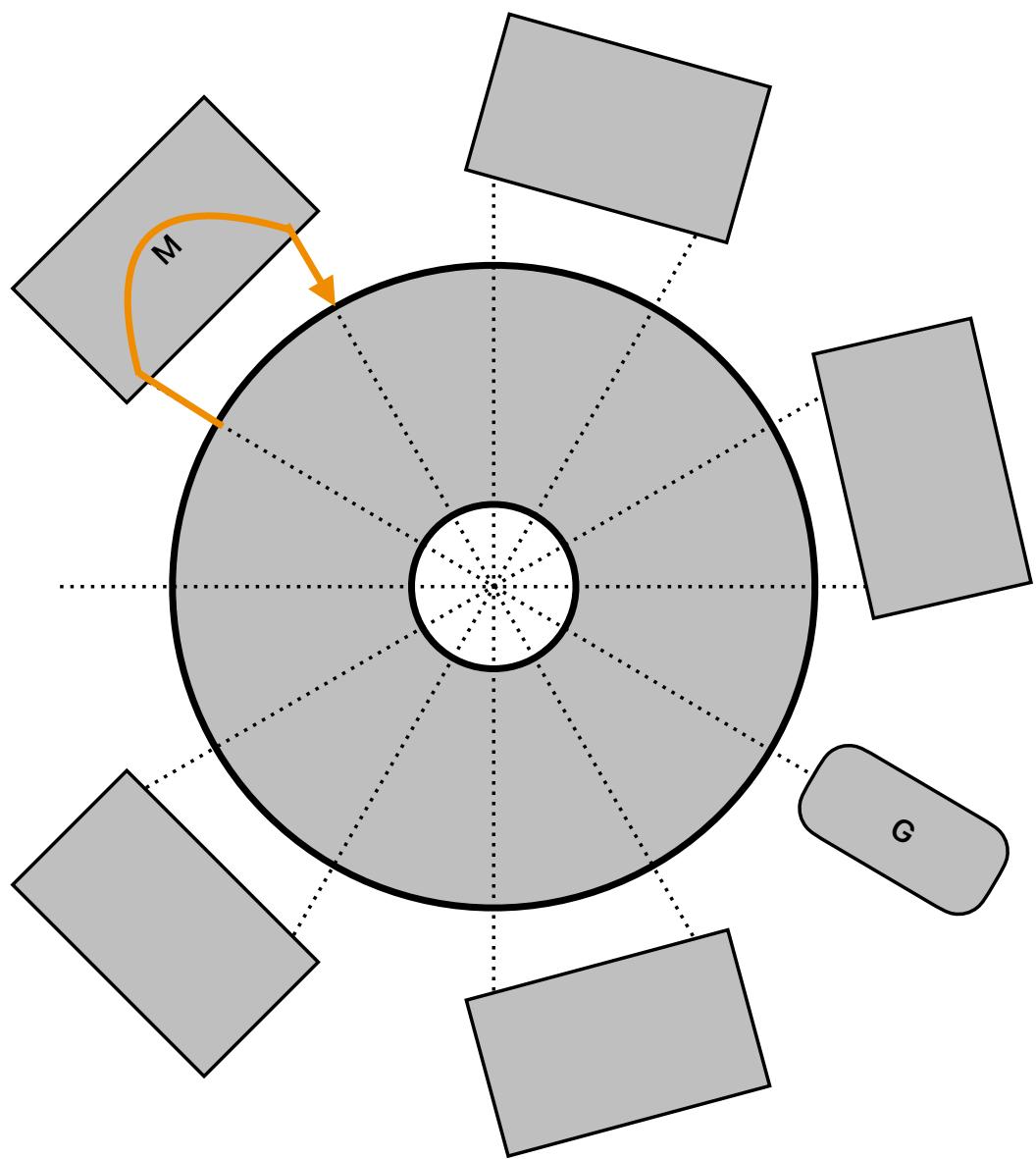


Figure 2.9: $[\frac{3T}{4}, T]$ time frame of a rhodotron

2.3. Numerical Integration Methods

2.3.1. Leapfrog

The Leapfrog method is a numerical method commonly used to solve ordinary differential equations (ODEs) that involve second-order time derivatives. Such an ODE is shown below:

$$\ddot{x} = \frac{d^2x}{dt^2} = f(x) \quad (2.16)$$

The Leapfrog method is a variant of the finite difference method, and it approximates the solution of an ODE by discretizing both time and space. The method gets its name from the way it calculates the values of the solution at each time step, which resembles a "leapfrogging" motion. It is a simple and efficient algorithm that is often used in simulations of physical systems, such as celestial mechanics or molecular dynamics. The idea is straight forward; in the time interval Δt ,

$$\begin{aligned} a(t_0) &= f(x_0) \\ x(t_0 + \Delta t) &= x(t_0) + v(t_0)\Delta t + a(t_0)\frac{\Delta t^2}{2} \end{aligned} \quad (2.17)$$

$$v(t_0 + \Delta t) = v(t_0) + \{a(t_0) + a(t_0 + \Delta t)\}\frac{\Delta t}{2} \quad (2.18)$$

For more stability, this version can be rearranged to what is called 'kick-drift-kick' form,

$$\begin{aligned} v(t_0 + \Delta t/2) &= v(t_0) + a(t_0)\frac{\Delta t}{2} \\ x(t_0 + \Delta t) &= x(t_0) + v(t_0 + \Delta t/2)\Delta t \end{aligned} \quad (2.19)$$

$$v(t_0 + \Delta t) = v(t_0 + \Delta t/2) + a(t_0 + \Delta t) \frac{\Delta t}{2}$$

This version provides more time resolution to our calculation; however, it increases the number of calculations needed by about 50%.

2.3.2. Runge Kutta

The Runge-Kutta methods, named after the German mathematicians Carl Runge and Martin Wilhelm Kutta, family of numerical methods used to solve ordinary differential equations (ODEs) that are in the following form

$$\frac{\partial y}{\partial x} = f(x, y) \quad (2.20)$$

The basic idea behind the Runge-Kutta method is to approximate the solution of an ODE by taking small steps and using a weighted average of function evaluations at different points within each step.

$$y_{n+1} = y_n + \delta x \sum_{i=1}^m b_i k_i \quad x_{n+1} = x_n + \delta x \quad (2.21)$$

where

$$k_i = f(x_n + c_i \delta x, y_n + \delta x \sum_{j=1}^{i-1} a_{ij} k_j) \quad (2.22)$$

These *equations 2.21 and 2.22* define the family of methods. To specify a pertic-

ular method, order m , coefficients a_{ij} , b_i and c_i should be provided. The coefficients of any Runge-Kutta method can be visualized by a tableau called *Butcher Tableau*.

0					
c_1	a_{21}				
c_2	a_{31}	a_{32}			
\vdots	\dots				
c_m	a_{m1}	a_{m2}	\dots	$a_{m,m-1}$	
	b_1	b_2	\dots	b_m	

Figure 2.10: Butcher Tableau

The simplest Runge-Kutta method is the *Euler's method*. Its *Butcher tableau* is as follows.

0			
		1	

The most commonly used version of the Runge-Kutta method is the fourth-order Runge-Kutta method, also known as RK4. The RK4 method involves four function evaluations per step and has an error term that is proportional to the step size raised to the fifth power. Two of the most used *Butcher tableaus* for RK4 given below.

0					
$1/2$	$1/2$				
$1/2$	0	$1/2$			
1	0	0	1		
	1/6	1/3	1/3	1/6	
0					
$1/3$	$1/3$				
$2/3$	$-1/3$	1			
1	1	-1	1		
	1/8	3/8	3/8	1/8	

Figure 2.11: Butcher Tableau for RK4

The second tableau in *figure 2.11* is called the "*3/8 rule*". Its main advantage is that its error coefficients are smaller than the other. But it costs more floating point operations per step. Resulting in slower calculations.

2.4. Simulation

A simulation software is a computer program or tool that enables the creation and execution of simulations to model and analyze real-world systems or processes. It allows users to replicate the behavior, interactions, and outcomes of the system or process under study, providing insights and predictions that can be valuable for decision-making, optimization, or understanding complex phenomena.

Simulation software provides a virtual environment where users can define the parameters, variables, and rules of the system being simulated. The software then uses mathematical and physical models, algorithms, and computational techniques to simulate the behavior of the system over time.

2.4.1. Basic Concepts in Programming

2.4.1.1. Clock Cycle. In programming, a clock cycle refers to a fundamental unit of time measurement used in computer systems. It represents the basic rhythm or timing mechanism of a computer's central processing unit (CPU) and is typically measured in terms of the CPU's clock speed, expressed in hertz (Hz).

A clock cycle represents one complete pulse or oscillation of the CPU's clock signal. It serves as a synchronization mechanism, coordinating the execution of instructions and the timing of various operations within the CPU. Each clock cycle is associated with a specific duration, which is determined by the clock speed of the CPU.

The concept of clock cycles is often used when considering the performance and efficiency of algorithms and code. Since the execution time of instructions is influenced by the number of clock cycles required, minimizing the number of clock cycles needed for a program or algorithm can lead to faster and more efficient code execution. Table below shows the amount of clock cycles required for some mathematical calculations in various processors.

MEASURED CPU TIMES OF BASIC MATHEMATICAL OPERATIONS							
OPERATION	DOUBLE PRECISION					QUADRUPLE PRECISION	
	i486	Pentium	PA-7000	μ SPARC-II	Alpha	PA-7000	μ SPARC-II
$x = y \dots$	0.46	0.24	1.00	1.00	0.41	5.1	2.8
$x = y + z \dots$	0.91	1.00	1.00	1.00	1.00	131	11.3
$x = yz \dots$	1.09	1.00	1.00	1.00	1.00	457	99.5
$x = y/z \dots$	2.81	4.49	4.92	3.32	6.59	1100	168
$x = y^{1/2} \dots$	12.0	19.6	4.95	24.4	38.5	1100	149
$x = \sin y \dots$	19.8	23.3	49.8	22.0	18.5	9510	1740
$x = \log y \dots$	20.2	23.9	47.8	31.9	22.0	5420	1240
$x = \tan^{-1} y \dots$	19.8	27.3	46.3	37.3	23.5	5930	1270
$x = \tan y \dots$	18.0	28.3	63.2	30.0	35.1	8450	2040
$x = \exp y \dots$	24.9	32.7	42.0	19.0	23.2	6560	1370

Figure 2.12: Amount of clock cycle for mathematical operations in various processors.
[4]

Note that *division* is by far the most time consuming basic mathematical operation between addition, subtraction and multiplication.

2.4.1.2. Concurrency. Concurrency in programming refers to the ability of a program to execute multiple tasks or processes simultaneously. It allows different parts of a program to make progress independently, potentially improving performance, responsiveness, and resource utilization.

Concurrency in single core can be achieved by implementing clever scheduling of list of operations called *threads*. This results in non-blocking execution of multiple threads, but only one thread would be executed at any given time. True concurrency on the other hand, can only be achieved by using multiple cores. Each core would be able to execute one thread at any time.

Programs that implement concurrency using multiple cores executes more operations per clock cycle. However, this does not directly lead to an improved performance due to the heavy burden of scheduling and managing multiple threads.

Threads utilize *mutexes*, short for "mutual exclusion," to synchronize and manage access to shared resources in a multi-threaded or multi-process environment.

2.4.1.3. Object Oriented Programming. Object-Oriented Programming (OOP) is a programming paradigm that revolves around the concept of objects. Definition called class serves as a blueprint that defines the structure and behavior of objects. Each object encapsulates both data, and the functions that manipulate that data. This encapsulation promotes modular code design and enhances data security by controlling access to the object's internal state.

In OOP, *inheritance* enables the creation of new classes based on existing ones, facilitating code reuse and hierarchy. *Polymorphism* allows objects of different classes to be treated uniformly through a common interface, enhancing flexibility and extensibility. By providing abstraction, OOP simplifies complex systems by focusing on essential features and interactions, making software development more manageable, scalable, and maintainable.

3. TOOLS

3.1. Poisson Superfish

Poisson Superfish is a software package developed by the *Los Alamos National Laboratory* for the design and analysis of electromagnetic fields, particularly those related to particle accelerators and high-energy physics experiments. It is commonly used in the field of accelerator physics to simulate and optimize the behavior of charged particle beams as they pass through various electromagnetic structures, such as cavities and magnets.

3.2. CST Studio Suite

CST Studio Suite is a powerful software package developed by *Simulia* for electromagnetic simulation and analysis. It's widely used in various industries, including electronics, telecommunications, automotive, aerospace, and more, to design and optimize electromagnetic devices and systems. The software provides a comprehensive set of tools for simulating and analyzing the behavior of electromagnetic fields and their interactions with different materials, structures and charged particles.

3.3. ROOT

ROOT is a widely used open-source software framework developed by CERN (European Organization for Nuclear Research) for data analysis, visualization, and storage in the field of high-energy physics, particularly in experiments conducted at particle accelerators like the Large Hadron Collider (LHC). It has become an essential tool in particle physics research, and it's used by physicists around the world to analyze and interpret data.

3.4. gnuplot

GNUPLOT is a popular open-source software used for creating and visualizing data plots and graphs. It is widely utilized in various fields, including scientific research, data analysis, engineering, and more, to represent and analyze data in a graphical format.

4. DESIGN

4.1. Cavity Design

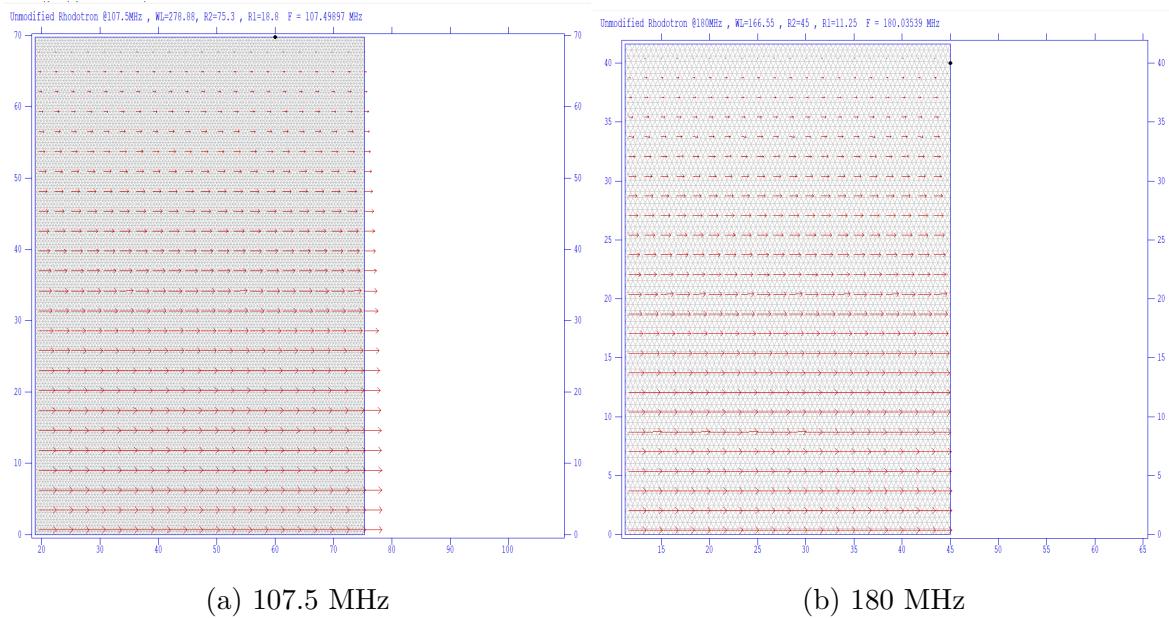
The first and the most important design parameter for a cavity is the operating RF frequency. After an operating RF frequency is set and the desired $R1/R2$ relation *figure 2.4* is selected, design parameters of acceleration plane of the cavity is fully determined.

By following the cylindrical design mentioned in *section 2.2.4*, the only main design parameter remaining is the height of the cylindrical cavity. This parameter can be found using the constraint mentioned in *section 2.2.2*; the fact that operating RF frequency must be equal to resonant frequency of the cavity. For simple coaxial cavity, the height should be $\lambda/2$, where λ is the wavelength of the external RF supply [1]. Simulation tools such as *CST Studio*, *Poisson SUPERFISH* can be used to confirm this condition.

In the following examples, $p = 1$ from *figure 2.4* was used, and *it will be the focus of all further calculations*. Using $f_{RF} = 107.5$ MHz, $f_{RF} = 180$;

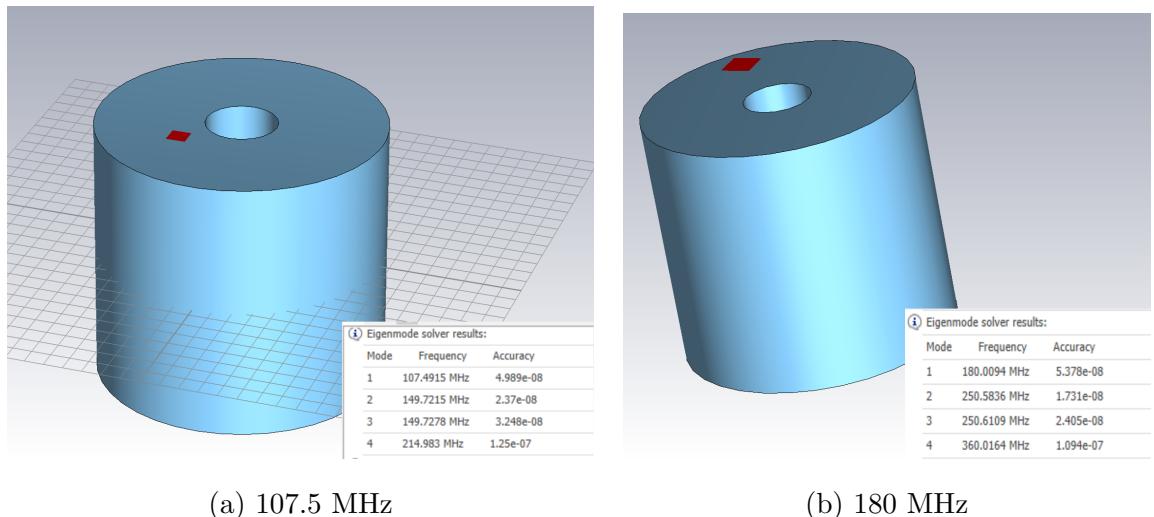
$$\begin{aligned}
 f_{107.5} &= 107.5 \text{ MHz} & f_{180} &= 180 \text{ MHz} \\
 \lambda_{107.5} &= \frac{c}{f_{107.5}} = 2.789m & \lambda_{180} &= \frac{c}{f_{180}} = 1.666m \\
 R_2 &= 0.27 \times \lambda_{107.5} = 0.753m & R_2 &= 0.27 \times \lambda_{180} = 0.450m \\
 R_1 &= \frac{R_2}{4} = 0.188m & R_1 &= \frac{R_2}{4} = 0.113m \\
 h &= \frac{\lambda}{2} = 1.394m & h &= \frac{\lambda}{2} = 0.833m
 \end{aligned} \tag{4.1}$$

In the following figures, *Poisson Superfish* and *CST* simulations of two cavities defined by *equation 4.1* can be observed.



(a) 107.5 MHz

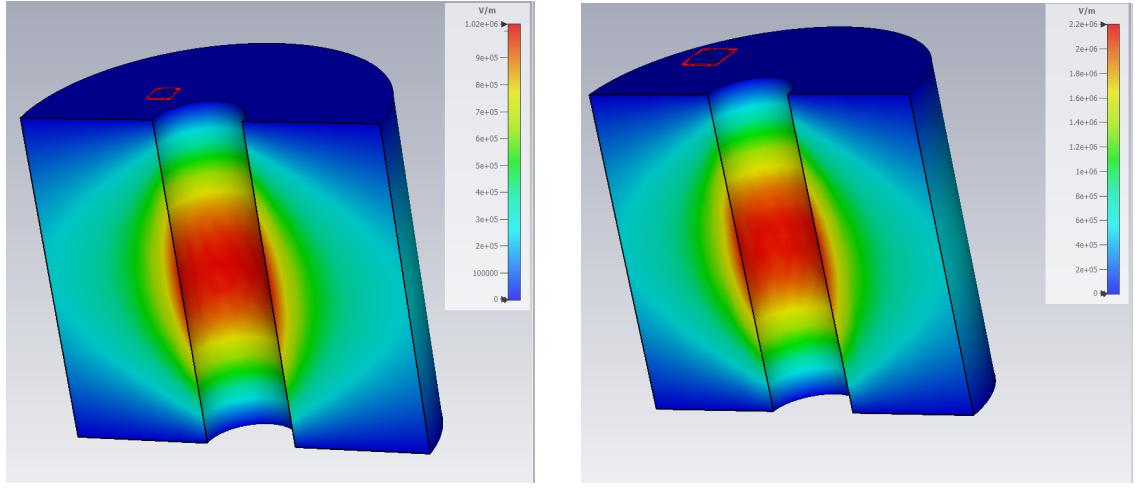
(b) 180 MHz

Figure 4.1: Poission Superfish results for *equation 4.1*

(a) 107.5 MHz

(b) 180 MHz

Figure 4.2: CST Eigenmode results for *equation 4.1*

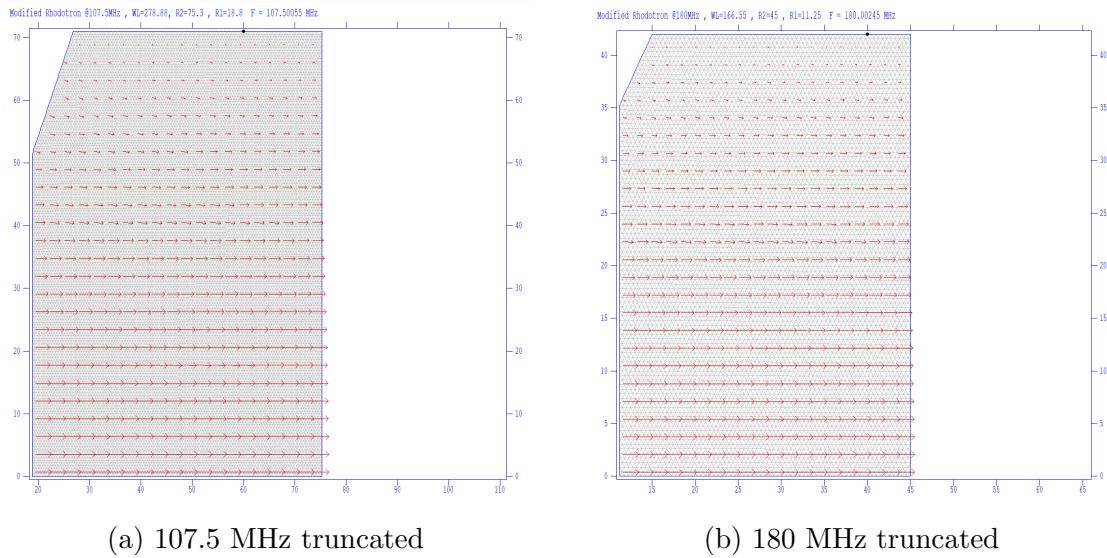


(a) 107.5 MHz

(b) 180 MHz

Figure 4.3: CST Electric field solutions for *equation 4.1*

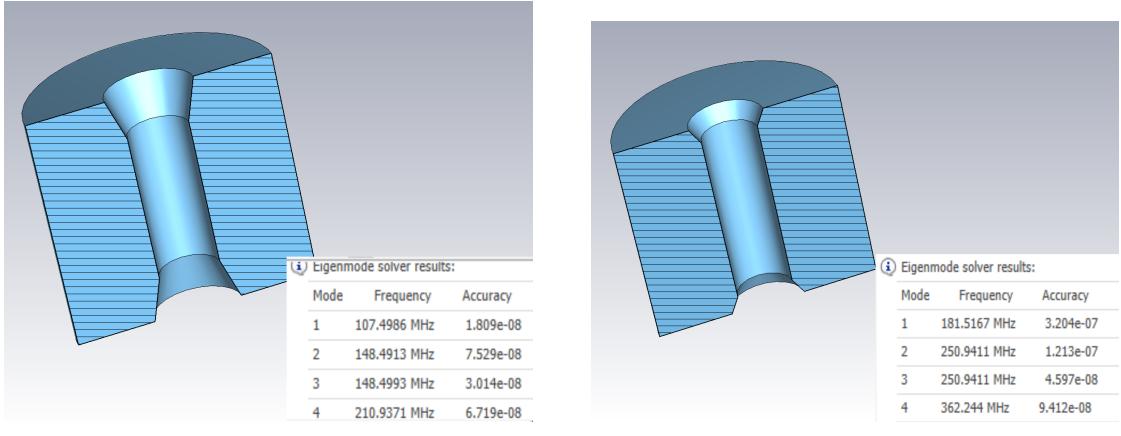
As mentioned by POTTIER, truncated cone terminations in inner cylinder can improve the shunt impedance Z of the cavity [1]. Below, *Poisson superfish* results of such a modification with matching height increase to keep resonant frequency can be found.



(a) 107.5 MHz truncated

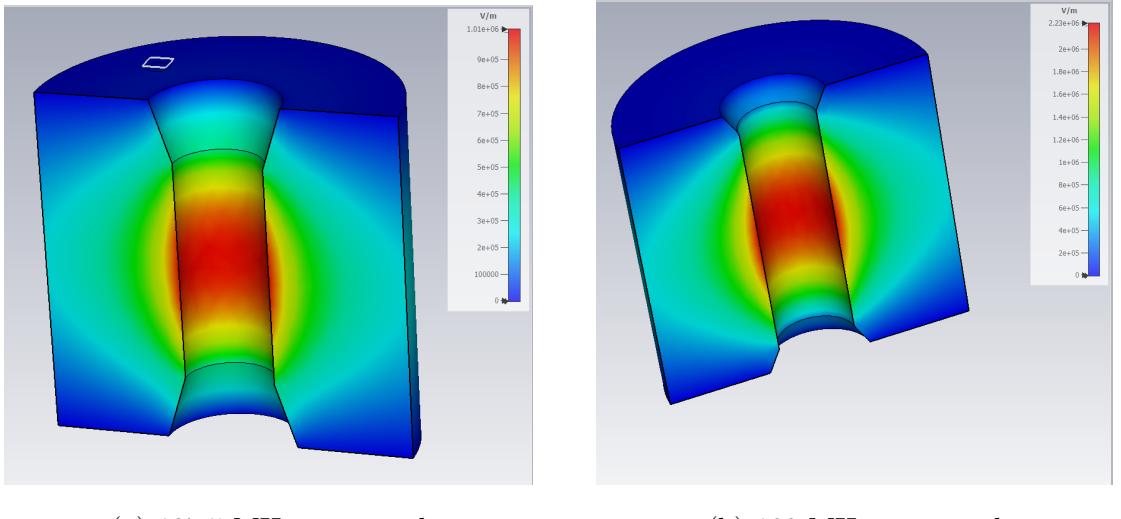
(b) 180 MHz truncated

Figure 4.4: Poisson SUPERFISH field results of truncated cavities



(a) 107.5 MHz truncated (b) 180 MHz truncated

Figure 4.5: CST eigenmode results of truncated cavities



(a) 107.5 MHz truncated (b) 180 MHz truncated

Figure 4.6: CST electric field results of truncated cavities

All calculated values below refer to the mesh geometry only.		All calculated values below refer to the mesh geometry only.
Field normalization (NORM = 0): EZERO = 1.00000 MV/m		Field normalization (NORM = 0): EZERO = 1.00000 MV/m
Frequency = 107.49897 MHz		Frequency = 180.09530 MHz
Normalization factor for E0 = 1.0000 MV/m = 1000.202		Normalization factor for E0 = 1.0000 MV/m = 998.526
Stored energy = 0.0087215 Joules/cm		Stored energy = 0.0031127 Joules/cm
Using standard room-temperature copper.		
Surface resistance = 2.78407 milliohm		Surface resistance = 3.50058 milliohm
Normal-conductor resistivity = 1.72410 microohm-cm		Normal-conductor resistivity = 1.72410 microohm-cm
Operating temperature = 20.0000 C		Operating temperature = 20.0000 C
Power dissipation = 120.2895 W/cm		Power dissipation = 92.9927 W/cm
Q = 48972.0 Shunt impedance = 4697.002 Mohm/m		Q = 37863.7 Shunt impedance = 3629.317 Mohm/m
r/Q = 54190.222 Ohm Wake loss parameter = 9.15951 V/pC		r/Q = 32350.861 Ohm Wake loss parameter = 9.14850 V/pC
Average magnetic field on the outer wall = 2654.74 A/m, 953.183 miliGauss		Average magnetic field on the outer wall = 2655.62 A/m, 1.23436 W/cm^2
Maximum H (at X,Y = 61.05,69.72) = 2654.52 A/m, 953.03 miliGauss		Maximum H (at X,Y = 43.95,41.63) = 2654.93 A/m, 1.23372 W/cm^2
Maximum E (at X,Y = 75.3,0.0) = 0.999999 MV/m, 0.085835 KiloV.		Maximum (at X,Y = 45.8,0.0) = 0.999992 MV/m, 0.070717 KiloV.
Ratio of peak fields Bmax/Emax = 3.3359 mT/(MV/m)		Ratio of peak fields Bmax/Emax = 3.3368 mT/(MV/m)
Peak-to-average ratio Emax/E0 = 1.0000		Peak-to-average ratio Emax/E0 = 0.9998
Wall segments:		Wall segments:
Segment Xend (cm) Yend (cm) Emax (MV/m) Power (W) P/A (W/cm^2) dF/dX (MHz/mm) dF/dY (MHz/mm)		Segment Xend (cm) Yend (cm) Emax (MV/m) Power (W) P/A (W/cm^2) dF/dX (MHz/mm) dF/dY (MHz/mm)
1 18.800 0.0000 0.9999 33.22 476.5 1.1154E-06 0.000		1 11.250 0.0000 0.9998 25.68 0.6168 1.1704E-05 0.000
2 75.300 69.720 0.9999 7.2838E-03 53.85 953.0 0.000 -0.1542		2 45.000 41.630 0.9998 1.5343E-02 41.64 1.234 0.000 -0.4325
3 75.300 0.0000 1.000 33.22 476.5 1.1140E-06 0.000		3 45.000 0.0000 0.9998 25.68 0.6168 1.1756E-05 0.000
Total 120.3		Total 92.99

(a) 107.5 MHz unmodified (b) 180 MHz unmodified

Figure 4.7: Poisson Superfish calculations with *unmodified* cavities

```

-----  

; All calculated values below refer to the mesh geometry only.  

; Field normalization (NORM = 0): EZERO = 1.00000 MV/m  

; Frequency = 107.50055 MHz  

; Normalization factor for E0 = 1.0000 MV/m = 10177.202  

; Stored energy = 0.0007482 Joules/cm  

; Using standard room-temperature copper.  

; Surface resistance = 2.70499 milliohm  

; Normal-conductor resistivity = 0.72x10 microOhm-cm  

; Operating temperature = 20.0000 C  

; Power dissipation = 117.3652 W/cm  

; Q = 50246.6 Shunt impedance = 4914.034 Mohm/m  

; r/Q = 54024.066 Ohm Wake loss parameter = 9.12258 V/pC  

; Average magnetic field on the outer wall = 2685.64 A/m, 975.509 mW/cm^2  

; Maximum H (at X,Y = 73.55,71) = 2707.68 A/m, 991.587 mW/cm^2  

; Maximum E (at X,Y = 75.3,0.0) = 1.00401 MV/m, 0.088182 Kilp.  

; Ratio of peak fields Bmax/Emax = 3.3890 mT/(MV/m)  

; Peak-to-average ratio Emax/E0 = 1.0040  

;  

; Wall segments:  

; Segment Xend Vend Emax Power P/A dF/dX dF/dY  

; (cm) (cm) (MV/m) (W) (mA/cm^2) (MHz/mm) (MHz/mm)  

-----  

; 1 18.800 0.0000  

; 1 18.800 51.400 0.9956 16.34 317.9 4.2318E-02 0.000  

; 2 26.800 71.000 0.3137 18.47 872.4 -4.5918E-02 -1.8742E-02  

; 3 75.300 71.000 4.0276E-02 47.31 975.5 0.000 -0.1350  

; 4 75.300 0.0000 1.004 35.25 496.4 -3.5967E-03 0.000  

;  

; Total 117.4  

-----  

-----  

939 All calculated values below refer to the mesh geometry only.  

940 Field normalization (NORM = 0): EZERO = 1.00000 MV/m  

941 Frequency = 180.00245 MHz  

942 Normalization factor for E0 = 1.0000 MV/m = 10088.517  

943 Stored energy = 0.0031145 Joules/cm  

944 Using standard room-temperature copper.  

945 Surface resistance = 3.50026 milliohm  

946 Normal-conductor resistivity = 1.72410 microOhm-cm  

947 Operating temperature = 20.0000 C  

948 Power dissipation = 98.5379 W/cm  

949 Q = 38965.5 Shunt impedance = 377.000 Mohm/m  

950 r/Q = 320.000 180 Ohm Wake loss parameter = 9.14325 V/pC  

951 Average magnetic field on the outer wall = 2665.56 A/m, 1.24351 W/cm^2  

952 Maximum H (at X,Y = 43.5,42) = 2678.17 A/m, 1.2553 W/cm^2  

953 Maximum E (at X,Y = 45.0,0) = 1.00154 MV/m, 0.070042 Kilp.  

954 Ratio of peak fields Bmax/Emax = 3.3603 mT/(MV/m)  

955 Peak-to-average ratio Emax/E0 = 1.0015  

956 -----  

957 Wall segments:  

958 Segment Xend Vend Emax Power P/A dF/dX dF/dY  

959 (cm) (cm) (MV/m) (W) (mA/cm^2) (MHz/mm) (MHz/mm)  

960 -----  

961 -----  

962 1 11.250 0.0000  

963 1 11.250 35.250 0.9981 17.74 0.5033 7.6469E-02 0.000  

964 2 15.000 42.000 0.1761 9.122 1.181 -8.1176E-02 -4.5998E-02  

965 3 45.000 42.000 3.1924E-02 37.29 1.243 0.000 -0.3870  

966 4 45.000 0.0000 1.002 26.38 0.6282 -4.6971E-03 0.000  

967 -----  

968 -----  

969 Total 98.54
-----
```

(a) 107.5 MHz truncated

(b) 180 MHz truncated

Figure 4.8: Poisson Superfish calculations with *truncated* cavities

One can observe the shunt impedance gain between truncated and unmodified cavities in *figures 4.7 and 4.8*:

$$\Delta Z_{107.5} = 2.5\% \quad \Delta Z_{180} = 2.7\%$$

4.2. Magnet Design

Magnet design in rhodotron type accelerators depend heavily on the design of the cavities. Because the limiting factors usually are frequency and total volume of the accelerator, magnet design parameters are considered after cavity design has been completed. Considering the nature of coaxial cavity, simplest path for beams inside magnets is major segment of a circle, which was used as the reference for following discussions.

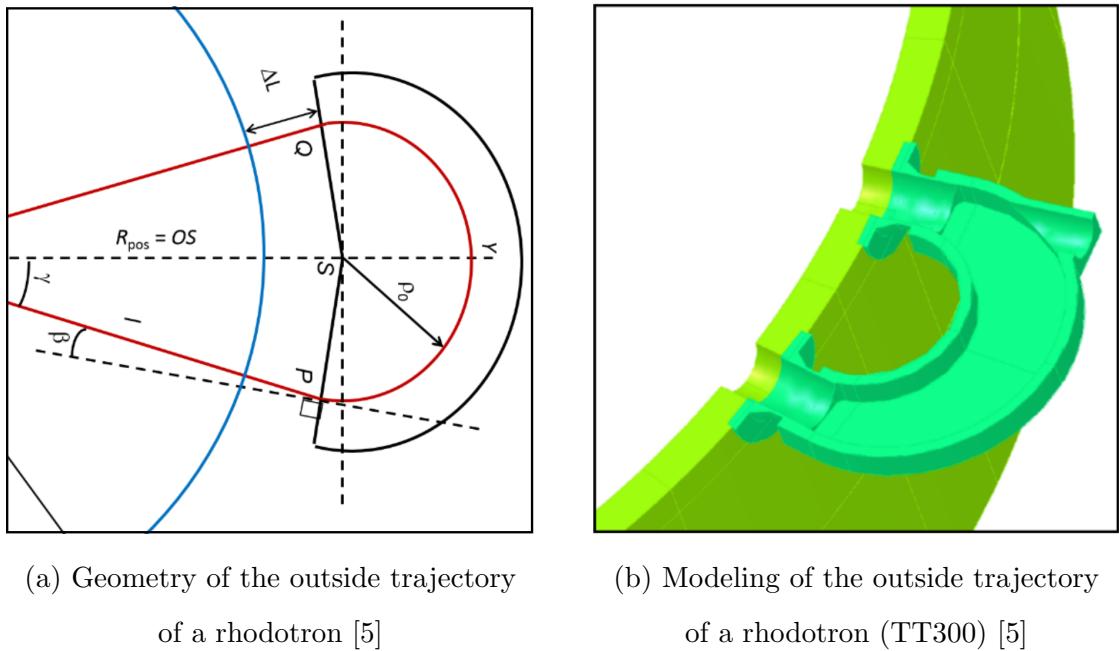


Figure 4.9: From "Design and Simulation Tools for the High-Intensity Industrial Rhodotron Electron Accelerator," by W.Kleeven, M.Abs, J.Brison, E.Forton, J.Hubert and J.Van de Walle

In rhodotron accelerators, magnets are used for not only guiding the electron beam to desired path, but also for synchronizing them with the RF field in the cavity. This synchronization, also called phase stability, is the fundamental constraint of magnet design. Time spent outside of the cavity should be precisely tuned to maximize acceleration in the succeeding pass.

Two different approaches have been used to achieve phase stability. These ap-

proaches are discussed further in the following sections.

4.2.1. $n\lambda$ Technique

First approach for phase stability considiration assumes that from the start of the current pass, $\beta_{avg} \approx 1$ for a synchronous particle.

To maintain the phase synchronization, a particle that started the current pass at $t = 0$, should start the next pass at $t = nT$, where n is an integer and T is the period of RF field inside cavity. As discussed earlier, by assuming the electron is traveling at c , phase stability can be achieved by designing the whole trajectory to be $n\lambda$, where λ is the wavelength of the RF field. In other words, total trajectory of synchronous particle needs to satisfy the following equality;

$$L_{pass} = n\lambda$$

Since $n > 1$ would require unnecessarily long and expensive beam guide and magnets, $n = 1$ is the most efficient choice.

$$L_{pass} = \lambda \quad (4.2)$$

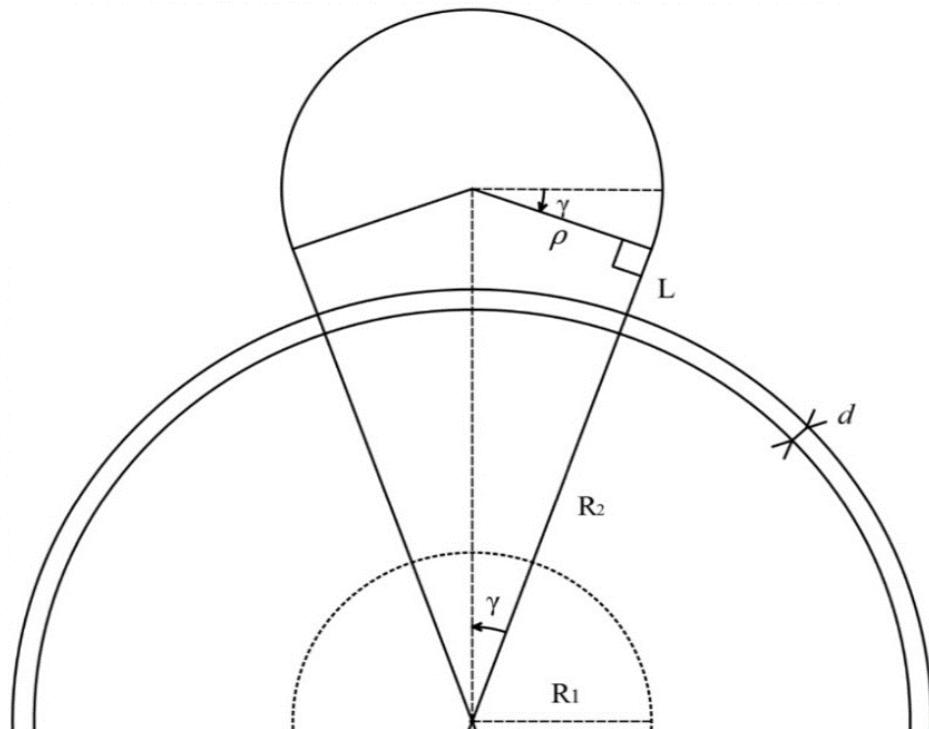


Figure 4.10: Trajectory of a particle in single pass

From *figure 4.10*, where γ is half the angle between trajectories of successive passes, ρ is the radius of the circular path inside magnet, L is the length of the magnet guide, d is the thickness of the cavity wall;

$$\begin{aligned}
 L_{in} &= 2R_2 \\
 L_{out} &= 2L + 2d + (\pi + 2\gamma)\rho \\
 L_{pass} &= L_{in} + L_{out} \\
 \lambda &= 2R_2 + 2d + 2L + (\pi + 2\gamma)\rho
 \end{aligned} \tag{4.3}$$

After the cavity design is complete, only L , γ and ρ remain, which are the design parameters of magnets. Observing the *figure 4.10* and using *equation 4.3*,

$$\begin{aligned} \tan(\gamma) &= \frac{\rho}{R_2 + d + L} \\ \rho &= \tan(\gamma)(R_2 + d + L) \end{aligned}$$

$$\begin{aligned} 2\rho &= \tan(\gamma)(\lambda - (\pi + 2\gamma)\rho) \\ \rho(2 + \tan(\gamma)(\pi + 2\gamma)) &= \tan(\gamma)\lambda \\ \rho &= \frac{\lambda}{\pi + 2\gamma + \frac{2}{\tan(\gamma)}} \end{aligned} \tag{4.4}$$

$$L + d = \frac{\rho}{\tan(\gamma)} - R_2 \tag{4.5}$$

Equations 4.4 and 4.5 define 2 constraints. As already discussed, it has been assumed that cavity design is set (*i.e* R_2 , λ , d are already defined). Therefore only the magnet design parameters ρ , γ and L are left. Together with the constraints, one free variable defines the whole magnet design. Considering the importance of γ for maximum number of possible passes, it will be used as the free magnet design variable in the further discussions.

4.2.2. L_{out} Parameter Sweep

As mentioned previously in *section 4.2.1*, *equation 4.3* assumes that the particles are fast enough so that $\Delta t_{pass} \approx nT$. This assumption is not guaranteed however, especially in the first few passes if the particles are not accelerated enough. This scenario can happen when RF power is not sufficiently high.

Consider an RF supply with $f = 1$ GHz, $P = 50$ kW. Using *equation 2.15*, after the first pass the synchronous electron that entered the accelerator with 40 keV and $\phi = 15^\circ$ phase lag,

$$W_{gain} = 2.14 \times 0.2998^{1/4} \times 50000^{1/2} = 354keV$$

$$W_{total} = 394keV$$

$$\beta_{40keV} \approx 0.374$$

$$\beta_{394keV} \approx 0.825$$

It is clear that β_{avg} is not fast enough to sustain phase stability if the magnet is designed for $\beta = 1$. For these cases, *equation 4.3* fails to deliver phase stability. Another approach for designing a magnet would be to find a new constraint, K for total path length.

$$L_{pass} = K$$

$$L_{in} + L_{out} = K$$

Since L_{in} is set previously from cavity design, we can remove it from our equation and continue with

$$L_{out} = K \quad (4.6)$$

To start, an optimization criteria for the following pass such as

- Ensure the phase stability of the synchronous electron
- Maximize the energy gain of the synchronous electron
- Minimize the energy spread of the beam
- Minimize the phase lag spread of the beam
- All of the above with decided weights

needs be selected. Using a simulation software such as *CST Studio Particle Module*, the optimum value of K for the criteria can be found by simulating for different values of K (*sweeping*) and analyzing the results. This process can be repeated for each magnet until the desired beam characteristics in the end of the accelerator are achieved.

However, one caviat of this technique is that, available simulation softwares are not well suited for this kind of process. As mentioned above, *CST Studio Particle Module* has a *parameter sweep* functionality. But the calculation times are too slow to be useful in this particular problem. *A custom built software offering magnet optimizing sweep functionality will be discussed in the following chapter.*

4.3. Initial Design at KAHVELab

A rhodotron with operation frequency of 107.5 MHz was decided to be built at KAHVELab. After considering the earlier simulations mentioned in *sections 4.1 and 4.2*, an initial design was created.

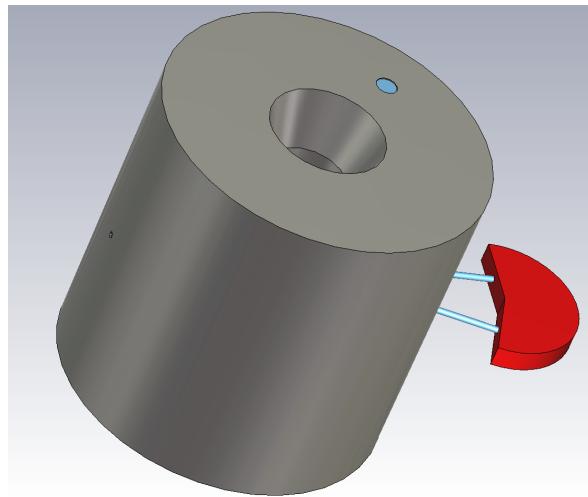
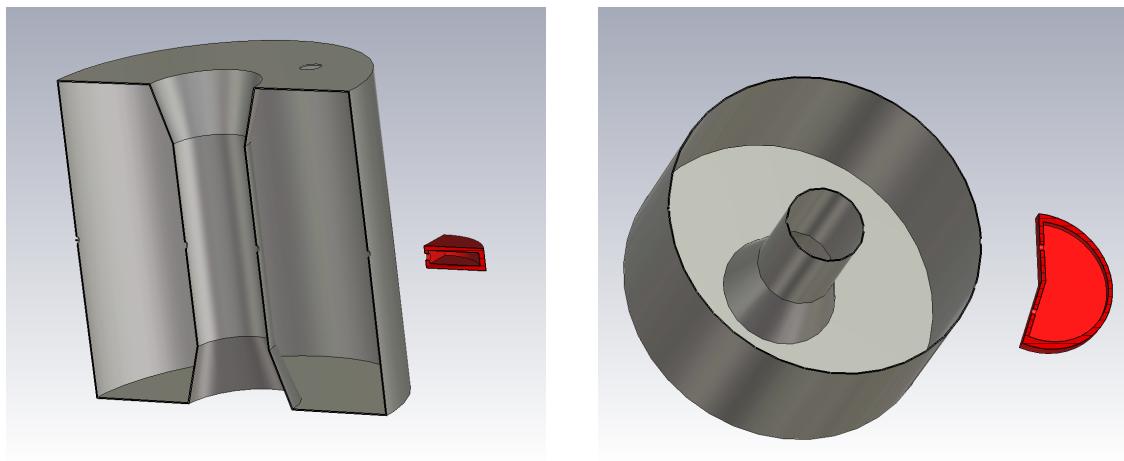


Figure 4.11: Initial design of the proposed rhodotron

$$\gamma = 9^\circ$$



(a) Axial cross section.

(b) Cross section at acceleration plane.

Figure 4.12: Initial design of the proposed rhodotron.

The magnets used in this design is the simplest in terms of design parameters. It consists of an iron casing encapsulating the shape of desired magnetic field, which are

created by two coils inside.

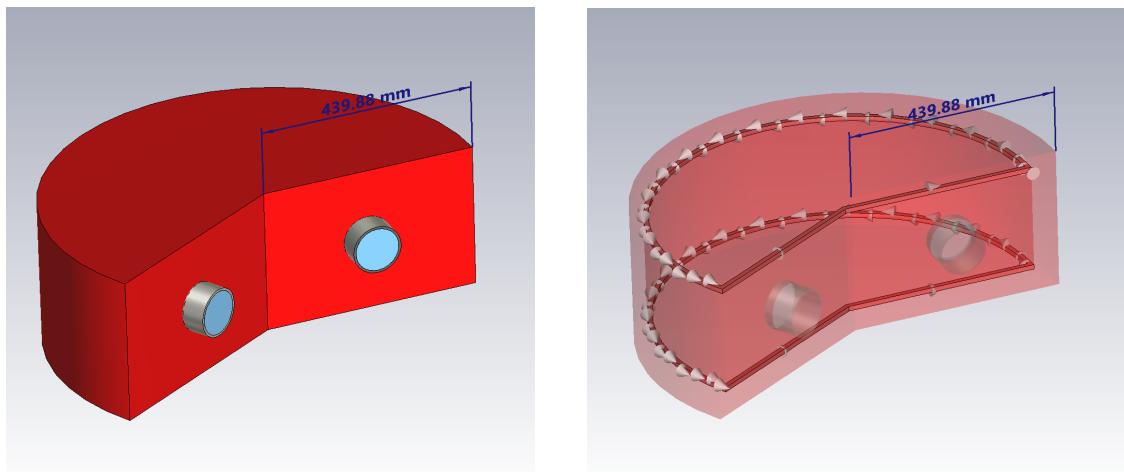


Figure 4.13: Initial magnet design.

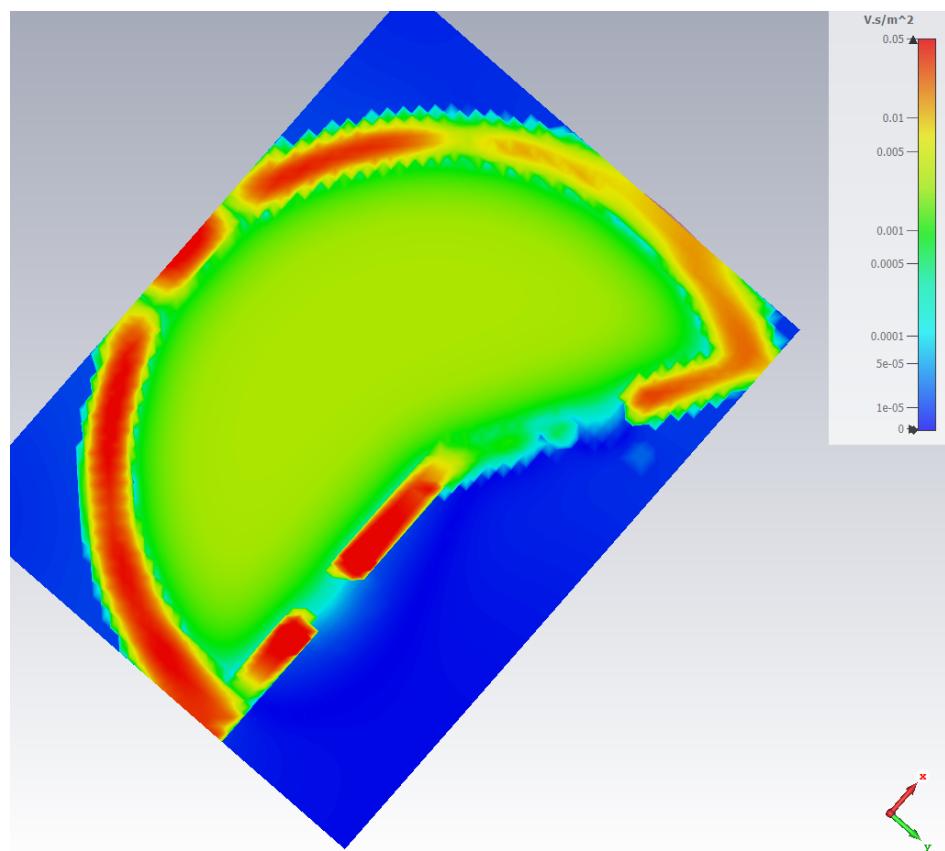


Figure 4.14: Magnetic field in acceleration plate of initial magnet design.

figure 4.14 shows that this design provides relatively uniform magnetic field inside, although magnetic field gradient in the openings can be improved.

After adding two more magnets and using $\gamma = 15^\circ$, the following second prototype design was created

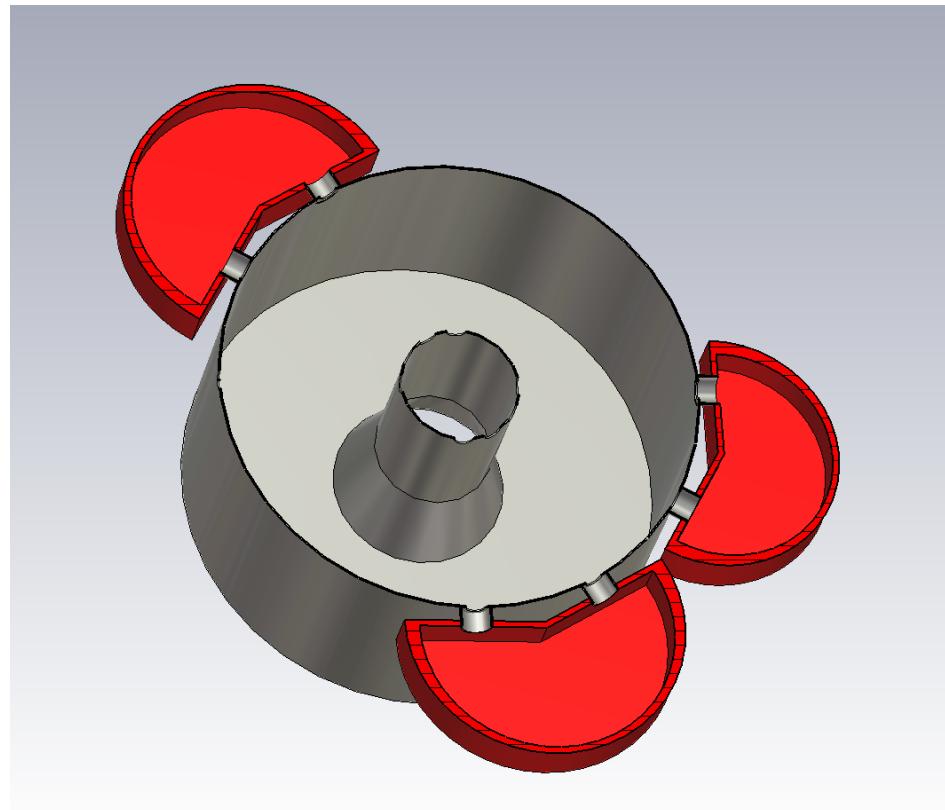
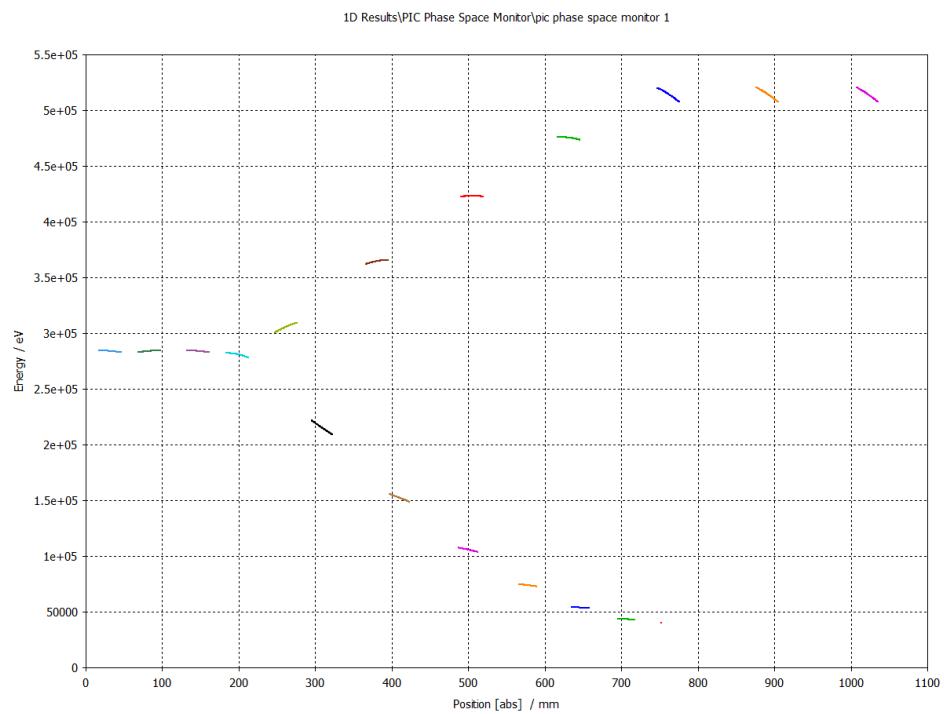


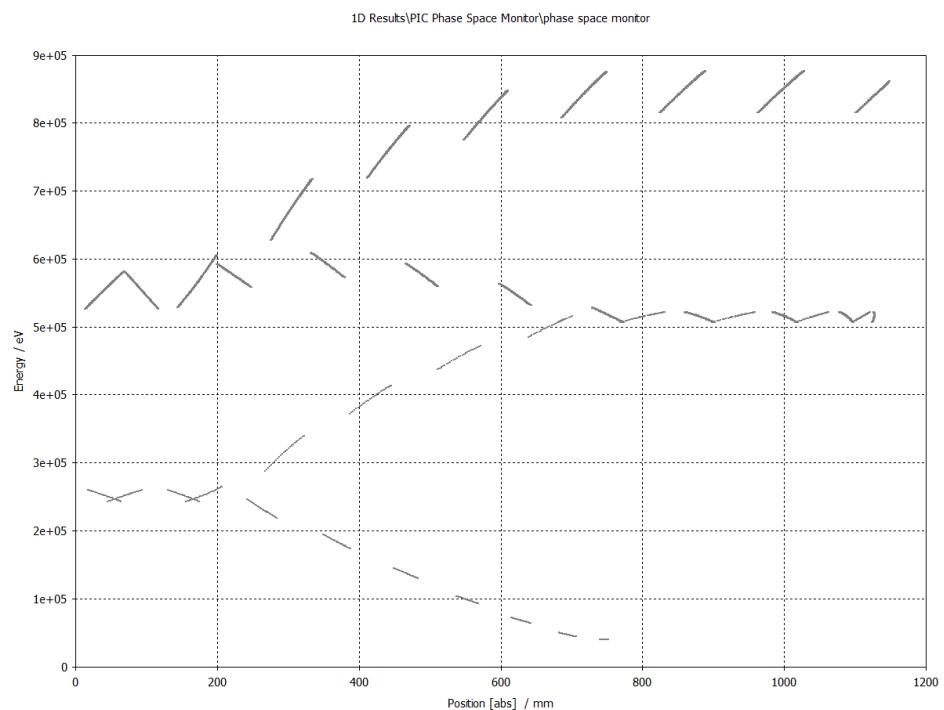
Figure 4.15: Initial cavity with three magnets as shown in *figure 4.13*

$$\gamma = 15^\circ$$

These designs, together with 40 keV e^- gun injecting with phase lag of 15° for 1 ns, was simulated at RF power of 12 kW using *CST Studio Particle Module PIC Solver*. Results of these simulations can be found in *figure 4.16*.



(a) One pass simulation of figure 4.11



(b) Two pass simulation of figure 4.15

Figure 4.16: $|r|$ vs *Energy* simulation of initial design.

The behavior of the beam in the *figure 4.16* after the first pass indicates that phase stability is not maintained. When the simulation results are examined further, it was observed that the beam started the second pass at $t \approx 12$ ns after the injection. Considering the starting phase lag $\phi_{lag} = 15^\circ$, this would mean phase lag of the second pass was $\phi_{lag2} = 120^\circ$. This result can be observed in *figure 4.16*, as the beam decelerates after a short acceleration in the beginning of the second pass.

After various similar observations in *CST* simulations, underlying cause of this problem was thought to be the $L_{pass} = n\lambda$ approach on magnet design failing in low energies, as anticipated and discussed earlier in the *section 4.2.2*. *The necessity for an alternative tool, capable of aiding researchers in improving beam behavior, was deemed evident.*

In KAHVELab, several improvements were made to these initial designs together with *Mr. Sinan ÖZ*. They include curved edges on cavity which reduce the power loss on cavity walls, and redesigned iron casing on magnets for ease of production and maintenance. Improved designs can be seen in the figures below.

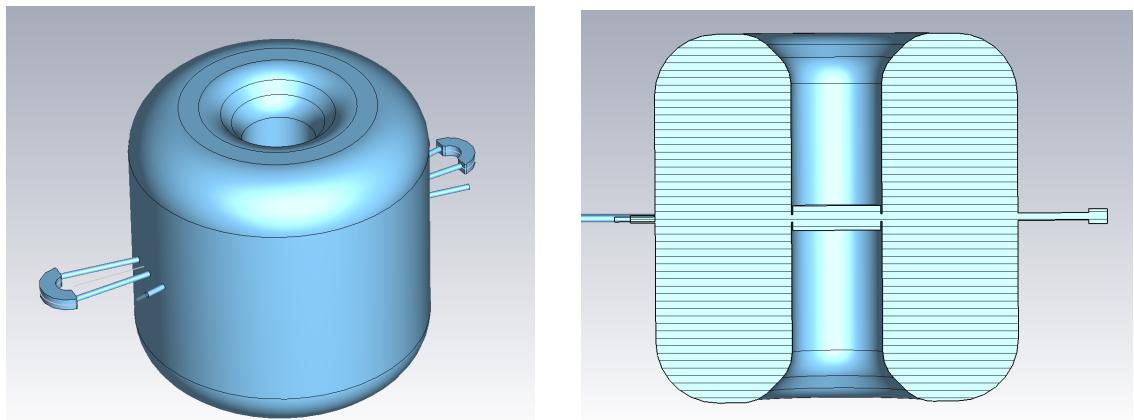


Figure 4.17: Improved cavity design.

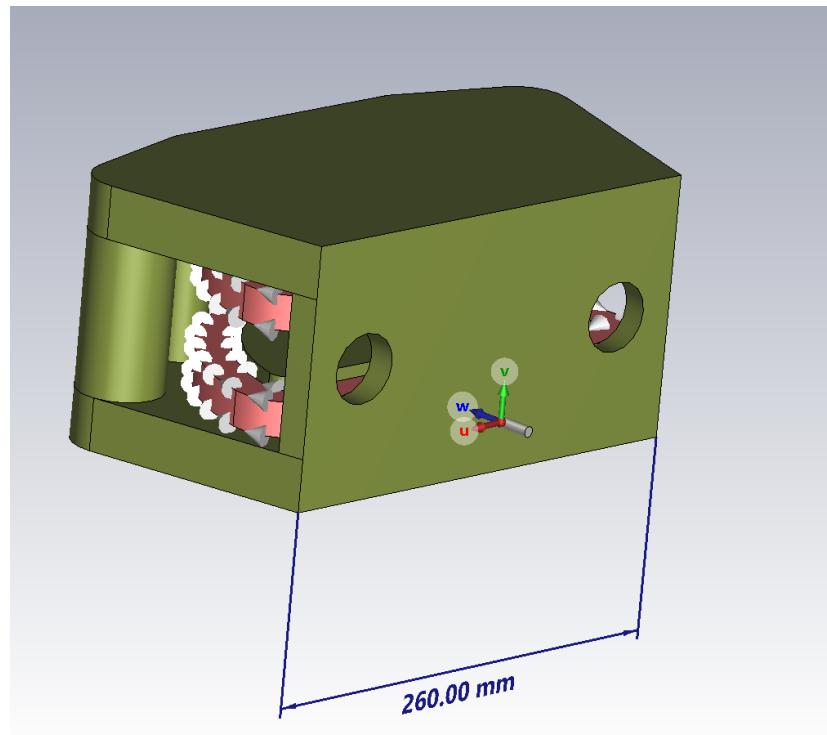


Figure 4.18: Improved magnet design.

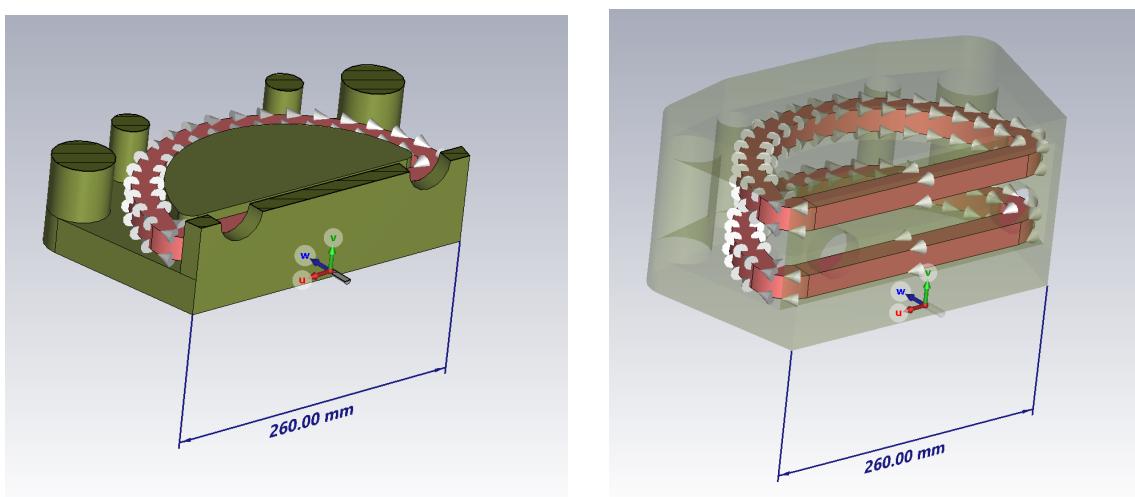


Figure 4.19: Cross section and coils of improved magnet design.

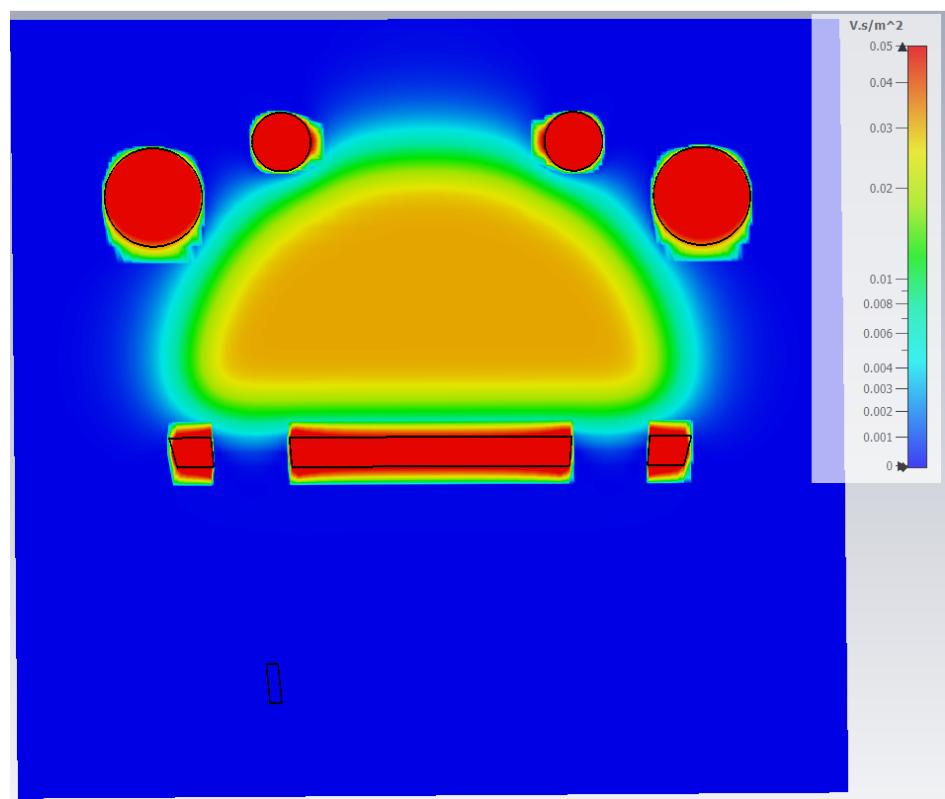


Figure 4.20: Magnetic field in acceleration plate of improved magnet design.

From figures 4.14 and 4.20, it can be observed that improved magnet design reduces magnetic field leak in enter and exit openings considerably. This reduction helps containing the spread of entering beam, decreasing the amount of focusing needed further into trajectory.

5. SIMULATION

5.1. Proof of Concept

Once the need for a custom built tool for designing and simulating a Rhodotron type accelerator was apparent, a simple 1D proof of concept was implemented.

This proof of concept worked with interacting an electron with the EM field provided, at discrete time steps dt . The core sequence of this software is provided below. Where \vec{r} is the position, \vec{v} is the velocity of the e^- ;

- Get electron energy E_{in}
- Get the RF field definitions
- Get the magnet design parameters if there is any
- Do following until the simulation time has ellapsed
 - (i) Calculate $\vec{E}_{||}(\vec{r})$
 - (ii) Calculate $\vec{a}_{E||}$ using *equation 2.11*
 - (iii) Calculate $\vec{r}(t + dt)$ using *equation 2.17*
 - (iv) Calculate $\vec{v}(t + dt)$ using *equation 2.18*
 - (v) Calculate new E_t

The implemenation of this logic can be observed in *figure 5.1*.

```

1  for (double t=0; t<SimuTime; t+=dT){
2      double RelBeta = v/c;
3      double RelGamma = 1.0 / sqrt(1.0-RelBeta*RelBeta);
4
5      double ef=Eradiat(r_pos*1000,t,0); // convert position to mm
6      double acc=ef*1E6*eQMratio/(RelGamma*RelGamma*RelGamma);
7
8      r_pos = r_pos + v * dT*ns + 1/2*acc*(dT*ns)*(dT*ns);
9      v = v + acc*dT*ns;
10
11     RelBeta = v/c;
12     RelGamma = 1.0 / sqrt(1.0-RelBeta*RelBeta);
13     Et=RelGamma*E0;
14 }
```

Figure 5.1: Core logic loop of the POC

After simulating the *synchronous electron* mentioned in *section 2.2.4.1* with $\phi_{lag} = 15^\circ$ for one pass, the results from *POC*, *CST* and *equation 2.14* are compared.

$$\begin{aligned}
 E_{Pottier} &= 0.565\text{MeV} \\
 E_{CST} &= 0.872\text{MeV} \\
 E_{POC} &= 0.873\text{MeV}
 \end{aligned} \tag{5.1}$$

$$P = 40\text{kW}, R_1 = 0.188\text{m}, R_2 = 0.753\text{m}, f = 107.5\text{MHz}$$

As can be observed in *equation 5.1*, *POC software* and *CST* produced very similar results. However, we cannot make the same claim regarding *equation 2.14*. This inconsistency between $E_{Pottier}$ and E_{CST} was noticed in the earlier simulations as well, which supported the idea of using another simulation software to reduce reliance on *CST*. Since using a single software would have made the process error prone.

After the results obtained from *POC* were found to be promising, a decision was made to develop a more robust simulation software. This software was named *Rhodotron Simulation* for the current stage and started development in October, 2021. The development and improvement efforts are still ongoing.

During the development of *Rhodotron Simulation* software, several intermediate builds have been implemented and tested. The following sections in this chapter investigates their implementations further.

5.2. Intermediate Versions

5.2.1. L_{out} Optimization For Single e^-

Initial step of improving *POC* towards *Rhodotron Simulation* was to implement L_{out} optimizations to help optimazing magnet designs, as discussed in *section 4.2.2*. First approach was to hang the e^- outside of the cavity for $t_{out} = L_{out}/v$, then inject it back to the cavity with reversed \vec{v} . One can sweep the t_{out} parameter to find the optimal value. This simple implementation can be found in *figure A.1* of *Appendix A*.

Although the results from this optimization sweep were promising after they were simulated with *CST*, simulating one particle would not be sufficiently useful for designing a magnet.

5.2.2. ϕ_{lag} Optimization For Bunches

After successfully accelerating single e^- , particle bunches were implemented to approximate a real e^- gun. They were modeled as N electrons fired from an e^- gun at even time intervals. This approach was taken because the amount of time gun fires, defined as *Gun Active Time*, t_g , is a crucial part of pulsed e^- gun design.

Addition of bunches would immediately proven useful when finding optimal phase lag for bunch entry. ϕ_{lag} for a bunch was defined as the RF phase when the first e^- of the bunch has entered the cavity and it defines the starting time of the current pass. To use the *parameter sweep* method, as used in L_{out} optimization, relevant bunch characteristics are defined as follows:

- μE : Average energy
- E_{rms} : Root mean square of energy
- R_{rms} : Root mean square of e^- positions

Optimal ϕ_{lag} would produce maximum μE while minimizing E_{rms} & R_{rms} . For the first pass, E_{rms} and R_{rms} would be vaguely dependent of each other; therefore, initial implementation of ϕ_{lag} sweep was based on minimizing E_{rms} during simulation (*see figure 5.2*). For μE considerations, data from the software would be analyzed either manually or by using external tools such as *ROOT framework*.

```

1 int phase_opt(int phase_sweep_range){
2     double minrms = 1;
3     int opt_phase;
4     for(int RFphase = -phase_sweep_range; RFphase <= phase_sweep_range; RFphase++){
5         Bunch bunch1(RFphase);
6         double t1 = 0;
7         bunch1.bunch_gecis_t(t1);
8         bunch1.reset_pos();
9
10        if( bunch1.E_rms() < minrms ){
11            minrms = bunch1.E_rms();
12            opt_phase = RFphase;
13        }
14    }
15    return opt_phase;
16 }
```

Figure 5.2: ϕ_{lag} Optimization For Initial Bunch Design

Since ϕ_{lag} is relatively easy to change after production, another version of *figure 5.2* that was modified for given magnet design parameters was also implemented (*see figure A.2 in Appendix A*). This version can be useful for optimizing ϕ_{lag} in case of production issues in magnets.

After the bunch and ϕ_{lag} sweep implementations, L_{out} sweep was also updated to minimize E_{rms} . ρ and L calculations, using *equations 4.4 and 4.5*, were also integrated. Two example runs from this intermediate build can be found in *figures B.1 and B.2 of Appendix B*.

5.2.3. Simulation in 3D

After successfully implementing $e^- - \vec{E}$ interaction in 1D and confirming the usefulness of this tool, the decision was made to proceed with implementing a 3D version of the *Rhodotron Simulation*. Although complete refactoring of the software was necessary, this upgrade was crucial for implementation of $e^- - \vec{B}$ interaction. The refactoring effort included proper implementation of *OOP*, details of which can be seen in *figures A.13, A.14, A.15, A.16, A.17 and A.18* of *Appendix A*.

Magnets were modeled as major segments of a circle, defined with \vec{r}_{mag} , \mathbf{R} and $|\mathbf{B}|$. For the initial implementation, \vec{B} assumed to be uniform and has no leaks outside the magnet boundary (See *figure A.4* in *Appendix A*).

Interaction logic for $e^- - \vec{E}$ and $e^- - \vec{B}$ in 3D can be found in *figure 5.3*.

```

1  vector3d CoaxialRFField::actOn(Electron& e){
2      vector3d Efield = getField(e.pos);                      // Calculate E vector
3      vector3d F_m = Efield*1E6*eQMratio;                  // Calculate F/m vector
4      vector3d acc = (F_m - e.vel*(e.vel*F_m)/(c*c))/e.gamma(); // Calculate a vector
5      return acc;
6  }



---


1  vector3d MagneticField::actOn(Electron& e){
2      if (isInside(e.pos) == -1)
3          return vector3d(0,0,0);
4      vector3d Bfield = getField(e.pos);                      // Calculate B vector
5      vector3d F_m = (e.vel % Bfield)*eQMratio;            // Calculate F/m vector
6      vector3d acc = (F_m)/e.gamma();                         // Calculate a vector
7      return acc;
8 }
```

Figure 5.3: $e^- - \mathbf{EM}$ interaction logic from *equation 2.12*

Where $*$ and $\%$ are, dot-product and cross-product respectably. (See *figure A.3* of *Appendix A*)

Simulating in 3D had one other benefit; it was now possible to visualize the

results by rendering the interaction data. For this purpose, *gnuplot* was integrated into *Rhodotron Simulation* to produce 2D visualization of acceleration plane. Rendered results could be stored as *gif* animations. Two of such renders can be seen in *figure 5.4*.

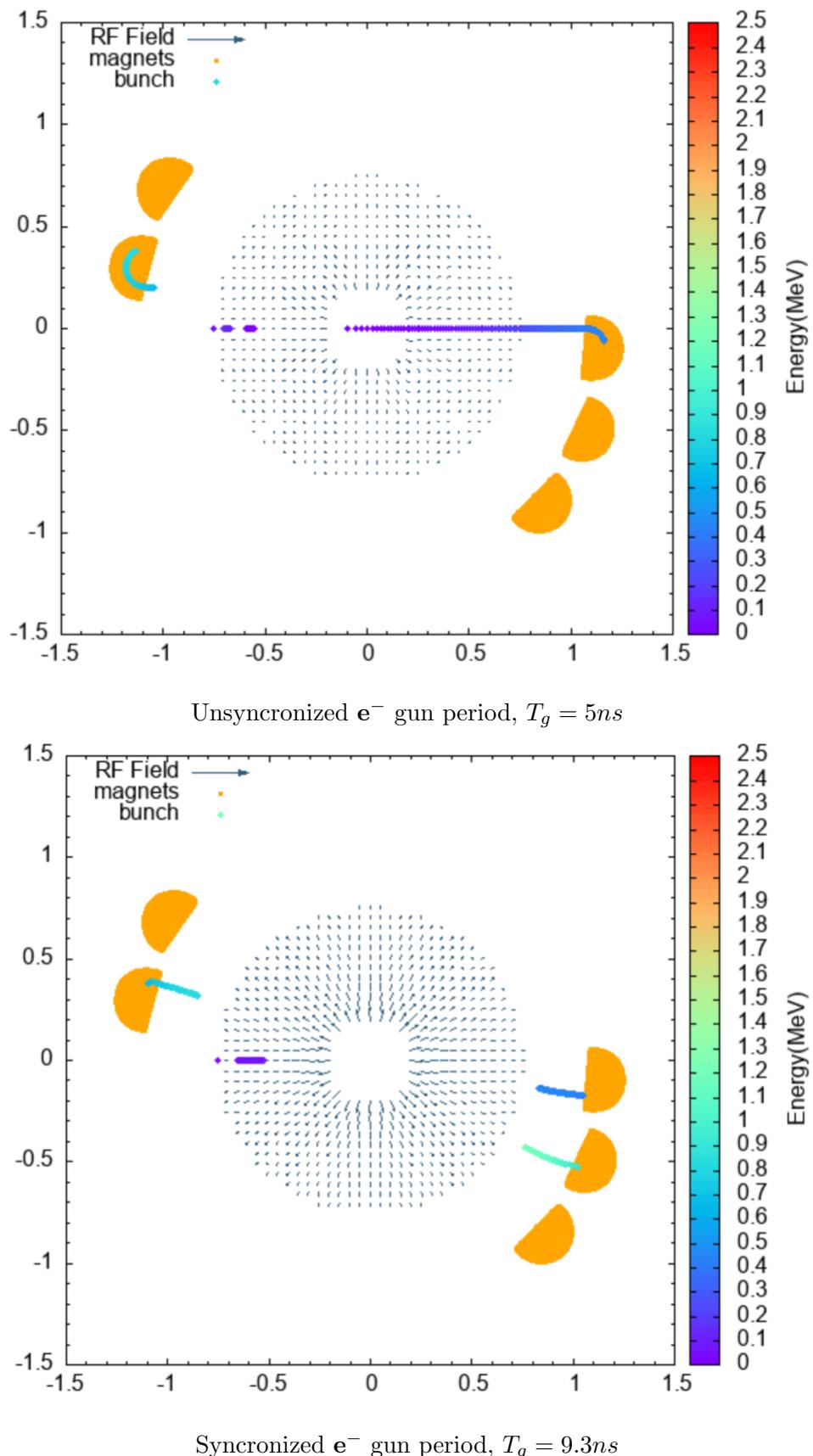


Figure 5.4: Example *gnuplot* renders of *Rhodotron Simulation*

$P = 12\text{kW}$, $f = 107.5\text{MHz}$

5.2.4. Acceleration in Magnetic Field

An issue regarding the $e^- - \vec{B}$ interaction became apparent when energy gain during these interactions was observed. A setup simulation was implemented in which a bunch of $100e^-$ at 1MeV was fired into a uniform magnetic field of 0.1T located in $x > 0.05m$. Initial results at $dt = 0.01ns$ proved the suspicion of $e^- - \vec{B}$ interaction being broken. However, the energy gain would decrease tremendously as dt decreased.

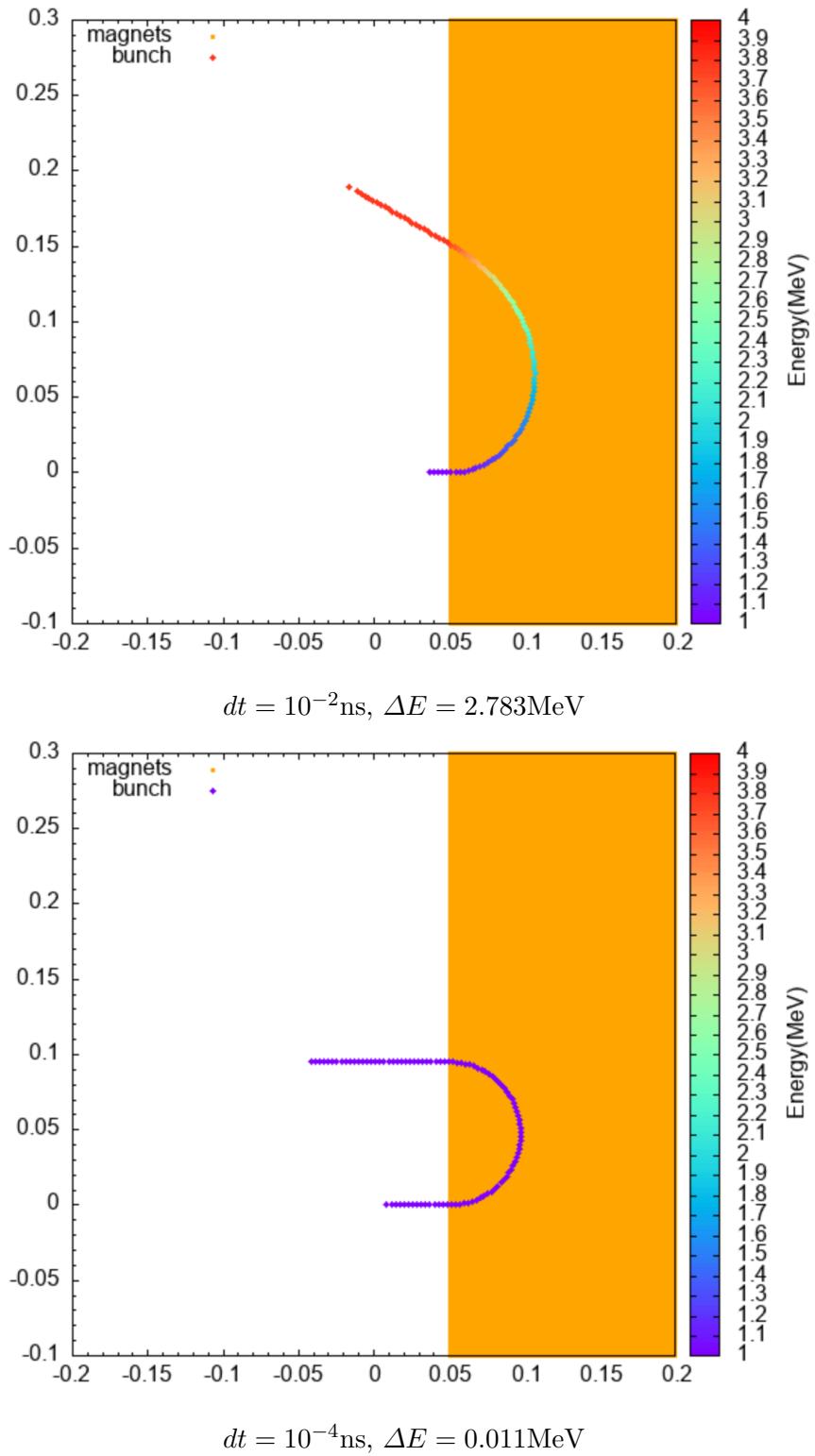


Figure 5.5: Energy gain of 1MeV bunch in $\mathbf{B}=0.1\text{T}$

Decreasing dt would be the best way to increase accuracy of the results; however,

this is not sustainable due to computing power limitations. Until this point, *Rhodotron Simulation* have been using *section 2.3.1* for e^- - **EM** interactions. To test newer approaches, two additional version of e^- - **EM** interaction that are using *section 2.3.2* were added.

5.2.4.1. RK4-1. First approach for integrating *section 2.3.2* into e^- - **EM** interaction was to calculate \vec{a}_E and \vec{a}_B from *equation 2.12* using *RK4*. After $\vec{a}_{EM} = \vec{a}_E + \vec{a}_B$ was calculated, e^- would move and accelerate using the *Leap-frog* method. The idea was to produce more refined interaction results, leading to improved accuracy especially in e^- - \vec{B} . RF field was kept static during the *RK4* computation, due to ongoing *multithreading* implementation efforts. The implementation can be found in *figures A.5 and A.6*.

5.2.4.2. RK4-2. Following the implementation of *RK4-1*, revisions were made to the integration method for *RK4* to replace *Leap-frog*. Instead of calculating \vec{a}_{EM} using *RK4*, \vec{r} and \vec{v} would be determined directly.

These three methods were then tested in the same setup as *figure 5.5*. The results can be found in *figure C.1* of *Appendix C*. *RK4-1* was decided to be abandoned as it produced the same accuracy in twice the simulation time of *RK4-2*.

More rigorous testing was done with *Leap-frog* and *RK4-2* however. Still using the setup in *figure 5.5*, each dt configuration was simulated 10 times, calculating average and standard deviations afterwards. Results from these tests can be observed in *figure 5.6*.

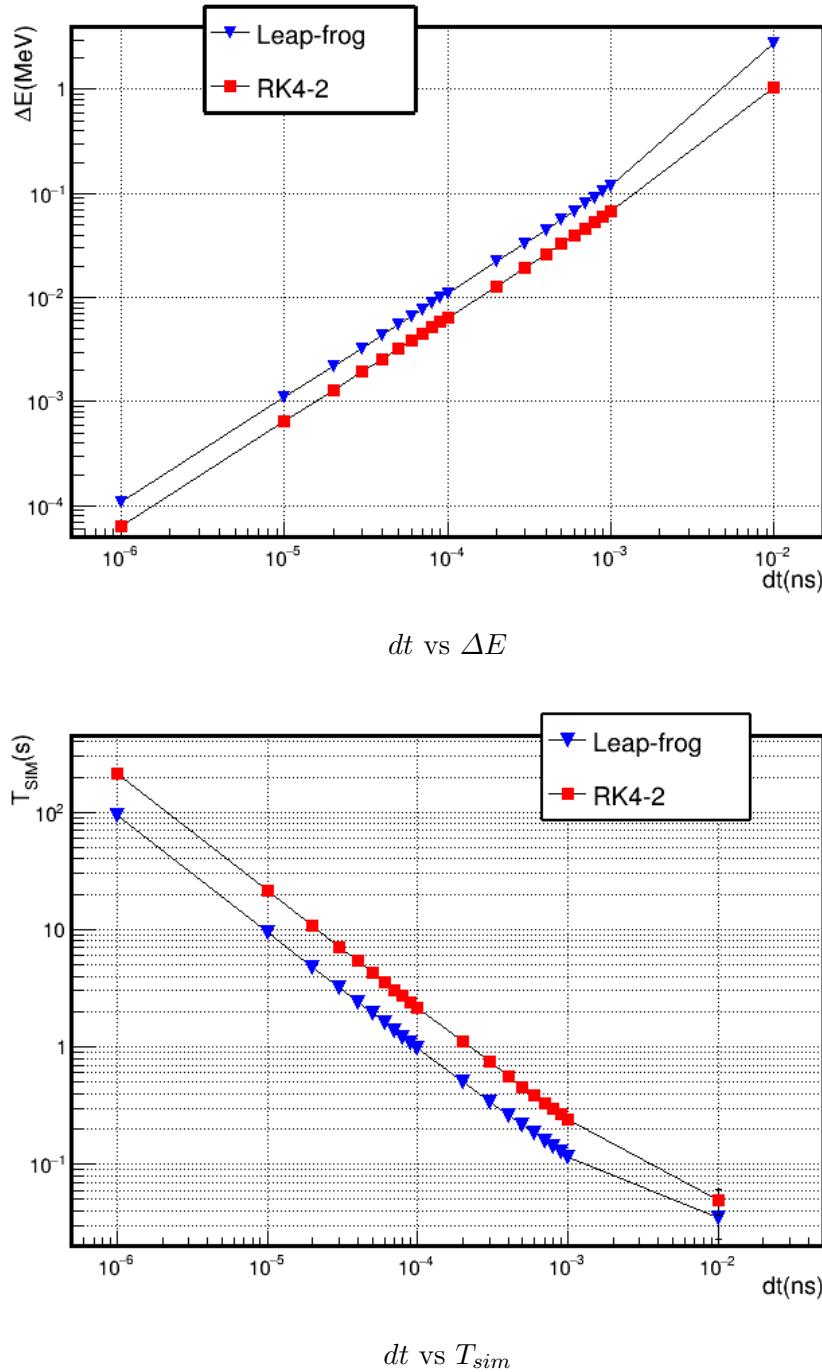


Figure 5.6: Comparing Leap-frog, RK4-2 performance on $e^- - \vec{B}$ interaction

$$E_{in} = 1\text{MeV}, \mathbf{B}=0.1\text{T}, t_{end} = 5\text{ns}$$

The data from these tests can be found in *Tables C.1 and C.2 of Appendix C*. To

investigate the data further, one can define a performance measurement, F .

$$F \propto 1/T$$

$$F \propto 1/\Delta E$$

When $dt = 10^{-5}$ ns was taken as reference point due to providing a good balance of accuracy and computational intensity,

$$\begin{aligned} \Delta E_{LF} &= 110 \times 10^{-5} MeV \\ \Delta E_{RK}^1 &= 64 \times 10^{-5} MeV \\ T_{LF}^1 &= 9.44 \pm 0.03s \\ T_{RK}^1 &= 21.33 \pm 0.02s \\ \Delta E_{LF} \times T_{LF} &= 104 \times 10^{-4} \pm 10^{-4} MeVs \\ \Delta E_{RK} \times T_{RK} &= 137 \times 10^{-4} \pm 2 \times 10^{-4} MeVs \\ F_{LF}/F_{RK} &= \frac{\Delta E_{RK} \times T_{RK}}{\Delta E_{LF} \times T_{LF}^1} = 1.32 \pm 0.01 \end{aligned} \quad (5.2)$$

Also, observing from the data,

$$\begin{aligned} \Delta E_{LF}(dt = 3 \times 10^{-5}) \approx \Delta E_{RK}(dt = 5 \times 10^{-5}) &\approx 32.5 \times 10^{-4} MeV \\ T_{LF}(dt = 3 \times 10^{-5}) &= 3.18 \pm 0.02s \\ T_{RK}(dt = 5 \times 10^{-5}) &= 4.30 \pm 0.01s \\ F_{LF}/F_{RK} = \frac{T_{RK}(dt = 5 \times 10^{-5})}{T_{LF}(dt = 3 \times 10^{-5})} &= 1.4 \pm 0.1 \end{aligned} \quad (5.3)$$

Uncertainty of *equation 5.3* was taken high due to the approximation. After combining *equations 5.2 and 5.3*, F can be calculated as

$$F_{LF}/F_{RK} = 1.36 \pm 0.05 \quad (5.4)$$

Therefore, *Leap-frog* was found to be the better choice as it provided with 1.36 ± 0.05 times accuracy in $\mathbf{e}^- - \vec{\mathbf{B}}$ interactions at a given time with respect to *RK4*. However, *RK4* was promising in situations where decreasing the stepsize, dt , is not viable. Therefore both were integrated into *Rhodotron Simulation* for the user to decide. *RK4-2* renders from these test can be found in *figure B.3 of Appendix B*.

5.2.5. Acceleration in Electric Field

After the accuracy concerns regarding the $\mathbf{e}^- - \vec{\mathbf{B}}$ interaction were raised, it was decided to test $\mathbf{e}^- - \vec{\mathbf{E}}$ and compare the performance of *Leap-frog* and *RK4*.

As test setups, two simulation configurations were made. They aimed to test the accuracy of acceleration of a beam in parallel and perpendicular static uniform electric fields. Both configurations had an \mathbf{e}^- -gun located at $(-0.753, 0, 0)$ m, directed at $(1, 0, 0)$ firing electrons with the kinetic energy of 1MeV.

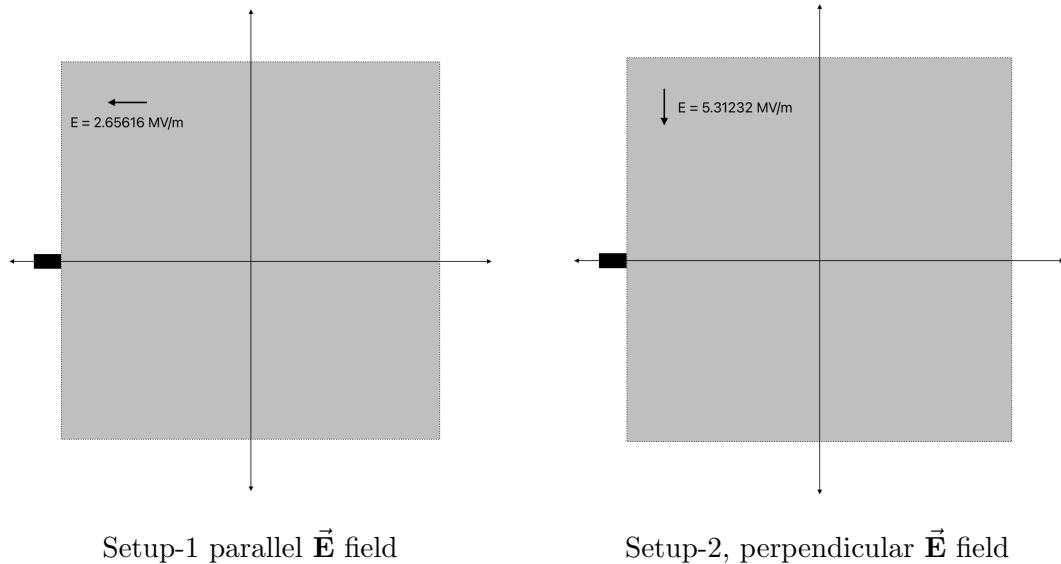


Figure 5.7: Illustration of test setups.

In the first test, the beam would be injected into a static uniform electric field $\vec{\mathbf{E}} = (-2.65616, 0, 0)$ MV/m where $-0.753 < x < 0.753$ and $-0.753 < y < 0.753$, $\vec{\mathbf{E}} = 0$ elsewhere.

Considering the $\vec{\mathbf{E}}$ is parallel to the beam path, potential difference V in the trajectory until (-0.753, 0, 0)m is

$$\Delta V^1 = - \int \vec{\mathbf{E}} \cdot d\vec{\mathbf{s}} \quad (5.5)$$

$$= - \int_{-0.753}^{0.753} -2.65616 \times dx \quad (5.6)$$

$$= 2.65616 \times 1.506 \quad (5.7)$$

$$\Delta V^1 = 4MV \quad (5.8)$$

$$\Delta E^1 = 4MeV \quad (5.9)$$

$$E_{exitTH}^1 = 5MeV \quad (5.10)$$

In the second test on the other hand, the beam would be injected into a different static uniform electric field,

$\vec{\mathbf{E}} = (0, -5.31232, 0)$ MV/m where $-0.753 < x < 0.753$ and $-0.753 < y < 0.753$,
 $\vec{\mathbf{E}} = 0$ elsewhere.

$$\Delta V^2 = - \int \vec{\mathbf{E}} \cdot d\vec{\mathbf{s}} \quad (5.11)$$

$$= - \int_0^{0.753} -5.31232 \times dy \quad (5.12)$$

$$= 5.31232 \times 0.753 \quad (5.13)$$

$$\Delta V^2 = 4MV \quad (5.14)$$

$$\Delta E^2 = 4MeV \quad (5.15)$$

$$E_{exitTH}^2 = 5MeV \quad (5.16)$$

Therefore, in the both tests, the beam was expected to exit $\vec{\mathbf{E}}$ with $E_{exitTH} = 5$ MeV. To also measure the variance in simulation completion times, T_{sim} , set of 10 runs were completed at the configuration for each dt value.

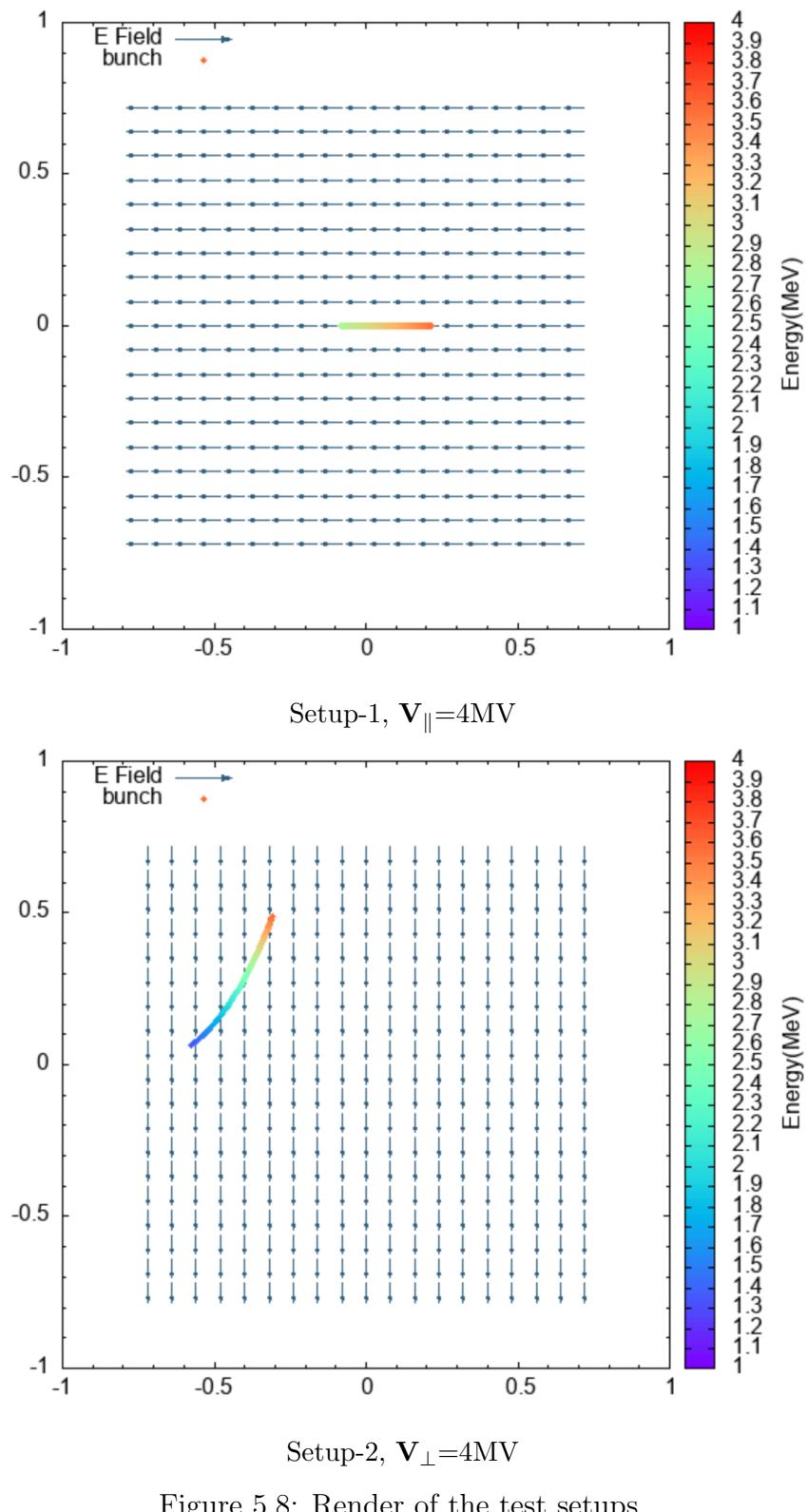


Figure 5.8: Render of the test setups.

$$E_{in} = 1 \text{ MeV}, t_{end} = 6\text{ns}$$

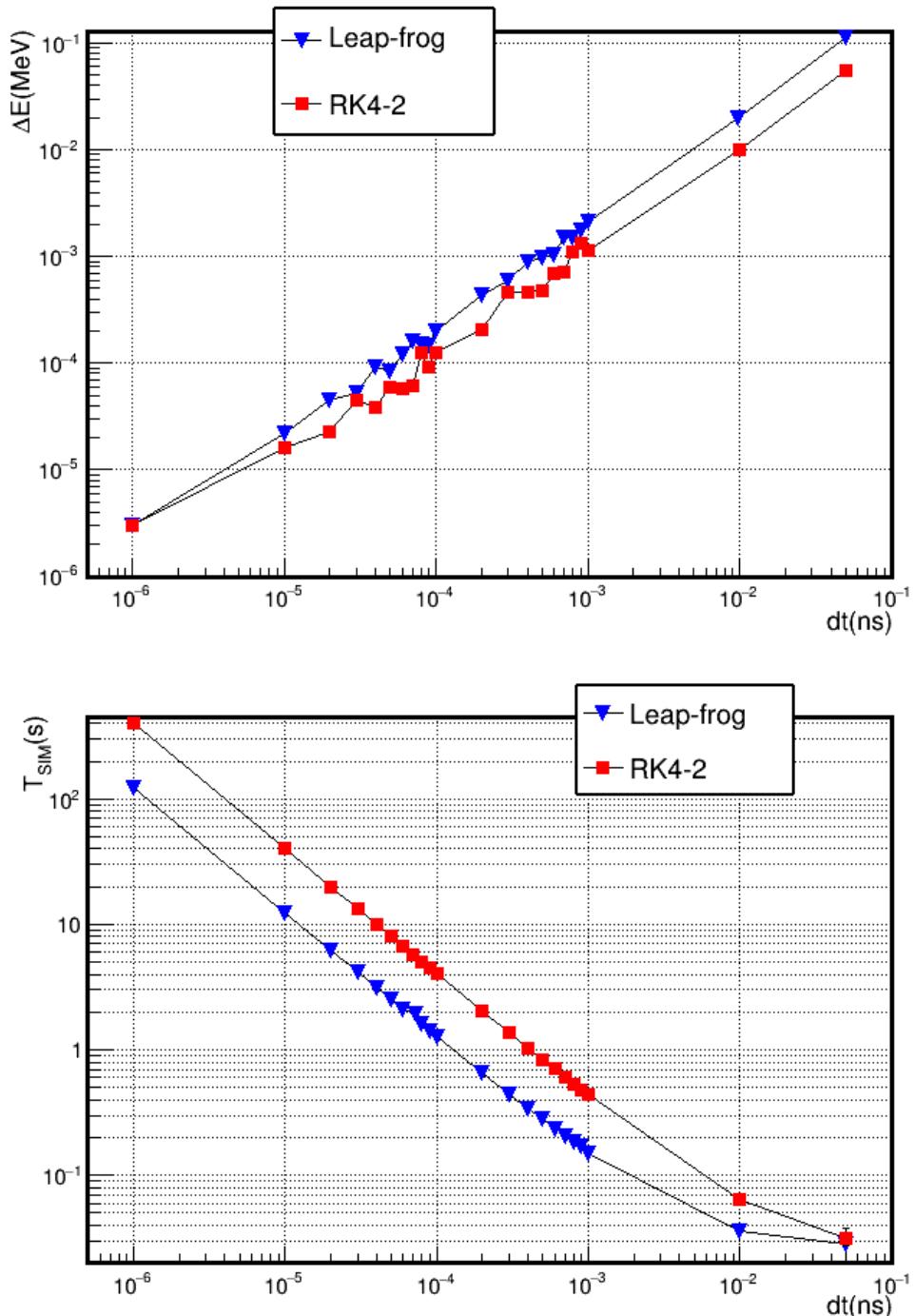


Figure 5.9: Comparing Leap-frog, RK4-2 performance on e^- - \vec{E} interaction

$$E_{in} = 1 \text{ MeV}, \mathbf{V}_{||} = 4 \text{ MV}, t_{end} = 6 \text{ ns}$$

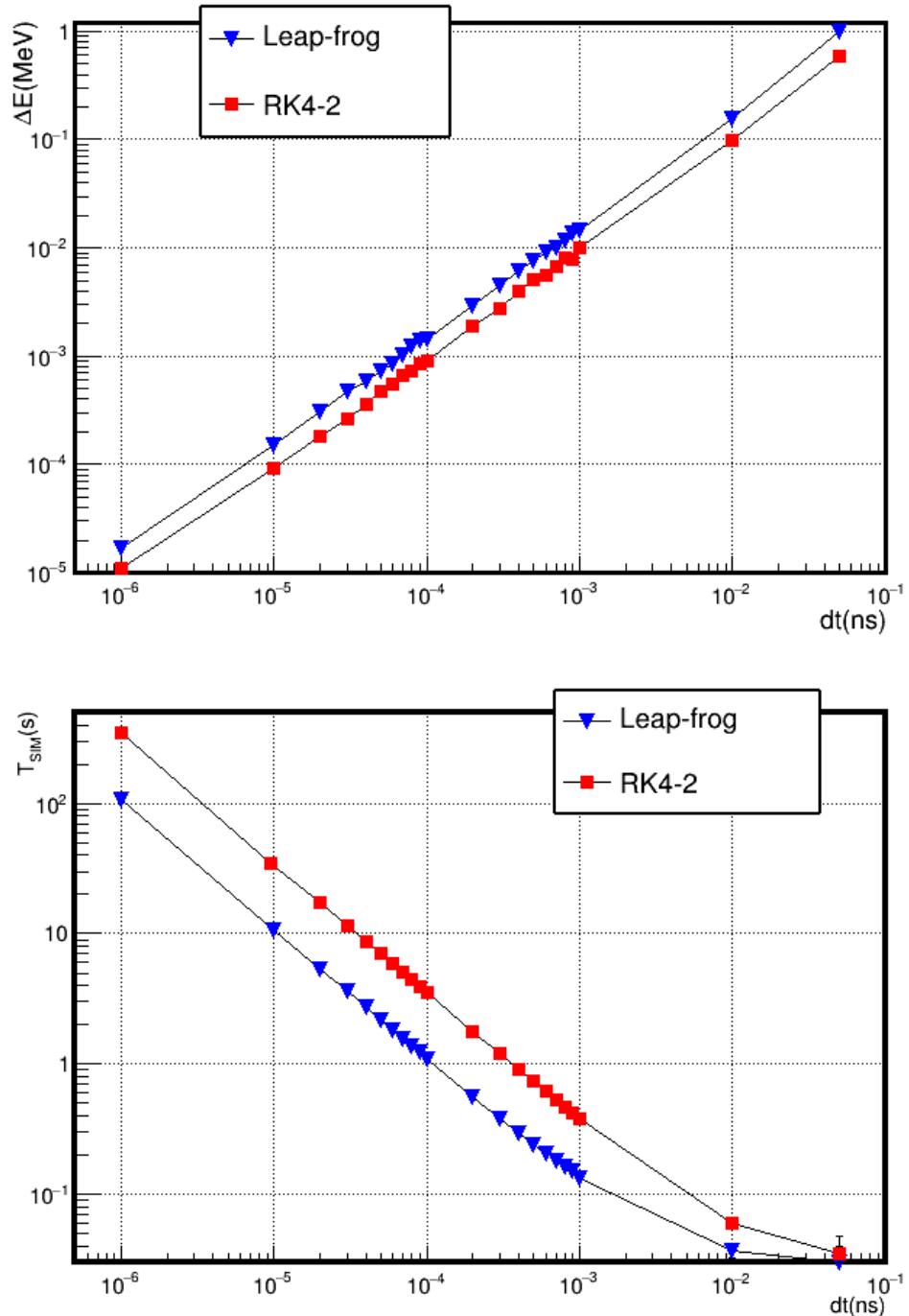


Figure 5.10: Comparing Leap-frog, RK4-2 performance on $e^- - \vec{E}$ interaction

$$E_{in} = 1 \text{ MeV}, \mathbf{V}_\perp = 4 \text{ MV}, t_{end} = 6 \text{ ns}$$

Taking the $dt = 10^{-5}$ ns for the reference point as before, the relative performance

can be calculated.

$$\begin{aligned}
 \Delta E_{LF}^1 &= 22 \times 10^{-6} MeV & \Delta E_{LF}^2 &= 15 \times 10^{-5} MeV \\
 \Delta E_{RK}^1 &= 16 \times 10^{-6} MeV & \Delta E_{RK}^2 &= 9.3 \times 10^{-5} MeV \\
 T_{LF}^1 &= 12.40 \pm 0.04s & T_{LF}^2 &= 10.75 \pm 0.04s \\
 T_{RK}^1 &= 40.08 \pm 0.12s & T_{RK}^2 &= 34.93 \pm 0.39s
 \end{aligned}$$

$$\begin{aligned}
 \Delta E_{LF}^1 \times T_{LF}^1 &= 270 \times 10^{-6} \pm 10^{-6} MeVs \\
 \Delta E_{RK}^1 \times T_{RK}^1 &= 640 \times 10^{-6} \pm 2 \times 10^{-6} MeVs \\
 F_{LF}^1/F_{RK}^1 &= \frac{\Delta E_{RK}^1 \times T_{RK}^1}{\Delta E_{LF}^1 \times T_{LF}^1} = 2.4 \pm 0.02
 \end{aligned} \tag{5.17}$$

$$\begin{aligned}
 \Delta E_{LF}^2 \times T_{LF}^2 &= 160 \times 10^{-5} \pm 10^{-5} MeVs \\
 \Delta E_{RK}^2 \times T_{RK}^2 &= 320 \times 10^{-5} \pm 10^{-5} MeVs \\
 F_{LF}^2/F_{RK}^2 &= \frac{\Delta E_{RK}^2 \times T_{RK}^2}{\Delta E_{LF}^2 \times T_{LF}^2} = 2.0 \pm 0.03
 \end{aligned} \tag{5.18}$$

Leap-frog provides 2.4 times and 2.0 times less overacceleration per simulation time than *RK4* in parallel and perpendicular electric fields respectably. This results can be tested further with obsering from the data (see *Tables C.3, C.4, C.5 and C.6* of *Appendix C*),

$$\begin{aligned}
 \Delta E_{LF}^1(dt = 2 \times 10^{-5}) &= \Delta E_{RK}^1(dt = 3 \times 10^{-5}) = 45 \times 10^{-6} MeV \\
 T_{LF}^1(dt = 2 \times 10^{-5}) &= 6.23 \pm 0.02s
 \end{aligned}$$

$$\begin{aligned} T_{RK}^1(dt = 3 \times 10^{-5}) &= 13.40 \pm 0.03s \\ F_{LF}^1/F_{RK}^1 &= \frac{T_{RK}^1(dt = 3 \times 10^{-5})}{T_{LF}^1(dt = 2 \times 10^{-5})} = 2.15 \pm 0.02 \end{aligned} \quad (5.19)$$

$$\Delta E_{LF}^2(dt = 3 \times 10^{-5}) \approx \Delta E_{RK}^2(dt = 5 \times 10^{-5}) \approx 470 \times 10^{-6} MeV$$

$$\begin{aligned} T_{LF}^2(dt = 3 \times 10^{-5}) &= 3.62 \\ T_{RK}^2(dt = 5 \times 10^{-5}) &= 7.00 \\ F_{LF}^2/F_{RK}^2 &= \frac{T_{RK}^2(dt = 5 \times 10^{-5})}{T_{LF}^2(dt = 3 \times 10^{-5})} = 1.9 \pm 0.1 \end{aligned} \quad (5.20)$$

The last uncertainty was taken high due to the approximation of $467 \approx 471$. After combining *equations 5.17, 5.18, 5.19 and 5.20*, the performance of *Leap-frog* relative to *RK4* in e^- - \vec{E} interaction can be calculated as in *equation 5.21*.

$$F_{LF}/F_{RK} = 2.11 \pm 0.03 \quad (5.21)$$

Therefore, it can be concluded that *Leap-frog* outperforms *RK4* with the relative performance of 2.11 ± 0.03 in e^- - \vec{E} interactions with static uniform \vec{E} field.

5.2.6. Multithreading

As already mentioned before, multithreading implementation efforts were ongoing since right after the start of this project. There have been a number of different approaches for implementation. First implemenation was done when the *Rhodotron Simulation* was only capable of 1D simulations. After the sizable refactoring done for 3D capabilities, this implementation was obselete.

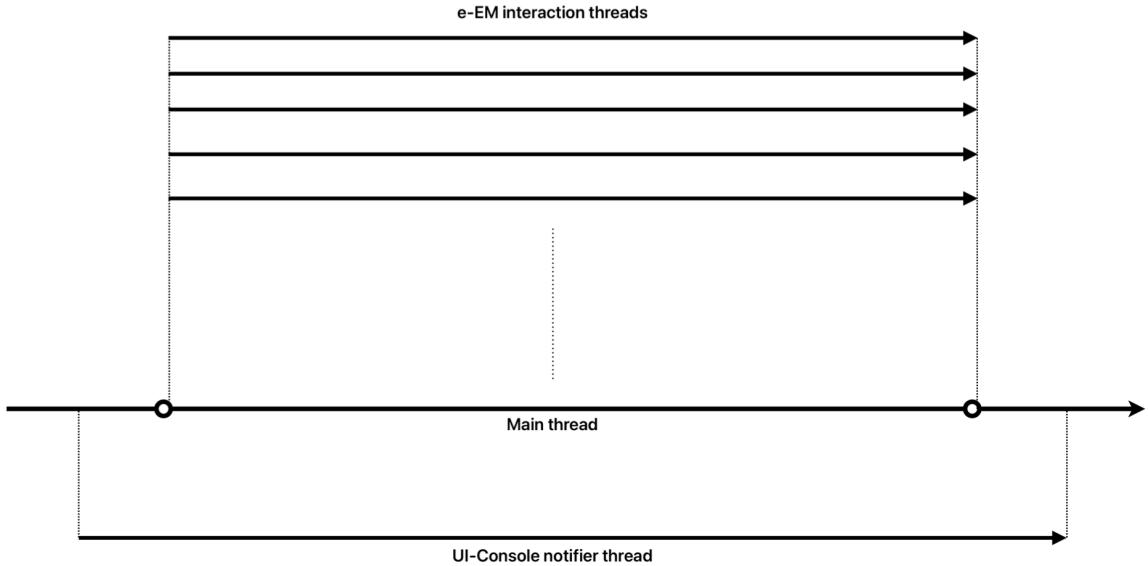


Figure 5.11: An Illustration of the multithreading architecture.

For the current version, a main thread that spawns and manages several other worker threads would be used as can be seen in *figure 5.11*. The UI-Console thread would handle incoming and outgoing communication, notifying the user about the status of simulation (see *figure B.5* in *Appendix B* for example console notification, *figure A.12* in *Appendix A* for implementation) or communicating with the *GUI* that will be discussed in the next section.

Focus of this section is the worker threads, also known as e^- - **EM** interaction threads. There were four competing architecture for these worker threads,

- (i) Have a thread pool that calculates e^- - **EM** in a queue
- (ii) Assign a thread to each bunch
- (iii) Assign random electrons to each thread and calculate e^- - **EM** with global time
- (iv) Assign random electrons to each thread and calculate e^- - **EM** with local thread time

Architecture 1 would require constant waiting in worker threads to get mutexes of \vec{E} field, especially in *RK4*.

Architecture 2 was performing well in configurations with a large number of bunches, but was not increasing performance in lower bunch count configurations as expected.

Architecture 3 was inefficient and wasteful since all the worker threads would wait the main thread to get the next time after they finish calculation, while the main thread would be waiting for the slowest worker thread.

Architecture 4 was thought to be the best performer. It would give freedom to calculate the whole simulation to each thread while giving up the global time. This would also mean thread-safety is ensured since there is no shared data between the worker threads. However, this architecture can cause issues if $e^- - e^-$ interactions were decided to be implemented in the future.

Another caveat is that a copy operation of \vec{E} and \vec{B} objects for each worker thread would be needed. This would lead to larger memory allocations and more time spent setting up simulations. Therefore, *Architecture 4* is not ideal for fast calculations and when the memory is an important constraint. An implementation that can use both **Architecture 2 & Architecture 4** when necessary would be a better approach considering these methods; nevertheless, **Architecture 4** was chosen to be implemented.

The implementation can be found in *figures A.8, A.9, A.10 and A.11* in *Appendix A*.

5.3. Graphical User Interface

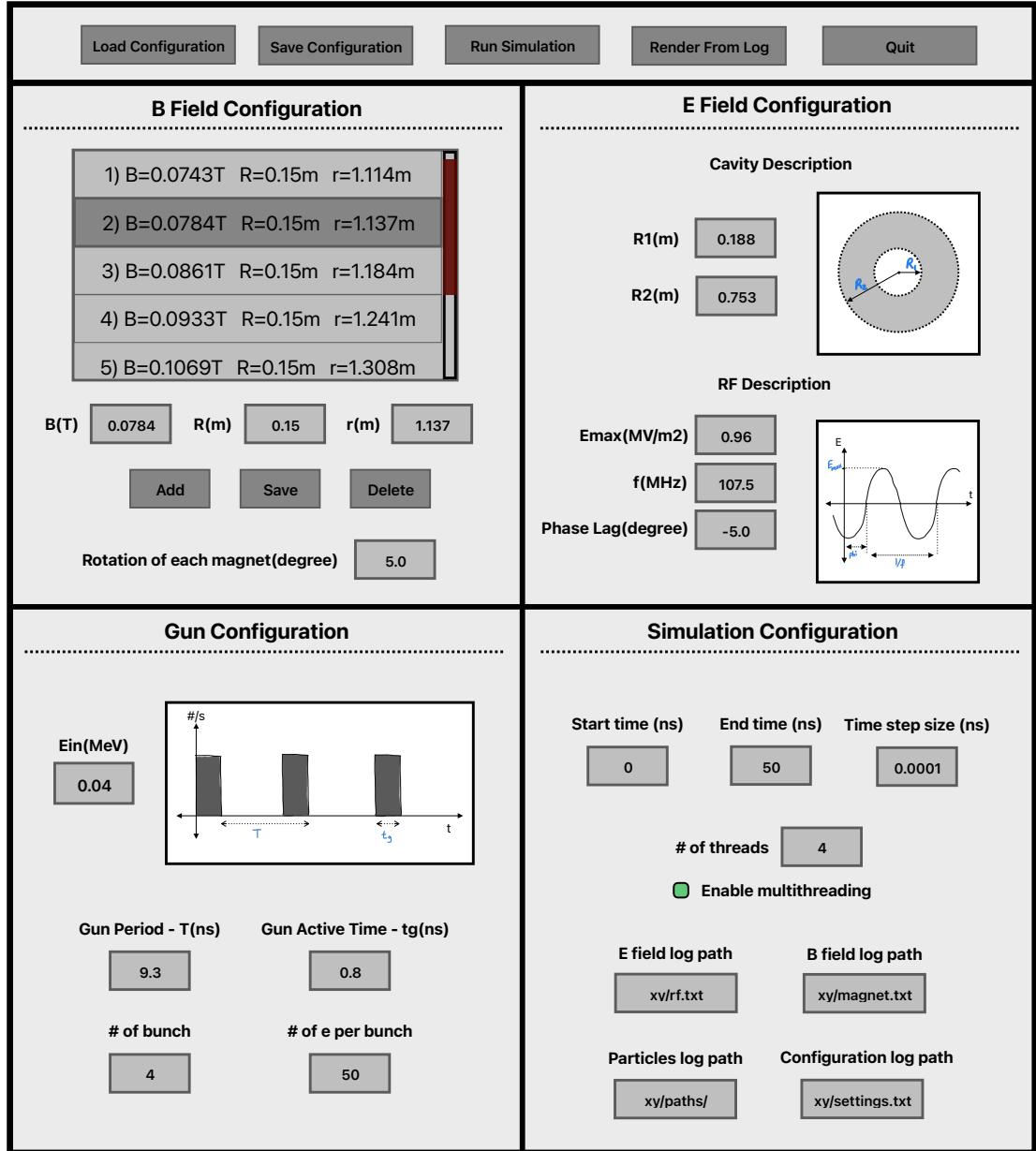


Figure 5.12: An Illustration of the first GUI design.

Until this point, *Rhodotron Simulation* could be used with a configuration file, defining the problem that would be simulated. An example of this configuration file can be found in *figure B.4* of *Appendix B*. This approach was simple and fast; however, it was not suitable for the average target user since required basic knowledge of command

line interface and was not up to modern standards. For this reason, a GUI was decided to be built, using *ROOT framework*. This would also enable *Rhodotron simulation* to make use of analysis tools offered by *ROOT*.

The initial design of the *Rhodotron Simulation GUI* can be observed in *figure 5.12.*

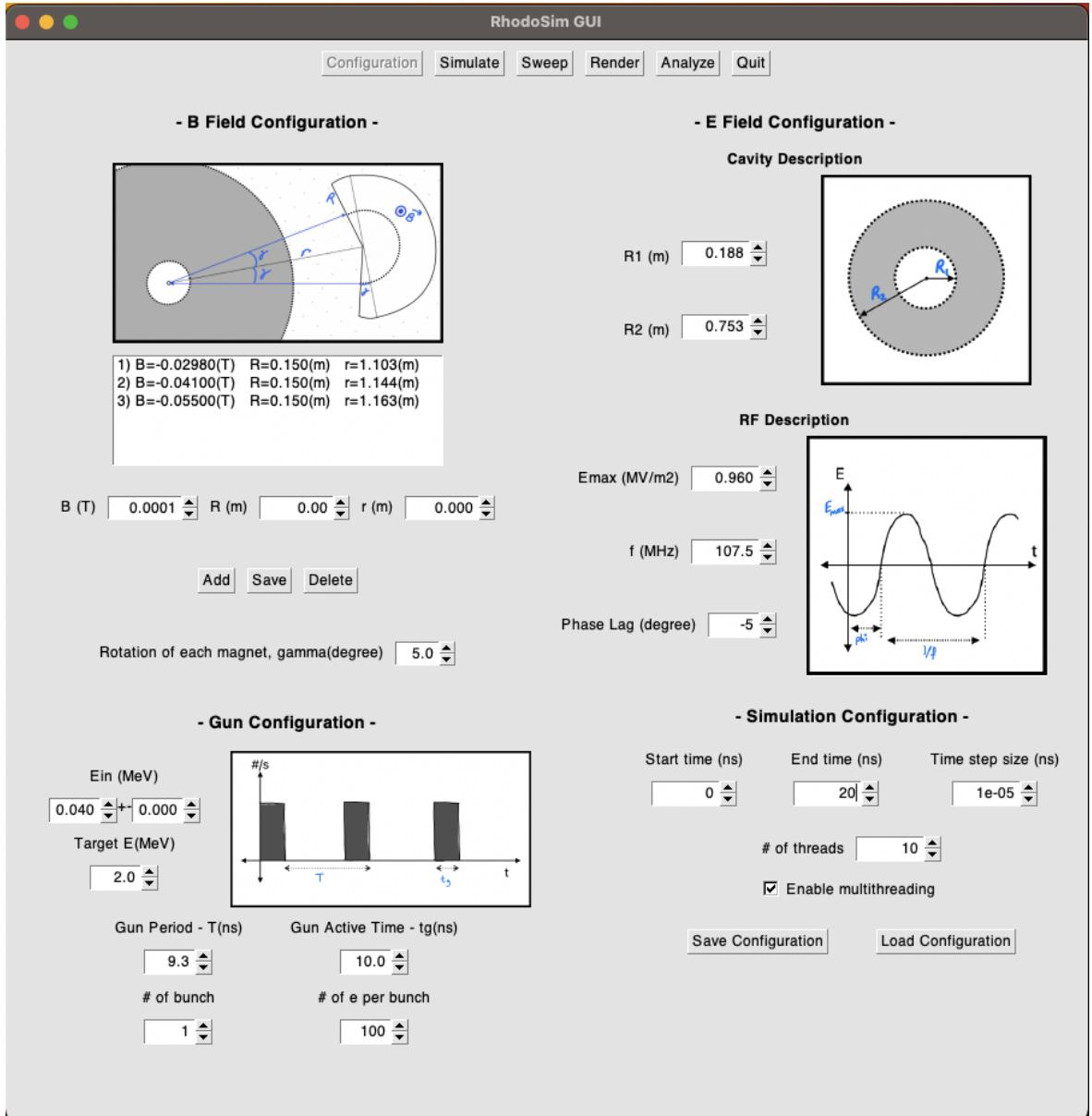


Figure 5.13: The configuration frame of implemented first GUI design.

The *GUI* would be a standalone application, running the *Rhodotron Simulation* as a service when needed. For this reason, the now called *simulation engine* was

updated to be able to run as a background service of *GUI*. By this approach, one could also ignore the *GUI* altogether and use the *simulation engine* as before, as these are two separate products.

In the *figure 5.13*, the implemented version of *figure 5.12* can be observed. This version of the *GUI* has the following frames;

5.3.1. Configuration Frame

This frame provides an interface for specifying, saving or loading a configuration, consists of $\vec{\mathbf{B}}$ field, $\vec{\mathbf{E}}$ field, *RF* description, e^- -gun , *simulation* configuration regions. Each one having an illustration of what the parameters are as can be seen in *figure 5.13*.

5.3.2. Simulation Frame

This frame spawns and manages the *simulation engine*, configures and starts the simulation. It has a progress bar that shows the current progress of the simulation, communicating with *UI handler thread* in *simulation engine*. Simulation frame after a simulation run has been completed can be observed in *figure 5.14*.

5.3.3. Render Frame

Since the rendering capabilities of *ROOT* is superior than *gnuplot*, the user can render a playback of the simulation in real time, see a specific time frame, export as snapshot or animated gif file. Two example renders can be seen in *figures 5.15 and 5.16*



Figure 5.14: Simulate frame of *GUI*.

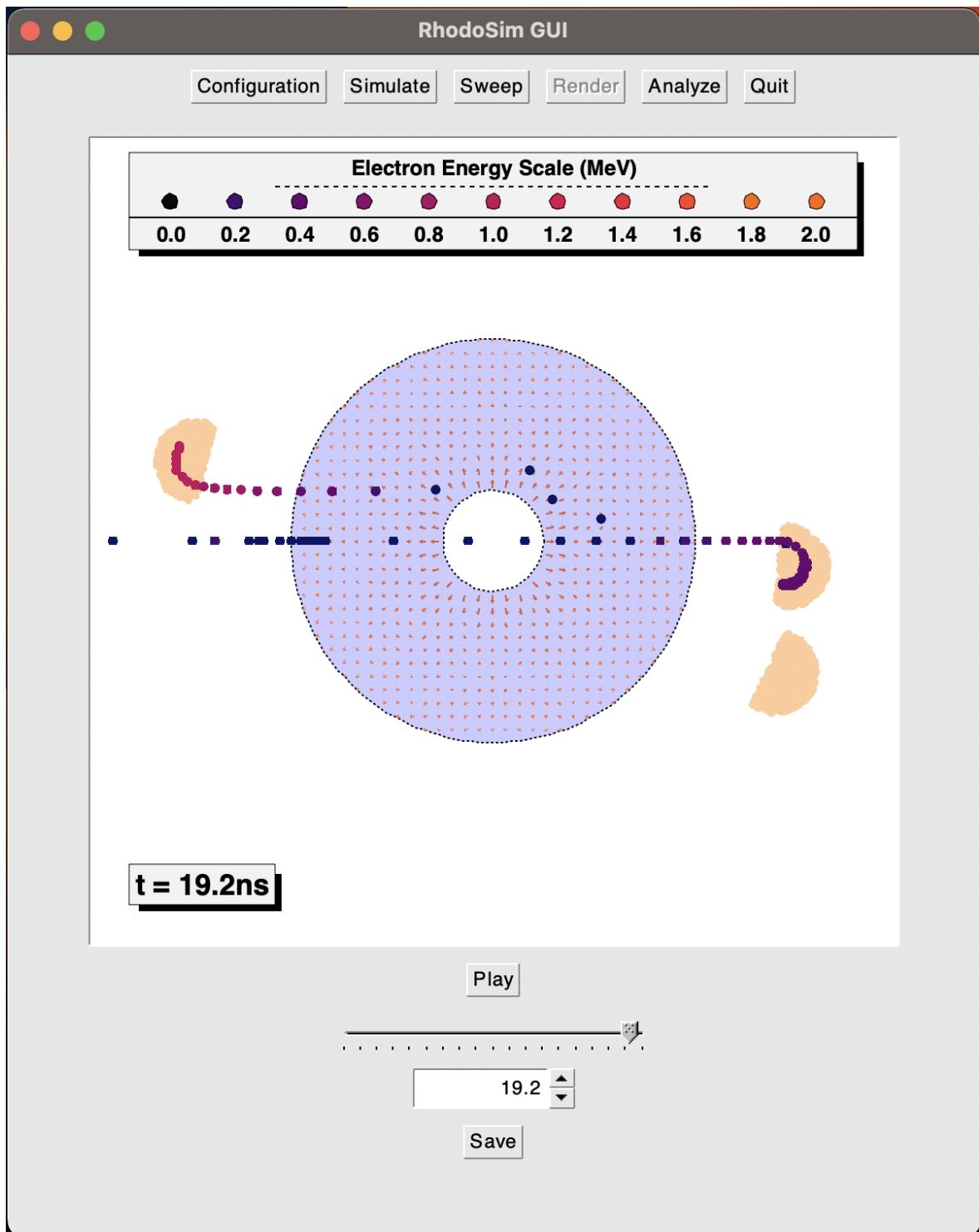
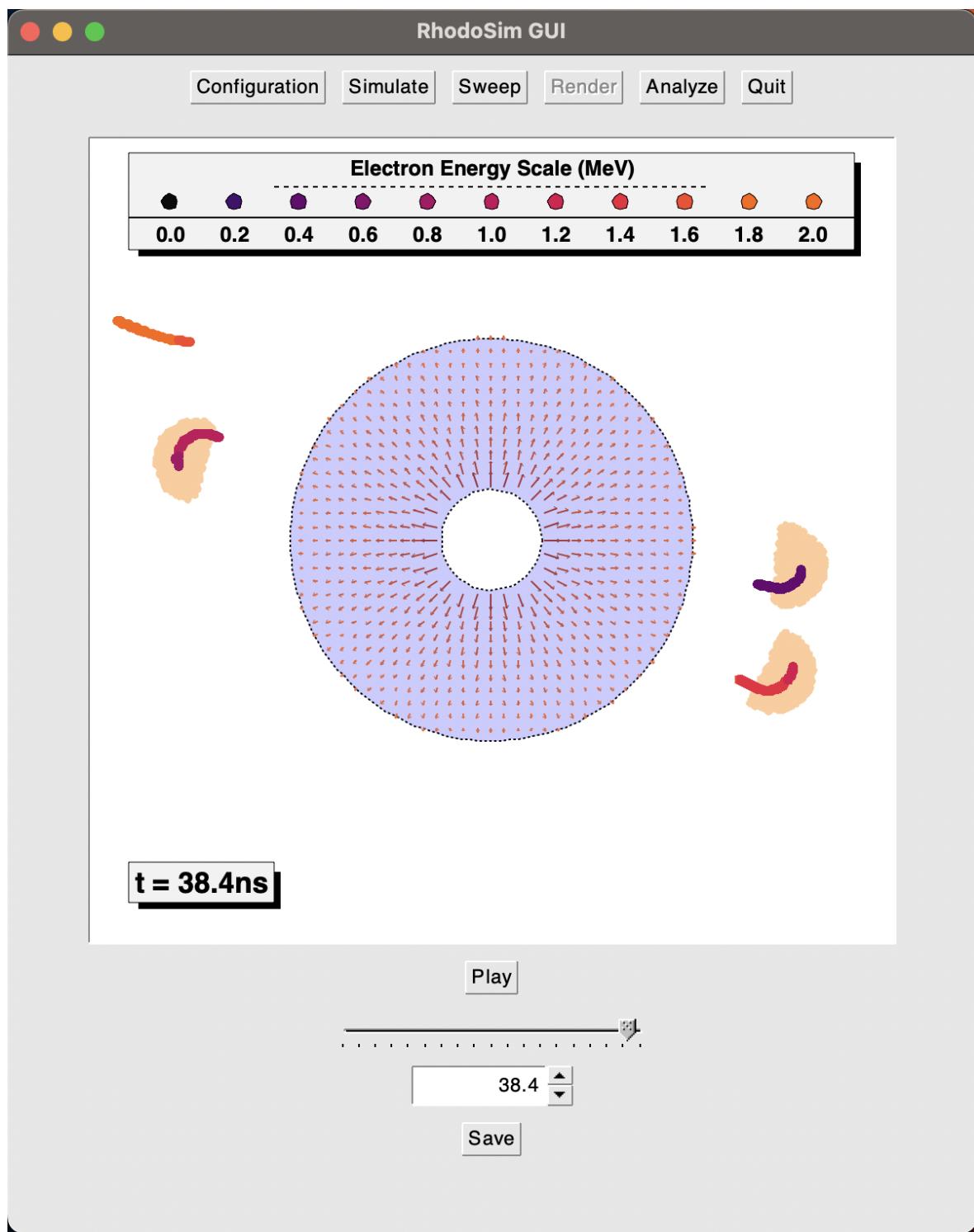


Figure 5.15: Render frame of *GUI*.

Figure 5.16: Render frame of *GUI*.

5.3.4. Analyze Frame

Analyze frame provides tools for analyzing and visualizing the simulation data. In the current version, **E** distribution histogram and $(E)(t)$ graph of each electron are implemented into this frame. This frame is a work in progress and will be the focus of improvement and become a really powerfull tool in the near future. In *figure 5.17*, *Analyze Frame* plotting **E** distribution histogram, and in *figures 5.18 and 5.19*, *Analyze Frame* plotting $(E)(t)$ graphs can be observed.

5.3.5. Sweep Frame

Parameter sweep method that was discussed in *sections 5.2.1 and 5.2.2* was integrated into *GUI* in a seperate frame named *Sweep Frame*. ϕ_{lag} sweep was the first parameter to be implemented. It takes the range of sweep, draws $\phi_{lag} vs \mu E$, $\phi_{lag} vs \sigma E$ and $\phi_{lag} vs \sigma R$ mentioned in *section 5.2.2*. This enables user with an already built accelerator to optimize e^- -gun parameters quickly. In the following *figures 5.20, 5.21, 5.22 and 5.23*, this frame can be inspected in detail.

As mentioned before, *GUI* is the latest and ongoing development effort in this project. New analysis and sweep features will be implemented and current capabilities will be improved in the near future.

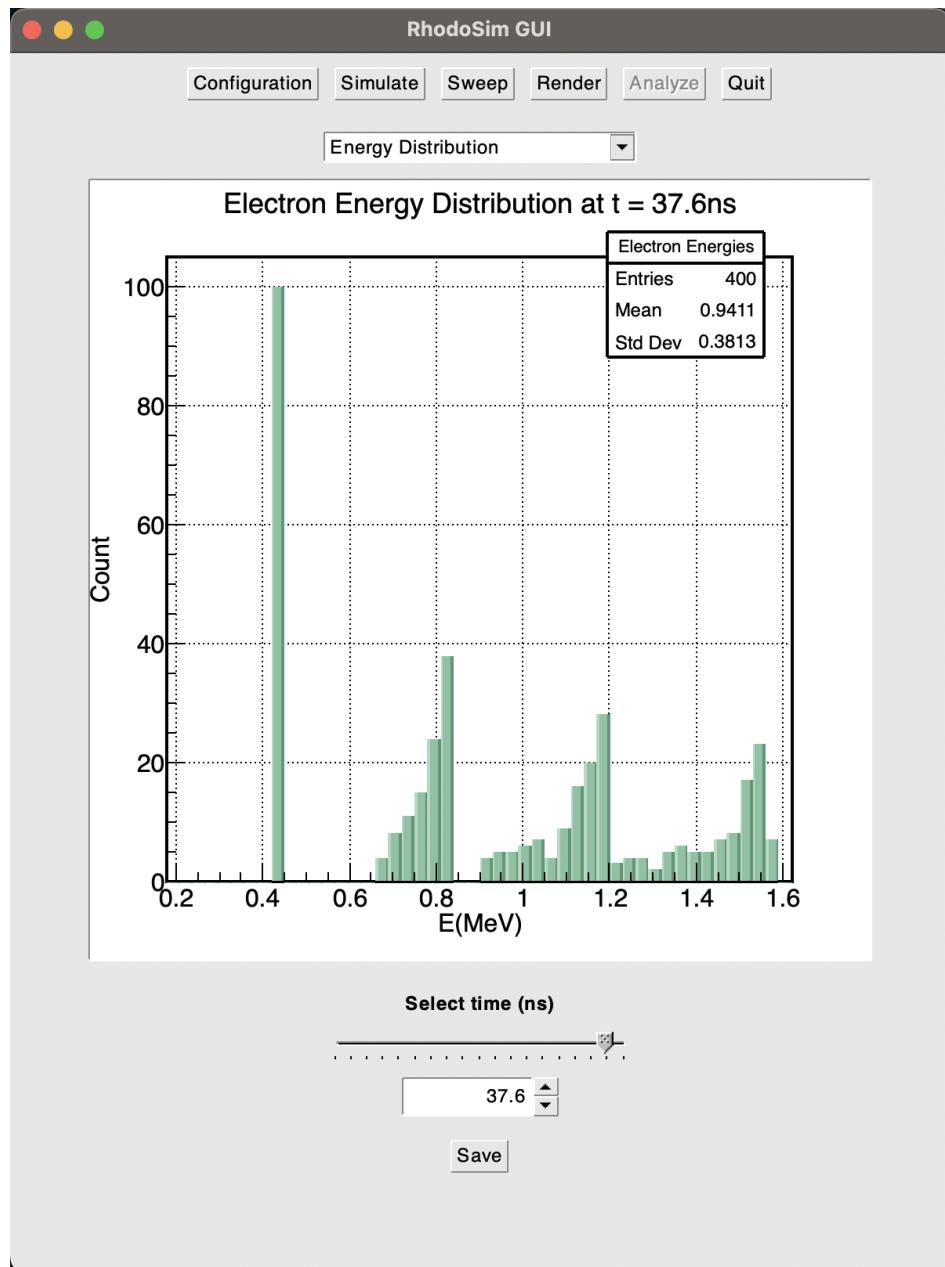
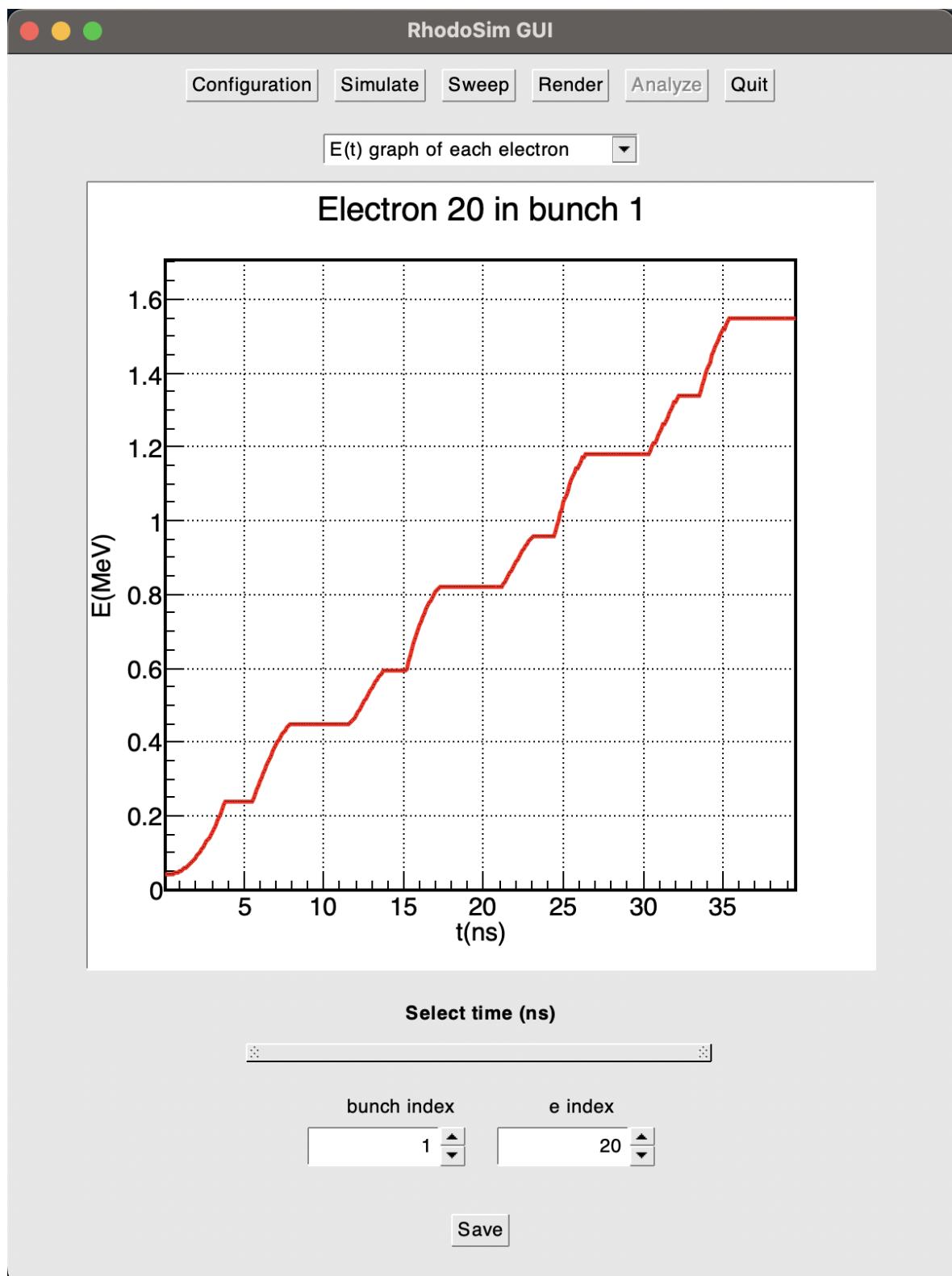


Figure 5.17: Analyze frame of *GUI* E distribution.

Figure 5.18: Analyze frame of *GUI E(t)*.

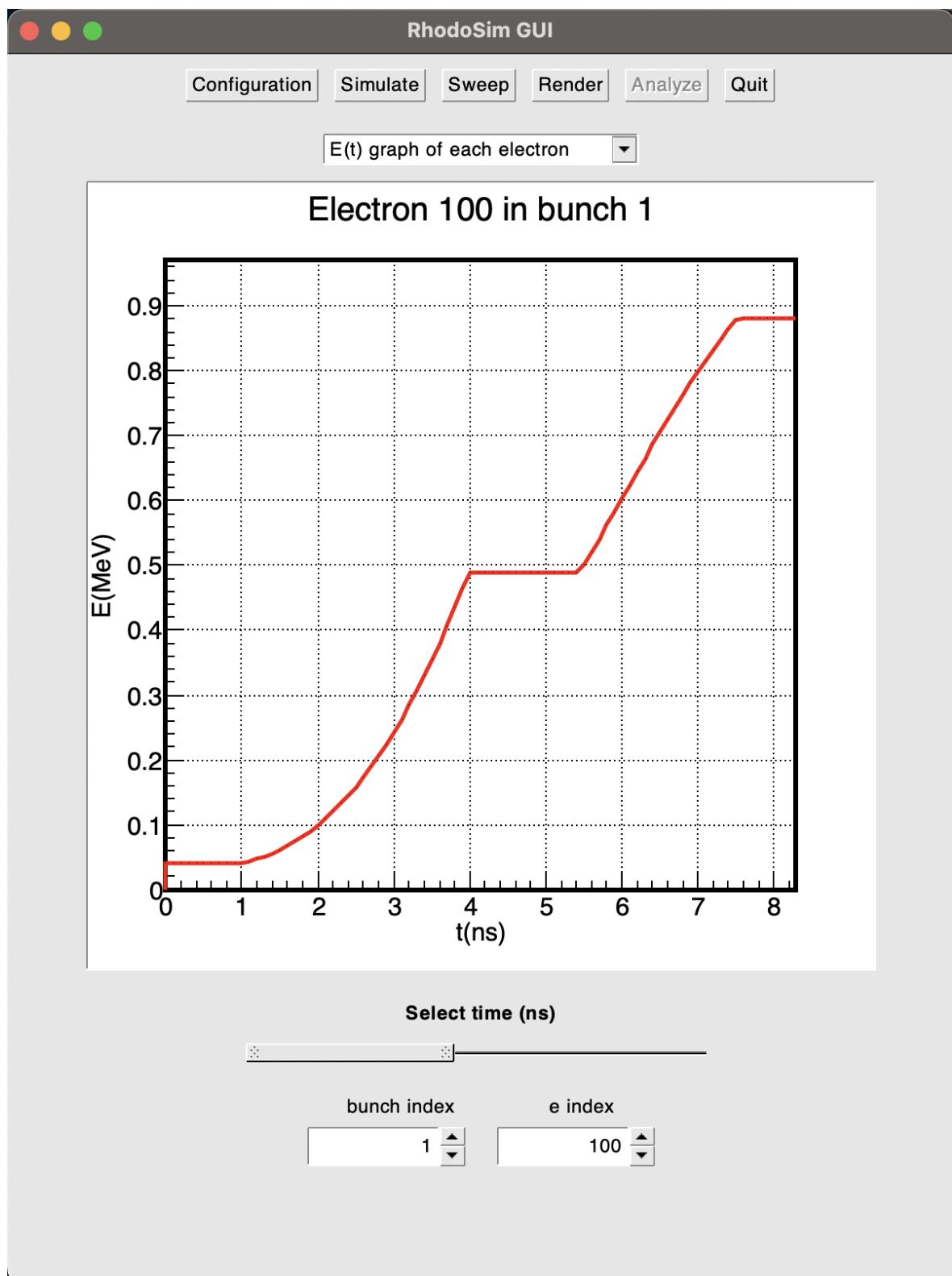
Figure 5.19: Analyze frame of *GUI E(t)*.



Figure 5.20: Sweep frame of *GUI*

ϕ_{lag} sweep running.

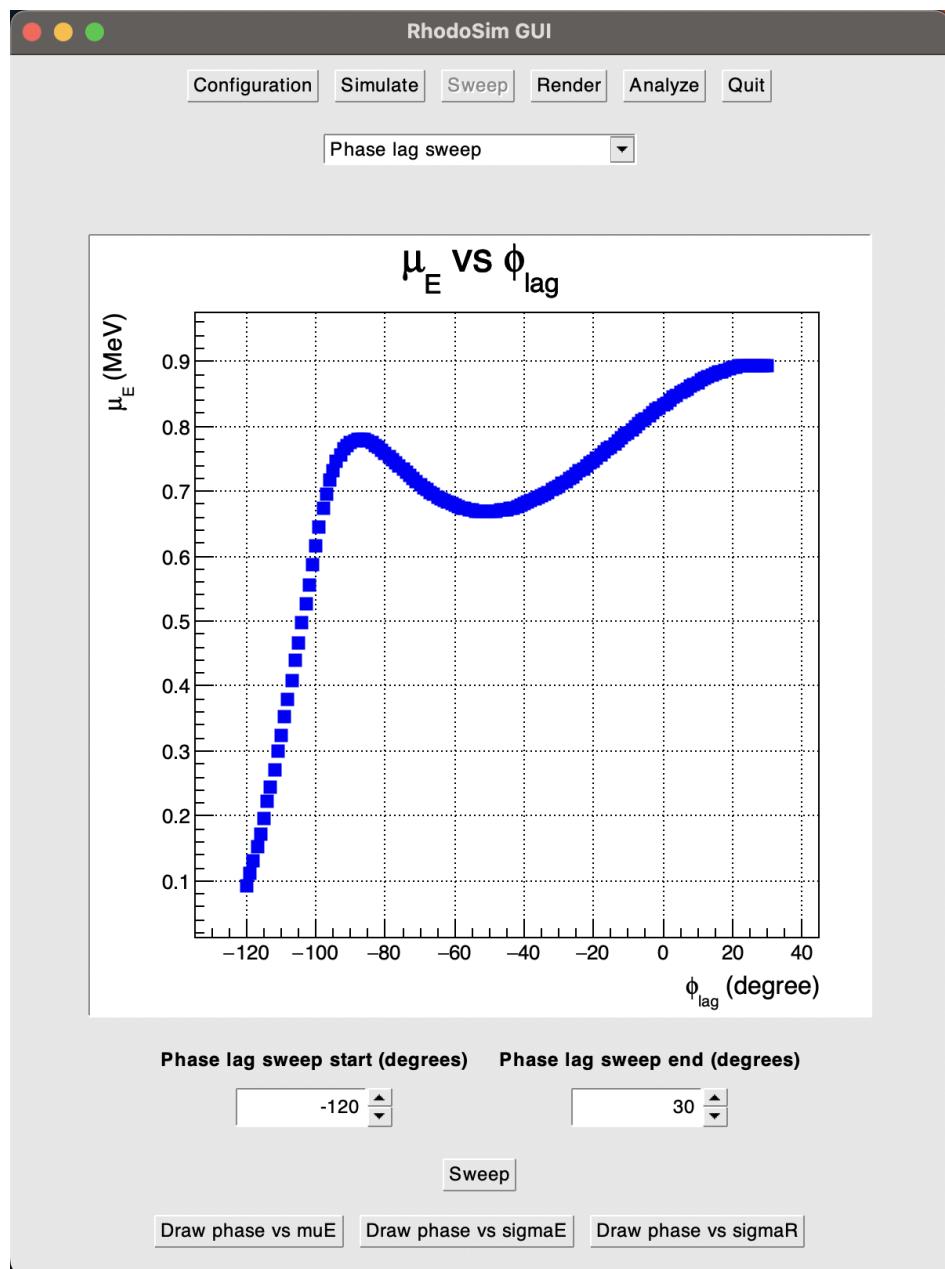


Figure 5.21: Sweep frame of *GUI*

ϕ_{lag} μE result.

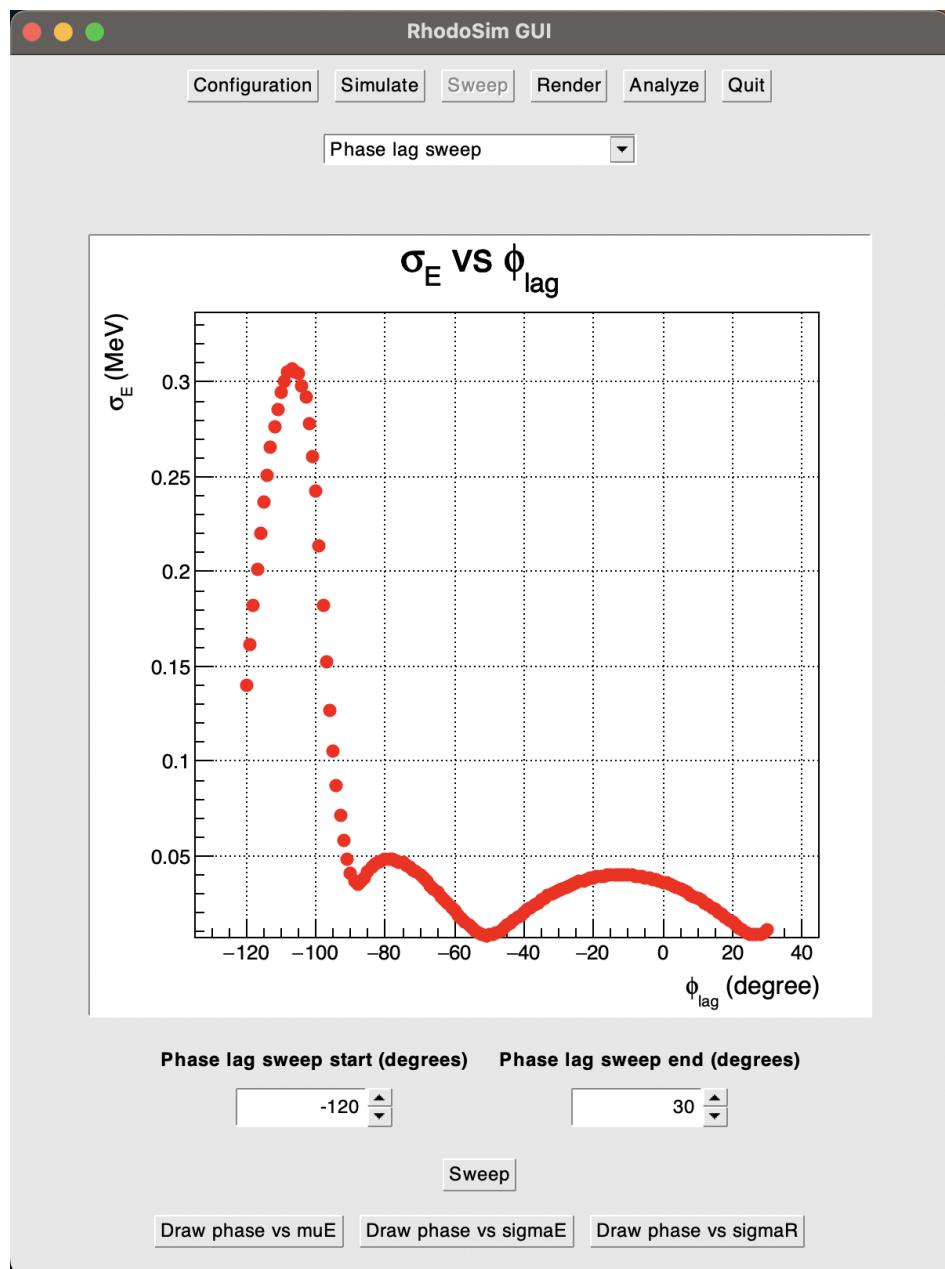


Figure 5.22: Sweep frame of *GUI*

ϕ_{lag} sweep σE result.

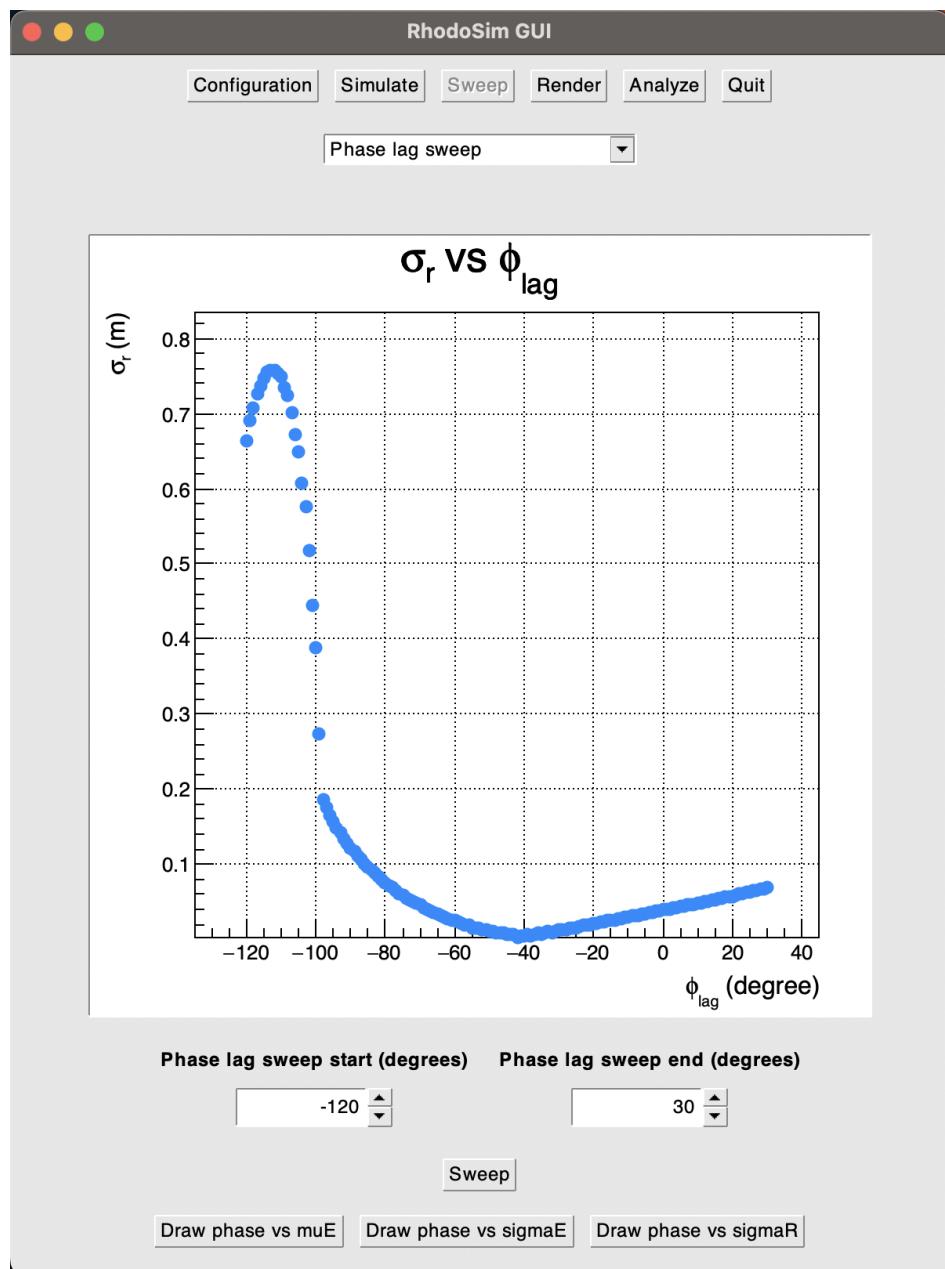


Figure 5.23: Sweep frame of *GUI*

ϕ_{lag} sweep σR result.

6. PRODUCTION

6.1. Final Design

As discussed before, production efforts of a rhodotron type accelerator is still ongoing in KAHVELab. This accelerator will be used at $f = 107.5\text{MHz}$ $P = 50 - 100\text{kW}$. In the following figures, rendered visuals of the final design by *Mr. Sinan Öz* can be observed.

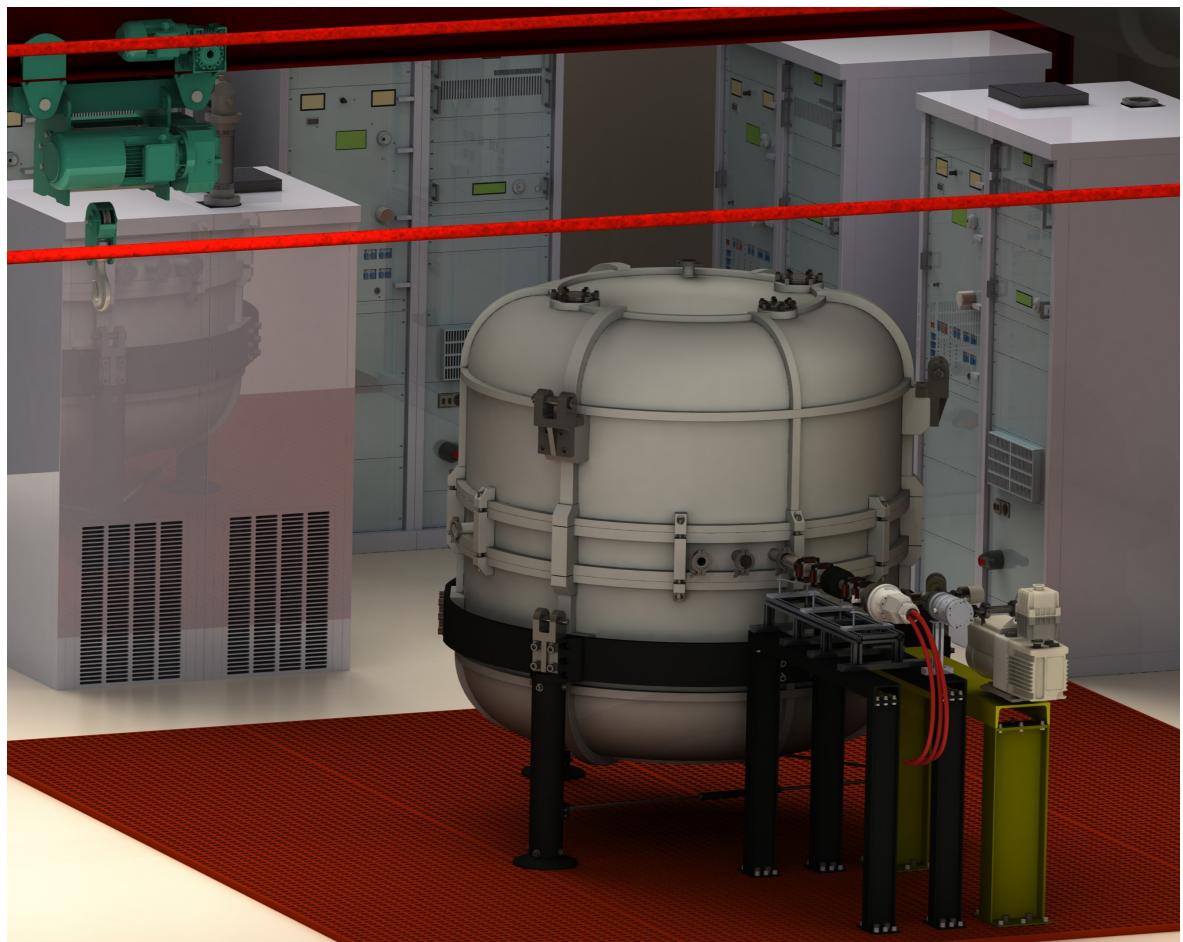


Figure 6.1: Rhodotron in KAHVELab



Figure 6.2: Rhodotron cavity and e^- gun

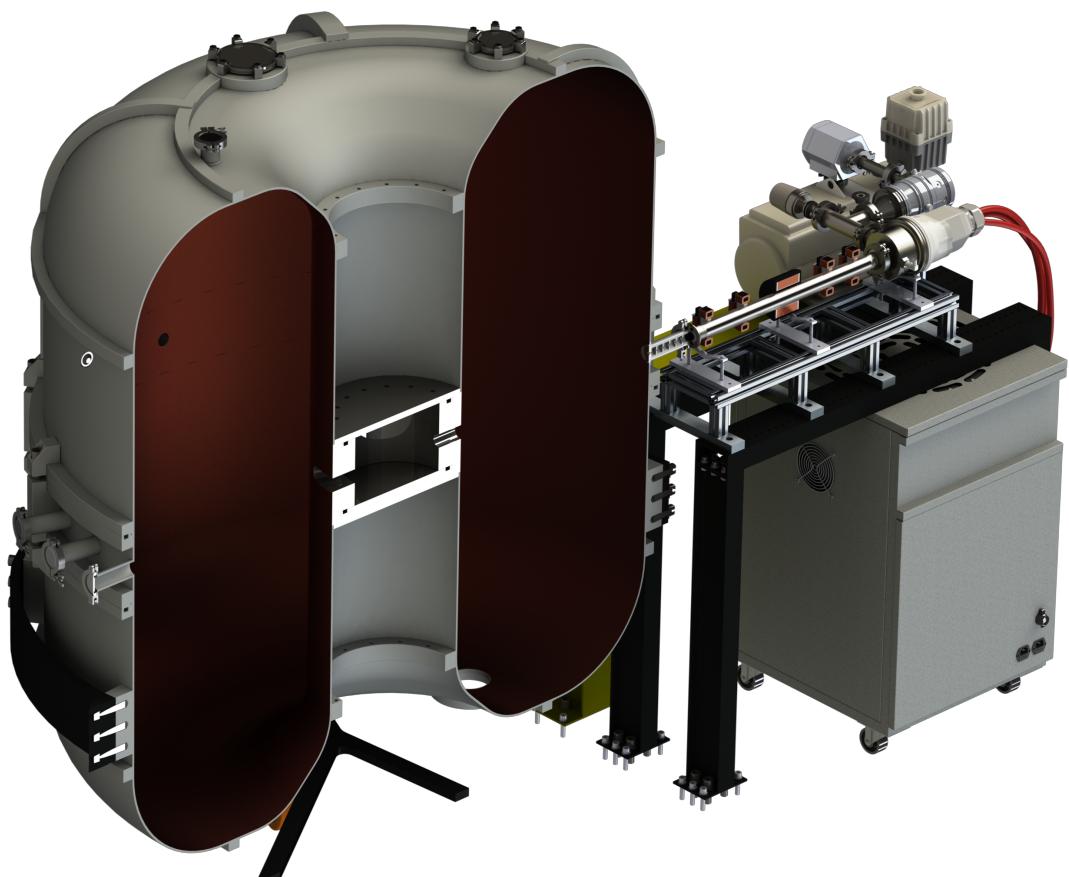


Figure 6.3: Vertical cross section of the beam injection line

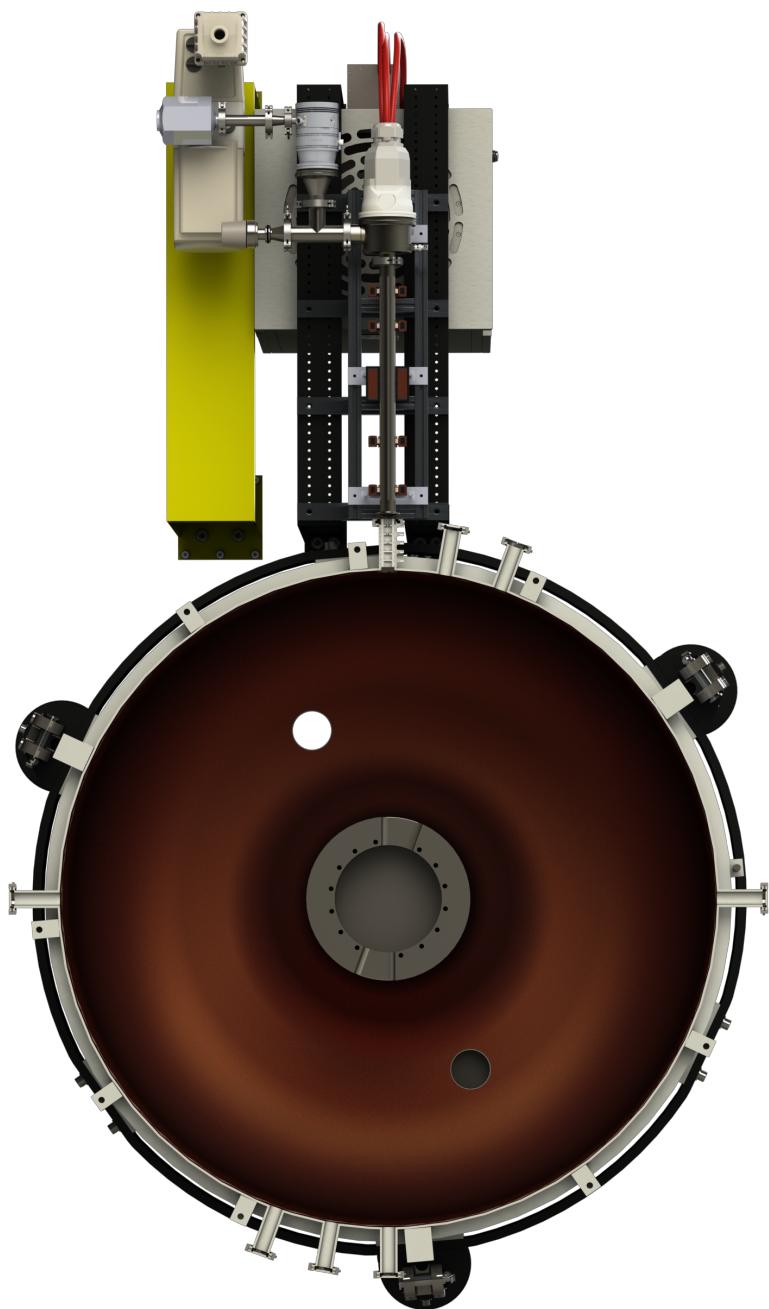


Figure 6.4: Cross section of the acceleration plane

6.2. Technical Drawing

Technical drawings of the design mentioned in *section 6.1* can be found in following figures:

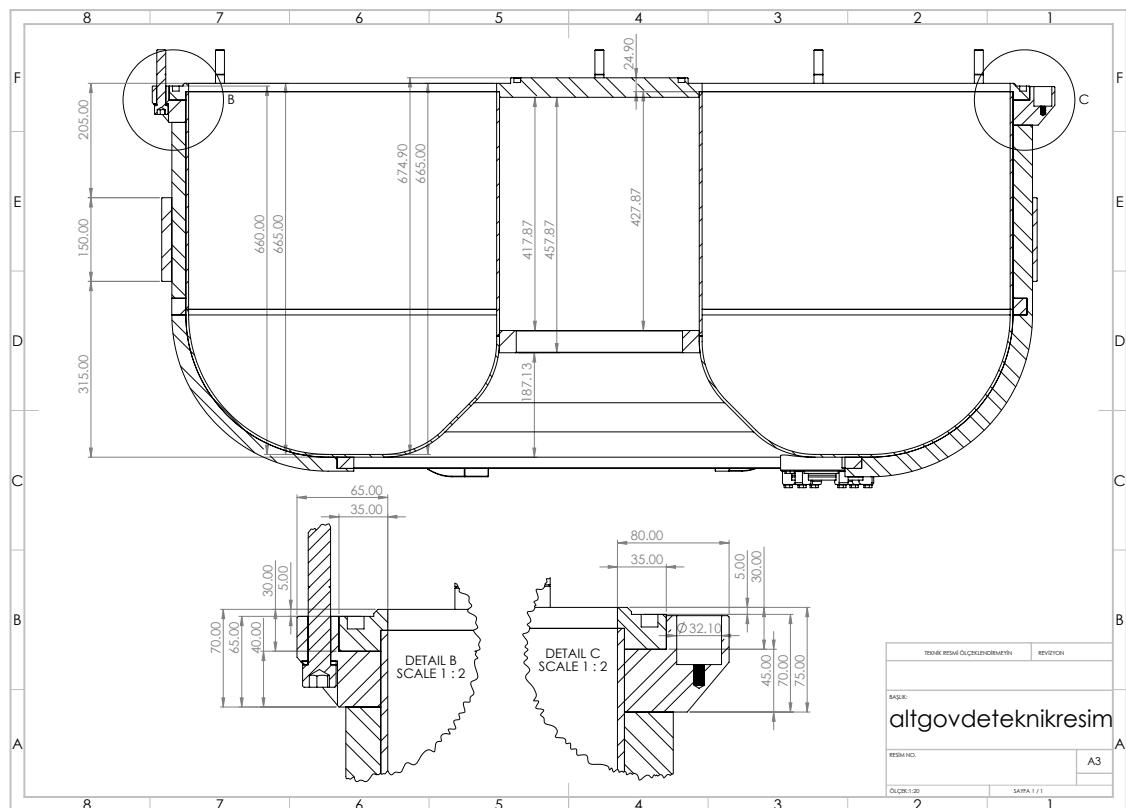


Figure 6.5: Technical drawing of bottom part cross section of the cavity.

Courtesy of Mr. Sinan Öz

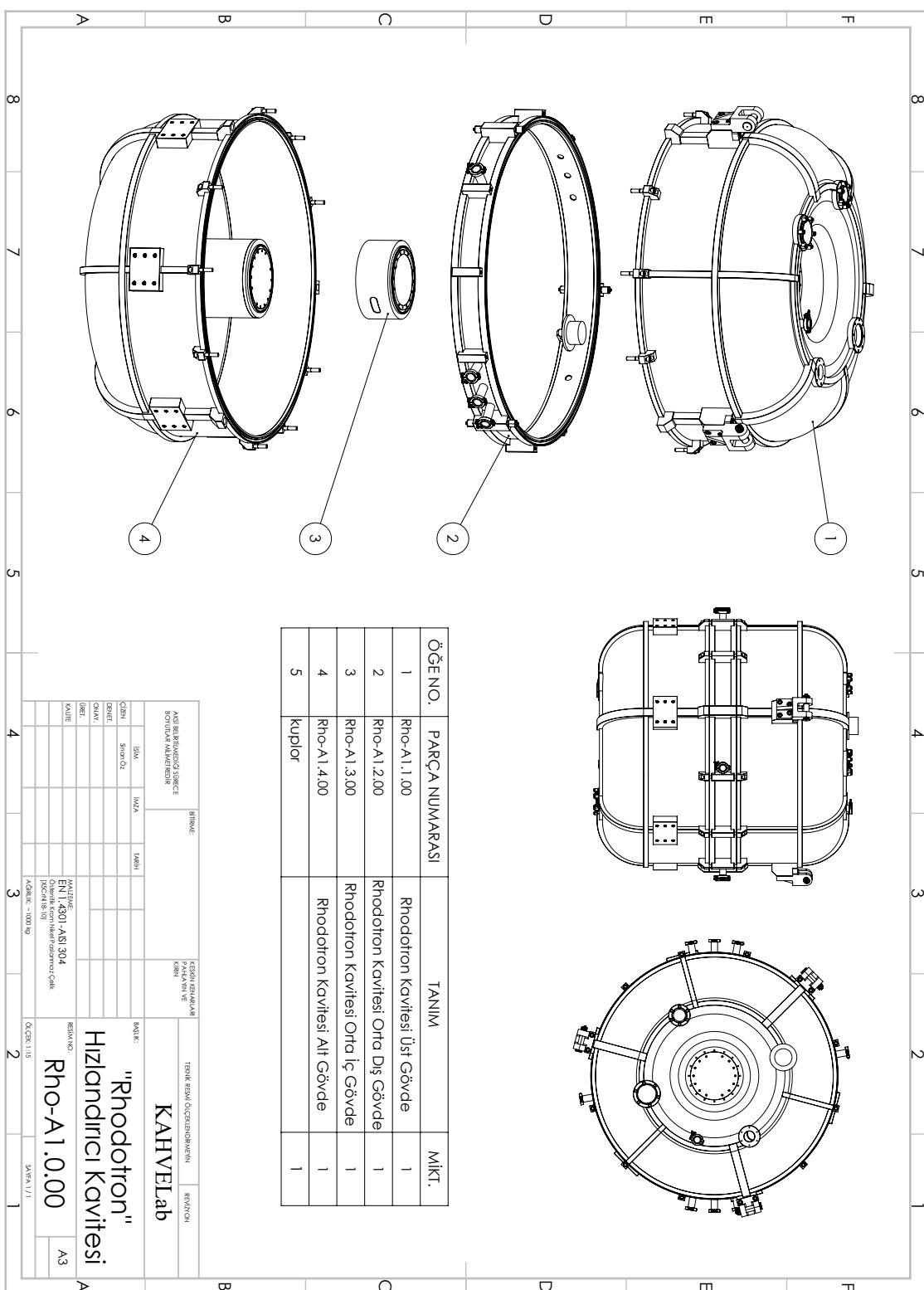


Figure 6.6: Technical drawing of the rhodotron cavity.

Courtesy of Mr. Sinan Öz

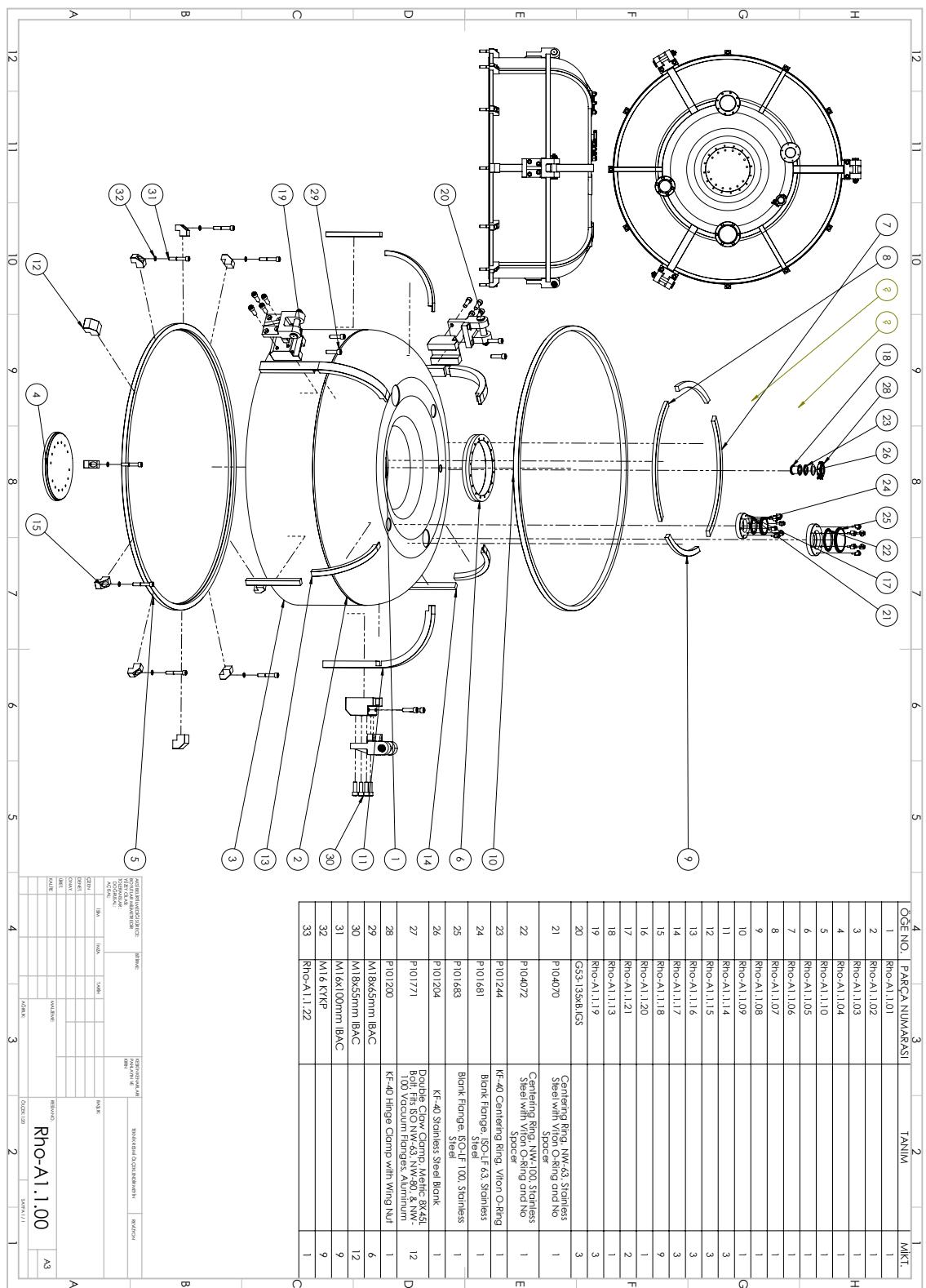


Figure 6.7: Technical drawing of upper part of the cavity.

Courtesy of Mr. Sinan Öz

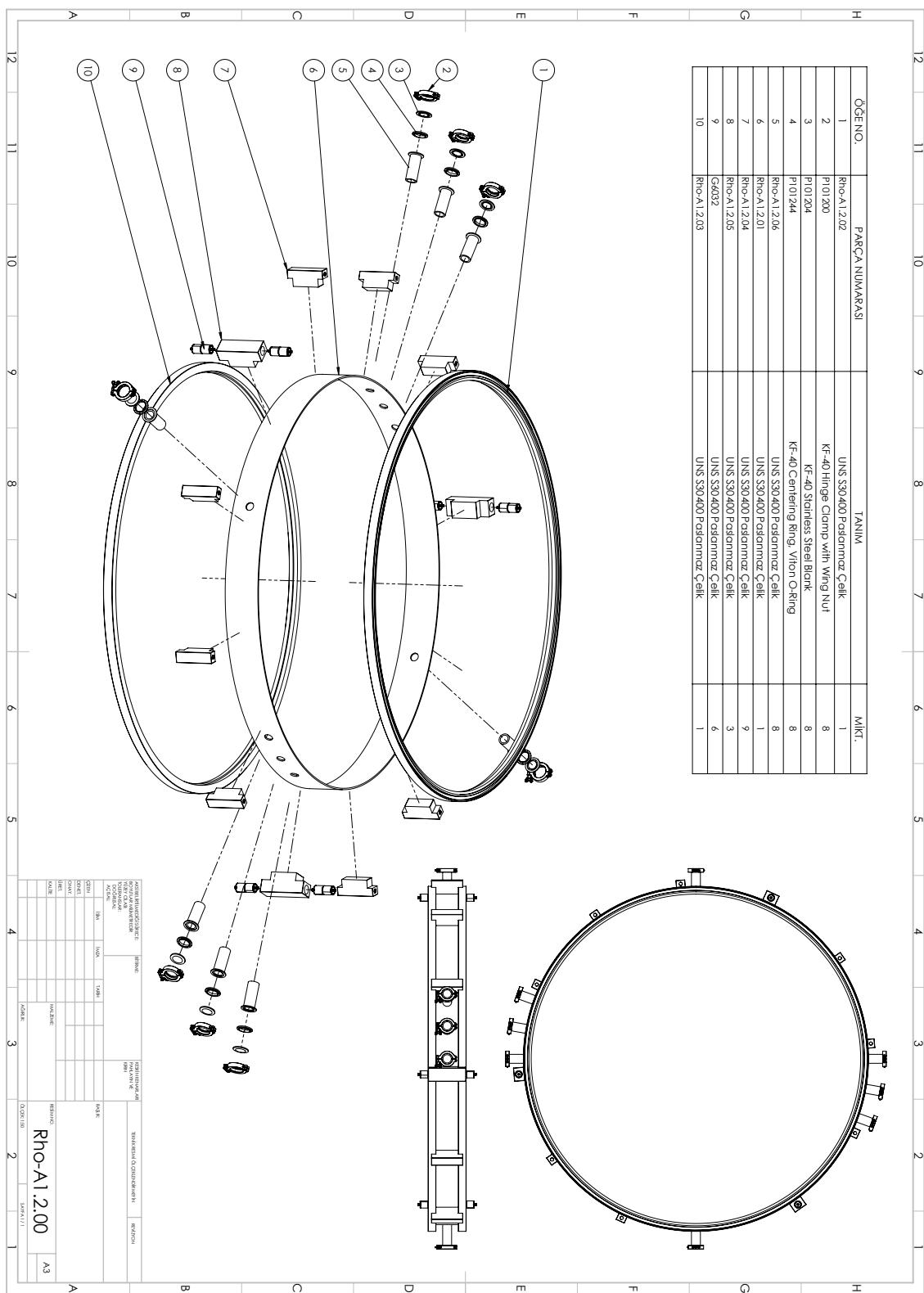


Figure 6.8: Technical drawing of middle part of the cavity which contains the beam line.

Courtesy of Mr. Sinan Öz

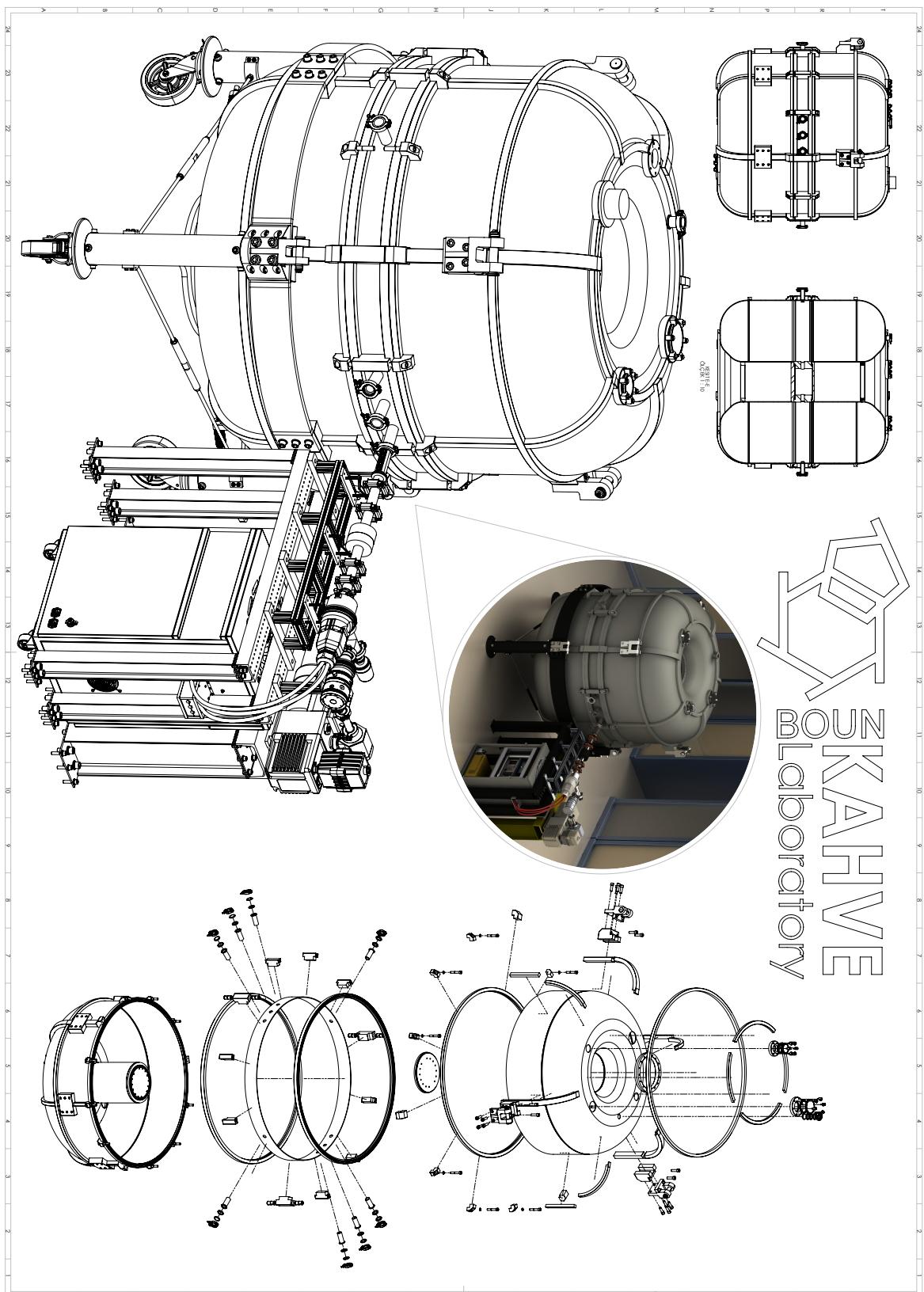


Figure 6.9: Technical drawing of middle part of the cavity which contains the beam line.

Courtesy of Mr. Sinan Öz

6.3. Manufacturing

Manufacturing of the rhodotron cavity has been ongoing, planned to be completed in the following months.

The cavity itself was manufactured as 5 main parts, using 5mm thick stainless steel sheet.

Elastic version of the 304, 304L was chosen to be the production material for ease of bending. These sheets were then pressed to achieve the shapes of the parts. Several flanges were then machined, including

- 4.5in EIA RF flange for RF input
- ISO100-KF vacuum connector flanges for vacuum pumps
- ISO-63 KF flange for probe insertion
- KF40 flanges for beam line vacuum gauges

After desired shapes were produced inside surface of the sheets were polished with 2000 grits. 304 stainless steel sheet bars of different sizes were then added by TIG welding to achieve pressure resistance.

Because the main body of the cavity was 5mm thick and 304L was used, deformations on cylindrical symmetry were encountered. This problem was solved by welding 6 10mm thick toroidal sheets between the coaxial cylinders, which would be removed after the heat treatment.



(a) Bottom part of the cavity while polishing. (b) Top part of the cavity after polishing.

Figure 6.10: Polishing of the inner surface of cavity.



(a) Bottom part of the cavity with the exposed beam line in inner cylinder.

(b) Bottom part of the cavity with the beam line assembled.

Figure 6.11: Deformation prevention measure, 10 mm thick toroidal sheets.



Figure 6.12: Welded sheet bars.



Figure 6.13: Middle part of the cavity containing the beam line.



(a) Welding sheet bars.

(b) Welding toroidal sheets.

Figure 6.14: TIG welding.



Figure 6.15: Assembled cavity as the current stage.

7. FUTURE WORK

At the time of publication, production efforts of a new rhodotron type electron accelerator are currently continuing in KAHVELab. Present bottleneck is the fabrication of the cavity.

7.1. Cavity

The cavity manufacturing process is in its concluding stages. The sheet bars and the flanges of the upper and the lower parts have been produced and are being welded and getting ready for heat treatment. After the heat treatment operation, the supporting toroidal sheets will be removed and the inside surface will be polished before cyanide copper plating.

7.2. Magnets

Magnet production was decided to be kept on hold in the early stages of the project, due to the disagreement on energy gain amount between the calculations of *J. POTTIER* [1] and the simulation results from *CST Studio Suite & Rhodotron Simulation*. The software is planned to be used in magnet design and beam optimizations in KAHVELab after the initial tests are completed.

7.3. Rhodotron Simulation

Rhodotron Simulation, which has been the main focus of this thesis, is ready and waiting to be tested with the cavity that is being manufactured; several improvements and new feature implementations are underway in the mean time. They include the following;

- Electric and magnetic field import
- Field generator module to directly produce field files from specified cavity
- Synchrotron radiation calculations to determine radiation and energy loss
- $e^- - e^-$ interactions
- Redesign of the GUI
- 3D render in GUI
- Refactoring of the $e^- - \mathbf{EM}$ interaction and logging to improve performance
- Refactoring of the GUI Render Frame to improve render speed
- Refactoring the magnet class to introduce field leaks
- Extension of analysis tools in Analyze Frame
- L_{out} sweep in Sweep Frame

7.4. Conclusion

In conclusion, the performance and the accuracy of this new tool, *Rhodotron Simulation*, cannot be determined before the cavity production is completed and the performance is compared with the predictions of the tool; however the capabilities it presents, coupled with the analogous results obtained from other extensively utilized simulation tools, lead to the inference that this tool holds promise.

APPENDIX A: Intermediate Codes

```

1  for(double i = 2; i < 9; i += dT_out){
2      t_dum += i;
3      double enow = gecis(r_pos, vel, Et, t_dum);
4      if( enow > maxE ){
5          maxE = enow;
6          t_opt = i;
7      }
8      t_dum = t;
9  }



---


1  double gecis(double r_pos, double vel, double Et, double &t){
2      for(; r_pos >= -R2 && r_pos <= R2 ; t+=dT){
3          vel = c*sqrt(Et*Et-E0*E0)/Et;
4          double RelBeta = vel/c;
5          double RelGamma = 1.0 / sqrt(1.0-RelBeta*RelBeta);
6
7          double ef=Eradiat(r_pos*1000,t,RFphase*deg_to_rad);
8
9          double acc=ef*1E6*eQMratio/(RelGamma*RelGamma*RelGamma);
10
11         r_pos = r_pos + vel * dT*ns + 1/2*acc*(dT*ns)*(dT*ns);
12         vel=vel+acc*dT*ns;
13         RelBeta = vel/c;
14         RelGamma = 1.0 / sqrt(1.0-RelBeta*RelBeta);
15         Et=RelGamma*E0;
16     }
17     return Et;
18 }
```

Figure A.1: L_{out} Optimization For Single e⁻

```

1 int phase_opt(const vector<double>& Louts, int phase_sweep_range){
2     double minrms = 1;
3     int opt_phase;
4     for(int RFphase = -phase_sweep_range; RFphase <= phase_sweep_range; RFphase++){
5         Bunch bunch1(RFphase);
6         double t1 = 0;
7         bunch1.bunch_gecis_t(t1);
8         bunch1.reset_pos();
9
10        for(int i = 0; i < Louts.size(); i++){
11            bunch1.bunch_gecis_d(Louts[i]);
12            bunch1.reset_pos();
13        }
14
15        if( bunch1.E_rms() < minrms ){
16            minrms = bunch1.E_rms();
17            opt_phase = RFphase;
18        }
19    }
20    return opt_phase;
21 }
```

Figure A.2: ϕ_{lag} Optimization For Initial Bunch Design

```

1 double vector3d::operator* (const vector3d& other){
2     double dot = 0;
3     dot += this->x * other.x;
4     dot += this->y * other.y;
5     dot += this->z * other.z;
6     return dot;
7 }

1 vector3d vector3d::operator% (const vector3d& other){
2     double x_ = (this->y * other.z) - (this->z * other.y);
3     double y_ = (this->z * other.x) - (this->x * other.z);
4     double z_ = (this->x * other.y) - (this->y * other.x);
5     vector3d crossed(x_, y_, z_);
6     return crossed;
7 }
```

Figure A.3: * and % operators of *vector3d* class

```

1  bool isInsideHalfSphere(vector3d e_position, double r, vector3d hs_position){
2      vector3d relative = e_position - hs_position;
3      // r/5 can be changed, use this for now
4      if (relative.magnitude() <= r && relative * hs_position.direction() >= -r/5){
5          return true;
6      }
7      return false;
8 }
```

Figure A.4: Logic of is e^- inside the shape of magnet

```

1  vector3d Electron2D::interactB_RK(const MagneticField& B, double time_interval){
2      if (B.isInside(pos) == -1){
3          return vector3d(0,0,0);
4      }
5      Electron2D e_dummy;
6      e_dummy.Et = Et;
7      e_dummy.pos = pos;
8      e_dummy.vel = vel;
9      double time_halved = time_interval*0.5;
10     // get k1
11     vector3d F_m = (e_dummy.vel % B.getField(pos))*eQMratio;
12     vector3d k1 = F_m * e_dummy.gamma_inv();
13     // get k2
14     e_dummy.move(time_halved);
15     e_dummy.accelerate(k1, time_halved);
16     F_m = (e_dummy.vel % B.getField(pos))*eQMratio;
17     vector3d k2 = F_m * e_dummy.gamma_inv();
18     // get k3
19     e_dummy.vel = vel;
20     e_dummy.accelerate(k2, time_halved);
21     vector3d k3 = F_m * e_dummy.gamma_inv();
22     // get k4
23     e_dummy.vel = vel;
24     e_dummy.move(time_halved);
25     e_dummy.accelerate(k3, time_interval);
26     F_m = (e_dummy.vel % B.getField(pos))*eQMratio;
27     vector3d k4 = F_m * e_dummy.gamma_inv();
28
29     return (k1 + k2*2 + k3*2 + k4)/6;
30 }
```

Figure A.5: RK4-1 implemenation of $e^- \cdot \vec{B}$

```
1 void Electron2D::interactRK_ActorE(const RFField& E, const MagneticField& B, double time_interval){
2     vector3d run_kut_E = interactE_RK(E, time_interval);
3     vector3d run_kut_B = interactB_RK(B, time_interval);
4
5     vector3d acc = run_kut_E + run_kut_B;
6
7     move(acc, time_interval/2);
8     accelerate(acc, time_interval);
9     move(acc, time_interval/2);
10 }
```

Figure A.6: RK4-1 implemenation of e^- - EM

```

1 void Electron2D::interactRK(RFField& E, MagneticField& B, const double time, double time_interval){
2     Electron2D e_dummy;
3     e_dummy.Et = Et;
4     e_dummy.pos = pos;
5     e_dummy.vel = vel;
6
7     // Calculate k1
8     vector3d acc_E = E.actOn(e_dummy);
9     vector3d acc_B = B.actOn(e_dummy);
10    vector3d acc = acc_E + acc_B;
11    e_dummy.move(time_interval);
12    e_dummy.accelerate(acc, time_interval);
13    vector3d pos_k1 = e_dummy.pos, vel_k1 = e_dummy.vel;
14
15    // Calculate k2
16    e_dummy.pos = (pos + pos_k1)*0.5;
17    e_dummy.vel = (vel + vel_k1)*0.5;
18    e_dummy.Et = e_dummy.gamma()*E0;
19    E.update(time + time_interval*0.5);
20
21    acc_E = E.actOn(e_dummy);
22    acc_B = B.actOn(e_dummy);
23    acc = acc_E + acc_B;
24    e_dummy.move(time_interval);
25    e_dummy.accelerate(acc, time_interval);
26    vector3d pos_k2 = e_dummy.pos, vel_k2 = e_dummy.vel;
27
28    // Calculate k3
29    e_dummy.pos = (pos + pos_k2)*0.5;
30    e_dummy.vel = (vel + vel_k2)*0.5;
31    e_dummy.Et = e_dummy.gamma()*E0;
32    E.update(time + time_interval*0.5);
33
34    acc_E = E.actOn(e_dummy);
35    acc_B = B.actOn(e_dummy);
36    acc = acc_E + acc_B;
37    e_dummy.move(time_interval);
38    e_dummy.accelerate(acc, time_interval);
39    vector3d pos_k3 = e_dummy.pos, vel_k3 = e_dummy.vel;
40
41    // Calculate k4
42    E.update(time + time_interval);
43
44    acc_E = E.actOn(e_dummy);
45    acc_B = B.actOn(e_dummy);
46    acc = acc_E + acc_B;
47    e_dummy.move(time_interval);
48    e_dummy.accelerate(acc, time_interval);
49    vector3d pos_k4 = e_dummy.pos, vel_k4 = e_dummy.vel;
50
51    E.update(time);
52    pos = (pos_k1 + pos_k2*2 + pos_k3*2 + pos_k4)/6;
53    vel = (vel_k1 + vel_k2*2 + vel_k3*2 + vel_k4)/6;
54    Et = gamma()*E0;
55 }
```

Figure A.7: RK4-2 implementation of e^- - EM

```

1 void RhodotronSimulator::_runMT(){
2     gun.fireAllWithFireTimesMT();
3
4     MTEngine.setupPool(time_interval, start_time, end_time, gun, E_field, B_field, gun.thread_bunchs);
5
6     STEPS_TAKEN = 0;
7     simulation_time = start_time;
8     while (simulation_time < end_time + time_interval ){
9         if (STEPS_TAKEN % log_interval() == 0){
10            E_field.update(simulation_time);
11            logEfield(simulation_time, simulation_time + time_interval > end_time);
12            notifyUI(MTEngine.getAverageTime());
13        }
14        simulation_time+=time_interval;
15        STEPS_TAKEN++;
16    }
17    bool end = false;
18    while (!end){
19        double time = MTEngine.getAverageTime();
20        notifyUI(time);
21        if ( time >= end_time ){
22            end = true;
23        }
24        this_thread::yield();
25    }
26    MTEngine.join();
27
28 }
```

Figure A.8: Multithreading main-thread logic.

```

1 void Gun::fireAllWithFireTimesMT(){
2     std::random_device rd;
3     std::mt19937 e2(rd());
4     std::normal_distribution<double> Edist(Ein, sEin);
5
6     for(_fired_bunch= 0; _fired_bunch < bunch_count; _fired_bunch++){
7         for(_fired_e_in_current_bunch= 0; _fired_e_in_current_bunch < e_per_bunch; _fired_e_in_current_bunch++){
8
9             double E = (sEin == 0 ) ? Ein : Edist(e2);
10
11             double fire_time = (ns_between_each_electron_fire * _fired_e_in_current_bunch) + _fired_bunch*gun_period;
12
13             auto burrowed_e = bunchs[_fired_bunch].AddElectronGiveAddress(E, gunpos, gundir, fire_time);
14
15             int thread_index = (_fired_e_in_current_bunch + _fired_bunch*e_per_bunch)%thread_bunchs.size();
16
17             thread_bunchs[thread_index]->push_back(burrowed_e);
18         }
19     }
20 }
```

Figure A.9: Multithreading electron assign logic.

```

1 void MultiThreadEngine::setupPool( double _time_interval, double _start_time, double _end_time, Gun& gun,
2   CoaxialRFField& RF, MagneticField& B, vector<shared_ptr<vector<shared_ptr<Electron2D>>>& e_list){
3   threads.reserve(thread_count);
4
5   for(int i = 0; i < thread_count && i == threads.size(); i++){
6
7     child_notifier_mutexes.push_back(make_shared<mutex>());
8     child_times.push_back(make_shared<double>());
9
10    auto _E = RF.Copy();
11    auto _B = B.LightWeightCopy();
12    double time_between_fires = thread_count*gun.getGunActiveTime()/gun.getElectronsPerBunch();
13    double first_fire_time = i*gun.getGunActiveTime()/gun.getElectronsPerBunch();
14    ThreadArguments thread_arguments(i, _time_interval, _start_time, _end_time, &gun, _E, _B, e_list[i], first_fire_time, time_between_fires);
15    thread_arguments.parent_notifier_mutex = child_notifier_mutexes[i];
16    thread_arguments.current_thread_time = child_times[i];
17    threads.push_back(thread(threadLoop, thread_arguments));
18  }
19 }
```

Figure A.10: Multithreading worker-threads setup logic.

```

1 void threadLoop(ThreadArguments thread_arguments){
2   uint64_t count = 0;
3
4   double sim_time = thread_arguments.start_time;
5
6   while(sim_time < thread_arguments.end_time + thread_arguments.time_interval){
7     thread_arguments.E->update(sim_time);
8     thread_arguments.i_args.time = sim_time;
9
10    if (count % (unsigned long)(0.1/thread_arguments.time_interval) == 0){
11      saveElectronInfoForSingleThread(thread_arguments.i_args);
12      // Notify the main thread
13      if(thread_arguments.parent_notifier_mutex->try_lock()){
14        *thread_arguments.current_thread_time = sim_time;
15        thread_arguments.parent_notifier_mutex->unlock();
16      }
17    }
18
19    interactForSingleThread(thread_arguments.i_args);
20    // save electron info here
21    sim_time+= thread_arguments.time_interval;
22    count++;
23  }
24  thread_arguments.parent_notifier_mutex->lock();
25  *thread_arguments.current_thread_time = sim_time;
26  thread_arguments.parent_notifier_mutex->unlock();
27 }
```

Figure A.11: Multithreading worker-thread logic.

```

1 void UIThreadWork(UIThreadArgs args){
2     int UI_WORK_PIECE = SIM_WORK_MASK;
3     if ( !args.isService ) {
4         UI_WORK_PIECE = 50;
5     }
6
7     double piece = (args.end_time - args.start_time)/UI_WORK_PIECE;
8
9     if ( !args.isService ) {
10        std::string sim_running_msg = "...Simulation is running...";
11        for(int i = 0; i < 26 - sim_running_msg.size()/2 ; i++){
12            std::cout << " ";
13        }
14        std::cout << sim_running_msg <<"\n";
15    }
16
17    args.ui_mutex->lock();
18    double simtime = *(args.simulation_time);
19    args.ui_mutex->unlock();
20
21    if ( !args.isService ) {
22        std::cout << "V";
23        for(int i = 0; i < 51; i++){
24            std::cout << "_";
25        }
26        std::cout << "V\n[" << std::flush;
27    }
28
29    int count = 0;
30    bool running = true;
31    while(running && (simtime < args.end_time || count < UI_WORK_PIECE )){
32        if( simtime > count * piece ){
33            if ( !args.isService ) {
34                std::cout << "#" << std::flush;
35            }
36            count++;
37            args.state_mutex->lock();
38            running = *args.state_ptr & SIM_RUNNING;
39            *args.state_ptr = (*args.state_ptr & ~SIM_WORK_MASK) | (count & SIM_WORK_MASK);
40            if (args.isService) write(args._fd, args.state_ptr, SIGNAL_SIZE);
41            args.state_mutex->unlock();
42        }
43        std::this_thread::sleep_for(std::chrono::milliseconds(25));
44        args.ui_mutex->lock();
45        simtime = *(args.simulation_time);
46        args.ui_mutex->unlock();
47    }
48
49    if ( !args.isService ) {
50        std::cout << "#]\n" << std::flush;
51        std::cout << "      ...Simulation is finished successfully...\n" << std::flush;
52    }
53
54    args.state_mutex->lock();
55    *args.state_ptr |= SIM_RENDERING;
56    if ( args.isService ) write(args._fd, args.state_ptr, SIGNAL_SIZE);
57    args.state_mutex->unlock();
58 }
```

Figure A.12: UI-Console handler thread logic.

```

1  class Electron{
2      std::vector<ElectronLog> log;
3      uint64_t index;
4      double Et;
5      vector3d pos;
6      vector3d vel;
7      double fire_time;
8  public:
9      Electron();
10     Electron(double Ein, vector3d position, vector3d direction, double _fire_time = 0);
11     ~Electron();
12
13     void setEin(double E_in);
14     void print_electron_info();
15
16     void move(double dt);
17     void move(const vector3d& acc, double dt);
18     void move(const vector3d& acc, const vector3d& jerk, double dt);
19     void accelerate(const vector3d& acc, double dt);
20     void accelerate(const vector3d& acc, const vector3d& jerk, double dt);
21
22     void interactLF(RFField& E, MagneticField& B, double time_interval);
23     void interactRK(RFField& E, MagneticField& B, double time_interval);
24     void interactRK(RFField& E, MagneticField& B, const double time, double time_interval);
25     void interactRK_ActorE(const RFField& E, const MagneticField& B, double time_interval);
26
27     vector3d interactE_RK(const RFField& E, double time_interval);
28     vector3d interactB_RK(const MagneticField& B, double time_interval);
29
30     void saveInfo(double t);
31     void setLogSize(size_t size);
32     void loge(DataStorage& path);
33     const vector<ElectronLog>& getLog();
34
35     double vel();
36     double beta();
37     double beta2();
38     double gamma();
39     double gamma_inv();
40 };

```

Figure A.13: Electron class definition

```

1  class Bunch{
2  private:
3      int e_count = 0;
4      int index_fastest = 0;
5      double max_energy = 0;
6      double entry_time = 0;
7      double E_in = 0.04;
8
9      vector<Bunch> subBunchs;
10     vector<shared_ptr<Electron>> e;
11
12 public:
13     Bunch(unsigned int num_of_electrons, double Ein, vector3d gunpos, vector3d gundir, double gun_ns);
14     Bunch();
15     ~Bunch();
16
17     void saveInfo(double time);
18
19     Electron& getFastest();
20
21     void AddElectron(double Ein, const vector3d& gunpos, const vector3d& gundir, double fire_time);
22
23     shared_ptr<Electron> AddElectronGiveAddress(double Ein, const vector3d& gunpos, const vector3d& gundir, double fire_time);
24
25     void setEntryTime(double entry_time);
26     void setEin(double E_in);
27     void setNSLen(double len);
28
29     uint32_t get_e_count();
30     double getEin();
31     double E_ave();
32     double E_rms();
33
34     void interact(RFField& E, MagneticField& B, const double time, double time_interval);
35
36     void divide(unsigned int num);
37     void concat();
38     Bunch& subBunch(unsigned int index);
39     vector<Bunch*> subBunchPtr();
40
41     void reset();
42
43     void print_bunch_info();
44     void print_summary();
45
46     void logPaths(vector<DataStorage>& pathStorage, std::string header);
47 };

```

Figure A.14: Bunch class definition

```

1  class Gun{
2      double Ein;
3      double sEin;
4      double gun_period;
5      double gun_active_time;
6      double ns_between_each_electron_fire;
7
8      vector3d gunpos;
9      vector3d gundir = vector3d(1,0,0);
10
11     uint64_t bunch_count = 0;
12     uint64_t e_per_bunch = 1;
13
14     uint32_t _fired_bunch = 0;
15     uint32_t _fired_e_in_current_bunch = 0;
16     bool _firing = false;
17
18     bool _mt_enabled = false;
19     uint32_t _child_thread_count = 0;
20
21     vector<Bunch> bunchs;
22
23     mutex _gun_mutex;
24     vector<shared_ptr<vector<shared_ptr<Electron>>> thread_bunchs;
25
26     void setNSLen(double len);
27 public:
28     Gun();
29     Gun(double Ein, double gun_active_time, double pulse_interval, vector3d gunpos);
30     ~Gun();
31     void interact(RFField& E, MagneticField& B, const double time, double time_interval);
32
33     void fireIfActive(double time);
34     void fireAllWithFireTimesMT();
35
36     void addBunch(unsigned int num_of_electrons, double Ein);
37
38     void setGunActiveTime(double gt);
39     double getGunActiveTime(){return gun_active_time;}
40     void setGunInterval(double guninterval);
41     void setGunPos(vector3d gun_pos);
42     void setGunDir(vector3d gun_dir);
43     void setEin(double Ein) ;
44     void setEinStd(double EinStd) ;
45     void setNumberOfElectrons(uint64_t e_num);
46     void getElectronsPerBunch(){return e_per_bunch;}
47     void setNumberOfBunchs(uint64_t b_num);
48     void enableMT(uint32_t thread_count);
49
50     void saveInfo(double time);
51     void logPaths(vector<vector<DataStorage> >& pathsStorage, std::string pathsPath, std::string header);
52 };

```

Figure A.15: Gun class definition

```

1  class Simulator{      // E in MV/m,   En in MeV,   B in T,    t in ns
2  protected:
3      Gun gun;
4      double Emax; double freq = 107.5; double phase_lag = 0;
5      DataStorage EfieldStorage; DataStorage BfieldStorage;
6      vector<vector<DataStorage>> pathsStorage;
7      std::string configPath;
8      std::string pathsPath;
9      std::string EfieldLogHeader;
10     std::string BfieldLogHeader;
11     std::string eLogHeader;
12
13     double simulation_time = 0;
14     double dummy_time = 0;
15     double start_time = 0;
16     double end_time = 45;
17     double time_interval = 0.0001;
18     uint64_t STEPS_TAKEN = 0;
19     double GUN_ACTIVE_TIME = 1;
20     double GUN_PERIOD = 9.3;
21     uint64_t NUM_OF_ELECTRONS = 1;
22     uint64_t NUM_OF_BUNCHS = 1;
23     double Ein = 0.04;
24     double EinStd = 0;
25     vector3d gunPosition;
26     vector3d gunDirection = vector3d(1,0,0);
27
28     shared_ptr<mutex> ui_mutex;
29     UIHandler ui_handler;
30     shared_ptr<mutex> state_mutex;
31     uint8_t state = 0x0;
32
33     MultiThreadEngine MTEngine;
34     unsigned int MAX_THREAD_COUNT = 1;
35     bool MULTI_THREAD = false;
36 public:
37     Simulator();
38     ~Simulator(){}
39
40     void enableMultiThreading(unsigned int thread_count);
41     void setdT(double dT);
42     void setEin(double E_in);
43     void setEinStd(double Ein_Std);
44     void setNumberofElectrons(uint64_t num_of_electrons);
45     void setNumberofBunchs(uint64_t num_of_bunchs);
46     void setStartTime(double starttime);
47     void setEndTime(double end_time);
48     void setGunActiveTime(double gun_ns);
49     void setGunPeriod(double pi);
50     void setGunPosition(vector3d pos);
51     void setGunDirection(vector3d dir);
52     void setRFPPath(std::string path);
53     void setBPath(std::string path);
54     void setPathsPath(std::string path);
55     void setConfigPath(std::string path);
56
57     virtual void run();
58     void openLogs();
59     void closeLogs();
60     virtual void getConfig(Configuration& config) = 0;
61     virtual void logEfield(double time, bool) = 0;
62     virtual void logBfield() = 0;
63     void logPaths();
64     shared_ptr<mutex> getUIMutex();
65 };

```

Figure A.16: Simulator class definition

```
1 class RhodotronSimulator : public Simulator{
2     private:
3         double R1;
4         double R2;
5         CoaxialRFField E_field;
6         MagneticField B_field;
7
8         void _runMT();
9         void _runST();
10    public:
11        RhodotronSimulator(double phase_lag);
12        RhodotronSimulator(Configuration& config);
13
14        void StartUIHandler();
15        void StopUIHandler();
16        void DeclareService(std::string pipe_name);
17
18        void setFreq(double frequency);
19        void setPhaseLag(double phase_lag);
20        void setEmax(double E_max);
21        void setR1(double r1);
22        void setR2(double r2);
23        void addMagnet(double B, double r, vector3d position);
24        void addMagnet(Magnet m);
25
26        void updateSimulation();
27        void getConfig(Configuration& config);
28        void logEfield(double time, bool end);
29        void logBfield();
30
31        void run();
32        void notifyUI(double time);
33
34        void setState(uint8_t state_);
35    };
```

Figure A.17: RhodotronSimulator class definition

```

1  class RFField{
2  protected:
3      double E;                      // MV/m
4      double E_max = 0;               // MV/m
5      double frequency = 107.5;       // MHz
6      double phase_lag = 0;           // degree
7  public:
8      RFField();
9      RFField(double phase_lag);
10     virtual ~RFField() {}
11
12     virtual vector3d getField(vector3d position);
13     virtual double getField(double R);
14     virtual int log(DataStorage& rf, double time, bool end = false);
15     vector3d actOn(Electron& e);
16     vector3d actOnAndGetRungeKuttaCoef(Electron& e, double dt);
17
18     double getE() {return E;}
19     void setEmax(double E_max) {this->E_max = E_max; update(0);}
20     void setFreq(double freq){ frequency = freq;}
21     void setPhaseLag(double phaselag){ phase_lag = phaselag; update(0);}
22     virtual void update(double time);
23 };


---


1  class CoaxialRFField : public RFField{
2  private:
3      double r1 = 0.188;                // m
4      double r2 = 0.753;                // m
5      double E_max_pos = r1;           // m
6      std::vector<CoaxialRFField*> _childs;
7      double E_radial(double R) const;
8  public:
9      CoaxialRFField();
10     CoaxialRFField(double phase_lag);
11     ~CoaxialRFField() override;
12     void setR1(double r1);
13     void setR2(double r2);
14     void setEmaxPos(double Emaxpos);
15
16     vector3d getField(vector3d position) const override;
17     double getField(double R) const override;
18     vector3d actOn(Electron& e);
19     vector3d actOnAndGetRungeKuttaCoef(Electron& e, double dt);
20     void update(double time) override;
21
22     int log(DataStorage& rf, double time, bool end = false) override;
23
24     std::shared_ptr<CoaxialRFField> Copy();
25     void split(uint32_t amount_of_child);
26     CoaxialRFField* child(uint32_t index);
27 };

```

Figure A.18: CoaxialRFField class definition

APPENDIX B: Example Simulation Runs

```

1 Optimal phase with the least RMS : -5
2
3 Simulation settings :
4 ph = -5 deg, gt = 1 ns, enum = 1000
5 dT = 0.001 ns, dT_out = 0.01 ns
6
7 For the 1th magnet:
8 Optimum out path = 0.81044 m
9 Magnet guide = 0.25852 m
10 Rho = 0.088477 m
11 Drift time of the first electron in the bunch : 7.688 ns
12 Drift time of the last electron in the bunch : 7.487 ns
13 Max energy = 0.47581 MeV
14 RMS = 0.0058165 MeV
15
16 For the 2th magnet:
17 Optimum out path = 1.0833 m
18 Magnet guide = 0.37766 m
19 Rho = 0.098898 m
20 Drift time of the first electron in the bunch : 5.597 ns
21 Drift time of the last electron in the bunch : 5.617 ns
22 Max energy = 0.89172 MeV
23 RMS = 0.0099018 MeV
24
25 For the 3th magnet:
26 Optimum out path = 1.1705 m
27 Magnet guide = 0.41573 m
28 Rho = 0.10223 m
29 Drift time of the first electron in the bunch : 5.314 ns
30 Drift time of the last electron in the bunch : 5.325 ns
31 Max energy = 1.298 MeV
32 RMS = 0.013879 MeV
33
34 Electron with the most energy : 623) 1.6999 MeV,           RMS of bunch : 0.017981 MeV
35
36 Total steps calculated : 12468052652
37 Simulation finished in : 632050015 us      ( 632.1 s )
38

```

Figure B.1: ϕ_{lag} , ρ & L optimization at
 $P = 12\text{KW}$, $R_1 = 0.188\text{m}$, $R_2 = 0.753\text{m}$, $t_g = 1\text{ns}$, $E_{in} = 40\text{KeV}$

```

1 Optimal phase with the least RMS : 0
2
3 Simulation settings :
4 ph = 0 deg, gt = 0.8 ns, enum = 1000
5 dT = 0.001 ns, dT_out = 0.01 ns
6
7 For the 1th magnet:
8 Optimum out path = 0.80787 m
9 Magnet guide = 0.2574 m
10 Rho = 0.088379 m
11 Drift time of the first electron in the bunch : 7.629 ns
12 Drift time of the last electron in the bunch : 7.48 ns
13 Max energy = 0.47579 MeV
14 RMS = 0.0038689 MeV
15
16 For the 2th magnet:
17 Optimum out path = 1.0833 m
18 Magnet guide = 0.37765 m
19 Rho = 0.098898 m
20 Drift time of the first electron in the bunch : 5.589 ns
21 Drift time of the last electron in the bunch : 5.605 ns
22 Max energy = 0.89169 MeV
23 RMS = 0.0068848 MeV
24
25 For the 3th magnet:
26 Optimum out path = 1.1705 m
27 Magnet guide = 0.41573 m
28 Rho = 0.10223 m
29 Drift time of the first electron in the bunch : 5.311 ns
30 Drift time of the last electron in the bunch : 5.318 ns
31 Max energy = 1.298 MeV
32 RMS = 0.0096887 MeV
33
34 Electron with the most energy : 629) 1.6999 MeV,           RMS of bunch : 0.012318 MeV
35
36 Total steps calculated : 12455378454
37 Simulation finished in : 631136046 us      ( 631.1 s )

```

Figure B.2: ϕ_{lag} & ρ & L optimization at $P = 12\text{KW}, R_1 = 0.188\text{m}, R_2 = 0.753\text{m}, t_g = 0.8\text{ns}, E_{in} = 40\text{KeV}$

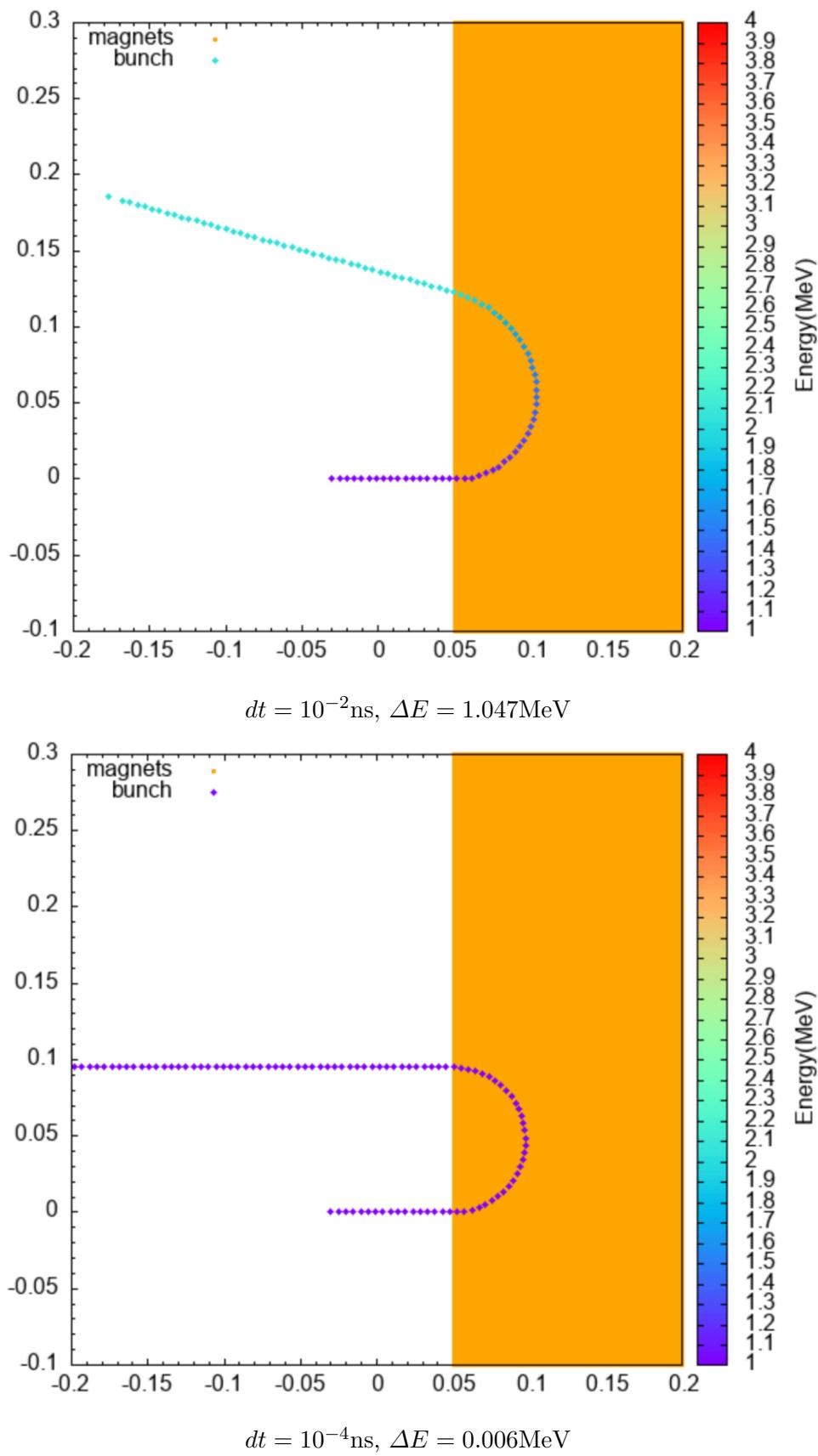


Figure B.3: Energy gain of 1MeV bunch in $\mathbf{B}=0.1\text{T}$ using RK4-2

```

1 # Rhodotron Simulation Configuration File
2 # =====
3 # M.Furkan Er 22/09/2022
4 # =====
5 #
6 # emax = Maximum electric field strength (MV/m)
7 # ein = Energy of electrons coming out of the gun (MeV)
8 # einstd = Standard deviation of energy of electrons coming out of the gun (MeV)
9 # targeten = Max energy on the output gif (MeV)
10 # freq = Frequency of the RF field (MHz)
11 # phaselag = phase lag of the first electrons (degree)
12 # starttime = time to start firing the gun (ns)
13 # endtime = ns to run the simulation (ns)
14 # dt = time interval to do the calculations (ns)
15 # guntime = how long a gun pulse is (ns)
16 # gunperiod = time between two gun pulses (ns)
17 # enum = number of electrons to simulate in a bunch
18 # bunchnum = number of times the gun fires
19 # r1 = radius of the inner cylinder (m)
20 # r2 = radius of the outer cylinder (m)
21 # epath = path to store the electric field data
22 # bpath = path to store the magnetic field data
23 # cpath = path to store the settings
24 # ppath = path to store electron data
25 # multh = enable or disable multithreading
26 # thcount = set the maximum thread to be used
27 # magrotation = degrees of rotation to enter each magnet
28 # addmagnet = takes 3 input. (B , R, < Radial distance of center >)
29 # output = output file name
30
31
32 # E FIELD CONFIGURATION
33 emax=1.170
34 freq=107.3
35 phaselag=10.0
36 r1=0.1840
37 r2=0.7380
38
39 # B FIELD CONFIGURATION
40 magrotation=5.0
41
42 # GUN CONFIGURATION
43 einmean=0.040
44 einstd=0.0000
45 targeten=2.0
46 guntime=1.0
47 gunperiod=9.3
48 enum=50
49 bunchnum=1
50
51 # SIM CONFIGURATION
52 starttime=0
53 endtime=10
54 dt=0.0000010000
55 epath=xy/rf.dat
56 bpath=xy/magnet.dat
57 cpath=xy/settings.dat
58 ppath=xy/paths/
59 multh=1
60 thcount=10

```

Figure B.4: An example *config.in* file.

```
1 -- Simulation Configuration --
2 Emax : 0.96      MV/m
3 Freq : 107.5     MHz
4 Phase Lag : -5   degree
5 EndTime : 100    ns
6 dT : 0.0001     ns
7 guntime : 0.8    ns
8 gunperiod : 9.3 ns
9 enum : 100
10 bunchnum : 2
11 R1 : 0.188241   m
12 R2 : 0.752967   m
13 Magnet count : 5
14 Ein : 0.04       MeV
15 TargetE : 2.5    MeV
16 -----
17
18          ...Simulation is running...
19 V-----V
20 [#####] ]
```

Figure B.5: Example of console output while simulation is running.

APPENDIX C: Data and Graphs

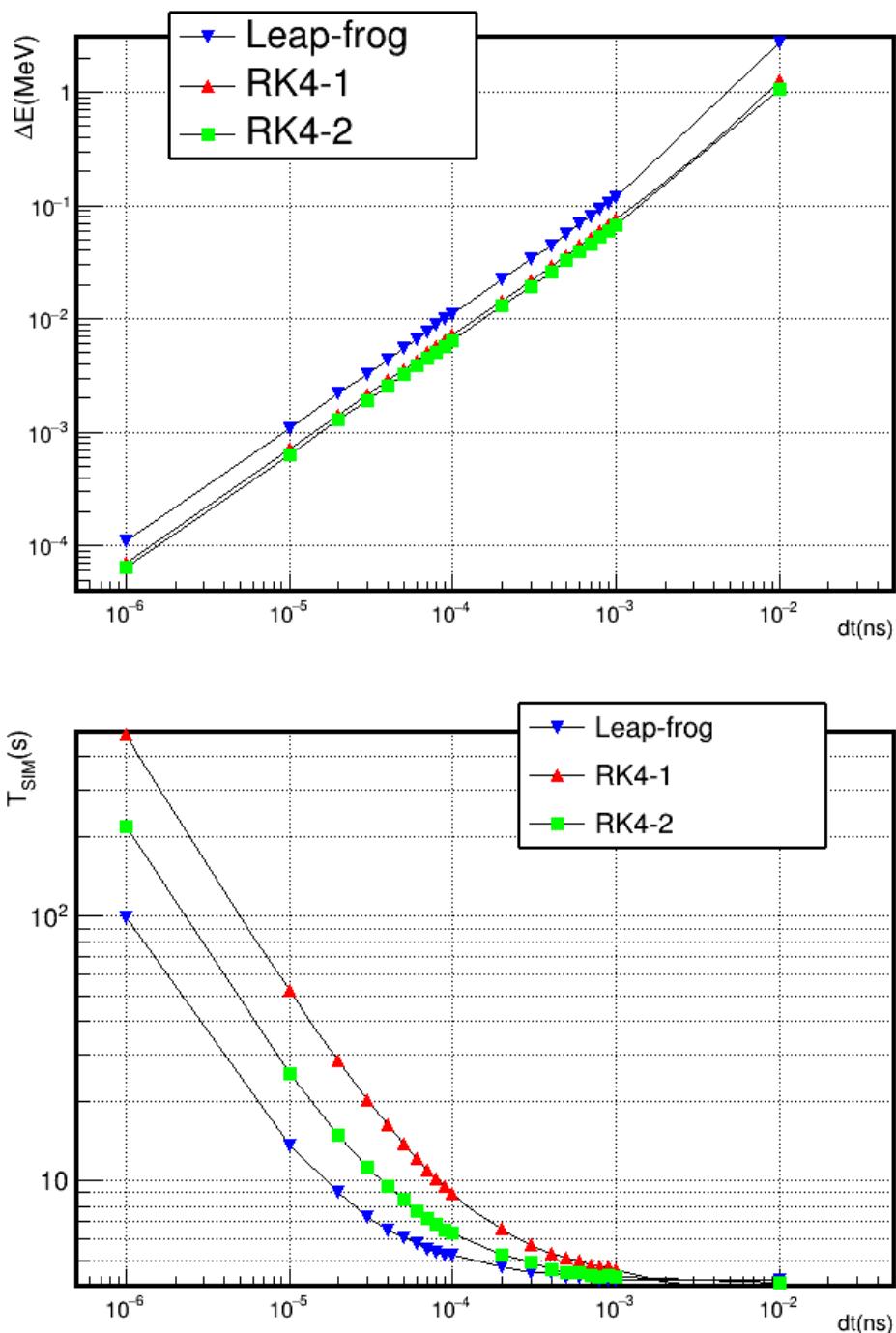


Figure C.1: Comparing Leap-frog, RK4-1, RK4-2 performance on $e^- - \vec{B}$ interaction

$$E_{in} = 1\text{ MeV}, \mathbf{B}=0.1\text{T}, t_{end} = 5\text{ ns}$$

Δt (ns)	ΔE_{avg} (MeV)	μT_{sim} (s)	σT_{sim} (s)
1e-02	2.783228	0.034660	0.011537
1e-03	0.117124	0.115775	0.001071
9e-04	0.104552	0.128983	0.001820
8e-04	0.092252	0.143251	0.002585
7e-04	0.080144	0.158808	0.003228
6e-04	0.068258	0.182359	0.002426
5e-04	0.056554	0.215104	0.002807
4e-04	0.044931	0.262952	0.005119
3e-04	0.033467	0.341552	0.002610
2e-04	0.022158	0.501784	0.005709
1e-04	0.011006	0.973145	0.005849
9e-05	0.009899	1.084032	0.010985
8e-05	0.008792	1.216145	0.012486
7e-05	0.007688	1.387908	0.019031
6e-05	0.006586	1.604475	0.011775
5e-05	0.005485	1.926505	0.014535
4e-05	0.004384	2.395898	0.009702
3e-05	0.003286	3.178265	0.014099
2e-05	0.002189	4.740706	0.022709
1e-05	0.001094	9.441138	0.027266
1e-06	0.000109	93.888320	0.290820

Table C.1: *Leap-frog* data on $E_{in} = 1\text{MeV}$, $\mathbf{B}=0.1\text{T}$, $t_{end} = 5\text{ns}$

dt(ns)	ΔE_{avg} (MeV)	μT_{sim} (s)	σT_{sim} (s)
1e-02	1.047130	0.048943	0.011642
1e-03	0.066912	0.239299	0.003483
9e-04	0.059899	0.268007	0.004530
8e-04	0.053028	0.296154	0.004146
7e-04	0.046183	0.333123	0.004259
6e-04	0.039452	0.384046	0.002458
5e-04	0.032734	0.456387	0.003888
4e-04	0.026072	0.563011	0.004803
3e-04	0.019474	0.742440	0.006169
2e-04	0.012926	1.103559	0.007649
1e-04	0.006437	2.178779	0.009733
9e-05	0.005791	2.411302	0.012266
8e-05	0.005145	2.704117	0.012683
7e-05	0.004500	3.078304	0.013281
6e-05	0.003856	3.589154	0.014472
5e-05	0.003212	4.297561	0.009546
4e-05	0.002568	5.369322	0.012127
3e-05	0.001925	7.136687	0.007845
2e-05	0.001283	10.679166	0.012126
1e-05	0.000641	21.325229	0.011661
1e-06	0.000064	212.824121	0.040967

Table C.2: $RK4-2$ data on
 $E_{in} = 1\text{MeV}$, $\mathbf{B}=0.1\text{T}$, $t_{end} = 5\text{ns}$

dt(ns)	ΔE_{avg} (MeV)	μT_{sim} (s)	σT_{sim} (s)
5.00E-02	0.114422	0.028398	0.009294
1.00E-02	0.019233	0.036245	0.002182
1.00E-03	0.002141	0.147979	0.00125
9.00E-04	0.001749	0.172161	0.00794
8.00E-04	0.001516	0.184629	0.004359
7.00E-04	0.00152	0.20445	0.001919
6.00E-04	0.001049	0.234947	0.004416
5.00E-04	0.000974	0.278938	0.010969
4.00E-04	0.0009	0.336847	0.003028
3.00E-04	0.000587	0.441771	0.005859
2.00E-04	0.000433	0.652066	0.006637
1.00E-04	0.0002	1.280059	0.016027
9.00E-05	0.000145	1.41908	0.013122
8.00E-05	0.000153	1.592492	0.011332
7.00E-05	0.000162	1.806561	0.009931
6.00E-05	0.000123	2.103571	0.011143
5.00E-05	0.000084	2.515006	0.01393
4.00E-05	0.000092	3.140471	0.014138
3.00E-05	0.000053	4.158715	0.009474
2.00E-05	0.000045	6.231734	0.015236
1.00E-05	0.000022	12.39871	0.039575
1.00E-06	0.000003	123.318	0.299559

Table C.3: *LF* data on

$E_{in} = 1\text{MeV}$, $\vec{\mathbf{E}} = (-2.65616, 0, 0) \text{ MV/m}$, $t_{end} = 6\text{ns}$

$dt(\text{ns})$	$\Delta E_{avg}(\text{MeV})$	$\mu T_{sim}(\text{s})$	$\sigma T_{sim}(\text{s})$
5.00E-02	0.055355	0.031054	0.003281
1.00E-02	0.009868	0.063656	0.000893
1.00E-03	0.001141	0.435864	0.005332
9.00E-04	0.001325	0.478216	0.00595
8.00E-04	0.001112	0.533789	0.003897
7.00E-04	0.000705	0.60586	0.005441
6.00E-04	0.000687	0.703851	0.009447
5.00E-04	0.000474	0.840173	0.00913
4.00E-04	0.00046	1.039166	0.005874
3.00E-04	0.000456	1.378822	0.007995
2.00E-04	0.000207	2.045464	0.008499
1.00E-04	0.000127	4.052472	0.014711
9.00E-05	0.000091	4.500401	0.015404
8.00E-05	0.000124	5.048922	0.014392
7.00E-05	0.000062	5.773208	0.031443
6.00E-05	0.000057	6.726501	0.020609
5.00E-05	0.00006	8.045931	0.014246
4.00E-05	0.000039	10.04809	0.012756
3.00E-05	0.000045	13.39669	0.028057
2.00E-05	0.000023	20.03889	0.006436
1.00E-05	0.000016	40.0818	0.120627
1.00E-06	0.000003	400.2421	0.554172

Table C.4: *RK4-2* data on

$E_{in} = 1\text{MeV}$, $\vec{\mathbf{E}} = (-2.65616, 0, 0) \text{ MV/m}$, $t_{end} = 6\text{ns}$

$dt(\text{ns})$	$\Delta E_{avg}(\text{MeV})$	$\mu T_{sim}(\text{s})$	$\sigma T_{sim}(\text{s})$
5.00E-02	0.986005	0.030314	0.009209
1.00E-02	0.154314	0.036833	0.004068
1.00E-03	0.01476	0.131783	0.002352
9.00E-04	0.013618	0.150984	0.005618
8.00E-04	0.011863	0.161336	0.003117
7.00E-04	0.010109	0.179693	0.002658
6.00E-04	0.009277	0.205356	0.002912
5.00E-04	0.007527	0.242352	0.003093
4.00E-04	0.006084	0.294269	0.001551
3.00E-04	0.00449	0.383982	0.001711
2.00E-04	0.002897	0.563762	0.004214
1.00E-04	0.001458	1.101252	0.008064
9.00E-05	0.001376	1.229321	0.0089
8.00E-05	0.001232	1.381441	0.01057
7.00E-05	0.001027	1.574994	0.011933
6.00E-05	0.000853	1.833183	0.006065
5.00E-05	0.00074	2.184948	0.013416
4.00E-05	0.000596	2.72503	0.016356
3.00E-05	0.000467	3.617576	0.017581
2.00E-05	0.000308	5.394629	0.027885
1.00E-05	0.00015	10.75182	0.03301
1.00E-06	0.000017	106.9198	0.373175

Table C.5: *LF* data on
 $E_{in} = 1\text{MeV}$, $\vec{\mathbf{E}} = (0, -5.31232, 0) \text{ MV/m}$, $t_{end} = 6\text{ns}$

dt(ns)	ΔE_{avg} (MeV)	μT_{sim} (s)	σT_{sim} (s)
5.00E-02	0.587777	0.034741	0.012735
1.00E-02	0.09943	0.059945	0.001318
1.00E-03	0.009987	0.38199	0.005941
9.00E-04	0.007946	0.417119	0.005683
8.00E-04	0.007999	0.466052	0.006398
7.00E-04	0.006732	0.525791	0.003782
6.00E-04	0.005541	0.612876	0.007114
5.00E-04	0.005022	0.732091	0.012129
4.00E-04	0.004031	0.906845	0.012156
3.00E-04	0.002779	1.199367	0.014536
2.00E-04	0.001897	1.782653	0.016993
1.00E-04	0.000895	3.52464	0.029778
9.00E-05	0.000858	3.916115	0.041066
8.00E-05	0.000731	4.409459	0.040828
7.00E-05	0.00066	5.012686	0.042727
6.00E-05	0.000561	5.851793	0.038488
5.00E-05	0.000471	7.002315	0.046152
4.00E-05	0.000363	8.747954	0.063936
3.00E-05	0.000264	11.64305	0.091404
2.00E-05	0.000184	17.44533	0.123198
1.00E-05	0.000093	34.93332	0.391012
1.00E-06	0.000011	348.494	3.237203

Table C.6: *RK4-2* data on

$E_{in} = 1\text{MeV}$, $\vec{\mathbf{E}} = (-2.65616, 0, 0) \text{ MV/m}$, $t_{end} = 6\text{ns}$

REFERENCES

1. Jacques Pottier, *A new type of rf electron accelerator: The rhodotron*, Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms, Volumes 40–41, Part 2, 1989, Pages 943-945,
2. J.M. Bassaler, J.M. Capdevila, O. Gal, F. Lainé, A. Nguyen, J.P. Nicolaï, K. Umiastowski, *Rhodotron: an accelerator for industrial irradiation*, Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms, Volume 68, Issues 1–4, 1992, Pages 92-95,A:
3. Y. Jongen. (2001). *Manufacturing of Electron Accelerators*. Ion Beam Applications s.a. (IBA) Chemin du Cyclotron 3, B-1348 Louvain-la-Neuve, Belgium.
4. Fukushima, Toshio. (2001). *Reduction of Round-off Errors in Symplectic Integrators*. The Astronomical Journal. 121. 1768-1775.
5. W. Kleeven, M. Abs, J. Brison, E. Forton, J. Hubert and J. Van de Walle, *Design and Simulation Tools for the High-Intensity Industrial Rhodotron Electron Accelerator*, 9th International Particle Accelerator Conference, 2018