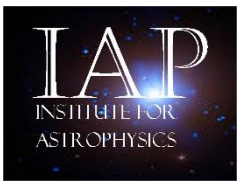
بسم الله الرحمن الرحيم



Inertial Electrostatic Confinement Fusion Reactor

Author: Maryam ABDEL-KARIM

Last Update: 30.08.2021 10:16

Contents

1 IEC Device

1.1 Introduction

Historically, most fusion research has focused on energy production in two basic configurations: magnetic confinement (magnetic fusion energy or MFE), and inertial confinement (inertial fusion energy or IFE). Each of these may lead eventually to a viable fusion reactor, yet they tend to be large, complicated, and expensive. This generates many physics and engineering difficulties, so the time frame for their development remains uncertain. The present note describes a fundamentally different fusion concept based on electrostatic focusing of ions into a dense core. Generically, such systems are called *inertial-electrostatic confinement* (IEC) fusion systems. In the 1950's, research was done on one form of IEC, purely electrostatic confinement, in which a voltage difference on concentric grids focuses charged particles. Ions accelerate down the electrostatic potential in spherical geometry, and convergence at the origin gives high density[1].

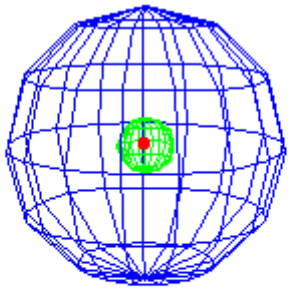


Figure 1: Grids plus energetic plasma core of the simplest IEC configuration.

1.2 Inertial Electrostatic Confinement Fusion Principle

Inertial electrostatic confinement fusion does not rely on any magnetic field, and instead uses an electric field to perform the plasma confinement. The device is formed by arranging two spherical metal meshes concentrically and putting an extremely large voltage across them, typically 10 to 100 kV in vacuum (Figure 1.2). The meshes are supported by a high-voltage stalk (Figure 1.3). Fusion fuel, in this case deuterium, is then released into the chamber where the large voltage ionizes a portion of the gas. These positively charged ions are accelerated by the electric field toward the center of the device, where most will pass straight through the cathode at the center and out the other side due to the large gaps in the metal mesh. Once on the other side, the electric field slows the ions and eventually turns them back towards the center, setting up a recirculation. The ion density in the center increases, and eventually fusion occurs. This does not require nearly as large a machine as the tokamak: typical IEC devices comfortably sit on tabletops.

[1] https://iec.neep.wisc.edu/overview.php

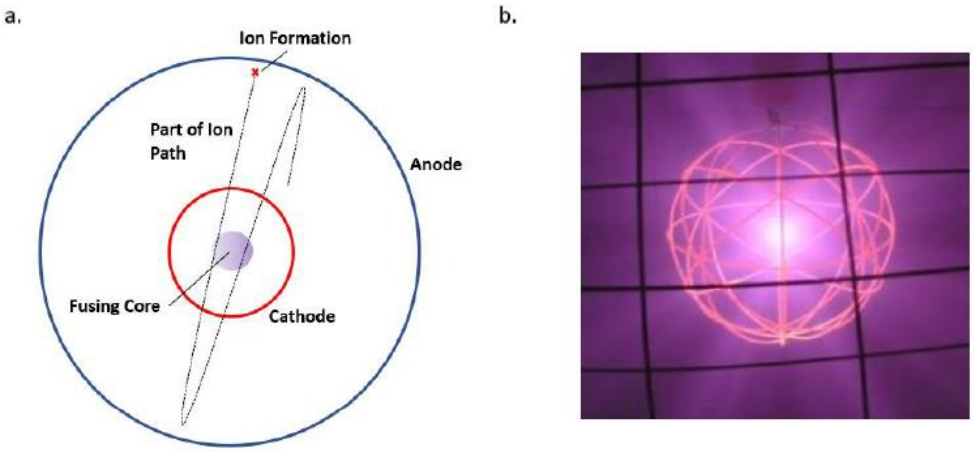


Figure 1.2 a) A general schematic of an idealized IEC fusion device. The red spherical cathode has a large negative voltage, while the larger blue sphere is a grounded anode. Deuterium is ionized near the anode, and these positive ions are accelerated by the radial electric field toward the center. The ions pass through the center and out the other side, where the electric field slows and turns them back. A high-density core sets up due to the presence of recirculating ions, and fusion occurs. b) A photo of an IEC device in operation at the University of Wisconsin-Madison [3]. The cathode grid glows orange from heat, while the anode (foreground) does not. Reproduced with permission.

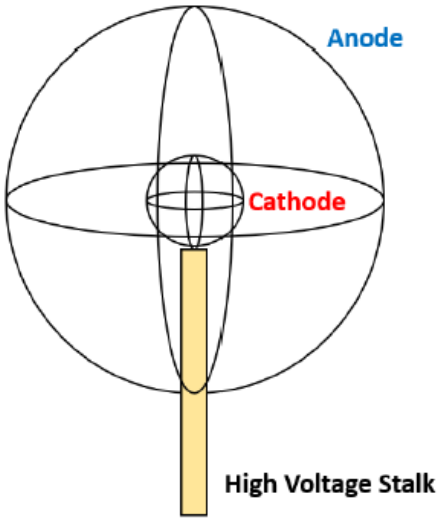


Figure 1.3 Realistic schematic of IEC device. The high voltage stalk supports the anode and cathode. The anode and cathode are far from being an ideal spherical surface, and are instead constructed from several wires. The stalk and wires are loss channels for ions recirculating within the device and inhibit ideal IEC operation.

[2]

[2] Modeling the Fusion Reaction in an Inertial Electrostatic Confinement Reactor with the Particle-in-Cell Method - Jacob van de Lindt - 2020

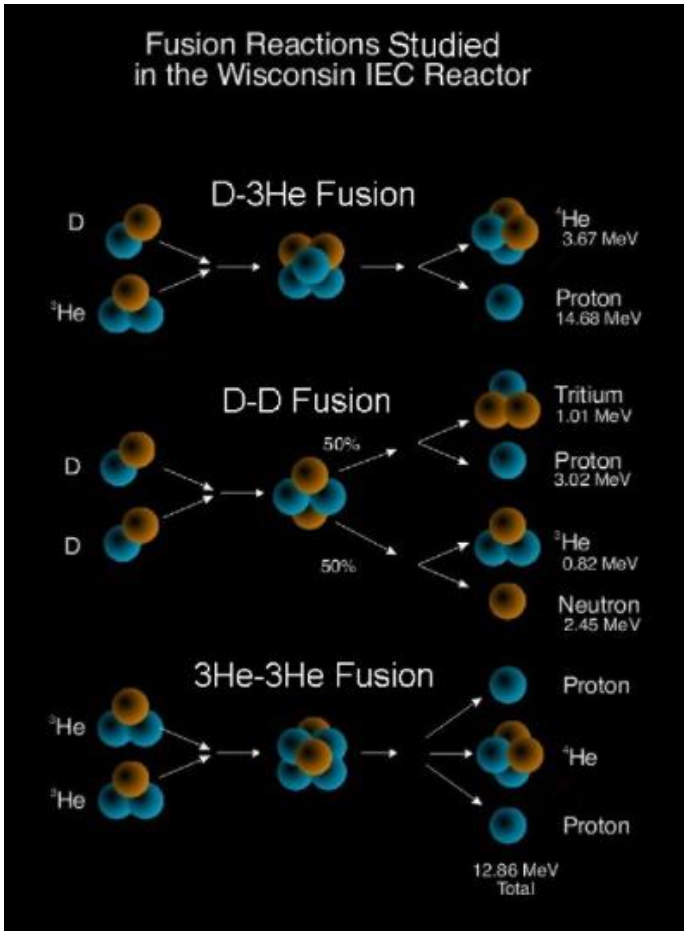
1.3 Nuclear fusion

Nuclear fusion is the process by which small elements and isotopes can combine into heavier isotopes. The fusion reaction products will have a net mass that is lower than the reactants that entered the fusion reaction, called the mass defect. This mass defect Δm is exactly the amount of kinetic energy gained by the products of the fusion reaction. This kinetic energy gain can be solved for using Einstein's famous equation $Q = \Delta m c^2$, where Q is the kinetic energy gained by the products and c is the speed of light in vacuum. Fusion reactions occur between two positively charged nuclei only when the two reactants come close enough for the strong force to fuse the two reactants. The two reactant nuclei must overcome the powerful Coulomb repulsion between the two positive nuclei for the fusion reaction to occur. The energy to overcome this Coulomb barrier can be supplied by the temperature of the products. If the thermal motion of the reactants where Q is the kinetic energy gained by the products and c is the speed of light in vacuum. Fusion reactions occur between two positively charged nuclei only when the two reactants come close enough for the strong force to fuse the two reactants. The two reactant nuclei must overcome the powerful Coulomb repulsion between the two positive nuclei for the fusion reaction to occur. The energy to overcome this Coulomb barrier can be supplied by the temperature of the products. If the thermal motion of the reactants is high enough, then the reactants will have enough kinetic energy to overcome the Coulomb barrier and initiate the fusion reaction. It is the high temperature requirements of fusion reactants that necessitate the merging of fusion science and plasma physics: most common fusion reactants, deuterium and tritium, exist in a plasma state at the temperatures required for fusion. The fusion reaction occurring in the device proposed here is D-D (deuterium-deuterium) fusion. An extremely small amount of D-T (deuterium-tritium) fusion could occur as tritium is a product of the D-D reaction. However, D-T fusion is likely not occurring at appreciable rates due to the low amount of D-D fusion occurring in the device and the short operating times. The branching ratio, defined as the probability of a certain set of products emerging from the same fusion reaction, is 50% for D-D products The two D-D reactions and the D-T reaction are

$$
\begin{aligned}
{}^{2}_{1}D + {}^{2}_{1}D &\rightarrow {}^{3}_{1}T + {}^{1}_{1}H, && Q = 4.03 \ \text{MeV} \\
{}^{2}_{1}D + {}^{2}_{1}D &\rightarrow {}^{3}_{2}He + n, && Q = 3.27 \ \text{MeV} \\
{}^{3}_{1}T + {}^{2}_{1}D &\rightarrow {}^{4}_{2}He + n, && Q = 17.58 \ \text{MeV}
\end{aligned}
$$

Where n is a neutron. The kinetic energy gain Q is shared between the products of the fusion reaction, with the lighter product particle receiving a larger proportion of the energy. The neutron produced by the D-D reaction has an energy of 2.45 MeV, and the proton produced by the D-D reaction has an energy of 3.02 MeV. These energy values are important for particle detection, which is discussed in Section 2.4. The neutron produced by the D-T reaction has an energy of

14.1 MeV. It is the 2.45 MeV neutron that can be measured to validate that fusion is occurring in the proposed device. The number of 2.45 MeV neutrons detected per second can be used to predict the total fusion rate occurring in the device.[3]



1.4 Applications

Depending on the fuel, fusion produces several unique and potentially useful products: high-energy neutrons (2-14 MeV), thermal neutrons, high-energy protons (3-15 MeV), and electromagnetic radiation (microwave to x-ray to gamma- rays). These fusion products have many potential commercial applications, including but not limited to:

- production of radioisotopes for medical applications and research, including isotopes for positron emission tomography (PET)
- detection of specific elements or isotopes in complex environments, including explosives in the form of improvised explosive devices (IEDs), landmines, fissile material, and concealed nuclear weapons

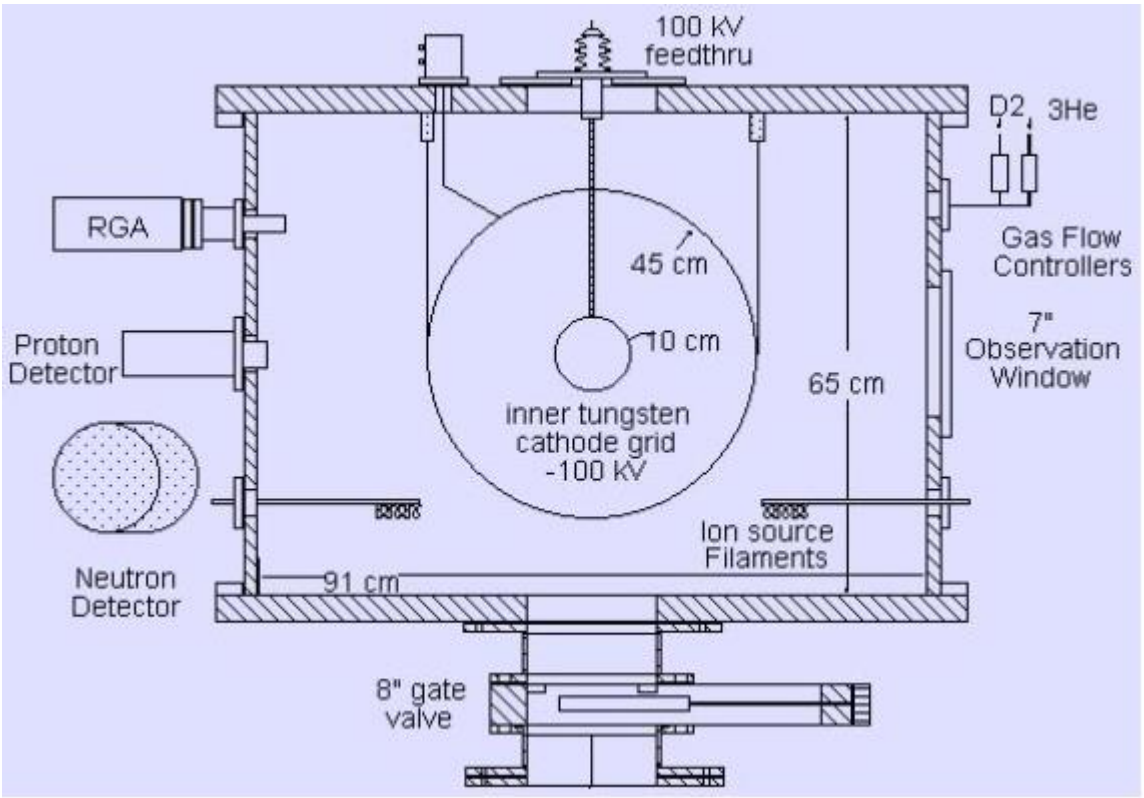
[3] Modeling the Fusion Reaction in an Inertial Electrostatic Confinement Reactor with the Particle-in-Cell Method - Jacob van de Lindt - 2020

- radiotherapy
- alteration of the electrical, optical, or mechanical properties of solids
- destruction of long-lived radioactive waste
- destruction of fissile material from nuclear warheads
- production of tritium for military and civilian applications
- food and equipment sterilization
- pulsed x-ray sources

1.5 University of Wisconsin-Madison Device

1.5.1 Design

The IEC reactor is a vacuum chamber filled with a fuel gas such as deuterium at low pressure. There are inner and outer spherical or cylindrical grids centered inside the chamber. The outer grid is held at nearly zero potential, and the inner, 90-99% transparent grid is held at a high negative potential, typically -100 kV. The potential difference between the grids accelerates ions inward to velocities of fusion relevance.



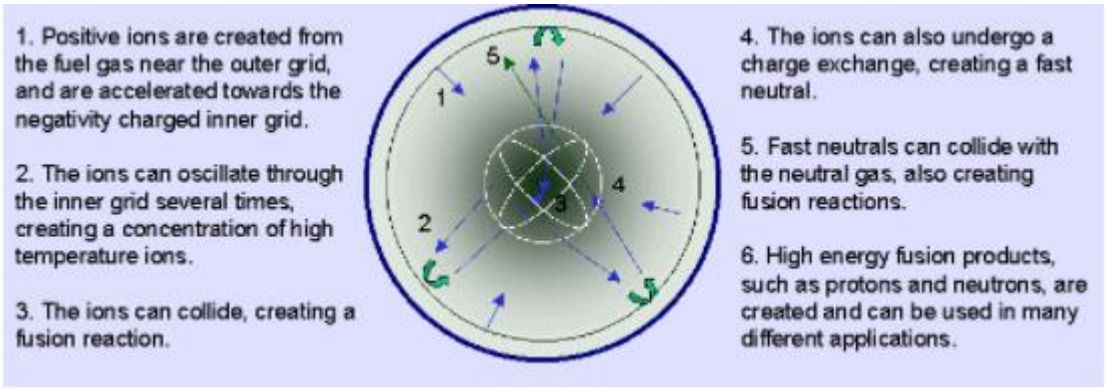
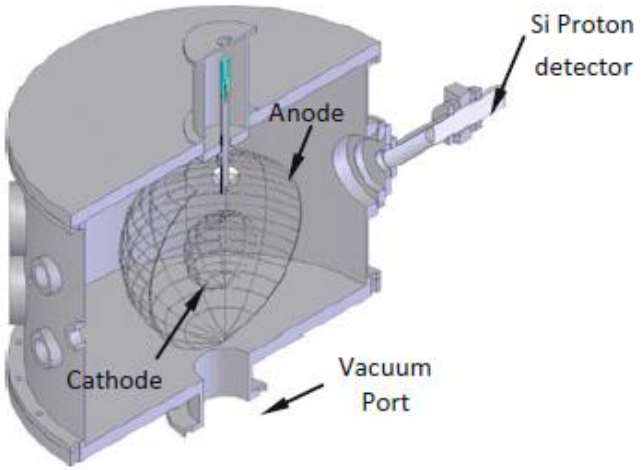
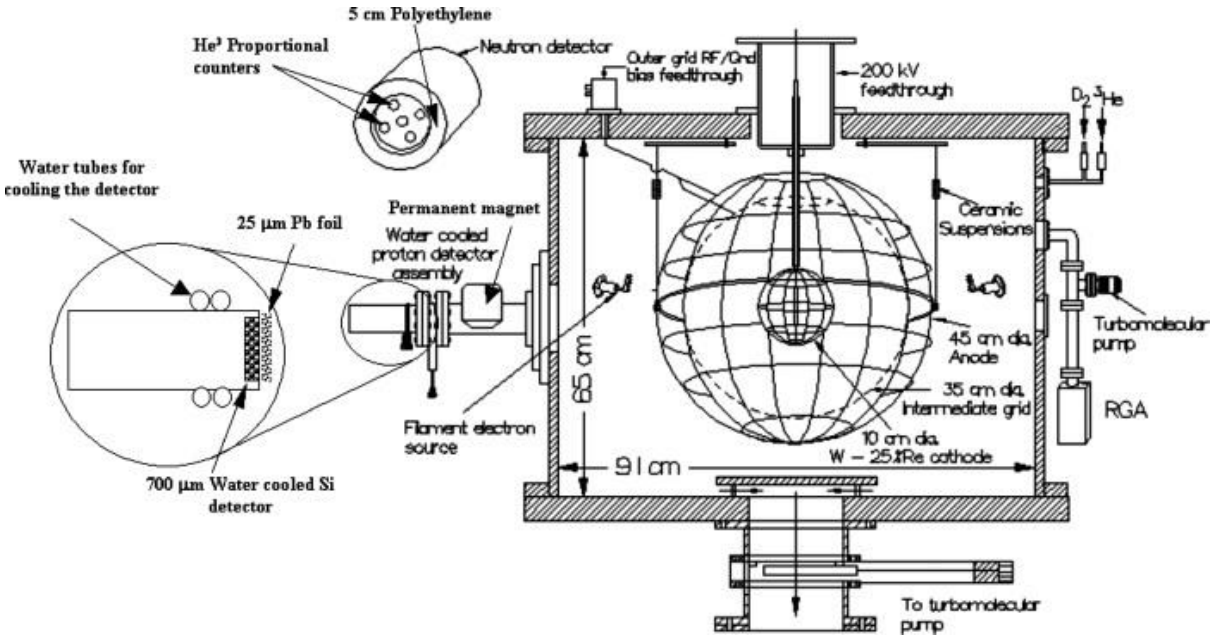


Fig. 1.6 Schematic of the IEC device (named Homer) at UW Madison. The cathode, anode, and the various detector ports are shown in the cross-sectional diagram [38]





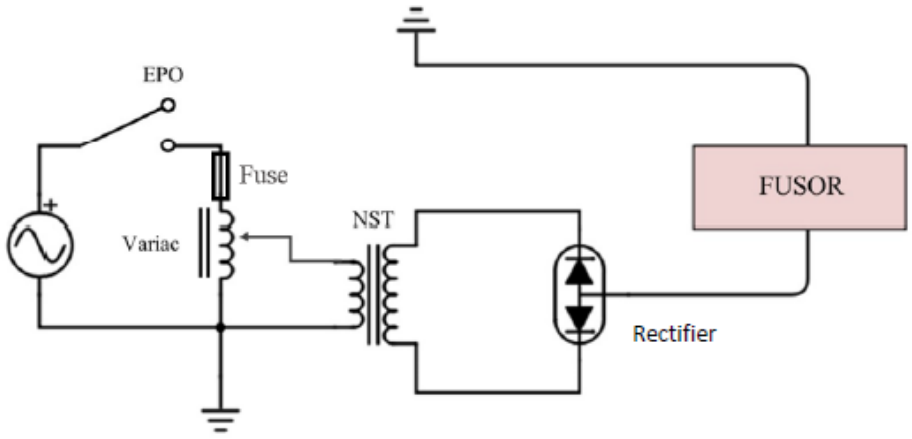


Figure 2.11 The general schematic of the proposed electrical system, which includes an emergency power off (EPO) switch and a fuse. The Variac and NST step up the wall voltage. The high voltage rectifier converts AC to DC [10].

1.5.2 High-Voltage Stalk Design for IEC

At the University of Wisconsin, Madison, the construction of the inner-grid stalk support was done to ensure the structural integrity and minimize the electrostatic stresses that are imparted to the stalk. Figure 4.7 shows the schematic of the side view of the high-voltage stalk assembly. Concentric tubes of aluminum and quartz were selected for the bulk stalk materials based on their rigidity and high voltage insulation properties. The ceramics are cut to size from the stalk using a diamond-bladed saw, and they are sealed together using ceramic cement and Torr-seal vacuum epoxy. The vacuum end of the stalk is also rounded using diamond files in order to minimize the electric stresses in that region.

The inner grids are tack-welded to rounded screws that then thread into the end of the stainless steel center tube.

The inner spherical grid hangs from a special ceramic stalk structure that is held in place by a ½" Cajon fitting atop a 100-kV isolation feedthrough. This independent attachment allows the inner cathode to be raised, lowered, and rotated with respect to the fixed outer grids and chamber walls. This arrangement provides external alignment of the cathode and further experimental flexibility (even under vacuum).

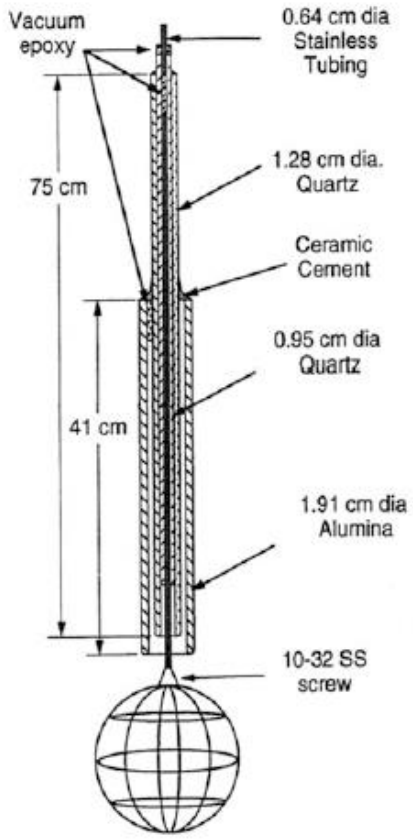


Fig. 4.7 Cross section of the complete stalk, along with the inner cathode grid, previously used at University of Wisconsin (UW), Madison [9]

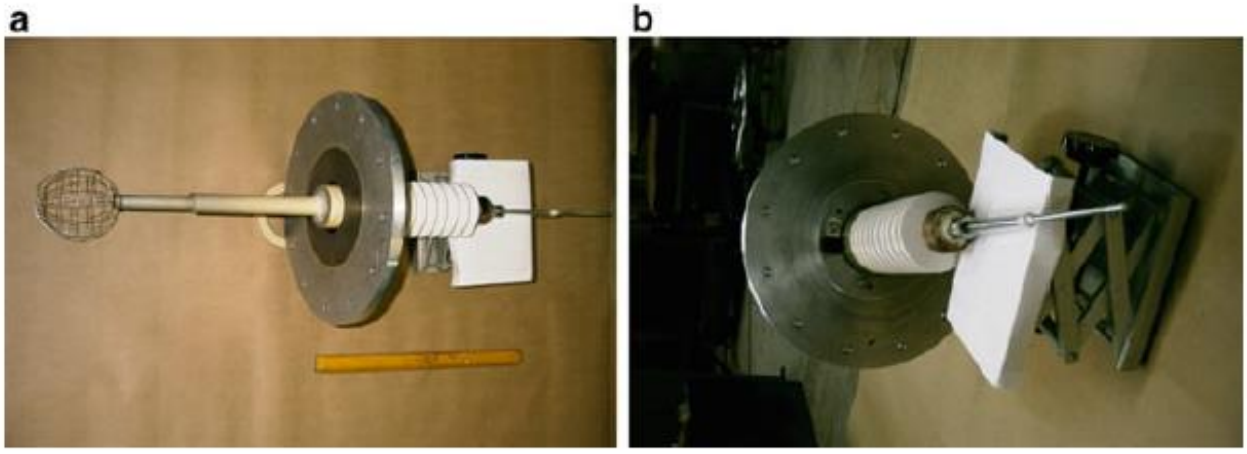


Fig. 4.8 Top view (a) and side view (b) of the stalk assembly [3]

1.5.3 Construction of grids

Prior to standardized grid manufacturing, most groups resorted to manual grid construction. Wires were typically made of stainless steel or tantalum. These wires were turned into loops using a spot welder prior to arranging them in the form of a grid and securing these loops together, once again using a spot welder. A"mold" cut from wood or other material was often used to hold wires in place while spot welding them. It was realized early on that despite its low

cost and ease of use, stainless steel is not ideal, as it sputters profusely and melts at a relatively low temperature. Thus, varieties of other materials have been used in some IECs. Though tungsten was a preferable alternative, it was not widely used due to the lack of accessible welding techniques. Attempts were made to use intermediate materials like nickel foil, placed between the two tungsten wires that needed spot welding. The nickel melted and secured the tungsten wires. However, the melting point of the nickel fundamentally limited these grids so the advantages of tungsten wires could not be fully exploited. As previously discussed, the University of Wisconsin group selected W-25 %Re for grid construction and testing. Experience with this alloy has been very encouraging. For example, the stainless steel wires previously used by Murali lasted for under a week depending on the power load. In contrast, with the W-25 %Re alloy, the grid lasted for over 2 years. It was eventually replaced when a cleaning attempt (using a sand blaster) unfortunately turned the smooth surface texture into a rough surface, rendering the grid useless (Ashley RP, 2002, University of Wisconsin, Madison, private communication). Otherwise the grid could have continued in use for some time.

A very successful rapid prototyping technique was developed at the University of Wisconsin, Madison. The technique is illustrated in Table 5.2 along with the description of the grid construction procedure. This procedure produces a reproducible, uniform grid, but unlike the NASA work, it still requires spot welding.

Another commonly used design for the cathode grid uses a series of plates/disks (such as shown in Fig. 5.14). The idea of using such disks for the construction of cathode grids was first suggested by P. T. Farnsworth in his original patents. The major advantage of using such grids is that while the grid transparency is maintained constant, the surface area for radiating the heat away from the grid increases. These disks are cut using lasers, though powerful water jets can also be used. One of the principal considerations while building a grid is to improve the grid transparency. As we explained earlier, operation in the "Star" mode (microchannel) depends more on the "effective" transparency versus the geometric transparency.

However, operation in other modes is very dependent on the geometric value, so improvements in the geometric transparency are very desirable. One such technique that was developed by Murali and colleagues at Los Alamos National lab for this purpose used a combination of several wires with different diameters to accomplish the desired grid transparency. Thicker wires were used to construct a strong skeleton structure, while thinner gauge wires were used to fill in the gaps to obtain the desired spacing. The grid constructed in this way is shown in Fig. 5.15. The geometric transparency achieved with this technique was 95%. With the large openings in these grids, microchannel form, providing an even higher effective transparency. Still, many ions eventually scatter out of the microchannel and hit grid wires. That factor in turn depends on the

geometric transparency. The high geometric transparency is valuable in the sense of reducing those collisions, increasing ion recirculation times.[4]

Table 5.2 Standardized grid construction procedures developed at the University of Wisconsin, Madison [10]

	Figure	Procedure
1.		Plastic model created using rapid prototyping
2.		Rubber mold created using the plastic template
3.		Wax is poured into the rubber mold and is left there until it solidifies
4.		A wax template is created using the rubber mold
5.		The wax template is used to keep the wires positioned until it is spot-welded
6.		After spot welding the wires, the wax template is melted away using a hot air gun. The wax can then be reused
7.		The grid is cleaned in an ultrasonic cleaner to remove any residual wax from the grid surface. The grid is now ready for use

[4] Inertial electrostatic confinement (IEC) fusion fundamentals and applications – George H. Miley S. Krupakar Murali

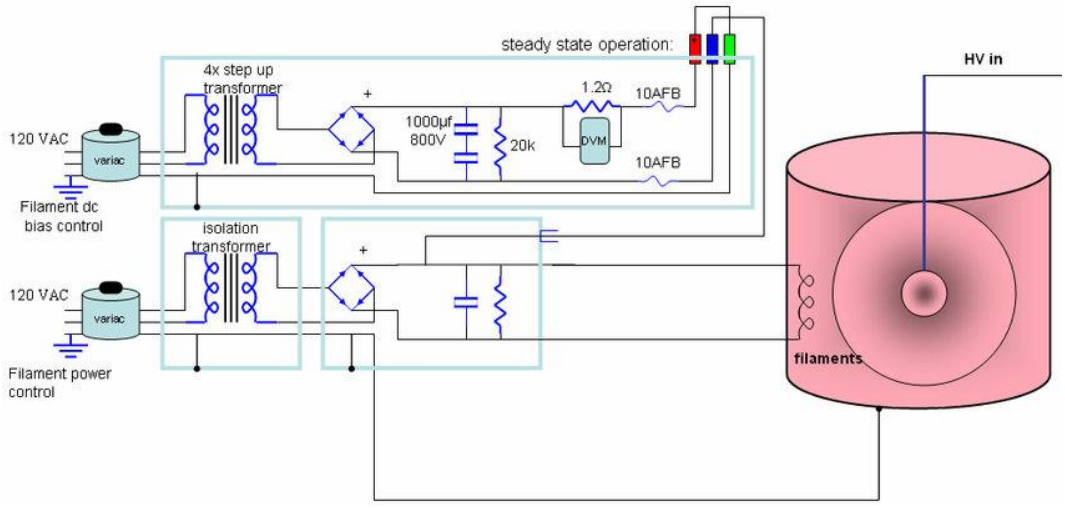
1.5.4 Operation[5]

It consists of a 91 cm diameter, 65 cm tall cylindrical aluminum vacuum chamber. Experiments run steady-state within the limitations of chamber heating. Partly to circumvent this constraint, a spherical, stainless-steel, water-cooled chamber has recently been brought into operation.

A 450 l/s turbo pump pumps the chamber down to a base pressure of 2 x 10-7 torr. The inner tungsten-rhenium (W-Re) cathode is a 10 cm diameter coarse-grid sphere of 0.8 mm tungsten wire supported on a 200 kV vacuum feed-through. It has operated at input power levels exceeding 8.5 kW at voltages ranging from -25 kV to -185 kV and at currents from 30-75 mA. A power supply provides up to 200 kV at 75 mA. The outer anode is typically a 50 cm diameter grid of stainless-steel wire.

The wire can be biased with variable amplitude AC and DC voltages. Electronically controlled gas flow regulators adjust the fuel flow ratio and amount into the system. A remote controlled throttle on the turbo pump is used to control the operating pressure of the gas mix.

{The deuterium plasma is created by adopting hot cathode discharge (i.e. filamentary discharge) method in which two thoriated tungsten filaments are placed at two diagonally opposite positions and at 10 cm away from the wall of the chamber. The filaments are heated to produce thermionic electrons and a discharge voltage and current of 80 V and 200−500 mA is maintained, respectively, at a working pressure of ~ 10−3 Torr. Then, negative voltage is applied to the cathode through the feedthrough}



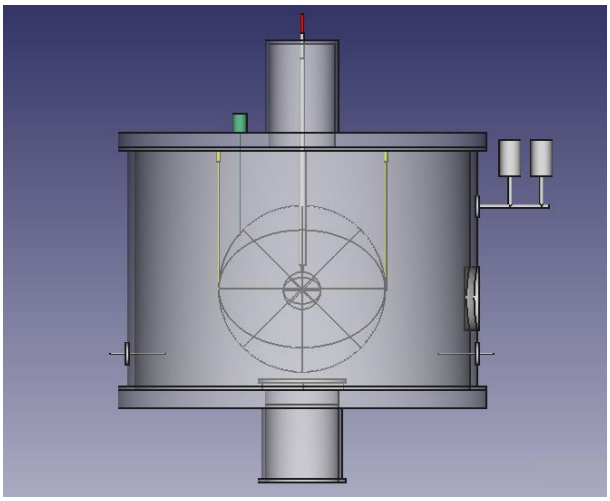
IEC source region filament control circuit.

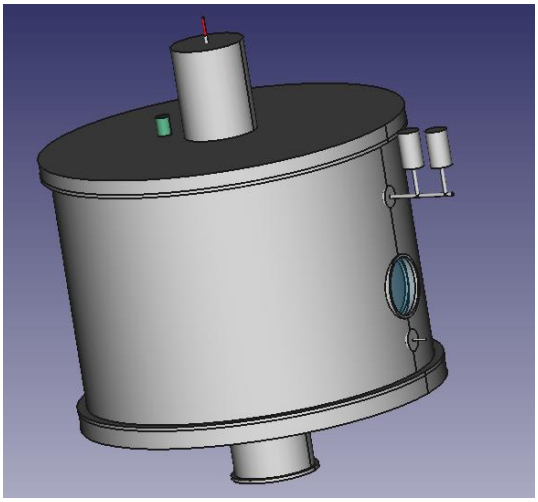
[5] Fusion Science and Technology - Overview of University of Wisconsin Inertial Electrostatic Confinement Fusion Research

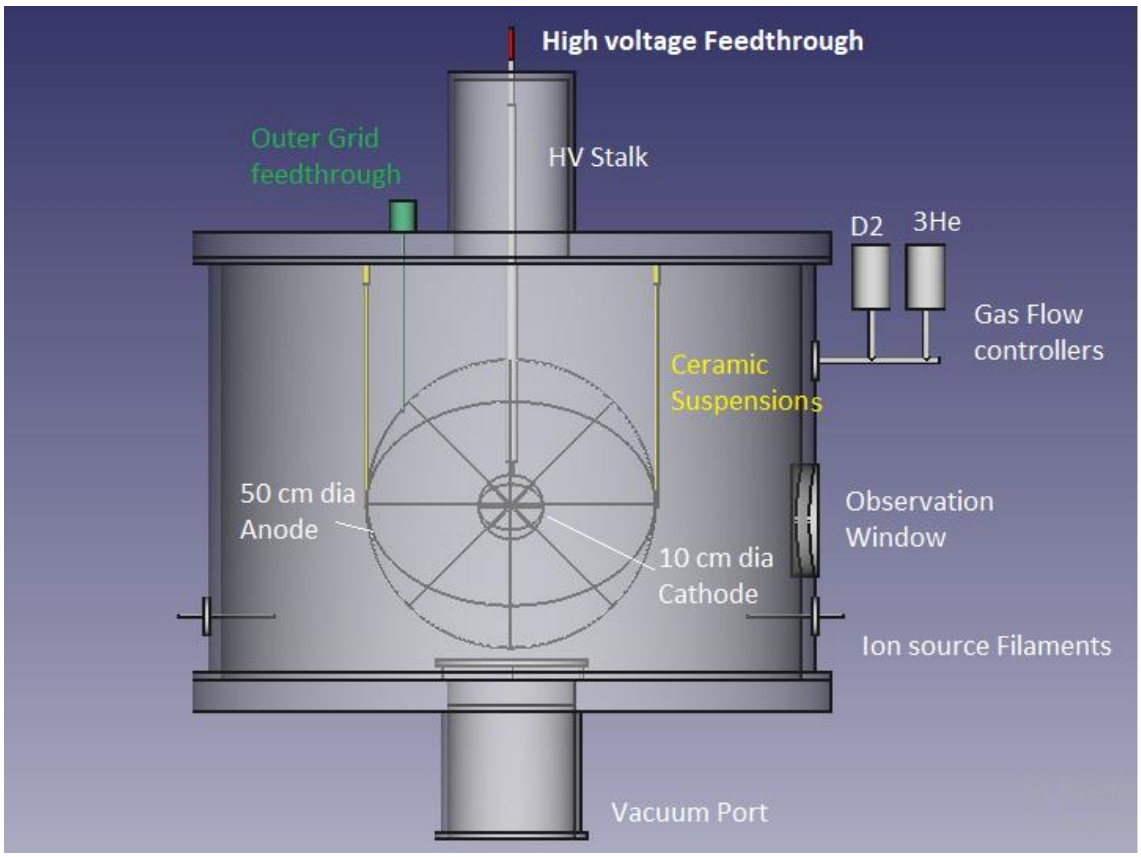
The fusion reactions in the IEC chambers produce neutrons, protons, electrons, helium-4, tritium, γ-rays, and x-rays, depending on the fuel. The present diagnostic set detects 2.45 MeV D-D neutrons, 3.02 MeV D-D protons, and 14.7 MeV D-3He protons. The proton detector is an EG&G Ortec Ultra solid-state detector, has an active area of 1200 mm2, and has a depletion region thickness of 700 μm. This thickness allows both the 3 MeV protons and the 14.7 MeV protons to be detected simultaneously. A helium proportional counter neutron detector, placed 60 cm from the center, is used to detect the 2.45 MeV DD neutrons. A residual gas analyzer (RGA) assesses neutral gas components.

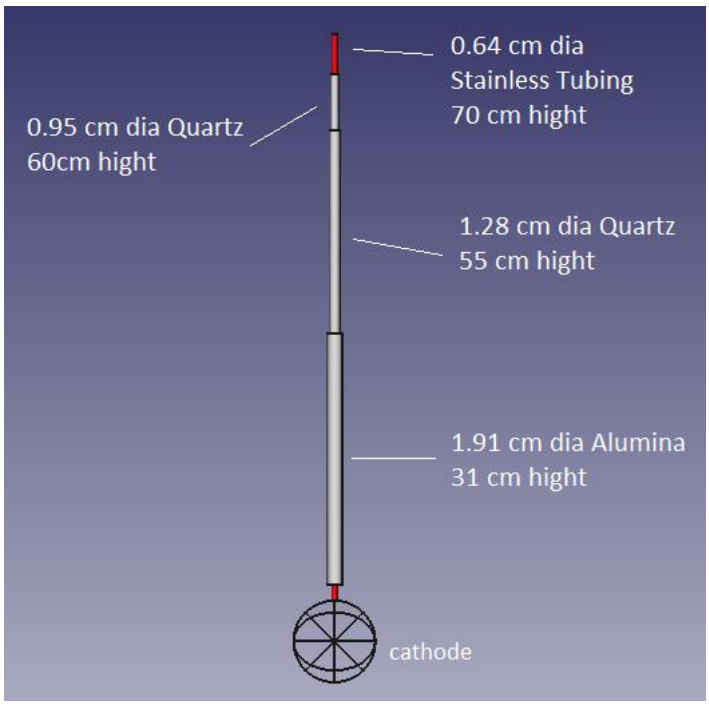
2 Our Device

2.1 Design









2.2 Specifications

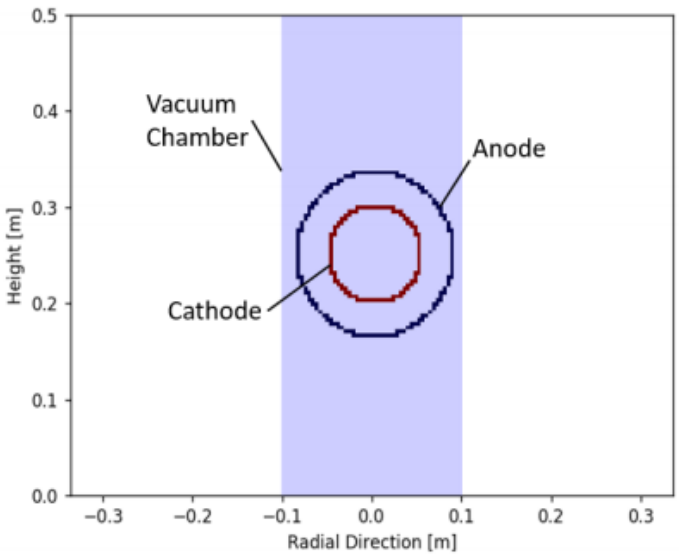
Part	Description	Material	Size
Anode wire	grounded,	stainless steel	45 cm diameter grid of stainless-steel wire
Cathod wire	-100KV DC, supported on a 200 kV vacuum feed-through. input power levels exceeding 8.5 kW at voltages ranging from -25 kV to -185 kV and at currents from 30-75 mA.	tungsten	10 cm diameter coarse-grid sphere of 0.8 mm tungsten wire
stalk (will hold the cathod)	stainless steel tubing Quartz tubing non-porous high alumina ceramic (Alumina ceramic (Aluminum Oxide or Al2O3) is an excellent electrical insulator)	stainless steel, quartz, alumnia	0.64 cm dia stanless 70 cm 1.28 cm dia quartz 55 cm 0.95 cm dia quartz 60 cm 1.91 cm dia Alumina 31 cm
Vacuum chamber	cylindric, base pressure of 2 x 10-7 torr, Experiments run steady-state within the limitations of chamber heating	Aluminum	91 cm diameter, 65 cm tall
Deuterium/ He gas tube	Electronically controlled gas flow regulators adjust the fuel flow ratio and amount into the system.		
Ion Source Filaments	2 filaments a discharge voltage and current of 80 V and 200–500 mA is maintained	thoriated tungsten filaments	
vacuum pump	450 l/s turbo pump		
high voltage supply	provides up to 200 kV at 75 mA		
neutron detector	helium proportional counter neutron detector, placed 60 cm from the center		

3 Simulation[6]

3.1 Computational Domain

- The computational domain was meshed using a regular Euclidian mesh with equal cell dimensions in x and y, despite the spherical geometry of the device.
- The computational model of the IEC device replaces the physical wire grids with two circular equipotential of the same radii.
- The potential of these circles is equal to the grid voltage.
- The computational domain is two-dimensional.
- The domain's outer-most boundary is the cylindrical vacuum chamber
- Assuming the anode and cathodes are closed circles, assigning a geometric transparency.
- The stalk was ignored in this study.

The computational domain used in this model is visualized in Figure 2.1.



3.1.1 Defining the Geometric Mesh

The regular Euclidian mesh making up the computational domain was extended over a region with N_x cells in x direction, and N_y cells in y direction. This domain can be translated into a set of N_xN_y equations, one for each cell corner by using the discretized Poisson's Equation:

$$\nabla^2\phi = -\frac{\rho}{\varepsilon_o}.$$

[6] Modeling the Fusion Reaction in an Inertial Electrostatic Confinement Reactor with the Particle-in-Cell Method - Jacob van de Lindt - 2020

Here ρ is the net charge density, given by $q(n_{ion} - n_e)$, the number density of ions minus the number density of electrons times q, the charge of a proton. ϕ is the electric potential, and ε_o is the permittivity of free space. This equation can be discretized in a finite difference form in three dimensions:

$$\frac{\phi_{i-1,j,k} - 2\phi_{i,j,k} + \phi_{i+1,j,k}}{(\Delta x)^2} + \frac{\phi_{i,j-1,k} - 2\phi_{i,j,k} + \phi_{i,j+1,k}}{(\Delta y)^2} + \frac{\phi_{i,j,k-1} - 2\phi_{i,j,k} + \phi_{i,j,k+1}}{(\Delta z)^2} = q\frac{n_e - n_i}{\varepsilon_o}. \quad (2.2)$$

Equation 2.2 in two dimensions is the basis for the computational simulation carried out in this work.

3.1.2 Initial Conditions and Boundary Conditions

At the radius of the cathode (red cells in Figure 2.1), an equipotential circle is defined with a Dirichlet boundary condition equal to the grid voltage imparted to the device. A similar condition is applied to the anode (dark blue cells), which is set to 0 V. The vacuum chamber walls are simulated as an open boundary: any particle that collides with this surface passes through and is lost from the computational domain. The domain is initiated with zero ions present, with a particle source near the anode. Particles are assumed to have no net initial velocity beyond thermal motion sampled from a Maxwellian distribution of initial ion velocities.

3.1.3 Particle-in-cell Technique

The PIC algorithm in this study utilizes the following steps in the main loop:

1. Determine the Charge Density
2. Determine the Electric Potential
3. Determine the Electric Field
4. Move the Macro particles and Check for Fusion
5. Sample Sources and Determine Losses
6. Repeat until Maximum Number of Time Steps is reached.

3.1.3.1 Determine the Charge Density

$$q_G = q\frac{(h_y)(h_x)}{\Delta x \Delta y}. \tag{2.4}$$

Here q_G is the charge contribution the green node receives from the grey macroparticle, h_x is the grey macroparticle position from the left of the cell, and h_y is the grey macroparticle position from the bottom of the cell.

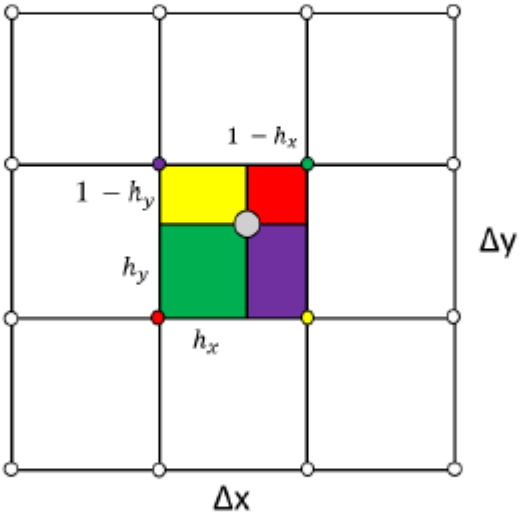


Figure 2.2 Visualizing the charge weighting to nearby numerical grid intersections. The grey macroparticle's charge is distributed to the nodes of the cell it is in. The proportion of charge that is received by the nodes is found by the area ratio using the particle's position in the cell as shown by Equation 2.4. This is visualized by the colored rectangles within the cell. The green node, for example, receives the green rectangle's proportion of the grey macroparticle's charge. Figure based on a discussion in [5].

3.1.3.2 Determine the Electric Potential

An important assumption that will be used in this study is that the electrons in the IEC plasma move instantly relative to the ions. In an electromagnetic field a particle's acceleration is proportional to the charge to mass ratio $\frac{q}{m}$. The deuterium mass is nearly 4,000 times larger than the electron mass, and hence to the ion the electron moves nearly instantly. The electrons are assumed to be a fluid with a temperature T_e. The electron density is assumed to maintain a Boltzmann distribution given by

$$n_e = n_o e^{\frac{q(\phi-\phi_o)}{k_B T_e}},$$

where ϕ_o is the reference ground potential, 0 V. Equation 2.5 can be substituted into Equation 2.2 for the electron number density in two dimensions with the cell length in the x and y direction equaling the Debye length, yielding Equation 2.6:

$$\frac{\phi_{i-1,j} - 2\phi_{i,j} + \phi_{i+1,j}}{(\lambda_D)^2} + \frac{\phi_{i,j-1} - 2\phi_{i,j} + \phi_{i,j+1}}{(\lambda_D)^2} = \frac{q}{\varepsilon_o}\left[n_o e^{\frac{q(\phi_{i,j} - \phi_o)}{k_B T_e}} - n_{ion}\right].$$

Equation 2.6 is no longer linear due to the exponential on the RHS containing the potential $\phi_{i,}$, so a simple matrix inversion by gaussian elimination cannot solve this system of equations. The potential $\phi_{i,}$ is solved for by setting the domain boundary conditions: the vacuum chamber walls and the anode are set to a potential of 0 V. The cathode is set to the cathode voltage (between -10 and -200 kV).
Next, the interior cells of the domain (excluding the vacuum chamber walls) are looped over and the potential at each cell is updated. The update formula is found by rearranging Equation 2.6:

$$\phi_{i,j}^{k} = \frac{\phi_{i-1,j}^{k} + \phi_{i+1,j}^{k-1} + \phi_{i,j-1}^{k-1} + \phi_{i,j+1}^{k}}{4} - \frac{{\lambda_D}^2 q}{4\varepsilon_o}\left[n_o e^{\frac{q(\phi_{i,j}^{k-1} - \phi_o)}{k_B T_e}} - n_{ion}\right].$$

3.1.3.3 Determine the Electric Field

$$E_{x,ij} = -\frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x},$$

$$E_{y,ij} = -\frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y},$$

$$E_{x,ij} = -\frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x},$$

$$E_{y,ij} = -\frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y}.$$

3.1.3.4 Move the Macroparticles and Check for Fusion

The macroparticles are moved using Newton's Second Law according to Equation 2.13:

$$\mathbf{v}^{s+0.5} = \mathbf{v}^{s-0.5} + \frac{q}{m_m}\mathbf{E}\Delta t.$$

Here the superscript s refers to the s^{th} timestep iteration. The particle position $\mathbf{r}$ at the next timestep is determined from the s^{th} timestep iteration as

$$\mathbf{r}^{s+1} = \mathbf{r}^{s} + \mathbf{v}^{s+0.5}\Delta t. \qquad (2.14)$$

The probability P that a macroparticle moving from $\mathbf{r}_s$ to $\mathbf{r}_{s+1}$ will undergo a fusion reaction is given by

$$P = w_s|\mathbf{v}^{s+0.5}\Delta t|\sigma_f(\mathbf{v})n_{ion}.$$

3.2 Particle-in-Cell Python Code

```
import numpy as np
import matplotlib.pyplot as plt
#setup constants
EPS0 = 8.854e-12        #permittivity of free space
QE = 1.602e-19          #elementary charge
kb = 1                  #boltzmann constant
AMU = 1.661e-27         #atomic mass unit
m_ion = 2*AMU           #ion mass (deuterium)

# Physical Grid Dimensions
R1 = 0.05             # Cathode [m]
GEO_C = .95           # Geometric transparency cathode (manual for now)
R2 = .085             # Anode [m]
GEO_A = .95           # Geometric transparency anode   (manual for now)
#Physical Chamber Dimensions
R_Chamber = .1        # [m]
H_Chamber = 0.5      # [m]
r_wire = .80*1e-3 / 2 # Radius of 20 guage wire in m
#Sourse Dimension and Distrobution
R_Sourse_min = .08  # smallest radius a sourse particle can appear at
sourse_spread = .0025 # largest radius a sourse particle can appear minus the minimum

#input settings
# n0 = 4.6e13
n0 = 1e16                 #electron background density in #/m^3
```

```
phi_Cathode = -210000        #cathode potential
phi0 = 0
# Te = np.abs(phi_Cathode)#reference potential
Te = 600    #electron temperature in eV
Ti = 0.1                    #ion velocity in eV (not used yet)
vth = np.sqrt(2*QE*Ti/m_ion)   #thermal velocity with Ti in eV
Operating_Pressure = 7      # Pa (Not used yet)
#calculate plasma parameters
#lD = np.sqrt(EPS0*Te/(n0*QE))      #Debye length (not used yet)
vth = np.sqrt(2*QE*Ti/m_ion)        #Thermal velocity with Ti in eV (not used yet)
lD = 0.004   # MANUAL SETTING FOR TESTING
dx = lD               #x cell size
dy = lD               #y cell size
#set simulation domain in the x dimension
nx = np.floor((2*R_Chamber)/dx).astype(int) #number of nodes in x direction, with a physical
                                            #domain +/- R_Chamber
#set simulation domain in the y dimension
ny = np.floor((H_Chamber)/dy).astype(int)   #number of nodes in y direction, with a physical
                                            #domain +/- H_Chamber / 2
ts = 800                                    #number of time steps

#calculate maximum expected velocity and timestep
E_av = (np.abs(phi_Cathode) - 0) / (R2 - R1)
a_av = E_av*QE / m_ion
v_max = np.sqrt(2*a_av*R2)
dt = 1e-10     #time step size, at vmax move 0.10dx


# Domain length in the x direction
Lx = (nx-1)*dx
# Domain length in the y direction
Ly = (ny-1)*dy
# SET UP THE POINTS REPRESENTING THE GRIDS
def get_discrete_Value(y, dy):
    # Takes in an array y and a grid spacing dy and returns an array
    # with the same physical dimensions as y but with a spacing of dy.
    actual = y / dy
    index = np.round(actual)
    return index*dy
```

```
# CATHODE SETUP
Thetav = np.arange(0, 2*np.pi, np.pi/1000)
x_cathode = R1*np.cos(Thetav)
y_cathode = R1*np.sin(Thetav)
X_cathode = get_discrete_Value(x_cathode, dx)
Y_cathode = get_discrete_Value(y_cathode, dy)

INDEX_X_Cathode = (X_cathode/dx).astype(int)
INDEX_Y_Cathode = (Y_cathode/dx).astype(int)

# ANODE SETUP
x_anode = R2*np.cos(Thetav)
y_anode = R2*np.sin(Thetav)
X_anode = get_discrete_Value(x_anode, dx)
Y_anode = get_discrete_Value(y_anode, dy)

INDEX_X_Anode = (X_anode/dx).astype(int)
INDEX_Y_Anode = (Y_anode/dy).astype(int)

# LEFT WALL SETUP
y_left_wall = np.arange(-H_Chamber/2, H_Chamber/2, dy)
x_left_wall = np.ones(y_left_wall.shape[0]) * (-R_Chamber)
INDEX_X_left_wall = (x_left_wall/dx).astype(int)
INDEX_Y_left_wall = (y_left_wall/dy).astype(int)
# RIGHT WALL SETUP
y_right_wall = np.arange(-H_Chamber/2, H_Chamber/2, dy)
x_right_wall = np.ones(y_right_wall.shape[0]) * (R_Chamber - dx)
INDEX_X_right_wall = (x_right_wall/dx).astype(int)
INDEX_Y_right_wall = (y_right_wall/dy).astype(int)
# BOTTOM WALL SETUP
x_bottom_wall = np.arange(-R_Chamber, R_Chamber, dx)
y_bottom_wall = np.ones(x_bottom_wall.shape[0]) * (-H_Chamber/2)
INDEX_X_bottom_wall = (x_bottom_wall/dx).astype(int)
INDEX_Y_bottom_wall = (y_bottom_wall/dy).astype(int)
# TOP WALL SETUP
x_top_wall = np.arange(-R_Chamber, R_Chamber, dx)
y_top_wall = np.ones(x_bottom_wall.shape[0]) * (H_Chamber/2 - dy)
```

```
INDEX_X_top_wall = (x_top_wall/dx).astype(int)
INDEX_Y_top_wall = (y_top_wall/dy).astype(int)

# Deleate repeate XY Pairs
Cathode_Id1 = np.zeros([2, INDEX_X_Cathode.shape[0]])
Cathode_Id1[0, :] = INDEX_X_Cathode
Cathode_Id1[1, :] = INDEX_Y_Cathode
Cathode_Id2 = np.unique(Cathode_Id1, axis=1)
INDEX_X_Cathode = Cathode_Id2[0, :].astype(int)
INDEX_Y_Cathode = Cathode_Id2[1, :].astype(int)

# Calculate specific weight and prepair to insert particles
np_insert = 400                      #insert 2 particles per anode cell.
#flux = 4.6e22
flux = 1.6e21*np.abs(phi_Cathode)/100000    #Flux of entering ions [ions per second]
npt = flux*dt
spwt = npt/np_insert                    #specific weight, real particles per macroparticle
mp_q = 1                                #macroparticle charge
max_part = 200000                       #buffer size

#allocate particle array
part_x = np.zeros([max_part,2]) #particle positions
part_v = np.zeros([max_part,2]) #particle velocities
sourse_storage = np.zeros([max_part,2]) #particle sourses
# Define the potential solver function

PHI_B = np.zeros([nx, ny])
idx = np.round(nx/2).astype(int)
idy = np.round(ny/2).astype(int)

#Potential Solver

def get_Potential(PHI_B, den, nx, ny, iters):
    for k in range(iters):
        PHI_OLD = PHI_B
        for i in range(1,nx-2):
            for j in range(1, ny-2):
                ni = den[i,j]
                rho = QE*(ni - n0*np.exp((PHI_OLD[i,j] - phi0)/(kb*Te))) / EPS0
```

```
                chrg = -rho*dx**2
                PHI_B[i,j] = (chrg - PHI_B[i+1, j] - PHI_B[i-1, j] - PHI_B[i, j+1] - PHI_B[i, j-
1])/(-4)
        PHI_B[INDEX_X_Cathode + idx, INDEX_Y_Cathode + idy] = phi_Cathode
        PHI_B[INDEX_X_Anode + idx, INDEX_Y_Anode + idy] = phi0
    return PHI_B


# Ion Source
def sample_Source(nump, R_S1, r_spread):
    # Updated to generate particles randomly in theta and in R.
    xv = np.zeros([nump])
    yv = np.zeros([nump])
    for i in range(nump):
        R_S = R_S1 + np.random.rand()*r_spread
        theta = np.random.rand(1)*2*np.pi   # Generate random polar angle
        x = R_S*np.cos(theta)                # Get x position
        y = R_S*np.sin(theta)                # Get y position
        xv[i] = x
        yv[i] = y
    return np.array([xv, yv])

# D-D Cross Section from 5 parameter fit
def fusion_Cross_Section(vx, vy):
    # Takes in velocity componants in m/s and returns a cross section in barns
    E = .5*m_ion*(vx**2 + vy**2)
    E = E*6.242e15  # convert J to KeV
    A1 = 46.097
    A2 = 372
    A3 = 4.36e-4
    A4 = 1.22
    A5 = 0
    AA1 = 47.88
    AA2 = 482
    AA3 = 3.08e-4
    AA4 = 1.177
    AA5 = 0
    term1 = A5 + A2/((A4 - A3*E)**2 + 1)
    term2 = E*(np.exp(A1/np.sqrt(E)) - 1)
```

```
    term3 = AA5 + AA2/((AA4 - AA3*E)**2 + 1)
    term4 = E*(np.exp(AA1/np.sqrt(E)) - 1)
    sig1 = term1/term2
    sig2 = term3/term4
    return sig1 + sig2


##############
# MAIN LOOP
#############

#INITIALIZE
PHI_M = np.zeros([nx, ny])
idx = np.round(nx/2).astype(int)
idy = np.round(ny/2).astype(int)
PHI_M[INDEX_X_Cathode + idx, INDEX_Y_Cathode + idy] = phi_Cathode
PHI_M[INDEX_X_Anode + idx, INDEX_Y_Anode + idy] = phi0
fuse_pos_x = np.array([])
fuse_pos_y = np.array([])
fuse_time = np.array([])
col_counter = 0
```

```
top_counter = 0
bot_counter = 0
left_counter = 0
right_counter = 0
anode_counter = 0
cathode_counter = 0
fuse_counter = 0
num_p = 0                          #Clear number of particles
iters = 600                        #Number of iterations used in the potential solver
P_FUS = 0 # set a checker to report max fusion probability
print('Beginning Main Loop. This could take a while. \n')

for it in range(ts):
    P_FUS = 0
    print('Time Step ', it, 'Particles ', num_p)
    #reset field quantities
    den = np.zeros([nx,ny])              #number density
```

```
efx = np.zeros([nx,ny])              #electric field, x-component
efy = np.zeros([nx,ny])              #electric field, y-component
chg = np.zeros([nx,ny])              #charge distribution
col_counter = 0

# *** 1. Calculate Charge Density ***
# deposit charge to nodes
for p in range(num_p):                          #loop over particles
    fi = (part_x[p, 0] + R_Chamber-dx)/dx       #real i index of particle's cell
    i = np.floor(fi).astype(int)                #integral part
    hx = fi - i                                 #the remainder
    fj = (part_x[p,1] + (H_Chamber/2)-dx)/dx #real i index of particle's cell
    j = np.floor(fj).astype(int)                #integral part
    hy = fj - j                                 #the remainder
    #interpolate charge to nodes
    chg[i, j] = chg[i, j] + (1-hx)*(1-hy)
    chg[i+1, j] = chg[i+1, j] + hx*(1-hy)
    chg[i, j+1] = chg[i, j+1] + (1-hx)*hy
    chg[i+1, j+1] = chg[i+1, j+1] + hx*hy

# Calculate the Density
den = spwt*mp_q*chg / (dx**2)
den[0,:] = 2*den[0,:]              # Double density since only half volume contributing
den[nx-1,:] = 2*den[nx-1,:]
den[:,0] = 2*den[:,0]
den[:,ny-1] = 2*den[:,ny-1]
den = den + 1e3                    # Add density floor to help the solver

# *** 2. Calculate the Potential
PHI_M = get_Potential(PHI_M, den, nx, ny, iters)
print('Potential Solution Complete.')
# *** 3. Calculate the Electric Field ***
efx[1:nx-2,:] = PHI_M[0:nx-3,:] - PHI_M[2:nx-1,:]   #central difference on internal nodes
efy[:,1:ny-2] = PHI_M[:,0:ny-3] - PHI_M[:,2:ny-1]   #central difference on internal nodes
efx[0,:] = 2*(PHI_M[0,:] - PHI_M[1,:])             #forward difference on x=0
efx[nx-1,:] = 2*(PHI_M[nx-2,:] - PHI_M[nx-1,:])     #backward difference on x=Lx
efy[:,0] = 2*(PHI_M[:,0] - PHI_M[:,1])              #forward difference on y=0
efy[:,ny-1] = 2*(PHI_M[:,ny-2] - PHI_M[:,ny-1])     #forward difference on y=Ly
```

```
    efx = efx / (dx**2)                                  #divide by denominator
    efy = efy / (dx**2)

    # *** 4. Generate New Particles
    print('Generating Particles')
    #insert particles randomly distributed and within the sourse region
    Posv = sample_Source(np_insert, R_Sourse_min, sourse_spread)
    part_x[num_p:num_p+np_insert, 0] = Posv[0,:]   #x position
    part_x[num_p:num_p+np_insert, 1] = Posv[1,:]   #y position
    #sourse_storage[num_p:num_p+np_insert, 0] = Posv[0,:]
    #sourse_storage[num_p:num_p+np_insert, 1] = Posv[1,:]
    #sample maxwellian in x and y
    pt1 = np.random.rand(np_insert)
    pt2 = np.random.rand(np_insert)
    pt3 = np.random.rand(np_insert)
    pt11 = np.random.rand(np_insert)
    pt12 = np.random.rand(np_insert)
    pt13 = np.random.rand(np_insert)
    part_v[num_p:num_p+np_insert,0] = (-1.5 + pt1 + pt2 + pt3)*vth     # x velocity
    part_v[num_p:num_p+np_insert,1] = (-1.5 + pt11 + pt12 + pt13)*vth  # y velcoity
    num_p = num_p + np_insert                                          #increment particle counter
# *** Move Particles ***
print('Moving Particles...   DO NOT INTERRUPT')
p=0
while p < num_p:        # Loop over particles
    fi = (part_x[p, 0] + R_Chamber-dx)/dx   # i index of particle's cell. Taking into account
    i = np.floor(fi).astype(int)         # that the physical domain is centered at (0,0)
    hx = fi - i
    fj = (part_x[p,1] + (H_Chamber/2)-dx)/dx
    j = np.floor(fj).astype(int)
    hy = fj-j
    #print('Fi: ', fi, 'Fj: ', fj)
    # gather electric field
    E = np.array([0,0])
    E = np.array([efx[i,j], efy[i,j]])*(1-hx)*(1-hy)      #contribution from (i,j)
    E = E + np.array([efx[i+1,j], efy[i+1,j]])*hx*(1-hy)            #(i+1,j)
    E = E + np.array([efx[i,j+1], efy[i,j+1]])*(1-hx)*hy            #(i,j+1)
    E = E + np.array([efx[i+1,j+1], efy[i+1,j+1]])*hx*hy            #(i+1,j+1)
```

```
    #print('E:', E)
    # Update Velocity and Position
    F = QE*E      # Lorenz force F = qE
    a = F/m_ion   # Acceleration
    part_v[p,:] = part_v[p,:] + a*dt
    part_x[p,:] = part_x[p,:] + part_v[p,:]*dt
    #print(part_v[p,:])
    #print(part_x[p,:])
    # Get Fusion probability
    vx = part_v[p, 0]
    vy = part_v[p, 1]
    delx = vx * dt * 1e4
    dely = vy * dt * 1e4
    path_len = np.sqrt(delx**2 + dely**2)
    sigma = fusion_Cross_Section(vx, vy)*1e-28 # microscopic cross section [m^2]

    # gather density at particle position
    Rho = den[i,j]*(1-hx)*(1-hy)      #contribution from (i,j)
    Rho = Rho + den[i+1,j]*hx*(1-hy)            #(i+1,j)
    Rho = Rho + den[i,j+1]*(1-hx)*hy            #(i,j+1)
    Rho = Rho + den[i+1,j+1]*hx*hy            #(i+1,j+1)

# Calculate macroscopic cross section
SIGMA = Rho * sigma
Prob_fusion = SIGMA * path_len * spwt
# Store
if Prob_fusion > P_FUS:
    P_FUS = Prob_fusion
# Prepare to check for fusion
random = np.random.rand(1)

# Process Boundries and Sinks
R = np.sqrt(part_x[p,0]**2 + part_x[p,1]**2)
#print('Finished Position and Velocity Update')
#print('Checking Top Wall')
# Top Wall
if part_x[p,1] > H_Chamber/2:
    part_x[p,:] = part_x[num_p-1,:]  # Kill particle by replacing with last particle
```

```
        part_v[p,:] = part_v[num_p-1,:]
        part_x[num_p-1,:] = np.array([0,0]) # Reset last particle to 0
        part_v[num_p-1,:] = np.array([0,0])
        num_p = num_p - 1              # Reduce particle count
        p = p - 1                      # Reduce particle index
        col_counter = col_counter + 1
        top_counter = top_counter + 1

    # Bottom Wall
    elif part_x[p,1] < -H_Chamber/2:
        #print('Checking Bottom Wall')
        part_x[p,:] = part_x[num_p-1,:]  # Kill particle by replacing with last particle
        part_v[p,:] = part_v[num_p-1,:]
        part_x[num_p-1,:] = np.array([0,0]) # Reset last particle to 0
        part_v[num_p-1,:] = np.array([0,0])
        num_p = num_p - 1              # Reduce particle count
        p = p - 1                      # Reduce particle index
        col_counter = col_counter + 1
        bot_counter = bot_counter + 1

    # Right Wall
    elif part_x[p,0] > R_Chamber:
        #print('Checking Right Wall')
        part_x[p,:] = part_x[num_p-1,:]  # Kill particle by replacing with last particle
        part_v[p,:] = part_v[num_p-1,:]
        part_x[num_p-1,:] = np.array([0,0]) # Reset last particle to 0
        part_v[num_p-1,:] = np.array([0,0])
        num_p = num_p - 1              # Reduce particle count
        p = p - 1                      # Reduce particle index
        col_counter = col_counter + 1
        right_counter = right_counter + 1

    # Left Wall
    elif part_x[p,0] < -R_Chamber:
        #print('Checking Left Wall')
        part_x[p,:] = part_x[num_p-1,:]  # Kill particle by replacing with last particle
        part_v[p,:] = part_v[num_p-1,:]
        part_x[num_p-1,:] = np.array([0,0]) # Reset last particle to 0
```

```
        part_v[num_p-1,:] = np.array([0,0])
        num_p = num_p - 1              # Reduce particle count
        p = p - 1                      # Reduce particle index
        col_counter = col_counter + 1
        left_counter = left_counter + 1

    # Process grids

    # Anode
    elif (R < R2 + r_wire) and (R > R2 - r_wire):
        #print('Check Anode')
        prob = np.random.rand(1)
        if prob > GEO_A:
            # Delete particle if it collides with the grid
            part_x[p,:] = part_x[num_p-1,:]  # Kill particle by replacing with last particle
            part_v[p,:] = part_v[num_p-1,:]
            part_x[num_p-1,:] = np.array([0,0]) # Reset last particle to 0
            part_v[num_p-1,:] = np.array([0,0])
            num_p = num_p - 1              # Reduce particle count
            p = p - 1                      # Reduce particle index
            col_counter = col_counter + 1
            anode_counter = anode_counter + 1
```

```
# Cathode
elif (R < R1 + r_wire) and (R > R1 - r_wire):
    #print('Check Cathode')
    prob = np.random.rand(1)
    if prob > GEO_A:
        # Delete particle if it collides with the grid
        part_x[p,:] = part_x[num_p-1,:]  # Kill particle by replacing with last particle
        part_v[p,:] = part_v[num_p-1,:]
        part_x[num_p-1,:] = np.array([0,0]) # Reset last particle to 0
        part_v[num_p-1,:] = np.array([0,0])
        num_p = num_p - 1              # Reduce particle count
        p = p - 1                      # Reduce particle index
        col_counter = col_counter + 1
        cathode_counter = cathode_counter + 1
```

```
        elif random <= Prob_fusion:
            print('FUSION!\n')
            fuse_pos_x = np.append(fuse_pos_x, part_x[p, 0])
            fuse_pos_y = np.append(fuse_pos_y, part_x[p, 1])
            fuse_time = np.append(fuse_time, dt*it)
            fuse_counter = fuse_counter + 1
            # Delete particle if it fused
            part_x[p,:] = part_x[num_p-1,:]  # Kill particle by replacing with last particle
            part_v[p,:] = part_v[num_p-1,:]
            part_x[num_p-1,:] = np.array([0,0]) # Reset last particle to 0
            part_v[num_p-1,:] = np.array([0,0])
            num_p = num_p - 1              # Reduce particle count
            p = p - 1                      # Reduce particle index

        p = p + 1                                # Move to the next particle
    print('Finished Moving Particles.')
    print('Net Change in Ion Population: ', np_insert - col_counter)
    print(col_counter, ' particles lost.')
    print('Max probability of fusion: ', P_FUS, '\n')

# Replace voltage with Phi_Cathode
np.savetxt('Density_210kV_1.csv', den, delimiter=',')
np.savetxt('Potential_210kV_1.csv', PHI_M, delimiter=',')
np.savetxt('fus_time_210kV_1.csv', fuse_time, delimiter=',')
np.savetxt('fus_pos_x_210kv_1.csv', fuse_pos_x, delimiter=',')
np.savetxt('fus_pos_y_210kV_1.csv', fuse_pos_y, delimiter=',')
np.savetxt('part_x_210kV_1.csv', part_x, delimiter=',')
np.savetxt('part_v_210kV_1.csv', part_v, delimiter=',')

av_fusion_rate = (fuse_counter / (dt*ts)) / 1e4
print('Field Properties and Fusion Data Saved to File.\n')
print('Average Fusion Rate: %.6f Fusions per Second\n' % av_fusion_rate)
```