eduPIC: an introductory particle based code for radio-frequency plasma simulation

Zoltán Donkó[1,*] ⓘ**, Aranka Derzsi**[1] ⓘ**, Máté Vass**[1,2] ⓘ**, Benedek Horváth**[1] ⓘ**, Sebastian Wilczek**[2] ⓘ**, Botond Hartmann**[3] **and Peter Hartmann**[1] ⓘ

[1] Institute for Solid State Physics and Optics, Wigner Research Centre for Physics, 1121 Budapest, Konkoly-Thege Miklós Str. 29-33, Hungary
[2] Department of Electrical Engineering and Information Science, Ruhr-University Bochum, D-44780, Bochum, Germany
[3] Óbudai Árpád Secondary Grammar School, 1034 Budapest, Hungary

E-mail: donko.zoltan@wigner.hu

CrossMark

Abstract

Particle based simulations are indispensable tools for numerical studies of charged particle swarms and low-temperature plasma sources. The main advantage of such approaches is that they do not require any assumptions regarding the shape of the particle velocity/energy distribution function (VDF/EDF), but provide these basic quantities of kinetic theory as a result of the computations. Additionally, they can provide, e.g. transport coefficients, under arbitrary time and space dependence of the electric/magnetic fields. For the self-consistent description of various plasma sources operated in the low-pressure (nonlocal, kinetic) regime, the particle-in-cell simulation approach, combined with the Monte Carlo treatment of collision processes (PIC/MCC), has become an important tool during the past decades. In particular, for radio-frequency (RF) capacitively coupled plasma (CCP) systems PIC/MCC is perhaps the primary simulation tool these days. This approach is able to describe discharges over a wide range of operating conditions, and has largely contributed to the understanding of the physics of CCPs operating in various gases and their mixtures, in chambers with simple and complicated geometries, driven by single- and multi-frequency (tailored) waveforms. PIC/MCC simulation codes have been developed and maintained by many research groups, some of these codes are available to the community as freeware resources. While this computational approach has already been present for a number of decades, the rapid evolution of the computing infrastructure makes it increasingly more popular and accessible, as simulations of simple systems can be executed now on personal computers or laptops. During the past few years we have experienced an increasing interest in lectures and courses dealing with the basics of particle simulations, including the PIC/MCC technique. In a response to this, the current paper (i) provides a tutorial on the physical basis and the algorithms of the PIC/MCC technique and (ii) presents a basic (spatially one-dimensional) electrostatic PIC/MCC simulation code, whose source is made freely available in various programming languages. We share the code in C/C++ versions, as well as in a version written in Rust, which is a rapidly emerging computational language. Our code intends to be a 'starting tool' for those who are interested in learning the details of the PIC/MCC technique and would like

to develop the 'skeleton' code further, for their research purposes. Following the description of the physical basis and the algorithms used in the code, a few examples of results obtained with this code for single- and dual-frequency CCPs in argon are also given.

(Some figures may appear in colour only in the online journal)

1. Introduction

Particle based simulations are indispensable tools for the investigation of plasma sources operated at conditions when kinetic effects prevail [1]. Such conditions are most prominent in weakly collisional settings, e.g. at low pressures where the mean free path of the charged particles (typically of the electrons) can be comparable or even longer than the dimensions of the system. Under such conditions, the transport has a nonlocal character [2–6] and although continuum (fluid) approaches can capture this phenomenon to some extent, it is usually achieved by including the corresponding energy/velocity distribution function (EDF/VDF) and nonlocal closure of the equations as an *input* to the model [7]. Particle methods, on the other hand, do not require any assumptions of this kind and provide the EDF/VDF as a *result* of the calculations. The price of using particle based approaches for the description of gas discharges is their computationally intensive nature. The efficiency of the simulations can largely be enhanced by carefully designed hybrid schemes [8, 9] where different (fluid/particle) approaches are used for the different plasma species. Such approaches are needed in the presence of, e.g. a complicated plasma chemistry, which may as well necessitate simulating processes on considerably different time scales [10]. Improvements to fluid models can widen the range of their applicability as it was shown in [11], however, at low pressures particle based approaches are still expected to be more accurate.

The history of particle based simulations, including the particle-in-cell (PIC) approach [12] dates back to the late 1950s when pioneering simulations [13, 14] have been implemented for studies of basic plasma properties and instabilities. Subsequently, for the simulation of plasma sources operating in the collisional regime, the PIC approach has been complemented with a stochastic, Monte Carlo type treatment of the collision processes, see e.g. [15–17]. This approach became known as 'particle-in-cell/Monte Carlo collisions', PIC/MCC.

We would like to note that Monte Carlo simulations of the charged particle transport in gaseous medium have been applied as well for decades, in swarm and discharge physics, e.g. [18–27].

While decades ago PIC and PIC/MCC codes could be executed on mainframe computers only, the advance of the computational resources has made the technique widely available by now. Today, electrostatic 1D simulations can be executed on PC-class computers and further spread of the technique is

expected with the advance of graphic processing units (GPUs), which considerably enhance computing performance [28–34] and allow carrying out simulations efficiently in 2D and 3D as well. Consequently, the computationally intensive nature of the particle based approaches, which has been emphasized as a disadvantage for decades, is becoming less constraining these days.

Based on the PIC/MCC approach, a wide variety of phenomena and effects have been investigated during the past decades in various plasma sources [35], e.g. the breakdown of the gases and the formation of the plasma [36, 37], the energy and/or angular distributions of ions at boundary surfaces [38–43], the operation of Hall thrusters [44–46], electron heating and heating mode transitions [47–49] (termed more correctly as electron power absorption and power absorption mode transitions in more recent literature [1, 50, 51]), the formation of striations [52], plasma series resonance oscillations [53, 54], electron dynamics in the afterglow [55], as well as the effects of ion-induced [56], and electron-induced [57, 58] secondary electrons on the plasma characteristics, the physics of fast-pulsed discharges [59] and atmospheric-pressure plasma jets [60]. Most of these investigations have found that the electron energy probability function (EEPF) in these plasmas is highly non-Maxwellian, which confirms the need for kinetic simulations.

A number of the above studies concerned low-pressure capacitively coupled plasma (CCP) sources. For these, actually, the PIC/MCC method became the most widely applied technique, which proved to be indispensable in the studies of plasma sources driven by various waveforms aimed at an enhanced control of the ion properties, i.e. the ion flux-energy distribution function (IFEDF), as well as the angular distribution of the ions at boundary surfaces. The paramount importance of these investigations stems from the applications of CCP discharges in microelectronics, photo-voltaic industry as well as in medical technologies that are based on the interaction of the active plasma species with surfaces like semiconductor wafers, medical implants, etc. [61–63]. The method has frequently been applied in studies of dual-frequency (DF) RF discharges [64, 65], plasma sources operated under the conditions where electrical asymmetry develops [66, 67], as well as in investigations of discharges driven by tailored voltage waveforms [68–71]. Related to this, the effects of various asymmetries, like those created by a magnetic field [72], by unequal secondary electron yields at the two electrodes [73, 74] have been investigated, as well as the interplay between geometrical and electrical asymmetry effects in CCPs

[6, 75]. A similarity law and the frequency scaling properties of CCPs were recently reported [76].

CCPs have been studied using the PIC/MCC method in various gases, e.g. in Ar [55, 77], He [78], O_2 [79–84], H_2 [85], CF_4 [49, 86–88], N_2 [89, 90] and Cl_2 [91], as well as in Ar–CF_4 [92, 93], Ar–O_2 [94], Xe–Ar [95], CH_4–H_2 [96], and Ar–H_2 [97] mixtures. The limitations and the optimization of the PIC/MCC method have been discussed in, e.g. [98–101], and the issues of code verification and validation have been addressed in, e.g. [102–104].

The PIC/MCC method belongs to the class of particle–mesh techniques. While charged particles move in space, their densities as well as the electric field and potential distributions (in *electrostatic* simulations) are computed on a discrete grid. The latter quantities are defined via the Poisson equation that takes into account the space charge generated by the ensemble of the charged particles (electrons and ions), as well as the potential distributions at the electrode surfaces that appear as boundary conditions of the differential equation. The Poisson equation has to be solved typically several thousand times within an RF cycle, at the usual (∼10 MHz) RF frequencies. The densities of the charged particles are computed in these time steps as well, and their positions are updated, too, according to the actual value of the electric field at their positions. In the meantime, particles may reach the electrodes, where various processes (absorption, reflection, emission of additional particles) may take place as defined by the surface model, and particles may as well collide within the gas phase with the atoms/molecules of the background gas. The above 'ingredients' of the PIC/MCC method will be discussed in considerable detail in section 2.

While above we introduced some general characteristics of electrostatic PIC/MCC simulations, for completeness, it has to be mentioned that such simulations can also fully account for *electromagnetic* effects which emerge in systems where the time derivative of the magnetic field cannot be neglected. This is the scenario, e.g. in inductively coupled plasmas, and also in CCPs when the physical dimensions of the system become comparable to the wavelength of the excitation signal (that actually shrinks due to the permittivity of the plasma) [105, 106]. Electromagnetic simulations incorporate the solution of Maxwell's equations, see e.g. [107, 108].

PIC/MCC codes have been developed by many groups worldwide. Most of these serve own research purposes of the respective groups, but some are available as freely accessible useful resources, e.g. [109]. Excellent descriptions of the PIC/MCC method, e.g. [110–112] have been published both for electrostatic and electromagnetic cases, and for 1D and higher dimensions as well. Codes have been developed in a variety of languages, e.g. Pascal [113], C/C++ [114, 115], Fortran [116, 117], MATLAB [118], Julia [119], Java [120]. The choice of the language can be motivated by considerations about compiler and code performance, availability of code libraries, quality of the development and debugging environments, availability of graphical user interface and real time interaction with the user and client-server operations etc., but primarily remains a matter of personal preference.
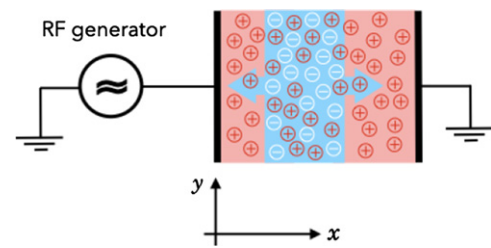

Figure 1. The model system considered in the eduPIC simulation code. In experiments, the RF power is coupled to the plasma via a matching network containing a blocking capacitor. A symmetric RF discharge driven by a single-harmonic waveform, such as considered here, can be, however, modeled based on this simplified setup.

In this paper, we provide an elementary introduction to the PIC/MCC technique, especially for those who would like to understand every detail and are motivated to develop their 'research grade' code from the open source 'starting tool' provided here in C, C++ and Rust languages. This code is dedicated to assisting in the education of those interested in particle based plasma simulations, hence the name 'eduPIC'. Our intention has been to keep this code as simple and as transparent as possible; optimization and further development is left for the interested readers. The codes that we provide are ∼1000 lines in length, in a single file. We include minimum diagnostics, which is, however, already sufficient for the analysis of some phenomena, as will be demonstrated in sections 4.1 and 4.2. While, as it has been mentioned earlier, PIC/MCC codes are capable of addressing the physics of various plasma sources and have been aiding the understanding of various physical effects, in this paper we will apply this technique exclusively for the description of CCPs. One should, however, keep it in mind that with proper modifications, use of the code presented here can be extended beyond CCPs.

Figure 1 presents the model system: we consider a CCP in which the electrodes are plane and parallel, and assume that the electrodes have a diameter that exceeds their separation considerably, reducing the spatial dependence of the plasma characteristics to one dimension. A rather general characteristic of such systems is that electrons are able to respond to the rapid variation of the electric field (on the nanosecond time scale), while the ions (of most gases) can only respond to the time average of the electric field [105]. As a consequence, the system splits into a bulk quasineutral region and sheath regions that appear/disappear near both electrodes in opposite phase during the RF cycles. The ion density distribution is therefore nearly stationary, while the electron density distribution varies significantly. Especially at low pressures, when the mean free path of the electrons may become comparable or even longer than the system size, kinetic effects are expected to arise and an accurate description of the system may be expected only from kinetic methods, like particle based simulations. Besides the electrons, the flight of the ions across the electrode sheaths may as well be strongly non-local and therefore an accurate calculation of the ion energy distribution at the surfaces also calls for a kinetic description. The PIC/MCC method clearly

qualifies for these purposes, and this is clearly reflected in its increasing popularity.

The paper is structured in the following way. Section 2 describes the details of the PIC/MCC approach. In section 3, we discuss the details of the implementation, including the basic simulation parameters, the cross sections, thoughts on random number generation, guidance for code compilation and execution, as well as about the data files created by the code (sections 3.1–3.5). The basic (C) version of the code is explored to algorithmic details in section 3.6, while a more general overview of the specific features of the C++ and Rust codes are given in sections 3.7 and 3.8, respectively. In section 4.1, we present results for various discharge characteristics obtained for a set of 'reference conditions', and in section 4.2, we analyze some of the characteristics of CCPs excited by a voltage waveform consisting of two components. The paper ends with a brief summary and a list of suggestions to the readers, who are interested in the further development of the eduPIC code (section 5).

2. The principles of the PIC/MCC method

As in CCPs the plasma density is typically in the range of $\sim 10^9$–10^{10} cm^{-3} and volumes of several hundreds to several thousands of cm^3 are usual, the number of electrons/ions in these systems can be in the $\sim 10^{13}$ range. Accounting for the mutual interaction and following the motion of so many particles would be impossible. Therefore, in the PIC approach, the direct (pair) interaction of the particles is omitted, the particles move under the influence of an electric field created by the potentials applied to the electrodes and modified by the space charge due to the presence of the charged particles. Additionally, instead of single particles, '*superparticles*' are traced that represent a given (high) number of individual particles. These two ideas make computations feasible with acceptable accuracy and within acceptable time.

Time is discretized into small $\Delta t = T_{RF}/N_t$ time steps, where T_{RF} is the period of the driving RF voltage and N_t is the number of time steps within the RF cycle. Besides time, the inter-electrode space $[0, L]$ is also discretized in the form of an equidistant computational grid having a division Δx and N_g points. Following the C programming language convention, the time steps are numbered from 0 to $N_t - 1$, while the grid points are numbered from 0 to $N_g - 1$. The distance between the grid points is $\Delta x = L/(N_g - 1)$. (We note, that the equidistant grid represents the simplest approach, and for specific applications, e.g. where strong field gradients are present in certain sub-domains of the computational region, a non-uniform grid can have advantages.)

The PIC/MCC method is equally suited to describe 'voltage-driven', as well as 'current-driven' discharges; these cases can be realized via implementing the proper boundary conditions [121]. At this point, a note about the possibility of comparison of the simulations with experimental systems is in order. In research laboratories the electrical (voltage and current) waveforms are often measured in experiments, these make a direct comparison with simulation results possible. In industrial plasma processing, however, the main control

parameter is usually the discharge power. This quantity can be computed from the simulations, but cannot be directly used as an input parameter in the form of a boundary condition. This means that one should either run simulations for several voltage values to find the conditions for a specified power, or should introduce an iteration loop into the simulation that adjusts the discharge voltage until the specified power level is reached. In our basic PIC/MCC code, we do not aim to reach this goal, the implementation of such an algorithm is left for interested readers (see also section 5).

Here, we consider the former case and, correspondingly, specify the driving voltage waveform. This waveform is directly applied to the discharge, i.e. we do not consider any external circuit. Furthermore, we assume that the electrodes are conductive. The driving voltage is applied at $x = 0$ (grid point 0) and the electrode at $x = L$ (grid point $N_g - 1$) is grounded.

As mentioned above already, the simulation traces superparticles that represent a high number of real particles. The number of real particles represented by a superparticle is called the *weight W*, in our definition. This key factor links the number of the superparticles with the density of the real charged particles (electrons and ions). To establish this relation, in 1D, one needs to define an arbitrary unit for the electrode surface area, which we set to $A = 1$ cm^2. With N_e superelectrons in the simulation, we have WN_e real electrons in the volume $V = AL$. The spatially averaged electron density is $\overline{n}_e = WN_e/V$. (As an example, assuming that we have an average electron density $\overline{n}_e = 4 \times 10^9$ cm^{-3} within an electrode gap of $L = 2.5$ cm, at a weight of $W = 10^5$ this density is established by $N_e = 10^5$ superelectrons.) Here, the same weight factor is used for electrons and ions. The spatially averaged ion density is $\overline{n}_i = WN_i/V$, where N_i is the number of superions in the simulation.

Finally, we note that in this simulation approach the superparticles are assumed to have a finite size, and their charge is spatially distributed according to specific cloud shapes, see, e.g. [16]. This finite size suppresses the short-range interaction between the particles, which can, however be considered by adding Coulomb collisions to the simulation [122]. These collisions between charged particles can be neglected at low enough densities, but not at densities exceeding typically $\sim 10^{11}$–10^{12} cm^{-3}, when the Coulomb interaction between closely separated charged particles may become significant and may influence the distribution function and the transport properties of these species. At such conditions, the EEDF tends to acquire a Maxwell–Boltzmann form and the plasma becomes nonideal. These effects should be kept in mind if simulations are conducted at elevated densities. For more detailed discussions of these effects we refer to [98, 122, 123].

In this paper, we choose a simple electropositive atomic gas (argon) as a medium for the creation of plasma and consider only a limited set of elementary processes. The charged particles that we consider are electrons and singly ionized atomic ions. This should be kept in mind during all the forthcoming discussions. One needs to note, however, that more complicated scenarios (e.g. electronegative discharges,
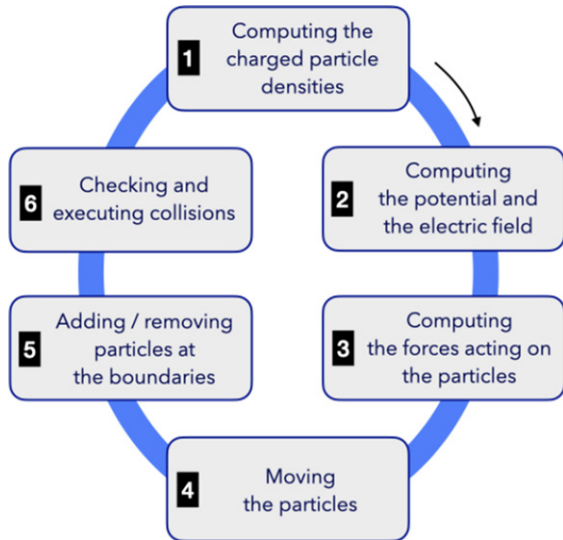
Figure 2. Steps of the basic PIC/MCC cycle.

Figure 3. Computation of the charged particle density at the grid points. Particle j resides at position x_j. The green line marks the computational domain, $0 \leqslant p \leqslant N_g - 1$ is the index of the grid points.

of a finite size [16] (mentioned in section 2)—here we assume a triangle-shaped charge cloud.

2.1.2. Computation of the potential and the electric field. The potential distribution is obtained as the solution of the Poisson equation:

$$\nabla^2 \phi = -\frac{\rho}{\varepsilon_0}. \tag{4}$$

This equation can be rewritten in the following finite difference form for a 1D problem:

$$\frac{-\phi_{p-1} + 2\phi_p - \phi_{p+1}}{\Delta x^2} = \frac{\rho_p}{\varepsilon_0}, \tag{5}$$

known as the discrete Poisson equation. This set of linear equations is solved on the grid: the potential distribution is calculated from the charge distribution, taking into account the potentials applied to the conductive electrodes as boundary conditions. By differentiating the potential distribution, we obtain the electric field at each grid point:

$$E_p = \frac{\phi_{p-1} - \phi_{p+1}}{2\Delta x}. \tag{6}$$

The boundary grid points need to be treated specially as:

$$E_0 = \frac{\phi_0 - \phi_1}{\Delta x} - \rho_0 \frac{\Delta x}{2\varepsilon_0}, \tag{7}$$

$$E_{N_g-1} = \frac{\phi_{N_g-2} - \phi_{N_g-1}}{\Delta x} + \rho_{N_g-1} \frac{\Delta x}{2\varepsilon_0}. \tag{8}$$

2.1.3. Computation of the forces acting on the particles. In this step, the electric field known at the grid points is interpolated to the positions of the particles. The electric field at the x_j position of particle j, located between grid points p and $p + 1$ (see figure 4), is obtained as

$$E_j = E_p \frac{x_{p+1} - x_j}{\Delta x} + E_{p+1} \frac{x_j - x_p}{\Delta x}, \tag{9}$$

and the force acting on the particle with a charge q_j is

$$F_j = E_j q_j. \tag{10}$$

2.1.4. Moving of the particles. The particles are advanced as dictated by the equations of motion. The new positions and velocities of the particles are obtained from the solution of the discretized equation of motion. Figure 5 illustrates the leapfrog integration method, where the particle velocities

multiple ionic species, various plasma-chemical reactions) can also be analyzed within the PIC/MCC framework, with proper extensions (not covered here).

2.1. The basic PIC/MCC cycle

Figure 2 shows the distinct steps of the PIC/MCC cycle (which is not to be confused with the RF cycle, as the latter normally comprises thousands of the PIC/MCC cycles, given by N_t), which are executed at every Δt time step. Note, that as the motion of the ions is much slower as compared to the motion of the electrons, unequal time steps for these two species can be adopted in the simulations. This *ion subcycling* can result in a substantial decrease of the computational time. When this approach is used, some of the basic steps shown in figure 2 are *not* executed in every cycle for the ions. In the detailed description of these steps, which follows, we keep on using a 'generic' Δt time step. This may, however, represent different values for the electrons and the ions.

2.1.1. Computation of the charged particle densities. In this step, we determine the densities of the charged particles at the points of the computational grid. The densities at the two grid points, which enclose the particle j (grid points p and $p + 1$, see figure 3) are incremented by:

$$\delta n_p = \left[(p+1) - x_j/\Delta x \right] \frac{W}{A \Delta x}, \tag{1}$$

$$\delta n_{p+1} = (x_j/\Delta x - p) \frac{W}{A \Delta x}, \tag{2}$$

where x_j is the position of the particle j and p is the integer part of $x_j/\Delta x$. This is done for all particles, resulting in the charge density distribution on the grid:

$$\rho_p = e(n_{i,p} - n_{e,p}), \tag{3}$$

where $n_{i,p}$ and $n_{e,p}$ are the densities of singly charged positive ions and electrons, respectively, at grid point p. The above procedure actually corresponds to the concept of using particles
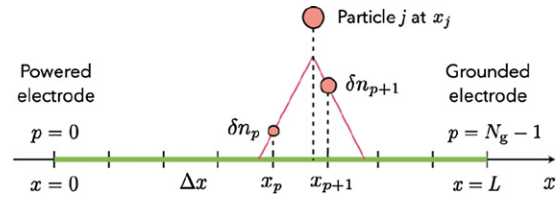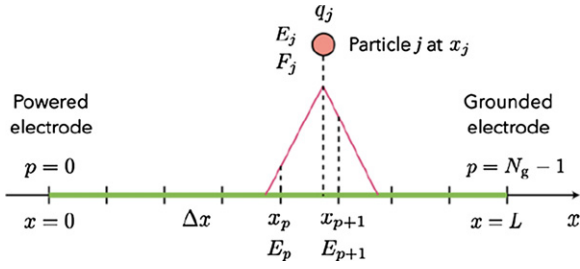
Figure 4. Interpolation of the electric field (known at the grid points) to the positions of the particles. Particle *j* with a charge q_j resides at position x_j. The green line marks the computational domain, $0 \leqslant p \leqslant N_g - 1$ is the index of the grid points.
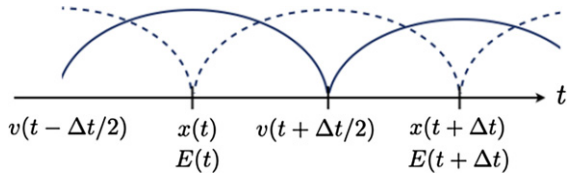

Figure 5. The leapfrog integration method for advancing the particles. The particle positions (x) and the electric field (E) are known at integer time steps, while the particle velocities (v) are defined at half integer time steps. The quantities are advanced according to equations (11) and (12).

(v) are defined at half integer time steps, while the particle positions are updated at integer time steps:

$$v(t + \Delta t/2) = v(t - \Delta t/2) + \frac{q}{m}E[x(t)]\Delta t, \quad (11)$$

$$x(t + \Delta t) = x(t) + v(t + \Delta t/2)\Delta t. \quad (12)$$

Note that as the measurement of a number of plasma characteristics requires position and velocity values at the same time, the computation of the velocity at the 'intermediate' time *t* is also required upon data collection.

2.1.5. Adding/removing particles at the boundaries. In this step, the particles that reached the boundary surfaces are identified, and their interaction with the surfaces (e.g. absorption, reflection, secondary electron emission) is accounted for. The particles absorbed at the surfaces are removed from the simulation. The new particles created at the surfaces (e.g. secondary electrons) can be added to the simulation, according to the surface model [57, 77, 78, 124–129]. In the present work, we use a very simple surface model: all particles reaching the electrodes are absorbed and no particles are emitted from the electrodes.

2.1.6. Checking and executing collisions. At every time step (Δt) of the given species, a decision about the occurrence of a collision has to be made for each particle. In the simplest manner, this is accomplished for each particle by (i) computing the collision probability, P_{coll}, and (ii) comparing this probability with a random number with uniform distribution on the [0, 1) interval, denoted by R_{01}: if $P_{coll} < R_{01}$ a collision will take place. (The R_{01} notation will often be used in this section,
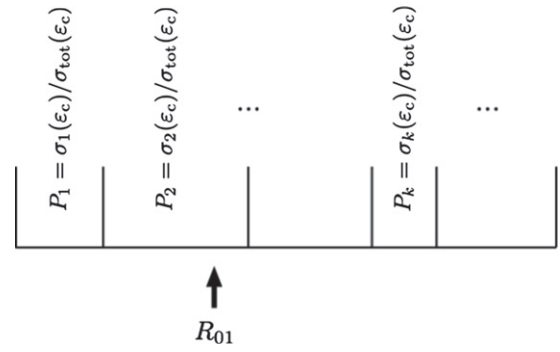

Figure 6. Choosing the actual elementary process by dividing the [0, 1) interval according to the probabilities (P_k) of occurrence of possible individual processes and selecting the type of the process by a uniformly distributed random number, R_{01}.

multiple occurrences of R_{01} always represent different random samples of the same uniform distribution.)

The collision probability is computed as:

$$P_{coll} = 1 - \exp[-n\sigma_{tot}(g)g\Delta t], \quad (13)$$

where *n* is the density of the target species, which is the background argon gas here, σ_{tot} is the *total cross section*, which is the sum of the cross sections of all possible collision processes of the given species, and

$$\mathbf{g} = \mathbf{v}_1 - \mathbf{v}_2 \quad (14)$$

is the *relative velocity* of the collision partners ($\mathbf{v}_1$ and $\mathbf{v}_2$ denote the velocity of the projectile and the target particles, respectively). When using equation (13) one has to make sure that the collision probability (that can be controlled by Δt) is small enough to make the chance for multiple collisions within a time step marginal (see also in section 2.2).

Whenever a collision occurs, its type is chosen in a Monte Carlo manner, by considering the values of the cross sections of all possible processes at the energy upon collision ($\varepsilon_c = \mu g^2/2$, where μ is the reduced mass). To select the type of the collision, the ratios of the individual cross sections to the total cross section, $P_k = \sigma_k(\varepsilon_c)/\sigma_{tot}(\varepsilon_c)$, are evaluated. These values add up to 1.0, which allows dividing the [0, 1) interval into segments of width P_k and choosing the type of process by identifying the interval into which a random sample R_{01} falls, as shown in figure 6.

Using the approach outlined above, one needs to check the collision probability of *each* particle in *every* time step using the computationally expensive mathematical expression (13). A more efficient selection of the colliding particles can be accomplished by the *null-collision* method [122, 130], where the *number* of colliding particles is given as $N_{coll} = NP_{coll}^*$, where *N* is the number of particles and $P_{coll}^* = \exp(-\nu^*\Delta t)$. Here, $\nu^* = \max\{n\sigma_{tot}g\}$ is the maximum collision frequency over the domains of interest of the parameters. For the particles chosen to collide, the process is selected as above, with the addition of a null-collision process. In the case of the occurrence of this latter process, the particle proceeds without any change of its velocity. For simplicity, we do not implement this method in the eduPIC code, but suggest this to readers

who would like to optimize and extend the code (see also section 5).

When the actual process is selected, the next task is to compute the post-collision velocity of the projectile. This is accomplished in several steps as follows.

Collision events are treated in the *center-of-mass* (COM) frame. In the laboratory (LAB) frame, the center of mass moves with the velocity

$$\mathbf{w} = \frac{m_1\mathbf{v}_1 + m_2\mathbf{v}_2}{m_1 + m_2}, \qquad (15)$$

where m_1 and m_2 are the masses of the collision partners. Equations (14) and (15) define two new variables from the 'original' two. These equations can easily be inverted both for the pre-collision velocities and for the post-collision velocities. For us, only the latter are interesting:

$$\mathbf{v}_1' = \mathbf{w}' + \frac{m_2}{m_1 + m_2}\mathbf{g}', \qquad (16)$$

$$\mathbf{v}_2' = \mathbf{w}' - \frac{m_1}{m_1 + m_2}\mathbf{g}', \qquad (17)$$

where $\mathbf{w}'$ and $\mathbf{g}'$ are the velocity of the COM and the relative velocity after the collision, respectively. The elementary classical treatment of the two-body interaction has two important conclusions: (i) due to the conservation of momentum, the velocity of the COM does not change during the encounter (i.e. $\mathbf{w}' = \mathbf{w}$), and (ii) the collision results in a change of the direction of $\mathbf{g}$ only in the case of elastic collisions (i.e. in this case $g' = g$), while in inelastic collisions the magnitude of $\mathbf{g}$ changes as well due to the loss of kinetic energy.

To execute the scattering, first we have to find the Euler angles θ and φ that define the Cartesian components of the relative velocity $\mathbf{g}$ before the collision (see figure 7(a)):

$$\mathbf{g} = \mathbf{v}_1 - \mathbf{v}_2 = \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} = g \begin{bmatrix} \cos\theta \\ \sin\theta\,\cos\varphi \\ \sin\theta\,\sin\varphi \end{bmatrix}. \qquad (18)$$

As the next step, we transform $\mathbf{g}$ to point in the *x* direction (see figure 7(b)); this can be achieved via two consecutive rotations of the velocity vector: (i) first, by an angle $-\varphi$ around the *x* axis and (ii) next, by an angle $-\theta$ around the *z* axis. Let us denote the matrices describing these operations by $\mathbf{T}_x(-\varphi)$ and $\mathbf{T}_z(-\theta)$. The transformed velocity vector is thus:

$$\mathbf{g}_T = \mathbf{T}_z(-\theta)\mathbf{T}_x(-\varphi)\mathbf{g} = g \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \qquad (19)$$

Actually, this transformation does not need to be carried out, as one can *assume* that $\mathbf{g}_T$ points in the *x* direction. The inverse of this transformation will, however, be needed later on.

The deflection and the change of the magnitude of the velocity vector is defined by the type of the collision. In the present model, we consider elastic, excitation and ionization collisions for the electrons, as well as elastic collisions for the ions. The scattering of the electrons is assumed to be isotropic in all types of processes, while for the ions a differential cross section that comprises an isotropic and a backward scattering part is adopted, as advised in [131].
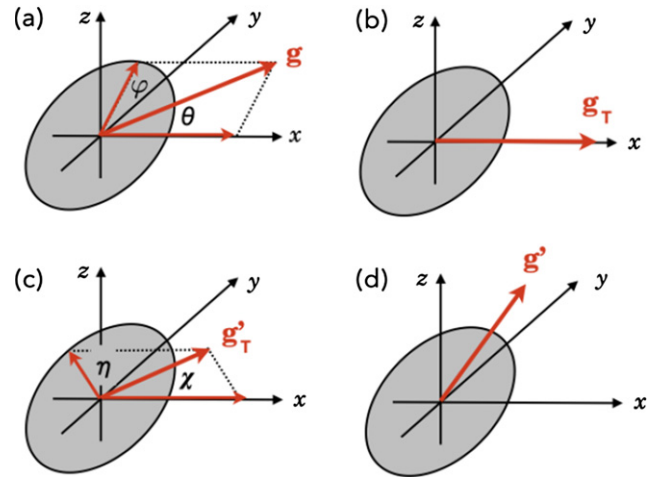

Figure 7. Steps of executing the scattering process.

In elastic collisions (both e$^-$ + Ar and Ar$^+$ + Ar), the magnitude of $\mathbf{g}$ is unchanged as dictated by the conservation of the total momentum. In this case, it is only the direction of $\mathbf{g}$ that has to be changed.

The deflection of $\mathbf{g}$ is defined by two angles: the scattering angle χ, and the azimuth angle η. The former has to be set according to the differential cross section of the scattering process, while the latter has a uniform distribution over the $[0, 2\pi)$ interval. These angles, in accordance with the Monte Carlo approach, are generated based on random numbers. For isotropic scattering, we compute χ as:

$$\chi = \arccos(1 - 2R_{01}), \qquad (20)$$

while for backward scattering:

$$\chi = \pi. \qquad (21)$$

The azimuth angle is set in all cases to be:

$$\eta = R_{01}2\pi. \qquad (22)$$

In case of inelastic collisions (that we consider for electrons only), the magnitude of the velocity vector has to be changed according to:

$$\frac{1}{2}\mu(g')^2 = \frac{1}{2}\mu g^2 - \Delta\varepsilon, \qquad (23)$$

where $\Delta\varepsilon$ is the excitation or ionization energy for the actual process and μ is the reduced mass of the projectile/target system.

In the case of ionization ($\Delta\varepsilon = \varepsilon_{\text{ion}}$), the scattered and ejected electrons share the remaining energy:

$$\varepsilon_s + \varepsilon_e = \varepsilon - \varepsilon_{\text{ion}}, \qquad (24)$$

where ε is the kinetic energy of the original projectile electron, ε_s and ε_e, respectively, are the energies of the scattered and ejected electrons. The partitioning between these latter two can be based on a distribution derived from experimental data [132]:

$$\varepsilon_e = w \tan\left[R_{01}\arctan\left(\frac{\varepsilon - \varepsilon_{\text{ion}}}{2w}\right)\right], \qquad (25)$$

where w is a characteristic parameter of the gas, in the case of Ar, $w = 10$. The scattering angles of the electrons are computed from

$$\cos \chi_s = \sqrt{\varepsilon_s / (\varepsilon - \varepsilon_{ion})}, \qquad \cos \chi_e = \sqrt{\varepsilon_e / (\varepsilon - \varepsilon_{ion})}, \tag{26}$$

whereas the azimuth angles are generated as

$$\eta_s = 2\pi R_{01}, \qquad \eta_e = \eta_s + \pi. \tag{27}$$

Having generated the scattering and azimuth angles, and having as well changed the magnitude of the velocity ($g_T \to g_T'$) due to the energy change, the velocity vector is deflected, as shown in figure 7(c):

$$\mathbf{g}_T' = g' \begin{bmatrix} \cos \chi \\ \sin \chi \cos \eta \\ \sin \chi \sin \eta \end{bmatrix}, \tag{28}$$

and is subsequently transformed back to the original coordinate system by the inverse rotation operations (see figure 7(d)):

$$\begin{aligned}
\mathbf{g}' &= \mathbf{T}_x(\varphi)\mathbf{T}_z(\theta)\mathbf{g}_T' \\
&= g' \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta \cos \varphi & \cos \theta \cos \varphi & -\sin \varphi \\ \sin \theta \sin \varphi & \cos \theta \sin \varphi & \cos \varphi \end{bmatrix} \\
&\quad \times \begin{bmatrix} \cos \chi \\ \sin \chi \cos \eta \\ \sin \chi \sin \eta \end{bmatrix},
\end{aligned} \tag{29}$$

which gives the final result for the post-collision velocity vector as:

$$\begin{aligned}
\mathbf{v}_1' &= \mathbf{w} + \frac{m_2}{m_1 + m_2} g' \\
&\times \begin{bmatrix} \cos \theta \cos \chi - \sin \theta \sin \chi \cos \eta \\ \sin \theta \cos \varphi \cos \chi + \cos \theta \cos \varphi \sin \chi \cos \eta - \sin \varphi \sin \chi \sin \eta \\ \sin \theta \sin \varphi \cos \chi + \cos \theta \sin \varphi \sin \chi \cos \eta + \cos \varphi \sin \chi \sin \eta \end{bmatrix}.
\end{aligned} \tag{30}$$

We note that for the e$^-$ + Ar collision we adopt the *cold gas approximation*, i.e. assume that the target Ar atoms are at rest. Correspondingly, $\mathbf{v}_2$ in equation (14) is set to zero. For the Ar$^+$ + Ar collisions, on the other hand, a random potential collision partner is sampled from the thermal distribution of the Ar atoms, before computing the collision probability. This sampling is aided by a random variable with normal distribution.

Finally, we would like to mention that the treatment of the collision processes as described above can be simplified in certain cases. When, e.g. one assumes isotropic scattering (as we actually do here), the determination of the Euler angles is not necessary, which makes the treatment simpler and the computation more efficient. Our intention with implementing the above procedures in the code was to provide a general treatment that would be applicable in the case of anisotropic scattering as well, which may be considered as an extension of the basic code (see section 5).

2.2. Accuracy and stability criteria

In order to have a stable simulation and to ensure that the results are accurate, a number of accuracy/stability conditions, which depend on the simulation settings, have to be fulfilled (see e.g. [107, 112]). These conditions are as follows:

(a) *The spatial grid has to resolve the electron Debye length,* i.e. $\Delta x / \lambda_D \lesssim 1$ should hold, where the Debye length $\lambda_D = \sqrt{\varepsilon_0 k_B T_e / n_e e^2}$ is evaluated with the effective electron temperature derived, e.g. from the mean electron energy;

(b) *The integration time step of the equations of motion has to be small enough to ensure that trajectories are computed accurately,* i.e. $\omega_{p,e} \Delta t_e < 2.0$ should hold. This condition is usually more restrictive for electrons, as electron oscillations represent the fastest movement in the system. In practice, a factor of 10 smaller time step is routinely used;

(c) *The collision probabilities should be sufficiently small,* in order to minimize the effect of 'missed' collisions (i.e. multiple collisions during a time step of the given species). It is a good practice to keep the collision probability of a given species (given in equation (13)) below $P_{coll} \cong 0.05$;

(d) *Particles should not fly a longer distance during a time step than the grid division, as expressed by the Courant–Friedrichs–Lewy condition* [133]. This condition states that $v_{max} \Delta t / \Delta x < 1$ should hold, where v_{max} is the maximum velocity of the particles. In practice, we loosen this condition by requiring it to hold up to a value $v_{max}^* < v_{max}$, where v_{max}^* corresponds to the energy, where the EEPF decays to a marginal value;

(e) *Accurate results require a high number of particles per grid cell.* Artificial numerical heating [98] appears in the system due to fluctuations, which are stronger when a smaller number of (super)particles is used. In an ideal PIC/MCC simulation, the results should not depend on the particle number (or on the weight, *W*). However, due to the above mentioned effect, this is not exactly the case, see e.g. [100, 101].

In practice, the first condition dictates the number of grid points, while the next three criteria define the magnitudes of the time steps for the electrons and ions. Which of them is most restrictive depends on the operating conditions.

The last condition in the list represents a real bottleneck in the PIC/MCC approach, the choice of the number of particles is always a compromise between accuracy and the execution time of the computer program.

3. Implementations

In the following subsections, we provide details about the basic simulation parameters (section 3.1), the cross sections used (section 3.2), random number generation (section 3.3), as well as about code compilation and execution (sections 3.4 and 3.5, respectively). In section 3.6, a detailed explanation of the structure of the 'basic' version of the code will be given, which has been written in C, with minimum C++ extensions. The

Table 1. Dictionary of the notations of some physical quantities and simulation parameters, which appear in the theoretical part (section 2) and in the parts of the code discussed in this section. (Following the conventions of the C language, constants appear in capital letters.)

Quantity	Notation	In the code
Constants:		
Voltage amplitude	V_0	VOLTAGE
Excitation frequency	f	FREQUENCY
Electrode gap	L	L
Gas pressure	p	PRESSURE
Gas temperature	T	TEMPERATURE
Number of grid points	N_g	N_G
Division of spatial grid	Δx	DX
Number of time steps	N_t	N_T
Electron time step	Δt_e	DT_E
Ion time step	Δt_i	DT_I
Subcycling ratio	N_s	N_SUB
Superparticle weight		WEIGHT
Number of entries in cross section tables		CS_RANGES
Energy resolution of cross section tables		DE_CS
Energy resolution of the EEPF		DE_EEPF
Energy resolution of the IFEDF		DE_IFED
Size of particle coordinate arrays	W	MAX_N_P
Variables:		
Potential	$\Phi(x)$	pot[ ]
Electric field	$E(x)$	efield[ ]
Electric charge density	$\rho(x)$	rho[ ]
Number of electrons	N_e	N_e
Positions of electrons	x_e	x_e[ ]
Electron velocity vector	$\mathbf{v}_e$	vx_e[ ],vy_e[ ],vz_e[ ]
Electron density	$n_e(x)$	e_density[ ]
Number of positive ions	N_i	N_i
Positions of ions	x_i	x_i[ ]
Ion velocity vector	$\mathbf{v}_i$	vx_i[ ],vy_i[ ],vz_i[ ]
Positive ion density	$n_i(x)$	i_density[ ]
Electron impact total cross section	$\sigma_e(\varepsilon)$	sigma_tot_e[ ]
Ion impact total cross section	$\sigma_i(\varepsilon)$	sigma_tot_i[ ]

latter include the use of constant declarations and the utilization of the random number generators (RNGs) from the C++ standard library. Despite the use of these C++ features, we refer to the basic code as a 'C code' throughout this paper. As already stated earlier, this version of the code is optimized for transparency and easy readability. Subsequently, more sophisticated versions of the code are also discussed, in sections 3.7 and 3.8, which have been written, respectively, in C++ and Rust languages.

When discussing the implementation details of the PIC/MCC approach in the eduPIC code we use the notations for the quantities as they appear in the codes. Table 1 presents a 'dictionary' between these notations and those used in the previous parts of the paper.

3.1. Basic simulation settings

The code simulates a CCP with conducting electrodes, placed at a distance L from each other. The space between them is filled with the background gas that has a spatially uniform density defined by the gas pressure (PRESSURE) and the temperature (TEMPERATURE). The electrode gap, including the boundaries, comprises N_G grid points, the spacing between the grid points is DX. For computational reasons, the inverse of this quantity, INV_DX is also defined, and in cases when a division by DX is required in the code, a more efficient multiplication operation by INV_DX is executed instead.

By default, one of the electrodes is driven by a cosine waveform characterised by the parameters VOLTAGE and FREQUENCY. The excitation waveform is specified within the solve_Poisson() function. One period of the RF excitation is divided into N_T time steps of length DT_E, which is the base time step for handling electrons. A longer time step DT_I is defined for the ions via the subcycling parameter N_SUB.

All input parameters (specifying the geometry, the driving waveform, the time steps, etc), as well as all the physical constants are specified in SI units in the code, with the exception of the constants that define the energy resolution of the EEPF and the IFEDF.
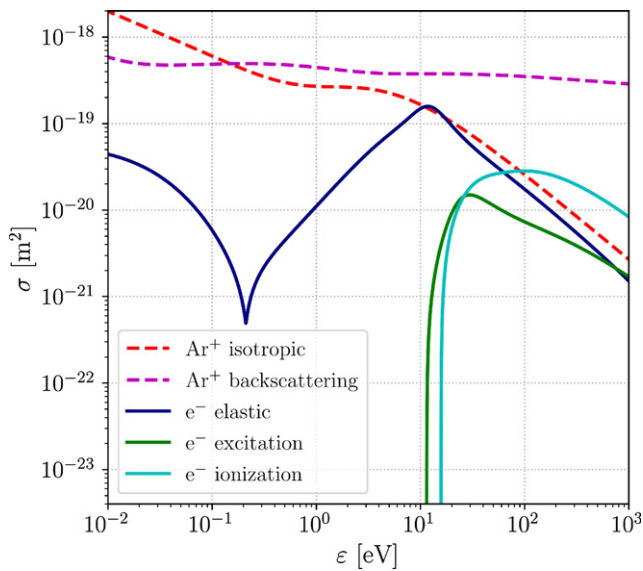
Figure 8. Cross sections for the collision processes of electrons (solid lines) and Ar$^+$ ions (dashed lines) considered in the eduPIC code. For the ions, ε is the COM energy.

Particle coordinates reside in static arrays. For the electrons, e.g. `x_e[ ]` stores the positions, while `vx_e[ ]`, `vy_e[ ]`, and `vz_e[ ]` store the Cartesian velocity components. The size of these arrays in the code is set by the constant `MAX_N_P`, which has a predefined value of 1 000 000, which is likely to be sufficient for applications of this code. Similar arrays are used for the coordinates of the ions.

3.2. Cross sections

The simulations here are performed in argon gas. The species in the simulation are electrons and Ar$^+$ ions. The electron impact cross sections are adopted from [134]. This set includes the elastic momentum transfer cross section, one excitation cross section that represents the sum of all excitation cross sections, and the ionization cross section. All e$^-$+ Ar collisions are assumed to result in isotropic scattering. In the case of Ar$^+$ + Ar collisions, only elastic collisions are considered. The cross section set for the ions includes an isotropic scattering part, as well as a backward scattering part (that corresponds to charge exchange), as advised in [131] (see also the discussion in section 2.1.6). These cross sections are displayed in figure 8.

The cross sections in [131, 134] are given by analytic forms. To avoid the need for evaluating these expressions at frequent times when cross section values are needed during the simulation, the functions `set_electron_cross_sections_ar()` and `set_ion_cross_sections_ar()` tabulate the cross sections at the beginning of the run. The data are stored in the array `sigma` for electrons and ions, consecutively, with an energy resolution specified by the constant parameter `DE_CS` (being equal to 0.001 eV as default).

For the electrons, the array stores the cross section values as a function of the LAB energy, while for the ions, the data are stored as a function of the COM energy. In [131], the data for the ions are given as a function of the laboratory energy, and conversion of the energy is done in the function `set_ion_cross_sections_ar()`. The function `calc_total_cross_sections()` computes the values of $n\sigma_{tot}$, as these are needed for the evaluation of the collision probabilities. These data are stored in the arrays `sigma_tot_e` and `sigma_tot_i`. The constant `CS_RANGES` specifies the size of the cross section arrays. (It has to be kept in mind that `DE_CS`*`CS_RANGES` should exceed the maximum particle energies expected.) During runtime, cross section values are found quickly from these lookup tables.

3.3. Random number generation

A central component of every Monte Carlo method based simulation is the pseudo-random number generator. We do not know about a rigorous derivation of the statistical requirements the RNG has to fulfill in order to guarantee the accuracy of the PIC/MCC scheme. Therefore, we rely on our empirical experience and general quality measures. The use of the basic RNG, included in the tested implementations of the C Standard General Utilities Library `stdlib.h`, was found to result in nonphysical distributions. For this reason, we utilize the Mersenne Twister 19 937 generator, included in the C++ standard library (beginning with version C++11) in the C and C++ versions of the code. The very same RNG is used to generate random samples from (i) the uniform distribution on the [0, 1) interval for general purposes, and (ii) the normal distribution used to generate random velocity components of background gas atoms in thermal equilibrium.

Although the Mersenne Twister RNG is available in Rust in an external library (or 'crate' as called in Rust terminology), but due to its limited functionality we have chosen to use the `ThreadRng`, which is a thread-local, cryptographically secure RNG based on the HC-128 algorithm [135].

3.4. Compilation

Our codes have been tested and benchmarked on an Ubuntu Linux computing cluster on nodes equipped with x86-64 based Intel® Xeon and AMD® Threadripper CPUs. The compilation of the C and C++ codes have been performed with both `icpc` Intel C++ compiler (ver. 2021.1.2, part of the freely available Intel oneAPI Base Toolkit) and `g++` from the GNU compiler collection (ver. 9.3.0). Best performance was achieved on Intel architecture using:

```
icpc -fast -o edupic eduPIC.cc
```

The same compilation line can be used with the C++ source file `eduPIC.cpp` as well. The Rust language compiler is best executed using the integrated package manager `cargo` with the option `build`. At default, the compiler creates a 'Debug' version of the program. To build the final 'Release' version with high level of compile time optimization turned on, the compile line.

```
cargo build --release
```

can be called from within the project folder.

3.5. Program execution and files created

Before executing the code, the simulation parameters have to be set in the source code, which then has to be compiled as explained above. Subsequently, the code can be invoked in a terminal window as

```
./edupic arg1
```

To start a new simulation, running an initialization cycle is required by setting `arg1` to 0, i.e. executing

```
./edupic 0
```

In this case, seeding of a number of initial particles, the simulation of a single RF cycle and saving of the state of the system is executed; further details are given in algorithm 1 of section 3.6.

The code can be run, subsequently, for any number of additional RF cycles specified by `arg1`, e.g. for 500 cycles, as

```
./edupic 500
```

In this run, the previously saved state of the system is restored, the given number of RF cycles is simulated, and at the end of the run, the state of the system is saved. (Please refer to algorithm 1 of section 3.6 for more details.) This procedure can be repeated any number of times, the simulation always continues from the previously saved state of the system. During these runs, the time evolution of the number of superparticles is stored in `conv.dat`, but no other data (results) are saved. Measurements on the system can be activated by specifying a second command line argument `m` when the code is invoked, e.g. for 1000 RF cycles, as

```
./edupic 1000 m
```

Here all built-in diagnostics are turned on. The number of RF cycles for which measurements are run affects the quality of the statistics of the results. Therefore, we recommend using ∼1000 RF cycles to obtain results with good signal to noise ratio. When measurements on the system are taken, the code saves the data into the following files:

- `density.dat`: time-averaged density distributions of electrons (second column) and ions (third column) as a function of the position (first column);
- `eepf.dat`: time-averaged EEPF in the central 10% spatial domain of the discharge (second column) as a function of the energy (first column). The data are normalized corresponding to (the discretized form of) $\int f(\varepsilon)\sqrt{\varepsilon}\mathrm{d}\varepsilon = 1$;
- `ifed.dat`: time-averaged IFEDF at the powered (second column) and grounded (third column) electrode as a function of the energy (first column). The data are normalized corresponding to (the discretized form of) $\int F(\varepsilon)\mathrm{d}\varepsilon = 1$;
- `pot_xt.dat`: spatio-temporal distribution of the potential;
- `efield_xt.dat`: spatio-temporal distribution of the electric field;
- `ne_xt.dat`: spatio-temporal distribution of the electron density;
- `ni_xt.dat`: spatio-temporal distribution of the ion density;

- `je_xt.dat`: spatio-temporal distribution of the electron current density;
- `ji_xt.dat`: spatio-temporal distribution of the ion current density;
- `powere_xt.dat`: spatio-temporal distribution of the power absorption by the electrons;
- `poweri_xt.dat`: spatio-temporal distribution of the power absorption by the ions;
- `meanee_xt.dat`: spatio-temporal distribution of the mean electron energy;
- `meanei_xt.dat`: spatio-temporal distribution of the mean ion energy;
- `ioniz_xt.dat`: spatio-temporal distribution of the ionization rate.

The number of the energy bins as well as the resolution of the EEPF and IFEDF are set by the constants `N_EEPF`, `DE_EEPF` and `N_IFED`, `DE_IFED`.

The 'xt files' that store the spatio-temporal variation of the given quantity contain a specific number of rows related to the number of grid cells `N_G`. The number of columns represents the temporal resolution. In order to improve the signal to noise ratio of the data, the number of time steps per RF cycle `N_T` is binned to a lower number `N_XT`. The number of time steps binned for the xt files can be set by the variable `N_BIN` in the code. At this point, care should be taken to ensure that (i) N_T is an integer multiple of N_BIN and that (ii) N_BIN is an integer multiple of N_SUB. This latter requirement ensures that information is gathered into each element of the XT matrices for the same number of ion time steps. All the output data are in SI units, except for the mean electron energy, as well as for the EEPF and the IFEDF, for which eV units are used.

Following a run in the 'measurement' mode, the code saves a file named `info.txt`. This file contains the operation parameters and simulation settings as a record of the run. The file also displays information about the stability and accuracy conditions. First, the conditions (a)–(c) (see section 2.2) are checked. These concern the relation of the grid spacing to the Debye length, the relation of the time step to the electron plasma frequency, and the collision probabilities during a time step. Whenever any of these checks fail, an error message is issued and no further diagnostics data are saved.

We note that the evaluation of the stability and accuracy criteria is meaningful only when measurements are taken over the converged state. The initial setting of the simulation parameters (like the grid size and the time steps) is normally based on an educated guess, these parameters can be refined later according to the information contained in `info.txt`. In case of need of modifications of the parameter settings, the code has to be re-compiled, the simulation has to be re-converged, and the stability and accuracy criteria have to be checked again.

In case the above conditions are met, the diagnostics data listed above are saved to the respective files, and the maximum electron energy for which the Courant–Friedrichs–Lewy condition (condition (d) in the list of section 2.2) still holds at the actual values of N_G and N_T, is also displayed in

Algorithm 1. The `main()` function of the program. When the code is started with a command line argument `arg1 = 0`, a given number (`N_INIT`) of electrons and Ar$^+$ ions are seeded within the gap at random positions and with zero initial velocities. Then, the simulation of a single RF cycle is executed by the `do_one_cycle()` function. Following this, the number of the particles and their coordinates are saved to the `picdata.bin` binary file. When the code is invoked with an integer command line argument `arg1 > 0`, the code simulates further `arg1` RF cycles. This run starts with loading the state of the system from the file `picdata.bin`. Then, `do_one_cycle()` is called `arg1` times, and finally the state of the system is saved again to `picdata.bin`. During these runs, the time evolution of the number of superparticles is stored in the file `conv.dat`, but no additional measurements on the system are being conducted. When a second command line argument `m` is specified, measurements on the system are activated and the data acquired are saved to data files (as described above).

```
// get an integer parameter from command line
    that controls code execution and number of
    RF cycles to be simulated:
strcpy(st0, argv[1])
arg1 = atol(st0)
// activate measurement mode if m is given as
    the second command line argument:
if argv[2] = m then
 |  measurement_mode = true
else
 |  measurement_mode = false
end
// call functions to compute and tabulate the
    cross sections:
set_electron_cross_sections_ar()
set_ion_cross_sections_ar()
calc_total_cross_sections()
// perform the simulation of a given number of
    RF cycles:
if arg1 = 0 then
    // perform an initialization cycle:
    no_of_cycles = 1
    init(N_INIT)        // seed initial particles
    Time = 0            // initialize simulation time
    do_one_cycle() // simulate one RF cycle
    cycles_done = 1
else
    no_of_cycles = arg1    // number of RF cycles
     to simulate
    load_particle_data()      // load the state of
     the system
    // simulate the given number of RF cycles:
    for 0 ≤ i < no_of_cycles do
     |  do_one_cycle()
    end
    cycles_done += no_of_cycles
end
save_particle_data()      // save the state of the
 system
// save results when measurements are
    activated:
if measurement_mode then
    save_density()
    save_eepf()
    save_ifed()
    save_all_xt()
    check_and_save()
end
```

Algorithm 2. PIC/MCC step 1—computation of the charged particle densities. At the beginning of each time step, the electron density is set to zero at the grid points. Then, for each of the electrons, the density at the two grid points which enclose the given electron, is incremented. Finally the electron density at the two edge points of the grid is multiplied by two, because the volume that belongs to these points is the half of that compared to the other grid points.

```
FACTOR_W = WEIGHT/DV     // scaling factor
  for the density
for 0 ≤ p < N_G do
 |  e_density[p] = 0
end
for 0 ≤ k < N_e do
    c0 = x_e[k] * INV_DX
    p = (int)(c0)                  // grid index
    e_density[p]+ = (p + 1 − c0) * FACTOR_W
    e_density[p + 1]+ = (c0 − p) * FACTOR_W
end
e_density[0]* = 2.0
e_density[N_G − 1]* = 2.0
```

Algorithm 3. PIC/MCC step 2—computation of the potential and the electric field. The discretized Poisson equation can be written in a matrix form with a tridiagonal matrix of constant coefficients. The equation is solved using the Thomas algorithm [136]. The electric field values at the grid points are also derived within the `solve_Poisson()` function.

```
for 0 ≤ p < N_G do
 |  rho[p] = E_CHARGE * (i_density[p] − e_density[p])
end
solve_Poisson(rho,Time)
// calculation of the potential and the
    electric field at the grid points
```

Algorithm 4. PIC/MCC steps 3 and 4—computation of the forces acting on the particles and moving the particles. For each electron, first, the velocity is updated according to the electric field interpolated to its position from the values at the grid points that enclose the given electron. Subsequently, the position is updated according to the leapfrog scheme.

```
FACTOR_E = DT_E / E_MASS * E_CHARGE
  // scaling factor for the change of velocity
for 0 ≤ k < N_e do
    c0 = x_e[k] * INV_DX
    p = (int)(c0)                   // grid index
    c1 = p + 1 − c0       // linear interpolation
     factors
    c2 = c0 − p
    e_x = c1 * efield[p] + c2 * efield[p+1]
    vx_e[k]− = e_x * FACTOR_E
    x_e[k]+ = vx_e[k] * DT_E
end
```

`info.txt`. To make sure that this condition holds for the *vast majority* of the electrons, one has to confirm that the EEPF decays to a small value at this energy. This is not done automatically in the code, the procedure is left to the user. We advise to observe the EEPF obtained in the center of the plasma, and to use a threshold value of $f(\varepsilon) \sim 10^{-6}$ eV$^{-3/2}$. For a more rigorous check, the complete space- and time-resolved EEPF should be considered.

Additional information about the particle characteristics at the electrodes and about the power absorption by the elec-

Algorithm 5. PIC/MCC step 5—adding/removing particles at the boundaries. Electrons whose updated position is outside the electrode gap are identified and removed. In such cases, the Boolean variable out (originally set to false) is set to true, the counter of the number of absorbed electrons at the given electrode is incremented, and the electron is removed from the array that stores the electron coordinates. Finally, N_e is decremented. For the ions (not shown here), the energy of the ion upon its impact at the electrode is used for building the IFEDF. Recall that we do not consider the reflection and emission of particles at/from the electrode surfaces.

```
k = 0
while k < N_e do
    // cycle over all electrons:
    out = false
    if x_e[k] < 0 then
        N_e_abs_pow++
        out = true
    end
    if x_e[k] > L then
        N_e_abs_gnd++
        out = true
    end
    if out then
        // remove electron:
        x_e[k] = x_e[N_e − 1]
        vx_e[k] = vx_e[N_e − 1]
        vy_e[k] = vy_e[N_e − 1]
        vz_e[k] = vz_e[N_e − 1]
        N_e − −
    else
        k++
    end
end
```

trons and ions is also saved to `info.txt` at the end of the simulation. The latter is computed as the spatial and temporal average of the product $j(x, t)E(x, t)$. This product is also saved to the files `powere_xt.dat` and `poweri_xt.dat` (corresponding to the power absorption rate by the electrons and ions, respectively) with spatial and temporal resolution.

3.6. Basic (C) code

Below, a detailed explanation of the structure and the operation of the C code is given at the level of algorithms. Algorithm 1 presents the `main()` function of the simulation code. Subsequently, in algorithms 2–6, details of the six steps of the PIC/MCC cycle (see section 2.1) are presented, which are parts of the `do_one_cycle()` function. In algorithms 2–5, the calculation routines are only shown for the electrons. Similar steps are applied for the positive ions, however, these are only performed every N_SUB_th time step as ions move slower in the plasma. By this subcycling method, remarkable computation time is gained without decreasing the accuracy of the calculation. In algorithm 6, though, which shows the collision routine, the steps applied for ions are also shown, since they are considerably different from the ones applied for electrons.

Measurements of the relevant physical quantities (listed above) are carried out at two points within the `do_one_cycle()` function. Accumulation of the data for the quantities that are defined at the grid points (potential, electric field, electron and ion density) is performed following

Algorithm 6. PIC/MCC step 6—checking and executing collisions. First, the collisions of the electrons are considered. Subsequently, the collisions of ions are handled, however, only in every N_SUB_th time steps, in accordance with ion subcycling. (t is the loop variable for the time steps within an RF cycle). For the ions, the potential collision partners are sampled from the thermal distribution of the background gas atoms.

```
for 0 ≤ k < N_e do
    // calculate the kinetic energy of the
        electron:
    v_sqr = vx_e[k] * vx_e[k] + vy_e[k] * vy_e[k]
        + vz_e[k] * vz_e[k]
    velocity = sqrt(v_sqr)
    energy = 0.5 * E_MASS * v_sqr/EV_TO_J
    // obtain the collision frequency of the
        electron with the background gas:
    energy_index = min( int(energy / DE_CS +
        0.5), CS_RANGES − 1) // variable indexing
        the energy bin the electron belongs to
    nu = sigma_tot_e[energy_index] * velocity
    // calculate the collision probability for
        the electron:
    p_coll = 1 − exp(−nu * DT_E)
    // perform electron-atom collision with the
        probability obtained, and increase the
        collision counter:
    if R01(MTgen) < p_coll then
        collision_electron(x_e[k], vx_e[k], vy_e[k],
            vz_e[k], energy_index)
        N_e_coll++
    end
end
if (t mod N_SUB) = 0 then
    for 0 ≤ k < N_i do
        // draw velocity components of a random
            target gas atom from the
            Maxwell-Boltzmann distribution:
        vx_a = RMB(MTgen)
        vy_a = RMB(MTgen)
        vz_a = RMB(MTgen)
        // calculate the relative velocity of
            the collision partners and the
            kinetic energy of the projectile:
        gx = vx_i[k] − vx_a
        gy = vy_i[k] − vy_a
        gz = vz_i[k] − vz_a
        g_sqr = gx * gx + gy * gy + gz * gz
        g = sqrt(g_sqr)
        energy = 0.5 * MU_ARAR * g_sqr/EV_TO_J
        // obtain the collision frequency of the
            ion with the background gas:
        energy_index = min( int(energy / DE_CS +
            0.5), CS_RANGES − 1)        // variable
            indexing the energy bin the ion belongs
            to
        nu = sigma_tot_i[energy_index] * g
        // calculate the collision probability
            for ions:
        p_coll = 1 − exp(−nu * DT_I)
        // perform ion-atom collision with the
            probability obtained, and increase
            the collision counter:
        if R01(MTgen) < p_coll then
            collision_ion (vx_i[k], vy_i[k], vz_i[k],
                vx_a, vy_a, vz_a, energy_index)
            N_i_coll++
        end
    end
end
```

the six steps of the PIC/MCC cycle. Data for some other quantities is accumulated in the part of the code where the particles are moved. The reason for this is that the positions and the velocities of the particles are not defined at the same time (recall that positions are defined at integer time steps, while position at half-integer time steps, see section 2.1). For an accurate calculation of, e.g. the power absorption by the particles their velocities have to be known at integer time steps as well. This is solved in the code by computing the particle velocities at an intermediate time as well (i.e. at the same time when the positions are known) and the measured quantities (e.g. mean velocity, mean energy) are evaluated with these average velocities. The measured quantities are updated at cells of two-dimensional data structures ('xt arrays') with a similar interpolation like the one used at the density assignment of the particles to the grid points. The details of these calculations are omitted from the forthcoming discussion of the algorithms of the PIC/MCC cycle.

3.7. C++ code

The C programming language falls into the category of 'procedural imperative' languages, in which the programmer instructs the machine how to change its state and instructions are grouped into procedures [137]. Representing one of the simplest programming paradigms [138] it is best suited for short codes and in cases where tight control over the hardware resources and timing is necessary. For a long time, its low level 'close to metal' nature made it popular (as an alternative to Fortran) in high performance computing applications, where the codes are optimized to perform one particular computational task.

With the introduction of the C++ programming language, another level of abstraction was added realizing the object oriented programming paradigm, which groups instructions with the part of the state they operate on. This pawed the way to the construction of large, multi-developer and multiple purpose programs, like modern operating systems and office program suites. Clearly, for the purposes of this study and the small size of the eduPIC code the programming overhead of object oriented implementation would not pay off, but it is a logical direction for further development targeting, e.g. a more complicated gas composition and chemistry. Another component of modern C++, as maintained by the 'Standard C++ Foundation' is the 'standard library' (std), a large collection of functions, types and objects of which the functionality is strictly defined but the implementations do continuously evolve and are optimized to a variety of hardware architectures.

The C++ implementation of the eduPIC code closely follows the structure of the C version but utilizes modern data structures, like std::array-s and std::vector-s, as well as a variety of vector transformation standard library routines. To illustrate the difference, let us focus on the first for loop in algorithm 3, the simple calculation of the charge density from the particle density distributions: in the C code, (i) a temporary variable p had to be declared and

initialized, (ii) the limit and increment of the loop had to be set, (iii) the input values had to be fetched from memory addresses calculated from the head-of-array addresses and the offsets given by p, (iv) the result of the calculation had to be directed to a specific memory address inside the output array. All these elementary steps had to be defined in the program explicitly. The disadvantage of having to define so many details is twofold: (i) errors can be introduced at any of these steps, (ii) by defining each elementary step and with this the order of execution, there remains very limited room for the compiler to perform optimization to the target hardware architecture. This latter point becomes increasingly important taking into account that the development of computing hardware in the last decade increasingly utilized multiple levels of parallelism (e.g. vector registers, multiple cores in CPUs, multiple CPUs per node, as well as massively parallel GPU acceleration options) at an increasing pace. Replacing the for loop by the std::transform function improves on both points, less details have to be defined and the best performing machine level code can be chosen by the compiler on the actual hardware architecture, including optional asynchronous parallel versions, if proper execution policies are applied. Similar statements can be formed about many other parts of the code, which are, however, not discussed further here.

3.8. Rust code

The Rust-language is a modern open-source, community-developed programming language maintained by the 'Rust Core Team' with its first stable release published in 2015. Rust is a multi-paradigm programming language, syntactically similar to C++, designed for performance and memory/thread safety [139]. The advantage of using a new but mature programming language is that one can utilize all successful modern tools without the bounds set by backward compatibility requirements to already superseded old concepts. Similarly to C++, all modern data structures like dynamic vectors, as well as a rich standard library are available. The Rust compiler (rustc) and package manager (cargo) provide user friendly error messages and effortless download, integration and version control of code libraries. The default behavior of the Rust language elements are tuned for best performance and safety. As an example, variables by default are immutable objects, meaning that after declaration their value can be initialized once during run-time and are read-only from there on within their scope (in contrast to constants, where the values have to be known already at compile-time). This introduces obvious optimization possibilities to the compiler and during execution, like cache memory management. Regarding the safety concerns, the handling of arrays and associated pointers are strict, making it virtually impossible to accidentally read or write out of bounds and trigger segmentation faults, or even worse, undefined behavior of the program.

Despite performing the same computations as the base C version, the Rust implementation of the eduPIC simulation features some adaptations in order to exploit certain features

of the language. Particle data (position and velocity components) are stored in a vector-of-structures dynamic container making particle management (adding and removing particles) easy and safe in contrast to the base code, where no intrinsic mechanism guarantees the consistency of the particle counters and the length of the independent position and velocity component arrays. Syntax wise a very appealing feature is that arithmetic function calls can be applied in an object-oriented style, which improves readability of complex expressions. Although in such a short and single source file code global variables do not represent safety hazards, following the general concept mostly relevant to more complex projects, we have omitted global variables from the Rust code.

The execution of the Rust version of the simulation can be performed by calling the executable file as described in section 3.5, or using the package manager `cargo` as

```
cargo run --release 1000 m
```

In this case the highly optimized 'release' version with arguments `1000` (number of RF cycles to simulate) and `m` (measurements turned on) will be executed after additional package download and code compilation steps, if needed to build the executable file, are performed.

The three implementations (C, C++ and Rust) described above, each utilizing different levels of abstractions and modern programming language features, are available to the interested reader [140]. All three implementations provide comparable execution times and equivalent simulation results of the computed physical quantities.

4. Results

In this section, first we present some 'reference results' that were obtained with the eduPIC code with a set of 'default' discharge parameters (section 4.1). These results are useful for demonstrating the capabilities of the code, as well as the basic operation characteristics of the simple CCP considered here. Subsequently, we address a physics problem, which can readily be studied without any extension of the code: in section 4.2 we illustrate the formation of the flux-energy distribution of Ar^+ ions at the electrode surfaces for the conditions of DF excitation. We note, that for all these calculations the number of superparticles was around 10^5, which normally provides acceptable accuracy, however, higher numbers are recommended for more accurate results [98, 100].

4.1. Reference results

Below, we illustrate the reference results obtained with our code. The discharge is driven by a single harmonic voltage waveform:

$$V(t) = V_0 \cos(2\pi f t). \tag{31}$$

V_0 and f, along with the other simulation parameters are listed in table 2. When downloaded, compiled, and executed, the code should reproduce these results with its 'default' parameters included in the source files. The following figures have been generated with Python and the corresponding plotting library Matplotlib.

Table 2. Set of 'default' parameters for the reference simulation run.

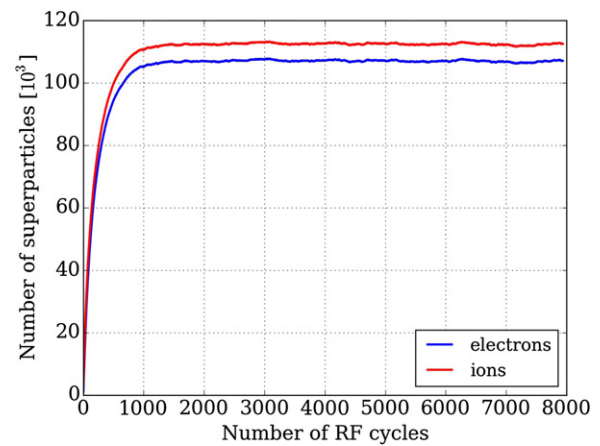| Quantity	Value
Driving voltage amplitude	$V_0 = 250$ V
Driving frequency	$f = 13.56$ MHz
Superparticle weight	$W = 7 \times 10^4$
Electrode gap	$L = 25$ mm
Ar pressure	$p = 10$ Pa
Gas temperature	$T_g = 350$ K
Number of grid points	$N_g = 400$
Number of time steps/RF cycle	$N_t = 4000$

Figure 9. The number of superparticles as a function of the number of RF cycles (datafile: `conv.dat`). Discharge conditions: argon $p = 10$ Pa, $L = 25$ mm, $T_g = 350$ K, $V = 250$ V and $f = 13.56$ MHz.

Figure 9 shows the dependence of the number of superparticles in the simulation on the RF cycles elapsed after the initialization of the simulation with 1000 initial superparticles of both species. One can see that convergence is reached after ∼1500 cycles. At this stage, the number of superparticles as a function RF cycles (or time) does not change anymore, it only fluctuates around the equilibrium value. During this initial phase no data for the plasma characteristics were collected except for the number of superparticles. An additional run for 1000 cycles was executed to collect the data shown in the subsequent figures. Recall, that the code does not consider any surface processes.

Figure 10 shows the temporally averaged electron and ion density as a function of space. This plot clearly illustrates the formation of a quasi-neutral plasma bulk in the center of the discharge and the two electron depletion regions close to both electrodes which are defined as the plasma sheaths.

In figure 11, we display the spatio-temporal distributions of several discharge characteristics: the density of the electrons and the ions (panels (a) and (b)), the potential and the electric field (panels (c) and (d)), the electron and ion current density (panels (e) and (f)), the power density absorbed by the electrons and the ions (panels (g) and (h)), the mean energy of the electrons and ions (panels (i) and (j)), as well as and the ionization source function (panel (k)).
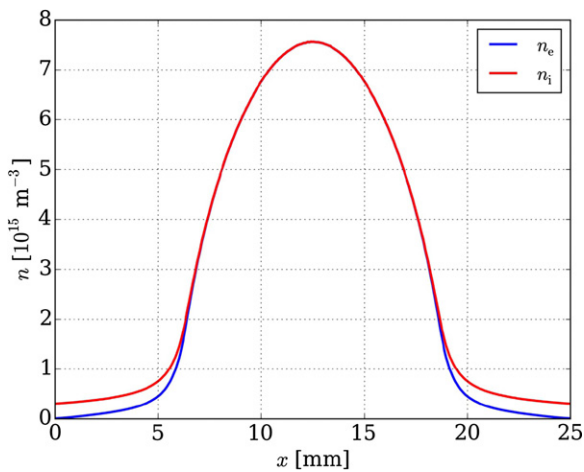
Figure 10. The temporally averaged electron (blue) and ion density (red) as a function of the spatial position between the electrodes (data file: `density.dat`). Discharge conditions: argon $p = 10$ Pa, $L = 25$ mm, $T_g = 350$ K, $V = 250$ V and $f = 13.56$ MHz.

Panels (a) and (b) clearly reveal the marked difference between the temporal dynamics of the electrons and the positive ions. While the spatial distribution of the ions is nearly time-independent, the motion of the electrons is governed by the temporal variation of the potential and the electric field (shown in panels (c) and (d), respectively). The electron density is depleted within the sheath regions that form near both electrodes with a shift of $T_{RF}/2$ (where $T_{RF} = 1/f$). The edge of the sheaths, found as advised in [141], is indicated by the gray lines in each panel. Figures 11(e) and (f) show the electron and ion current density, respectively. The electron current density is dominant in the plasma bulk (absolute values around 20 A m^{-2}) and almost zero within the sheath regions, due to electron depletion (see figure 11(a)). The ion current density increases from the centre of the discharge towards the electrodes from 0 to ± 0.4 A m^{-2}. The continuity of the total current is ensured by a significant displacement current (not shown) within the sheath regions.

The power absorption of the electrons (figure 11(g)) is prominent near the edges of the expanding sheaths. Upon sheath collapse, this quantity acquires negative values, corresponding to power loss. The 'vertical' features observable in this plot near the positions of the maximum extent of the sheaths show the effect of the ambipolar electric field (see e.g. [50]). The positive ions absorb power mostly within the sheath regions while they move towards the electrodes under the influence of the high electric field present in this domain (figure 11(h)). For these parameter settings the space- and time-averaged power density absorbed by the electrons is 1.58×10^3 W m^{-3}, while that absorbed by the ions is about a factor of two higher, 3.28×10^3 W m^{-3}.

As a consequence of the power absorption by the electrons in the vicinity of the edges of the expanding sheaths the mean energy of the electrons also exhibits a maximum in this domain of space and time (see figure 11(i)). Additional, smaller maxima of the mean energy are also observed in the vicinity of the electrodes, at times when the sheaths collapse; these are

caused by electric field reversals [142]. The mean energy of the electrons within the bulk plasma is nearly constant, around 1.5–2 eV. The mean energy of the ions, as illustrated in figure 11(j), is only slightly modulated in time and reaches maximum values of about 20 eV near both electrodes. The power absorption and the enhanced mean electron energy in the vicinity of the edges of the expanding sheaths [50, 51, 143], gives rise to a significant ionization (see figure 11(k)) within these times of the RF period, however, over a more extended spatial scale, as compared figure 11(g), due to the acceleration that the electrons experience towards the center of the discharge.

More information about the energy distribution of the electrons (via the EEPF, $f(\varepsilon)$, 'measured' at the central region of the plasma) and the ions (via their flux energy distribution, IFEDF, $F(\varepsilon)$, at the electrodes) is depicted in figures 12 and 13, respectively. The EEPF reveals a behavior that is expected in Ar CCPs at low pressures [144]. (In CCPs, the EEPF is known to exhibit a complex variation with position and time within the RF cycle. The average does not convey information about these changes, the investigation of which remains as future work of those readers who implement a straightforward extension of the eduPIC code to reveal this information.)

The IFEDF exhibits a series of peaks, characteristics for ion transport through the sheaths in the presence of charge-exchange collisions [145, 146].

The raw data obtained for this reference case can be found at [140]. We note that the three versions of the code generate identical results (within the limits of statistical fluctuations on the order of $\pm 0.3\%$ in the number of simulation particles), this is illustrated here by quoting the peak electron density values (which are results of simulations spanning 500 radiofrequency cycles in the converged state of the 'reference simulation case'):

- C code: $n_{e,max} = 7.55 \times 10^9$ cm^{-3},
- C++ code: $n_{e,max} = 7.52 \times 10^9$ cm^{-3},
- Rust code: $n_{e,max} = 7.51 \times 10^9$ cm^{-3}.

4.2. Ion properties and ionization dynamics in dual-frequency discharges

As it has already been mentioned in section 1, the control of the ion properties, viz their flux and mean energy is of utmost importance for controlling surface reactions in plasma processing applications [147–149]. Among the several approaches, the case of a voltage waveform that consists of two components with largely differing frequencies, i.e. the 'classical dual-frequency' (DF) excitation in geometrically symmetric reactors can readily be studied with our simple PIC/MCC code as the DC (direct current) self-bias voltage in such systems is negligible [150] and its self-consistent computation can therefore safely be omitted. Below, we show representative simulation results for argon discharges excited by waveforms

$$V(t) = V_{HF} \cos(2\pi V_{HF} t) + V_{LF} \cos(2\pi V_{LF} t), \quad (32)$$

where the lower frequency (LF), f_{LF}, is fixed at 2 MHz, and the higher frequency (HF), f_{HF}, is in the range of 10–40 MHz. The respective voltage amplitudes, V_{LF} and V_{HF}, and the additional
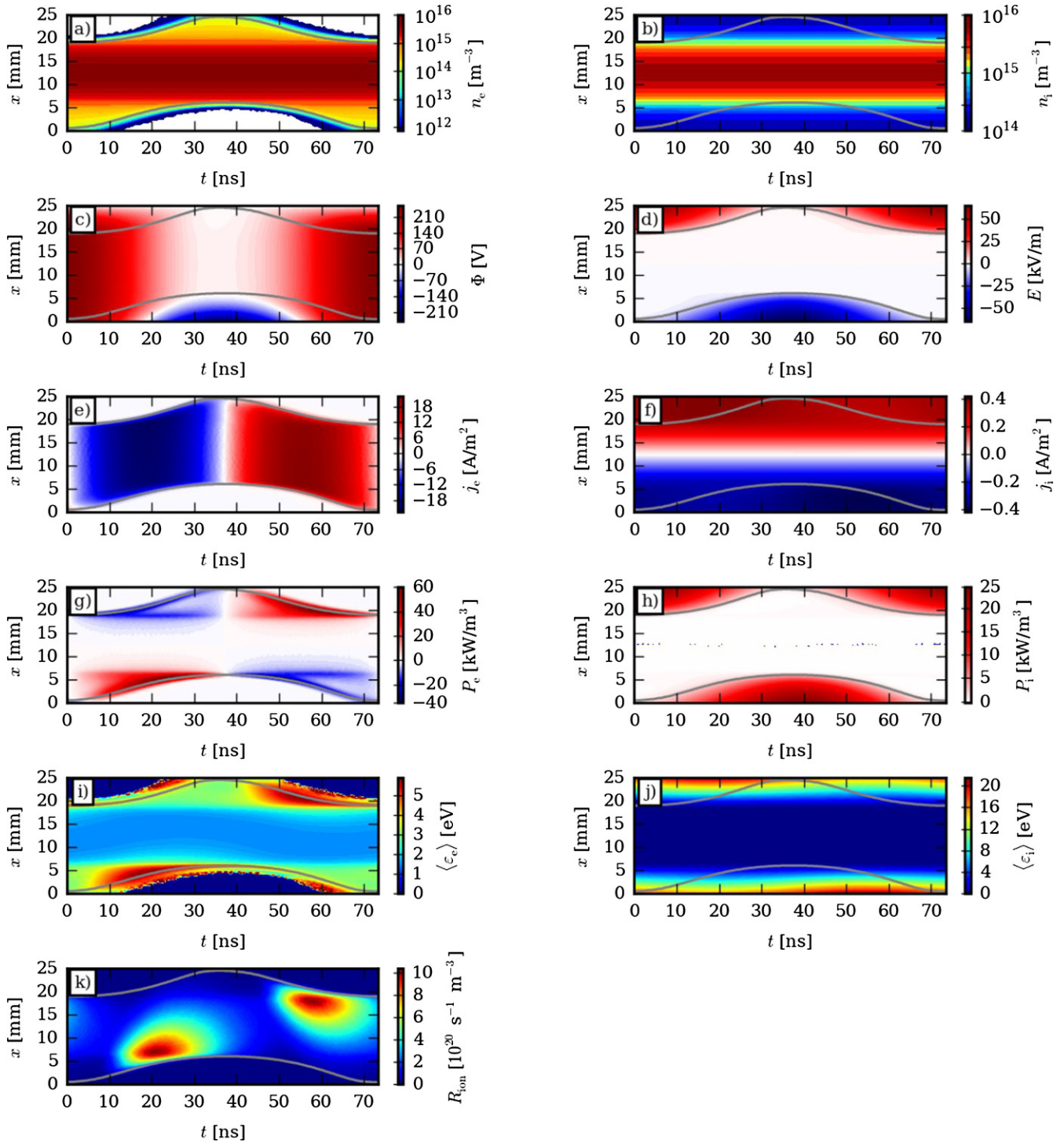
Figure 11. Spatially and temporally resolved distributions of (a) the electron density, n_e, in m^{-3} (data file: `ne_xt.dat`), (b) the ion density, n_i, in m^{-3} (data file: `ni_xt.dat`), (c) the potential, Φ, in V (data file: `pot_xt.dat`), (d) the electric field, E, in kV m^{-1} (data file: `efield_xt.dat`), (e) the electron current density, in A m^{-2} (data file: `je_xt.dat`), (f) the ion current density, in A m^{-2} (data file: `ji_xt.dat`), (g) the electron power density, in kW m^{-3} (data file: `powere_xt.dat`), (h) the ion power density, in kW m^{-3} (data file: `poweri_xt.dat`), (i) the mean electron energy, $\langle \varepsilon_e \rangle$, in eV (data file: `meanee_xt.dat`), (j) the mean ion energy, $\langle \varepsilon_i \rangle$, in eV (data file: `meanei_xt.dat`) and (k) the ionization rate, R_{ion}, in s^{-1} m^{-3} (data file: `ioniz_xt.dat`). The horizontal axis shows the time for one RF cycle and the vertical axis shows the distance from the powered electrode. The gray solid lines indicate the plasma sheath edge, calculated using the formula given in [141]. Discharge conditions: argon $p = 10$ Pa, $L = 25$ mm, $T_g = 350$ K, $V = 250$ V and $f = 13.56$ MHz.
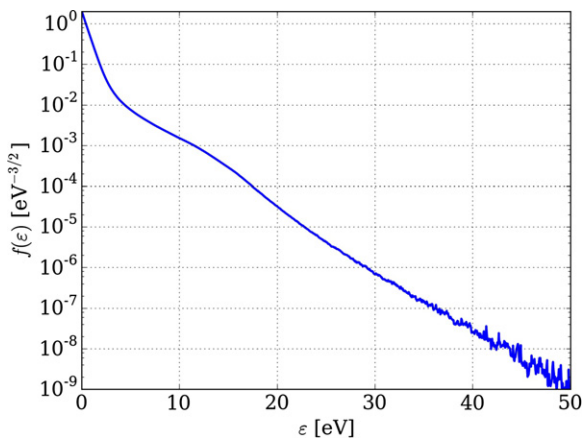
Figure 12. The temporally averaged electron energy probability function (EEPF) in the center of the discharge normalized as $\int f(\varepsilon)\sqrt{\varepsilon}\mathrm{d}\varepsilon = 1$ (data file: `eepf.dat`). Discharge conditions: argon $p = 10$ Pa, $L = 25$ mm, $T_g = 350$ K, $V = 250$ V and $f = 13.56$ MHz. The number of energy bins is `N_EEPF = 2000` and the resolution is `DE_EEPF = 0.05` (eV).
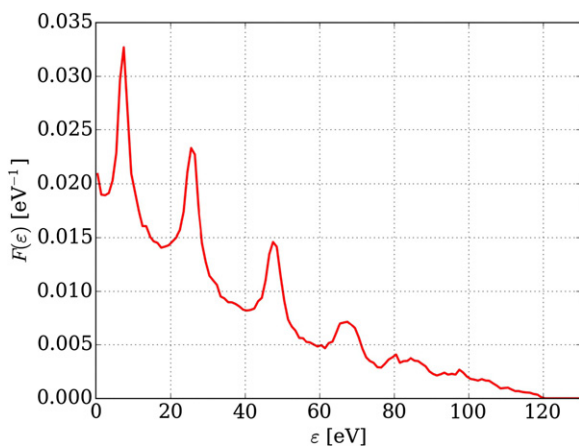

Figure 13. The temporally averaged flux-energy distribution (IFEDF) of the Ar^+ ions at the powered electrode normalized as $\int F(\varepsilon)\mathrm{d}\varepsilon = 1$ (data file: `ifed.dat`). Discharge conditions: argon $p = 10$ Pa, $L = 25$ mm, $T_g = 350$ K, $V = 250$ V and $f = 13.56$ MHz. The number of energy bins is `N_EEPF = 200` and the resolution is `DE_EEPF = 1` (eV).

discharge and simulation parameters are specified in table 3. We investigate two cases, with $p = 20$ Pa (CASE 1) and 3 Pa (CASE 2).

4.2.1. CASE 1: p = 20 Pa. We start the presentation of the results with the $p = 20$ Pa case, for which we used f_{HF} values of 10 MHz and 20 MHz, and V_{HF} values of 500 V and 250 V, respectively. Figures 14(a) and (b) present the flux-energy distributions of the Ar^+ ions ($F(\varepsilon)$), for these values of V_{HF}, along with a series of V_{LF} values. At $V_{LF} = 0$ (corresponding to a single-frequency excitation) the $F(\varepsilon)$ function peaks at low energy. This is a result of the collisional transport of the Ar^+ ions through the sheaths due to their mean free path being significantly shorter than the sheath length at this relatively high pressure. The ionization source function corresponding to the single-frequency case with $f_{HF} = 20$ MHz and $V_{HF} = 250$ V is

shown in figure 15(a). Ionization maxima in the vicinity of the expanding sheath edges can be seen to be repeated as dictated by the value of the high frequency. The application of a low frequency voltage results in a decay of the plasma density as it can be seen in figure 14(c). As a consequence of this, the ion flux to the electrodes also decreases with increasing V_{LF}, as shown in figure 14(d). The same figure also reveals that the mean energy of the ions hardly increases as a function of V_{LF}, which, at least in principle, should serve as the 'control parameter' for the mean ion energy and is supposed to have little influence on the ion flux. The results shown here confirm that this is not the case, i.e. the idea of using DF excitation for an independent control of the ion properties does not work efficiently under the conditions of CASE 1.

The decay of the plasma density and the ion flux can be attributed to the changes of the spatio-temporal distribution of the ionization rate, R_{ion} as a function of V_{LF}. An example for R_{ion}, obtained at $V_{LF} = 400$ V (at $f_{HF} = 20$ MHz and $V_{HF} = 250$ V) is displayed in figure 15(b). We observe a complicated variation of the sheath length, being modulated by both the low and high frequency voltages. As a result, (i) a more intensive ionization is observed as compared to the single-frequency case (figure 15(a)) when the sheath expands faster near the electrodes when the two applied voltage waveforms 'interfere' constructively, and (ii) a significantly lower ionization rate is obtained when the sheath length is large and the high frequency oscillations of the sheath length occur in the domain of higher ion density. The second effect is more pronounced and results in a decrease of the ionization rate, and consequently, in a decreased plasma density at $V_{LF} > 0$. This interplay between the two driving voltage components is called as the frequency coupling effect [151, 152]. Due to their increased length, the sheaths become even more collisional and the increased total voltage can just about compensate for the higher collisional energy losses of the ions, and results in about the same mean ion energy at any $V_{LF} > 0$. These observations hold for $f_{HF} = 20$ MHz as well, the mean energy of the ions increases only slightly with V_{LF} and the flux-energy distributions shown in figure 14(b) are rather similar at any V_{LF}, except for the disappearance of the peak created by charge transfer collisions [145]. We recall that the origin of these peaks is that ions that have undergone charge transfer collisions at times of a low sheath voltage can accumulate in certain spatial regions and get accelerated again when the sheath voltage reaches an appreciable value [146]. The periodicity of the decay and rise of the sheath electric field under single-frequency excitation can this way synchronize the motion of many ions, forming peaks in $F(\varepsilon)$. At DF excitation, however, the sheath voltage is low only during very limited times during the LF radio-frequency (RF) cycle, making such a synchronized ion motion hardly possible, which leads to the disappearance of the peaks.

4.2.2. CASE 2: p = 3 Pa. At this pressure, high frequency values of $f_{HF} = 30$ MHz and 40 MHz values are used. These higher values of the frequency, as compared to those used in CASE 1, are needed to establish sufficient ionization at the significantly lower value of the pressure considered now. In order

Table 3. Set of parameters for the dual-frequency discharge simulations.

V_{LF}	0 V	100 V	200 V	300 V	400 V
L			25 mm		
T_g			350 K		
f_{LF}			2 MHz		
			CASE 1		
p			20 Pa		
f_{HF}/V_{HF}			10 MHz/500 V		
W	1.1×10^5	1.0×10^5	8.2×10^4	6.1×10^4	4.2×10^4
N_g	600	600	600	600	600
N_t	31 100	31 200	31 400	31 700	33 200
f_{HF}/V_{HF}			20 MHz/250 V		
W	1.7×10^5	1.2×10^5	9.2×10^4	7.8×10^4	6.8×10^4
N_g	600	500	500	500	500
N_t	30 700	25 800	26 100	26 200	26 400
			CASE 2		
p			3 Pa		
f_{HF}/V_{HF}			30 MHz/250 V		
W	1.6×10^5	1.5×10^5	1.5×10^5	5.5×10^4	1.9×10^4
N_g	500	500	500	500	400
N_t	30 800	30 900	31 300	31 900	27 200
f_{HF}/V_{HF}			40 MHz/150 V		
W	1.5×10^5	1.3×10^5	1.1×10^5	8.8×10^4	2.8×10^4
N_g	500	500	500	500	400
N_t	30 700	30 900	31 100	31 300	26 300

to keep the plasma density near the same value as in CASE 1, V_{HF} values of 250 V and 150 V are used, respectively, for the f_{HF} values given in table 3.

The flux-energy distributions of the Ar$^+$ ions are shown in figures 16(a) and (b), respectively, for $f_{HF} = 30$ MHz and 40 MHz. In the single-frequency cases ($V_{LF} = 0$ V), the observed IFEDFs exhibit a prominent peak at high energy (that corresponds to the time-averaged sheath voltage) indicating a nearly collisionless transfer of the positive ions through the sheaths. With the application of the low-frequency excitation the IFEDFs change in a complicated manner. The most important change is that these functions extend towards higher energy with increasing V_{LF}, is rather obvious. The increased mean ion energy is also confirmed by the data shown in figure 16(d). The simultaneous increase of the low-energy part in the $F(\varepsilon)$ functions in figures 16(a) and (b), is however, an indication of the increase of the sheath length that was also observed at $p = 20$ Pa, discussed earlier. The increase of the sheath length is clearly demonstrated by the plot of the ionization rate, R_{ion}, in figure 17(b). Due to the lower pressure, the maximum of the ionization rate moves towards the center of the plasma as V_{LF} is increased, at otherwise same conditions. The coupling of the excitation harmonics results in this case as well in a decrease of the plasma density (figure 16(c)) and the ion flux (figure 16(d) with increasing V_{LF} in the absence of secondary electron emission from the electrodes [153]. Nonetheless, the mean ion energy increases significantly and can be tuned by the control parameter V_{LF} over an appreciable range, unlike in the case of higher pressures.

5. Summary and ideas for the further development of the code

In this work, we have described a simple particle-in-cell/Monte Carlo collisions code for 1D electrostatic simulations of capacitively coupled radiofrequency plasmas. Following an introduction to the basics of the approach, we provided comprehensive information about the structure of the eduPIC code, its compilation and execution, as well as about the content of the data files created. The operation of the code was explained at the level of algorithmic details. Subsequently, a set of reference results was presented, with the default parameter settings of the code. When downloaded and properly executed by a reader, the code should reproduce these results without any modifications of the code and the simulation settings. We also illustrated the capabilities of this basic code by presenting some simulation results for DF CCPs. The eduPIC code (with proper modifications of the input cross section set and the scattering model) has successfully been benchmarked with the data presented in [102].

As it has already been emphasized, this code intends to be a 'starting tool' that can be optimized in many respects (e.g. in handling the collisions [122, 154]) and can be developed further to address various research questions that the reader of this work may have. To assist such activities, we provide our code as an open source item, via Github, in three computing languages: the basic (C) version is complemented with more sophisticated versions available in C++ and Rust [140].
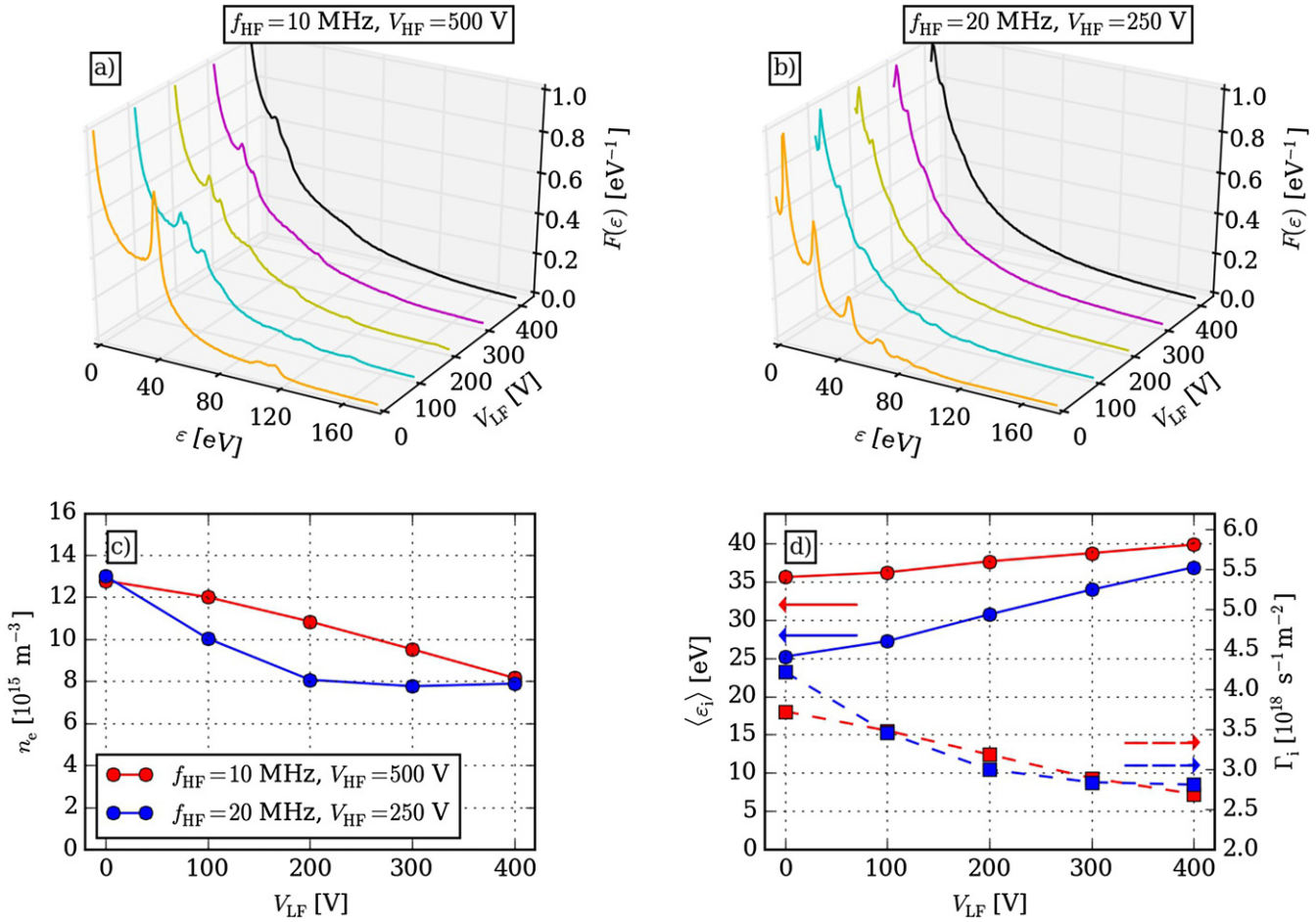
Figure 14. (a) and (b) IFEDFs as a function of the low frequency voltage for (a) $f_{HF} = 10$ MHz, $V_{HF} = 500$ V and (b) $f_{HF} = 20$ MHz, $V_{HF} = 250$ V. For a better comparison, the IFEDFs are normalized in a way that the maximum value is 1. (c) Peak electron density as a function of the low frequency voltage. (d) Mean ion energy at the powered electrode (solid lines, left axis) and the mean ion flux at the same electrode (dashed lines, right axis) as a function of the low frequency voltage. The colors of the lines in this panel correspond to those in panel (c). Discharge conditions: argon $p = 20$ Pa, $L = 25$ mm, $T_g = 350$ K and $f_{LF} = 2$ MHz.
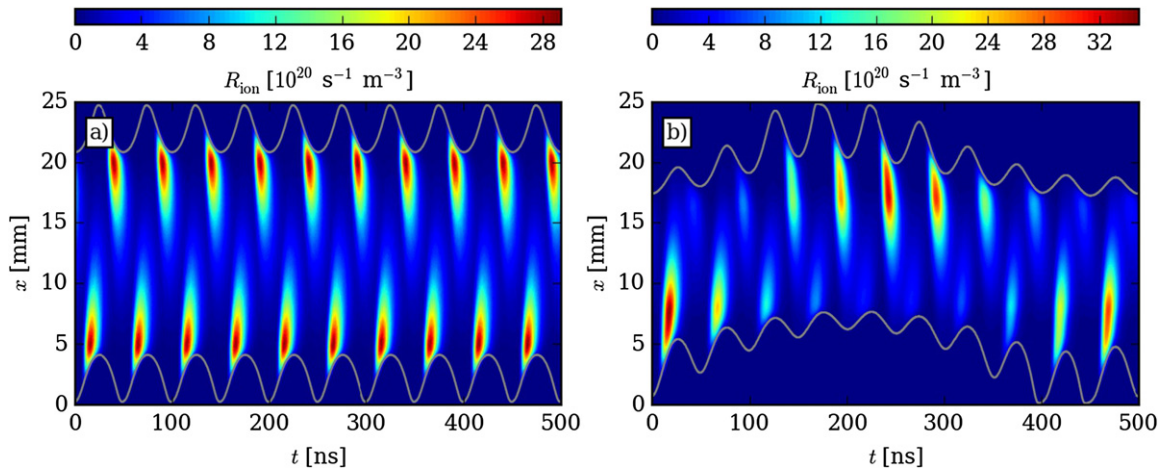
Figure 15. The spatial and temporal dynamics of the ionization rate for $f_{HF} = 20$ MHz and $V_{HF} = 250$ V. The low frequency voltage is $V_{LF} = 0$ V in panel (a) and $V_{LF} = 400$ V in panel (b). The gray solid lines represent the sheath edges. Discharge conditions: argon $p = 20$ Pa, $L = 25$ mm, $T_g = 350$ K and $f_{LF} = 2$ MHz. The time shown on the horizontal axis corresponds to one period of the LF excitation, $T_{LF} = 1/f_{LF}$.
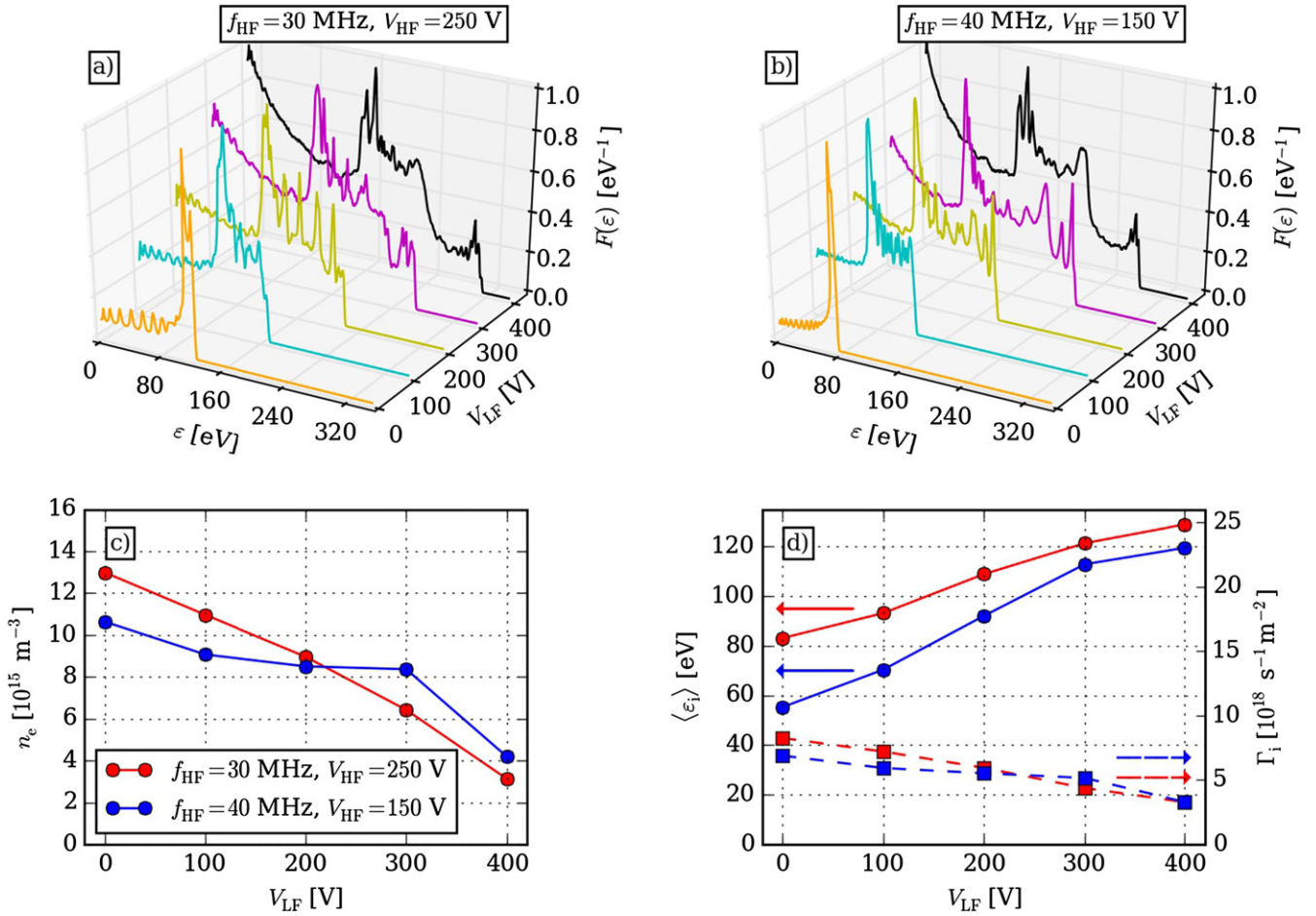
Figure 16. (a) and (b) IFEDFs as a function of the low frequency voltage for (a) $f_{HF} = 30$ MHz, $V_{HF} = 250$ V and (b) $f_{HF} = 40$ MHz, $V_{HF} = 150$ V. For a better comparison, the IFEDFs are normalized in a way that the maximum value is 1. (c) Peak electron density as a function of the low frequency voltage. (d) The colors of the lines in this panel correspond to those in panel (c). The mean ion energy at the powered electrode (solid lines, left axis) and the mean ion flux at the same electrode (dashed lines, right axis) as a function of the low frequency voltage. Discharge conditions: argon $p = 3$ Pa, $L = 25$ mm, $T_g = 350$ K and $f_{LF} = 2$ MHz.
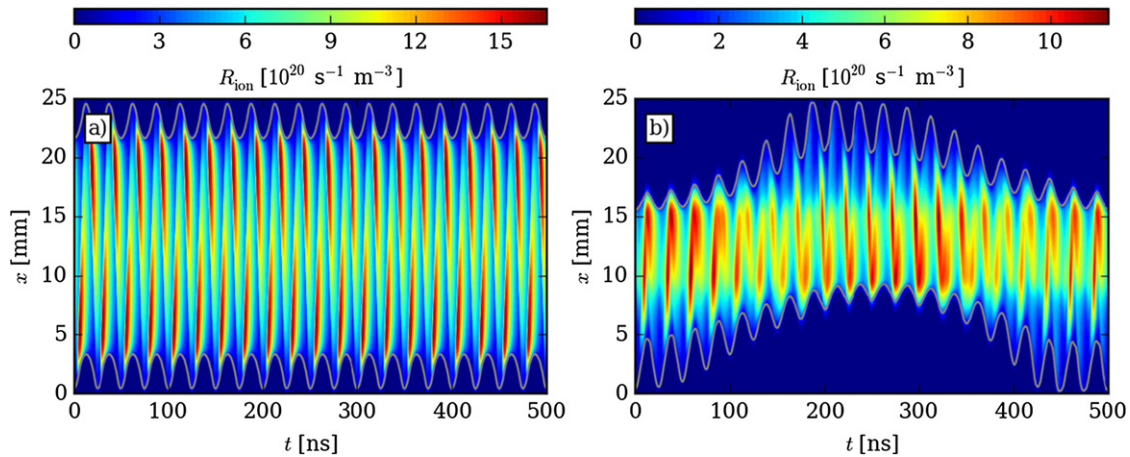
Figure 17. The spatial and temporal dynamics of the ionization rate for $f_{HF} = 40$ MHz and $V_{HF} = 150$ V. The low frequency voltage is $V_{LF} = 0$ V in panel (a) and $V_{LF} = 400$ V in panel (b). The gray solid lines represent the sheath edges. Discharge conditions: argon $p = 3$ Pa, $L = 25$ mm, $T_g = 350$ K and $f_{LF} = 2$ MHz. The time shown on the horizontal axis corresponds to one period of the LF excitation, $T_{LF} = 1/f_{LF}$.

Below we give several suggestions for the further development of the code, which will greatly widen its scope of applicability and/or enhance its performance.

- *Implementation of the emission of ion-induced secondary electrons from the electrodes*. In a stochastic approach the ion-induced secondary electron yield, γ, is understood as a probability: whenever an ion reaches the surface of an electrode, an electron is emitted with this probability (as long as $\gamma < 1$), i.e. when $R_{01} < \gamma$. The simplest approach is to use a constant γ, in a more sophisticated model, an energy-dependent γ can be adopted. For data for Ar^+ see [134], as well as [155] for corrections to some of the formulas given in [134].

- *Implementation of the elastic reflection of electrons at the electrodes*. Whenever an electron reaches the surface of an electrode, the electron is reflected elastically with a probability η_e. In the simplest approach, the probability of the elastic reflection can be set to a constant value (e.g. $\eta_e = 0.2$ [156], independently of the discharge conditions and surface properties). In a more complex approach, η_e is the function of the energy and angle of incidence of the electron and depends on the surface properties [57, 77]. (One can go even beyond considering elastic reflection and include as well inelastic reflection and secondary electron emission due to electron impact [57].)

- *Implementation of the computation of the self-bias voltage that develops when the discharge is driven by tailored waveforms*. Tailored excitation voltage waveforms provide a possibility to control the energy distribution of the ions (and electrons) reaching the electrodes (see e.g. [157]). The DC self-bias voltage can be computed by monitoring the currents of the oppositely charged particles at the electrodes. At each electrode, these currents have to compensate each other on time average, and the self-bias voltage can be adjusted in the simulation in an iteration cycle to achieve this balance. (The time-averaging should cover a number of RF cycles so that the statistical fluctuations of the particle currents should be sufficiently suppressed).

- *Implementation of different electrode materials at the powered and grounded electrode*. Electrodes made of different materials can be modeled by using different probabilities for the ion-induced secondary electron emission (γ) and/or the elastic reflection of electrons (η_e) at both electrodes. The DC self-bias voltage due to the secondary electron induced asymmetry of the discharge [73] and/or the asymmetry induced by electron reflection [125] can be determined as suggested above.

- *Improvement of the scattering model by incorporating anisotropic scattering for electron–atom collisions*. At high electron energy the scattering may be forward-peaked, which can be incorporated into the code via the proper differential cross sections. For guidance, see e.g. [158, 159].

- *Calculation of the position of the sheath edge*. For the definition of the sheath edge see [141]. Whenever a quasineutral region of the plasma exists, the procedure described in this reference gives a guidance for deriving the (time-dependent) position of the sheath edge from the (time-dependent) electron and ion density profiles. When the sheath length is known, further important information can be obtained from the existing data: (i) the voltage drop over the sheath can be found from the spatio-temporal distribution of the potential, and (ii) the net charge inside the sheath can be found from the density distributions of the charged particles.

- *Change the code into a cylindrical version*. By considering a long cylindrical setup with coaxial electrodes [16, 121, 160], plasma sources with unequal electrode surface areas can be simulated with a 1D approach. In this case, plasma parameters change as a function of the radial coordinate only. The computation of the charged particle densities, the solution of the Poisson equation and the integration of the equation of motion need modifications. In such a system, a self-bias voltage develops due to the unequal electrode surfaces. This voltage can be computed as described in one of the points above.

- *Include an external circuit*. Real plasma systems are driven by generators and matching networks to optimize power coupling into the plasma. In [121], a detailed description of the simultaneous solution of the Poisson equation and the equation for an external electrical circuit is provided, for various 1D geometries.

- *Introduce the null-collision approach to select colliding particles*. Instead of evaluating the collision probability using equation (13) that involves significant computation, for each particle, a certain number of particles can be assigned to undergo collisions, based on the maximum of the collision frequency of the given species. Using this approach the set of processes is expanded with the 'null collision' [17, 130, 161]—upon the occurrence of this, the particle proceeds without any change. The method significantly increases the efficiency of selecting colliding particles.

- *Implement an iteration algorithm that allows using the discharge power as an input quantity*. The computation of the discharge power is implemented in the voltage-driven eduPIC code. Try to implement an iteration cycle that adjusts the voltage automatically in order to reach the power level specified. This 'control loop' has to have sufficiently low gain to prevent diverging oscillations of the voltage as a result of its adjustment based on the difference between the specified and actual power levels. The latter should be measured over a sufficiently high number of RF cycles to provide an accurate measurement of the actual power.

- *Parallelization of the code*. Modern computer architectures provide access to various paradigms of parallel code execution. In order to fully utilize the compute capacity of the hardware, code parallelization is a key technique. At the lowest level, the vectorization of simple operations repeated on consecutive data elements is performed by modern compilers by default. For multi-thread (multi CPU core) execution several methods exist, like

OpenMP [162], pthreads [163], or those provided by recent generations of the C++ standard library. However, the PIC/MCC method is difficult to parallelize efficiently, as the Monte Carlo type of collision handling incorporates a high degree of code branching and unpredictable execution orders due to the heavy use of random numbers. A careful redesign of data structures and execution loops is necessary to optimize parallel performance. Alternatively, the application of massively parallel graphics processing units (GPUs) represent attractive architectures for code acceleration, as discussed in [34].

Apart from the above list of ideas and suggestions, there is a number of emerging topics that deserve interest and can be studied in CCPs via extending the eduPIC code, e.g. the investigation of the effects of magnetic fields on the plasma and various asymmetry effects (e.g. [164, 165]). There is also a possibility to modify the eduPIC code to be able to describe various plasma sources, e.g. pulsed discharges. The exploration and the discussion of these possibilities goes, however, beyond the scope of this work.

As a final remark, it should be kept in mind that the code and its possible extensions are always built on *discharge models*, which always represent a simplification of real systems. Therefore, one cannot expect simulations to perfectly reproduce the physics of the real systems. The expected accuracy greatly depends on the completeness of the model. Also, following any modifications of the code, it is a good practice to benchmark the results with those of independent studies reported in the literature, if available [102, 166].

Any development, in general, should be followed by thorough checks of the parts of the code that have been modified or extended. Discussing the methodologies of code verification goes beyond the scope of this paper but consulting references [103, 167, 168] is advised.

Disclaimer

Acknowledgments

Data availability statement

The data that support the findings of this study are available upon reasonable request from the authors.

ORCID iDs

Zoltán Donkó ⓘ https://orcid.org/0000-0003-1369-6150
Aranka Derzsi ⓘ https://orcid.org/0000-0002-8005-5348
Máté Vass ⓘ https://orcid.org/0000-0001-9865-4982
Benedek Horváth ⓘ https://orcid.org/0000-0002-2371-2444
Sebastian Wilczek ⓘ https://orcid.org/0000-0003-0583-4613
Peter Hartmann ⓘ https://orcid.org/0000-0003-3572-1310

References

[1] Wilczek S, Schulze J, Brinkmann R P, Donkó Z, Trieschmann J and Mussenbrock T 2020 *J. Appl. Phys.* **127** 181101
[2] Tsendin L D 2010 *Phys.-Usp.* **53** 133
[3] Gallagher J W 2012 *Nonequilibrium Effects in Ion and Electron Transport* (Berlin: Springer)
[4] Fu Y, Zheng B, Zhang P, Fan Q H, Verboncoeur J P and Wang X 2020 *Phys. Plasmas* **27** 113501
[5] Fu Y, Zheng B, Wen D-Q, Zhang P, Fan Q H and Verboncoeur J P 2020 *Plasma Sources Sci. Technol.* **29** 09LT01
[6] Wang L, Hartmann P, Donkó Z, Song Y H and Schulze J 2020 *Plasma Sources Sci. Technol.* accepted
[7] Held E D, Callen J D, Hegna C C, Sovinec C R, Gianakon T A and Kruger S E 2004 *Phys. Plasmas* **11** 2419–26
[8] Kushner M J 2009 *J. Phys. D: Appl. Phys.* **42** 194013
[9] Zhang Y, Zafar A, Coumou D J, Shannon S C and Kushner M J 2015 *J. Appl. Phys.* **117** 233302
[10] Economou D J 2017 *Plasma Process. Polym.* **14** 1600152
[11] Becker M M, Kählert H, Sun A, Bonitz M and Loffhagen D 2017 *Plasma Sources Sci. Technol.* **26** 044001
[12] Birdsall C K 1991 *IEEE Trans. Plasma Sci.* **19** 65–85
[13] Buneman O 1959 *Phys. Rev.* **115** 503–17
[14] Dawson J 1962 *Phys. Fluids* **5** 445–59
[15] Hockney R W and Eastwood J W 1988 *Computer Simulation Using Particles* (London: Taylor and Francis)
[16] Birdsall C and Langdon A 2004 *Plasma Physics via Computer Simulation* (*Plasma Physics and Fluid Dynamics*) (London: Taylor and Francis)
[17] Longo S 2006 *Plasma Sources Sci. Technol.* **15** S181
[18] Lin S L and Bardsley J N 1977 *J. Chem. Phys.* **66** 435–45
[19] Lin S L, Viehland L A, Mason E A, Whealton J H and Bardsley J N 1977 *J. Phys. B: At. Mol. Phys.* **10** 3567
[20] Sakai Y, Tagashira H and Sakamoto S 1977 *J. Phys. D: Appl. Phys.* **10** 1035
[21] Pitchford L C and Phelps A V 1982 *Phys. Rev.* A **25** 540
[22] Boeuf J P and Marode E 1982 *J. Phys. D: Appl. Phys.* **15** 2169
[23] Penetrante B M, Bardsley J N and Pitchford L C 1985 *J. Phys. D: Appl. Phys.* **18** 1087

[24] Dahl D A, Teich T H and Franck C M 2012 *J. Phys. D: Appl. Phys.* **45** 485201

[25] Grubert G K and Loffhagen D 2013 *J. Phys. D: Appl. Phys.* **47** 025204

[26] Dujko S, Raspopović Z M, White R D, Makabe T and Petrović Z L 2014 *Eur. Phys. J.* D **68** 1–9

[27] Ponomarev A A and Aleksandrov N L 2015 *Plasma Sources Sci. Technol.* **24** 035001

[28] Mertmann P, Eremin D, Mussenbrock T, Brinkmann R P and Awakowicz P 2011 *Comput. Phys. Commun.* **182** 2161–7

[29] Hanzlikova N 2015 Particle-in-cell simulations of highly collisional plasmas on the GPU in 1 and 2 dimensions *PhD Thesis* Dublin City University, Dublin, Ireland

[30] Shah H, Kamaria S, Markandeya R, Shah M and Chaudhury B 2017 A novel implementation of 2D3V particle-in-cell (PIC) algorithm for Kepler GPU architecture *2017 IEEE 24th Int. Conf. on High Performance Computing (HiPC)* (Piscataway, NJ: IEEE) pp 378–87

[31] Hur M Y, Kim J S, Song I C, Verboncoeur J P and Lee H J 2019 *Plasma Res. Express* **1** 015016

[32] Claustre J, Chaudhury B, Fubiani G, Paulin M and Boeuf J P 2013 *IEEE Trans. Plasma Sci.* **41** 391–9

[33] Fierro A, Dickens J and Neuber A 2014 *Phys. Plasmas* **21** 123504

[34] Juhasz Z, Ďurian J, Derzsi A, Matejčík Š, Donkó Z and Hartmann P 2021 *Comput. Phys. Commun.* **263** 107913

[35] Matyash K, Schneider R, Taccogna F, Hatayama A, Longo S, Capitelli M, Tskhakaya D and Bronold F X 2007 *Contrib. Plasma Phys.* **47** 595–634

[36] Radmilović-Radjenović M, Lee J K, Iza F and Park G 2005 *J. Phys. D: Appl. Phys.* **38** 950

[37] Liu Y-X, Wang X-Y, Zhang Q-Z, Donkó Z, Zhao K, Schulze J and Wang Y-N 2020 *Plasma Sources Sci. Technol.* **29** 12LT03

[38] Georgieva V, Bogaerts A and Gijbels R 2004 *Phys. Rev.* E **69** 026406

[39] Lee J K, Manuilenko O V, Babaeva N Y, Kim H C and Shon J W 2005 *Plasma Sources Sci. Technol.* **14** 89

[40] Wang S, Xu X and Wang Y-N 2007 *Phys. Plasmas* **14** 113501

[41] Donkó Z, Derzsi A, Vass M, Schulze J, Schuengel E and Hamaguchi S 2018 *Plasma Sources Sci. Technol.* **27** 104008

[42] O'Connell D, Zorat R, Ellingboe A and Turner M 2007 *Phys. Plasmas* **14** 103510

[43] Manuilenko O and Minaeva K 2006 *Problems of Atomic Science and Technology* **12** 116

[44] Sydorenko D, Smolyakov A, Kaganovich I and Raitses Y 2006 *Phys. Plasmas* **13** 014501

[45] Taccogna F, Schneider R, Longo S and Capitelli M 2008 *Plasma Sources Sci. Technol.* **17** 024003

[46] Lafleur T, Baalrud S D and Chabert P 2016 *Phys. Plasmas* **23** 053503

[47] Lafleur T, Chabert P and Booth J P 2014 *Plasma Sources Sci. Technol.* **23** 035010

[48] Gudmundsson J T and Snorrason D I 2017 *J. Appl. Phys.* **122** 193302

[49] Liu G-H, Liu Y-X, Wen D-Q and Wang Y-N 2015 *Plasma Sources Sci. Technol.* **24** 034006

[50] Schulze J, Donkó Z, Lafleur T, Wilczek S and Brinkmann R P 2018 *Plasma Sources Sci. Technol.* **27** 055010

[51] Vass M, Wilczek S, Lafleur T, Brinkmann R P, Donkó Z and Schulze J 2020 *Plasma Sources Sci. Technol.* **29** 025019

[52] Liu Y-X, Schüngel E, Korolov I, Donkó Z, Wang Y-N and Schulze J 2016 *Phys. Rev. Lett.* **116** 255002

[53] Donkó Z, Schulze J, Czarnetzki U and Luggenhölscher D 2009 *Appl. Phys. Lett.* **94** 131501

[54] Schüngel E, Brandt S, Korolov I, Derzsi A, Donkó Z and Schulze J 2015 *Phys. Plasmas* **22** 043512

[55] Proshina O V, Rakhimova T V, Kovalev A S, Vasilieva A N, Zotovich A I, Zyryanov S M and Rakhimov A T 2020 *Plasma Sources Sci. Technol.* **29** 015015

[56] Daksha M, Derzsi A, Wilczek S, Trieschmann J, Mussenbrock T, Awakowicz P, Donkó Z and Schulze J 2017 *Plasma Sources Sci. Technol.* **26** 085006

[57] Horváth B, Daksha M, Korolov I, Derzsi A and Schulze J 2017 *Plasma Sources Sci. Technol.* **26** 124001

[58] Horváth B, Schulze J, Donkó Z and Derzsi A 2018 *J. Phys. D: Appl. Phys.* **51** 355204

[59] Donkó Z, Hamaguchi S and Gans T 2019 *Plasma Sources Sci. Technol.* **28** 075004

[60] Korolov I, Donkó Z, Hübner G, Bischoff L, Hartmann P, Gans T, Liu Y, Mussenbrock T and Schulze J 2019 *Plasma Sources Sci. Technol.* **28** 094001

[61] Lieberman M A and Lichtenberg A J 2005 *Principles of Plasma Discharges and Materials Processing* 2nd edn (New York: Wiley)

[62] Makabe T and Petrovic Z L 2014 *Plasma Electronics: Applications in Microelectronic Device Fabrication* 2nd edn (Boca Raton, FL: CRC Press)

[63] Chabert P and Braithwaite N 2011 *Physics of Radio-Frequency Plasmas* (Cambridge: Cambridge University Press)

[64] Go Wakayama G and Nanbu K 2003 *IEEE Trans. Plasma Sci.* **31** 638–44

[65] Karkari S K and Ellingboe A R 2006 *Appl. Phys. Lett.* **88** 101501

[66] Donkó Z, Schulze J, Heil B G and Czarnetzki U 2008 *J. Phys. D: Appl. Phys.* **42** 025205

[67] Zhang Q-Z, Jiang W, Hou L-J and Wang Y-N 2011 *J. Appl. Phys.* **109** 013308

[68] Lafleur T and Booth J P 2012 *J. Phys. D: Appl. Phys.* **45** 395203

[69] Bruneau B, Novikova T, Lafleur T, Booth J P and Johnson E V 2014 *Plasma Sources Sci. Technol.* **23** 065010

[70] Bruneau B *et al* 2015 *Plasma Sources Sci. Technol.* **25** 01LT02

[71] Delattre P-A, Lafleur T, Johnson E and Booth J-P 2013 *J. Phys. D: Appl. Phys.* **46** 235201

[72] Yang S, Zhang Y, Wang H, Cui J and Jiang W 2017 *Plasma Process. Polym.* **14** 1700087

[73] Lafleur T, Chabert P and Booth J P 2013 *J. Phys. D: Appl. Phys.* **46** 135201

[74] Korolov I, Derzsi A, Donkó Z and Schulze J 2013 *Appl. Phys. Lett.* **103** 064102

[75] Zhang Q-Z, Zhao S-X, Jiang W and Wang Y-N 2012 *J. Phys. D: Appl. Phys.* **45** 305203

[76] Fu Y, Zheng B, Wen D-Q, Zhang P, Fan Q H and Verboncoeur J P 2020 *Appl. Phys. Lett.* **117** 204101

[77] Braginsky O, Kovalev A, Lopaev D, Proshina O, Rakhimova T, Vasilieva A, Voloshin D and Zyryanov S 2011 *J. Phys. D: Appl. Phys.* **45** 015201

[78] Sun A, Becker M M and Loffhagen D 2018 *Plasma Sources Sci. Technol.* **27** 054002

[79] Bronold F X, Matyash K, Tskhakaya D, Schneider R and Fehske H 2007 *J. Phys. D: Appl. Phys.* **40** 6583

[80] Schüngel E, Zhang Q-Z, Iwashita S, Schulze J, Hou L-J, Wang Y-N and Czarnetzki U 2011 *J. Phys. D: Appl. Phys.* **44** 285205

[81] Gudmundsson J T, Kawamura E and Lieberman M A 2013 *Plasma Sources Sci. Technol.* **22** 035011

[82] Teichmann T, Küllig C, Dittmann K, Matyash K, Schneider R and Meichsner J 2013 *Phys. Plasmas* **20** 113509

[83] Derzsi A, Lafleur T, Booth J-P, Korolov I and Donkó Z 2015 *Plasma Sources Sci. Technol.* **25** 015004

[84] Proto A and Gudmundsson J T 2018 *Atoms* **6** 65

[85] Diomede P, Longo S, Economou D J and Capitelli M 2012 *J. Phys. D: Appl. Phys.* **45** 175204

[86] Denpoh K and Nanbu K 1998 *J. Vac. Sci. Technol.* A **16** 1201–6

[87] Denpoh K and Nanbu K 2000 *Japan. J. Appl. Phys.* **39** 2804

[88] Proshina O V, Rakhimova T V, Rakhimov A T and Voloshin D G 2010 *Plasma Sources Sci. Technol.* **19** 065013

[89] Longo S, Hassouni K, Iasillo D and Capitelli M 1997 *J. Phys. III* **7** 707–18

[90] Kawamura E, Lieberman M A, Lichtenberg A J and Chabert P 2021 *Plasma Sources Sci. Technol.* **30** 035001

[91] Huang S and Gudmundsson J T 2013 *Plasma Sources Sci. Technol.* **22** 055020

[92] Georgieva V, Bogaerts A and Gijbels R 2003 *J. Appl. Phys.* **93** 2369–79

[93] Brandt S, Berger B, Donkó Z, Derzsi A, Schüngel E, Koepke M and Schulze J 2019 *Plasma Sources Sci. Technol.* **28** 095021

[94] Lee S H, Iza F and Lee J K 2006 *Phys. Plasmas* **13** 057102

[95] Babaeva N Y, Lee J K and Shon J W 2005 *J. Phys. D: Appl. Phys.* **38** 287

[96] Ivanov V, Proshina O, Rakhimova T, Rakhimov A, Herrebout D and Bogaerts A 2002 *J. Appl. Phys.* **91** 6296–302

[97] Voloshin D G, Mankelevich Y A, Proshina O V and Rakhimova T V 2017 *Plasma Process. Polym.* **14** 1600119

[98] Turner M M 2006 *Phys. Plasmas* **13** 033506

[99] Kawamura E, Birdsall C K and Vahedi V 2000 *Plasma Sources Sci. Technol.* **9** 413

[100] Erden E and Rafatov I 2014 *Contrib. Plasma Phys.* **54** 626–34

[101] Sun A, Becker M and Loffhagen D 2016 *Comput. Phys. Commun.* **206** 35–44

[102] Turner M M, Derzsi A, Donkó Z, Eremin D, Kelly S J, Lafleur T and Mussenbrock T 2013 *Phys. Plasmas* **20** 013507

[103] Turner M M 2016 *Plasma Sources Sci. Technol.* **25** 054007

[104] Turner M M 2017 *Plasma Process. Polym.* **14** 1600121

[105] Chabert P, Tsankov T V and Czarnetzki U 2021 *Plasma Sources Sci. Technol.* **30** 024001

[106] Jiankai L, Zhang Y, Kai Z, Deqi W and Younian W 2021 *Plasma Sci. Technol.* **23** 035401

[107] Lymberopoulos D P and Economou D J 1995 *J. Res. Natl Inst. Stand. Technol.* **100** 473

[108] Lapenta G 2016 *Plasma Modeling* 2053–563

[109] The Plasma Theory and Simulation Group https://ptsg.egr.msu.edu/

[110] Tskhakaya D, Matyash K, Schneider R and Taccogna F 2007 *Contrib. Plasma Phys.* **47** 563–94

[111] Verboncoeur J P 2005 *Plasma Phys. Control. Fusion* **47** A231–60

[112] Kim H C, Iza F, Yang S S, Radmilović-Radjenović M and Lee J K 2005 *J. Phys. D: Appl. Phys.* **38** R283

[113] Donkó Z and Petrović Z L 2007 *J. Phys.: Conf. Ser.* **86** 012011

[114] Verboncoeur J P 1995 Oopic: object oriented particle-in-cell code *Int. Conf. on Plasma Science (papers in summary form only received)* p 244

[115] Kühn C and Groll R 2020 picfoam: an openfoam based electrostatic particle-in-cell solver (arXiv:2012.14724)

[116] Kolev I and Bogaerts A 2006 *Plasma Process. Polym.* **3** 127–34

[117] Taccogna F, Minelli P and Longo S 2013 *Plasma Sources Sci. Technol.* **22** 045019

[118] Inusa E I, Sa'ad A U, Samuel A O and Sanusi L M 2014 *Arch. Appl. Sci. Res.* **4** 143–56

[119] Chaber B 2020 Particle-in-cell code for gas discharge simulations *2020 IEEE 21st Int. Conf. on Computational Problems of Electrical Engineering (CPEE)* pp 1–4

[120] Markidis S, Lapenta G, VanderHeyden W and Budimlic Z 2005 *Concurrency and Computation: Practice and Experience* **17** 821

[121] Verboncoeur J P, Alves M V, Vahedi V and Birdsall C K 1993 *J. Comput. Phys.* **104** 321–8

[122] Nanbu K 2000 *IEEE Trans. Plasma Sci.* **28** 971–90

[123] Wang C, Lin T, Caflisch R, Cohen B I and Dimits A M 2008 *J. Comput. Phys.* **227** 4308–29

[124] Derzsi A, Korolov I, Schüngel E, Donkó Z and Schulze J 2015 *Plasma Sources Sci. Technol.* **24** 034002

[125] Korolov I, Derzsi A, Donkó Z, Schüngel E and Schulze J 2016 *Plasma Sources Sci. Technol.* **25** 015024

[126] Hannesdottir H and Gudmundsson J T 2016 *Plasma Sources Sci. Technol.* **25** 055002

[127] Greb A, Niemi K, O'Connell D and Gans T 2013 *Appl. Phys. Lett.* **103** 244101

[128] Daksha M, Derzsi A, Mujahid Z, Schulenberg D, Berger B, Donkó Z and Schulze J 2019 *Plasma Sources Sci. Technol.* **28** 034002

[129] Derzsi A, Horváth B, Donkó Z and Schulze J 2020 *Plasma Sources Sci. Technol.* **29** 074001

[130] Skullerud H R 1968 *J. Phys. D: Appl. Phys.* **1** 1567–8

[131] Phelps A V 1994 *J. Appl. Phys.* **76** 747–53

[132] Opal C B, Peterson W K and Beaty E C 1971 *J. Chem. Phys.* **55** 4100–6

[133] Karimabadi H, Driscoll J, Omelchenko Y A and Omidi N 2005 *J. Comput. Phys.* **205** 755–75

[134] Phelps A V and Petrovic Z L 1999 *Plasma Sources Sci. Technol.* **8** R21

[135] Wu H 2008 *The Stream Cipher HC-128 New Stream Cipher Designs (Lecture Notes in Computer Science)* vol 4986 ed M Robshaw and O Billet (Berlin: Springer) pp 39–47

[136] Thomas L H 1949 Elliptic problems in linear difference equations over a network *Technical Report Watson Sci. Comput. Lab. Rept.* Columbia University New York

[137] Van Roy P and Haridi S 2004 *Concepts, Techniques, and Models of Computer Programming* (Cambridge, MA: MIT Press)

[138] Krishnamurthi S 2008 *SIGPLAN Not.* **43** 81–3

[139] Klabnik S and Nichols C 2018 *The Rust Programming Language* (San Francisco: No Starch Press)

[140] Donkó Z, Derzsi A, Vass M, Horváth B, Wilczek S, Hartmann B and Hartmann P 2021 eduPIC source codes https://github.com/donkozoltan/eduPIC/ accessed: 03 04 2021

[141] Brinkmann R P 2007 *J. Appl. Phys.* **102** 093303

[142] Schulze J, Donkó Z, Heil B G, Luggenhölscher D, Mussenbrock T, Brinkmann R P and Czarnetzki U 2008 *J. Phys. D: Appl. Phys.* **41** 105214

[143] Vass M, Wilczek S, Lafleur T, Brinkmann R P, Donkó Z and Schulze J 2020 *Plasma Sources Sci. Technol.* **29** 085014

[144] Godyak V A and Piejak R B 1990 *Phys. Rev. Lett.* **65** 996

[145] Wild C and Koidl P 1991 *J. Appl. Phys.* **69** 2909–22

[146] Schüngel E, Donkó Z and Schulze J 2017 *Plasma Process. Polym.* **14** 1600117

[147] Faraz T, Verstappen Y G P, Verheijen M A, Chittock N J, Lopez J E, Heijdra E, van Gennip W J H, Kessels W M M and Mackus A J M 2020 *J. Appl. Phys.* **128** 213301

[148] Bogdanova M, Lopaev D, Rakhimova T, Voloshin D, Zotovich A and Zyryanov S 2020 Virtual ied sensor for DF RF CCP discharges (arXiv:2012.14882)

[149] Kitajima T, Takeo Y, Petrović Z L and Makabe T 2000 *Appl. Phys. Lett.* **77** 489–91

[150] Czarnetzki U, Schulze J, Schüngel E and Donkó Z 2011 *Plasma Sources Sci. Technol.* **20** 024010

[151] Schulze J, Gans T, O'Connell D, Czarnetzki U, Ellingboe A R and Turner M M 2007 *J. Phys. D: Appl. Phys.* **40** 7008

[152] Waskoenig J and Gans T 2010 *Appl. Phys. Lett.* **96** 181501

[153] Donkó Z, Schulze J, Hartmann P, Korolov I, Czarnetzki U and Schüngel E 2010 *Appl. Phys. Lett.* **97** 081501

[154] Weisstein E W 2010 Sphere point picking from MathWorld-A Wolfram Web resource https://mathworld.wolfram.com/SpherePointPicking.html Last visited on 16 02 2021

[155] Phelps A, Pitchford L, Pédoussat C and Donkó Z 1999 *Plasma Sources Sci. Technol.* **8** B1

[156] Sheline R K 1956 *Encyclopedia of Physics* vol 21 ed S Flügge (Berlin: Springer)

[157] Lafleur T 2015 *Plasma Sources Sci. Technol.* **25** 013001

[158] Okhrimovskyy A, Bogaerts A and Gijbels R 2002 *Phys. Rev. E* **65** 037402

[159] Janssen J F J, Pitchford L C, Hagelaar G J M and van Dijk J 2016 *Plasma Sources Sci. Technol.* **25** 055026

[160] Wilczek S, Trieschmann J, Schulze J, Donkó Z, Brinkmann R P and Mussenbrock T 2018 *Plasma Sources Sci. Technol.* **27** 125010

[161] Vahedi V and Surendra M 1995 *Comput. Phys. Commun.* **87** 179–98

[162] Dagum L and Menon R 1998 *Computational Science & Engineering* vol 5 (Piscataway, NJ: IEEE) pp 46–55

[163] Narlikar G J and Blelloch G E 1998 Pthreads for dynamic and irregular parallelism *SC '98: Proc. 1998 ACM/IEEE Conf. on Supercomputing* p 31

[164] Wang L, Wen D-Q, Hartmann P, Donkó Z, Derzsi A, Wang X-F, Song Y-H, Wang Y-N and Schulze J 2020 *Plasma Sources Sci. Technol.* **29** 105004

[165] Skarphedinsson G A and Gudmundsson J T 2020 *Plasma Sources Sci. Technol.* **29** 084004

[166] Sarikaya C K, Rafatov I and Cakir S 2016 *Balkan Phys. Letters* **24** 36

[167] Riva F, Beadle C F and Ricci P 2017 *Phys. Plasmas* **24** 055703

[168] O'Connor S, Crawford Z D, Verboncoeur J P, Luginsland J and Shanker B 2021 *IEEE Trans. Plasma Sci.* **49** 1724–31