University of Maribor

Faculty of Electrical Engineering and Computer Science

FERI Faculty of Electrical Engineering and Computer Science

MATLAB SIMULATION FOR DIFFERENT NUMBER OF PILLARS OF A MICROFLUIDIC ROTATIONAL MOTOR DRIVEN BY CIRCULAR VIBRATIONS

Student: Iker Ruiz de Esquide Crespo

Student program: Engineering in industrial technologies, specialization in industrial electronics, Erasmus+ international exchange program.

Mentor: Prof. Dr. Riko Šafarič

Co-mentor: Assist. Prof. Dr. Suzana Uran

Maribor, August 2020

MATLAB SIMULATION FOR DIFFERENT NUMBER OF PILLARS OF A MICROFLUIDIC ROTATIONAL MOTOR DRIVEN BY CIRCULAR VIBRATIONS

Student: Iker Ruiz de Esquide Crespo

Student program: Engineering in industrial technologies, specialization in industrial electronics, Erasmus+ international exchange program.

Mentor: Prof. Dr. Riko Šafarič

Co-mentor: Assist. Prof. Dr. Suzana Uran

MATLAB SIMULATION FOR DIFFERENT NUMBER OF PILLARS OF A MICROFLUIDIC ROTATIONAL MOTOR DRIVEN BY CIRCULAR VIBRATIONS

Abstract: The purpose of this bachelor thesis is to create a MATLAB code able to simulate the velocity distribution of the liquid of a microfluidic rotational motor for a changeable number of pillars and parameters, so the results of different configurations of the motor can be theoretically shown.

The main idea of this microfluidic rotational motor is inducing a circular vibration to a glass plate with two perpendicular piezoelectric devices, so a circular flux is created in the liquid around the pillar that is glued perpendicular to the plate. This rotational movement is used to drive a rotor floating in the liquid. This motor has already been experimentally tested, but a possible improvement is searched by the change of the individual pillar for a “pot”.

It is especially interesting to observe the velocity distribution of the fluid when there are enough pillars creating a circle, so they overlap and create a pot shape configuration. This is interesting because the pot shape reduces the evaporation of the droplet which has been a persistent problem in the laboratory experimentation and could present improvements with respect to the pillar-motor. With this MATLAB code, it is possible to observe the velocity distribution for different changeable parameters of the motor, such as amplitude and frequency of the circular vibrations, distance between pillars and liquid chosen for the microfluidic motor.

In this bachelor thesis it has been shown that the microfluidic rotational motor with a “pot” has satisfactory theoretical velocity distribution results. A MATLAB code is provided so the user can calculate by itself which will be the theorical velocity distribution for different configurations of the motor.

Key words:

Microfluidic rotational motor, MATLAB code, velocity distribution, micro-size motor, pot shaped microfluidic motor.

INDEX

List of figures

List of symbols

A= amplitude of vibration

a= diameter of the pillar

D= distance between pillars

f= frequency of vibration

N= number of pillars

Ni= kinematic viscosity

List of abbreviations

CCW= counterclockwise

CW= clockwise

UI= user interface

1 INTRODUCTION

1.1 Previous work

This bachelor thesis is based on a microfluidic rotational motor developed by Suzana Uran, Božidar Bratina, Riko Šafarič and several students. The idea of this motor and the results of its experiments are explained in the paper called *Microfluidic Rotational Motor Driven by Circular Vibrations* [1]. The abstract of this article explains:

"The paper presents a rotational micro-sized motor (the diameter of the rotor is 350 µm) driven by low frequency (200–700 Hz) circular vibrations, made by two piezoelectric actuators, through the medium of a water droplet with diameter of 1 mm (volume 3.6 µL). The theoretical model presents how to produce the circular streaming (rotation) of the liquid around an infinitely long pillar with micro-sized diameter. The practical application has been focused to make a time-stable circular stream of the medium around the finite long vibrated pillar with diameter of 80 µm in the presence of disturbances produced by the vibrated plate where the pillar is placed. Only the time-stable circular stream in the water droplet around the pillar produces enough energy to rotate the micro-sized rotor. The rotational speed of the rotor is controlled in both directions from −20 rad/s to +26 rad/s. 3D printed mechanical amplifiers of vibrations, driven by piezoelectric actuators, amplify the amplitude of the piezoelectric actuator up to 20 µm in the frequency region of 200 to 700 Hz." [1]

Since the aim of this bachelor thesis is to create a MATLAB code for the simulation of the microfluidic rotational motor for N number of pillars, the computer simulation will be based on the theoretical model presented in the paper *Microfluidic Rotational Motor Driven by Circular Vibrations* [1]. That it is to say, a theoretical model of the motor for an infinitely long pillar with micro-sized diameter.

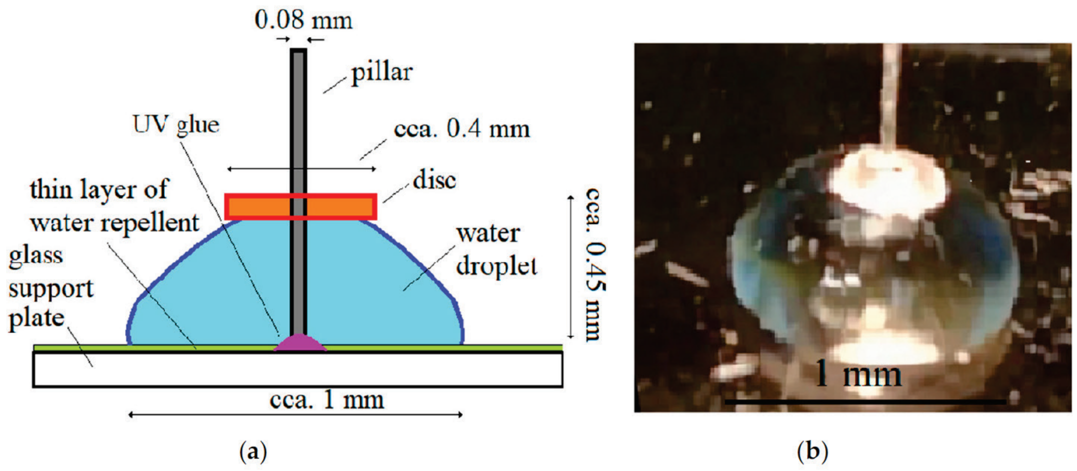


Figure 1-1: The microfluidic motor: (a) the scheme; (b) the photo. [1]

1.2 Motivation for the creation of the code

The main motivation of the creation of the MATLAB code for the simulation of the microfluidic vibrational motor for a changeable number of pillars is observing the velocity distribution of the motor’s fluid when there are enough pillars so they overlap and create a pot shape. This pot shape is achieved when it is not possible for the fluid to go between pillars. Obviously, for a greater number of pillars, a more precise pot shape will be obtained, but more computation time will be needed for the simulation.

In Figure 1-2 it is compared the pot shape for two different amounts of pillars. The left configuration shows 9 pillars and the right one shows 50. This figure is shown just as an example where the distance between pillars is 120 µm and the radius of the pillars is 40 µm. The number of pillars from which the pot shape starts to be created depends on the distance between pillars and the radius of these.

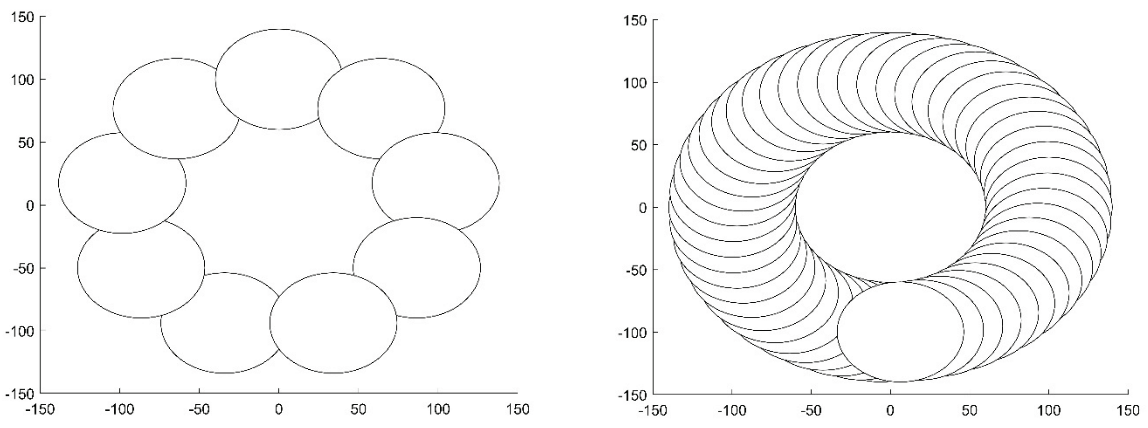


Figure 1-2: Pot shape example for 9 and 50 pillars

This pot shape is especially interesting for the motor configuration because it reduces the evaporation of the water droplet which has been a persistent problem in the laboratory when testing the motor experimentally and could present improvements with respect to the pillar-motor.

With the development of this code it will be possible to modify some parameters of the pot shaped motor such as the fluid used, the diameter of the “pot” and the frequency and amplitude of the circular vibrations. This will allow to choose some proper configurations for the experimentation in the laboratory.

In addition, it is also interesting the developing of this code to observe which is the velocity distribution of the motor’s fluid when several non-overlapped pillars are implemented in the motor, but this is not the main objective because its practical application it is not interesting in this case.

1.3 Objectives

The objective of this bachelor thesis is to create a MATLAB code which can show the velocity distribution of the microfluidic rotational motor`s fluid for a changeable number of pillars and changeable parameters. This code is based on the mathematical model descripted in the paper *Microfluidic Rotational Motor Driven by Circular Vibrations* [1] and the solution of this mathematical model is calculated with the code written by Prof. Dr. Riko Šafarič that it is attached to this document in the function *SimulationOnePillar*.

The changeable parameters are the following ones:

1. Liquid used in the microfluidic rotational motor.
2. Number of pillars.
3. Radius of the pillar/s.
4. Distance between pillars.
5. Frequency of the circular vibrations.
6. Amplitude of the circular vibrations.

The response of the motor is represented by two figures showing the velocity distribution of the fluid: one represents the tangential velocity and the other one the rotational velocity. The objective is to obtain similar figures to the shown in the article of Takeshi Hayakawa, Shinya Sakuma and Fumihito Arai: On-chip 3D rotation of oocyte based on a vibration-induced local whirling flow [2] – but in this case for a changeable number of pillars and changeable parameters.

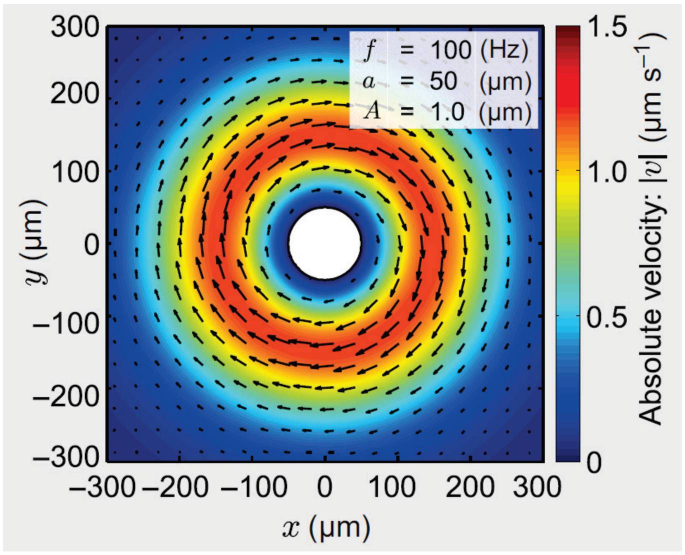


Figure 1-3: Graph type objective - Calculated flow velocity distribution [2]

Additionally, the MATLAB code must be efficient so the user can change the parameters of the motor and obtain a smooth update of velocity distribution figures.

2 THEORETICAL MODEL

The theoretical model used for the simulation of the microfluidic rotational motor of a unique pillar has been the one presented in the paper *Microfluidic Rotational Motor Driven by Circular Vibrations* [1], which it is based on Hayakawa, Sakuma, Fukuhara, Yokoyama and Arai's article [3]. The theorical model represents the induced flow around a micropillar when circular vibration is applied.

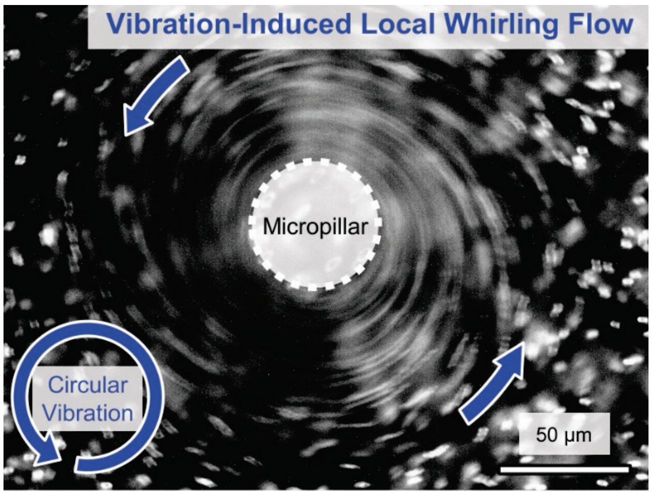


Figure 2-1: Vibration-induced local whirling flow around the micropillar [3]

They develop the model from the basic differential equations for the motion of an incompressible viscous fluid in two-dimensional space that can be written as:

$$\nabla^4\Psi - \frac{1}{\eta}\frac{\partial}{\partial t}\psi = \frac{1}{\eta}\mathrm{V}\cdot\nabla(\nabla^2\psi) = \frac{u}{\eta}\frac{\partial}{\partial x}\nabla^2\psi + \frac{v}{\eta}\frac{\partial}{\partial y}\nabla^2\psi \qquad (1)$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \qquad (2)$$

Here, Ψ is a stream function, η is the kinematic viscosity, v is the velocity vector, u and v are the velocity components in the x and y directions, respectively. [3]

If the equation is solved by assuming the convection term in the right side is smaller than the other terms:

$$\nabla^4\psi^{(0)} - \frac{1}{\eta}\frac{\partial}{\partial t}\psi^{(0)} = 0 \qquad (3)$$

Where $\psi^{(0)}$ *indicates the solution of zeroth order Equation (3).* [3]

For a circular vibration $A_0[\cos\omega t\hat{x} + \cos(\omega t \pm \pi)\hat{y}]$, the boundary conditions at the exterior part of the pillar and at an infinite distance from it can be written as:

$$\mathrm{v_r}|_{\mathrm{r=a}} = \mathrm{v_\theta}|_{\mathrm{r=a}} = 0 \tag{4}$$

$$\mathrm{v_r}|_{\mathrm{r}\to\infty} = \omega \mathrm{A_0}\ [\cos\theta\cos\omega t\ +\ \sin\theta\cos(\omega t \pm \pi)] \tag{5}$$

$$\mathrm{v_\theta}|_{\mathrm{r}\to\infty} = -\omega \mathrm{A_0}\ [\sin\theta\cos\omega t\ -\ \cos\theta\cos(\omega t \pm \pi)] \tag{6}$$

In these equations, v_r and v_θ are the velocity components in the r and θ directions, respectively, a is the radius of the micropillar, A_0 is the amplitude of vibration and ω is the angular frequency of the vibration. The positive (negative) sign of the phase indicates anticlockwise (clockwise) circular vibration. [3]

Solving the previous equation, the following zeroth order solution is obtained:

$$\psi^{(0)} = \omega \mathrm{A_0 a}\left[\sin\theta\mathbf{e}^{-\mathbf{i}\omega t} - \cos\theta\mathbf{e}^{-\mathbf{i}(\omega t\pm\pi)}\right]\left(\frac{2\mathrm{Y}}{\epsilon} - \frac{\mathrm{r}}{\mathrm{a}} - \frac{\mathrm{a}}{\mathrm{r}}\mathrm{C}\right) + \text{Complex conjugate} \tag{7}$$

Then, the method of successive approximation is used to treat the nonlinear convection term in Equation (1). By substituting the solution of Equation (7) into the right side of Equation (1), it is possible to obtain the first order equation:

$$\nabla^4\psi^{(1)} - \frac{1}{\eta}\frac{\partial}{\partial \mathrm{t}}\psi^{(1)} = \frac{1}{\eta}\mathrm{v}^{(0)}\ \cdot\ \nabla(\nabla^2\psi^{(0)}) \tag{8}$$

Developing this equation, it can be obtained the mathematical model of the velocity of the circular flux depending on the distance to the pillar. The model is descripted by the following equations:

$$\psi_{st} = r^4\left(\frac{1}{48}\int_a^r \frac{1}{x}\rho(x)dx + C1\right) + r^2\left(-\frac{1}{16}\int_a^r x\rho(x)dx + C2\right)$$
$$+\left(\frac{1}{16}\int_a^r x^3\rho(x)dx + C3\right) + \frac{1}{r^2}\left(-\frac{1}{48}\int_a^r x^5\rho(x)dx + C4\right) \quad (9)$$

Where:

$$\rho(x) = \frac{2\pi^3 f^3 A^2}{\eta^2}\left(2Y + 2Y^* - 2\frac{a^2}{r^2}\ CD - 2\frac{a^{\ 2}}{r^2}\ C^*Z - 4YY^* + 4ZZ^*\right) \quad (10)$$

$$C_1 = -\frac{1}{48}\int_a^\infty \frac{1}{x}\rho(x)dx \quad (11)$$

$$C_2 = \frac{1}{16}\int_a^\infty x\rho(x)dx \quad (12)$$

$$C_3 = \frac{a^4}{16}\int_a^\infty \frac{1}{x}\rho(x)dx - \frac{a^2}{8}\int_a^\infty x\rho(x)dx \quad (13)$$

$$C_4 = -\frac{a^6}{24}\int_a^\infty \frac{1}{x}\rho(x)dx - \frac{a^4}{16}\int_a^\infty x\rho(x)dx \quad (14)$$

Where the Y, D, C and Z abbreviation functions are expressed by Hankel functions of the first kind $H^{(1)}$ and second kind $H^{(2)}$ with the next equations:

$$Y = \frac{H_0^{(1)}(\varepsilon.x)}{H_0^{(1)}(\varepsilon.a)}, Z = \frac{H_2^{(1)}(\varepsilon.x)}{H_0^{(1)}(\varepsilon.a)}, C = \frac{H_2^{(1)}(\varepsilon.a)}{H_0^{(1)}(\varepsilon.a)}, D = \frac{H_2^{(2)}(\varepsilon z.x)}{H_0^{(1)}(\varepsilon.a)} \quad (15)$$

Where

$$\varepsilon = \left(i\frac{2\pi f}{\eta}\right)^{\frac{1}{2}}, \varepsilon z = \left(-i\frac{2\pi f}{\eta}\right)^{\frac{1}{2}} \quad (16)$$

With these equations, the tangential velocity v_t and a rotational velocity v_r of the rotational flux around the pillar can be calculated as:

$$v_t = \frac{\delta\psi}{\delta r}\ ,\ v_r = \frac{1}{r}\frac{\delta\psi}{\delta r} \quad (17)$$

The equations of this mathematical model can be solved with MATLAB in the function *SimulationOnePillar* and it is used for the simulation of the velocity distribution of the motor`s fluid in the MATLAB code created for this bachelor thesis. The code of this function has been written by Prof. Dr. Riko Šafarič based on the mathematical model's equations that had been exposed.

The function *SimulationOnePillar has as inputs:*

- Number of calculations (j)
- Differential radial distance for the calculations (dr)
- Frequency of circular vibrations (f)
- Radius of the pillar (a)
- kinematic viscosity (ni)
- Amplitude of circular vibrations (A)

And as outputs:

- Distance from the pillar (vr1)
- Tangential speed vector depending on distance vr1 (vv)

Then, with the function *SimulationOnePillar* it is possible to quickly calculate theoretically the tangential and rotational velocity of the liquid stream around the pillar against the distance from the pillar of the microfluidic rotational motor of a unique pillar with some specific parameters: frequency and amplitude of the circular vibrations, radius of the pillar and the kinematic viscosity of the fluid.

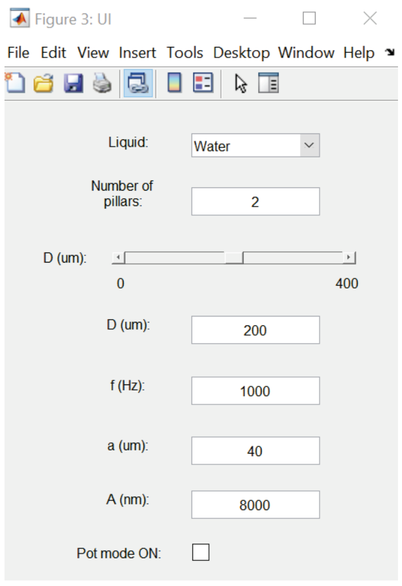
3 EXPLANATION OF THE CODE

The code creates two figures that represent the velocity distribution of the circular flux of the fluid around the pillar/s of the motor: one for the tangential velocity and the other one for the rotational velocity. The pillars are illustrated with a white circle, the "pot" with a white annulus and the circular flux of the fluid is represented in two different ways:

1. Colored backwound showing the magnitude of the velocity vector in each position. The value of each color is written in the colormap.
2. Vectors that show the sense, direction and magnitude (bigger or smaller arrow) of the velocity for each position represented.

A third figure is created for the user interface. The UI enables the user to change the following characteristics for the simulation of the motor:

7. Liquid used for the simulation (Liquid) - changes the kinematic viscosity used for the calculations. Water, Mercury or Galinstan can be chosen.
8. Number of pillars.
9. Distance between pillars (D) - diameter of the circumcircle of a regular polygon formatted by the vertices located in the central part of each pillar minus two times the radius of the pillar. Values from 0 to 400 µm can be chosen with the slider bar or an exact number can be written in the textbox. Units: µm.
10. Frequency of the circular vibrations (f). Units: Hz.
11. Radius of the pillar/s (a). Units: µm.
12. Amplitude of the circular vibrations (A). Units: nm.
13. Pot mode ON – It is the mode of display (PotOn). If selected, a "pot" will be displayed instead of the pillars.



Liquid: Water
Number of pillars: 2
D (um): 200
f (Hz): 1000
a (um): 40
A (nm): 8000
Pot mode ON: ☐

Figure 3-1: UI window example

3.1 Explanation of the Main code

The main code is structured in the following sections:

1. Initial configuration and simulation of the motor with one pillar.
2. Calculation of the position of the pillar/s.
3. Calculation of the position of the origin of the velocity vectors.
4. Calculation of the components of the velocity vectors for the previously calculated positions.
5. Interpolation of tangential and rotational velocity values and creating a mesh grid.
6. Sum of the velocity components in the mesh grid and calculation of the magnitude of the vectors.
7. Plots of tangential and rotational velocities.
8. User Interface.

3.1.1 Initial configuration and simulation of the motor with one pillar

Different necessary variables are defined and the calculations for the simulation of the motor with one pillar are done. These calculations can be represented in two plots: rotational velocity against distance from the pillar and tangential velocity against distance from the pillar. This graphs are not plotted by the original program because it is not considered necessary. The code for plotting these functions is commented inside the function *SimulationOnePillar*.

As an example, these two plots are represented for the following characteristics:

- Liquid: water.
- Frequency of the circular vibrations: 1 kHz.
- Amplitude of the circular vibrations: 8 µm.
- Radius of the pillar: 40 µm.
- 300 µm of simulation.

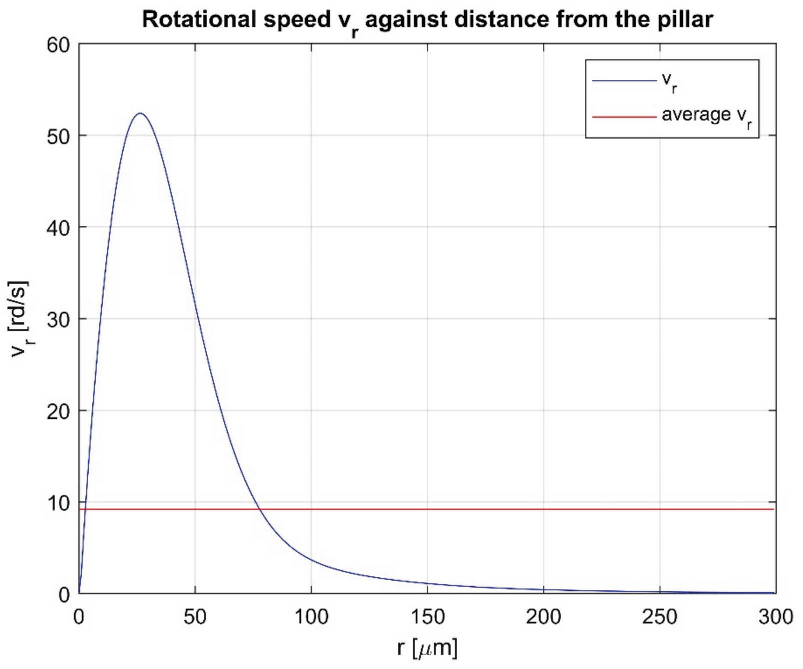


Figure 3-2: SimulationOnePillar – Rotational speed against distance from the pillar - example

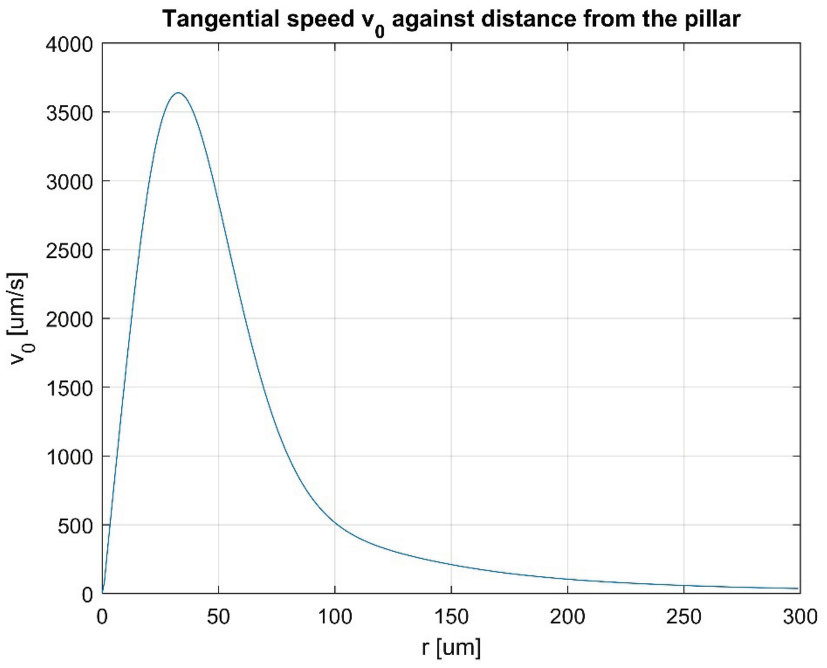


Figure 3-3: SimulationOnePillar – Tangential speed against distance from the pillar - example

3.1.2 Calculation of the position of the pillar/s

The x and y coordinates of the position of the center of the pillars are calculated for the number N of pillars selected by the user. For N>2, this is achieved with the position of the vertices of the regular polygon created by the MATLAB's function *nsidedpoly* [4].

This function has as inputs for creating the regular polygon: the number of vertices, the position of the center of the polygon (for this code it will always be [0,0]) and the length of the sides of the polygon. Since we are interested in having and specific diameter of the circumcircle, the side of the polygon must be determined by the following relation:

$$s = D1 * \sin \frac{\pi}{N} \qquad [5]$$

Where:

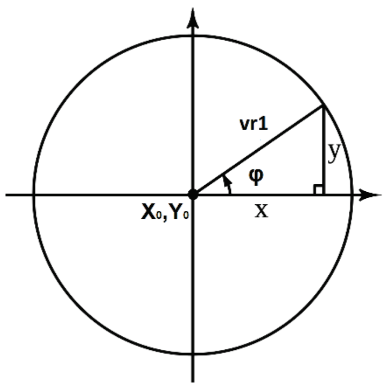
s = side of the regular polygon.

D1 = diameter of the circumcircle.

N = Number of pillars.

3.1.3 Calculation of the position of the origin of the velocity vectors

The x and y coordinates of the position of the origin of the velocity vectors are calculated based in the previously defined position of the center of the pillar/s, the angle ϕ (phi) and the distance vr1 (distance from the center of the pillar to the velocity vector).



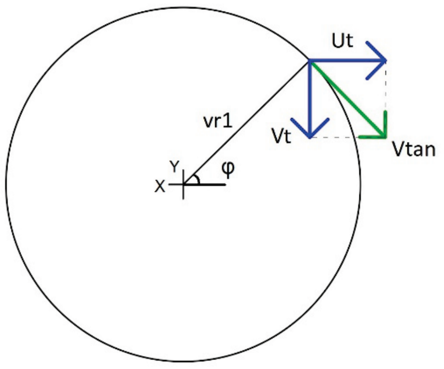
```
%Coordiantes of the velocity vectors
for k=1:N
    for i=1:length(vr1)
        for j=1:length(phi)
            X{k}(i,j)=xp(k)+vr1(i)*cos(phi(j)); %x-coordiante of the
                                                 %velocity vector
            Y{k}(i,j)=yp(k)+vr1(i)*sin(phi(j)); %y-coordinate of the
                                                 %velocity vector
        end
    end
    %Reshape matrices in a column vector
    Xcol{k}=reshape(X{k},[],1);
    Ycol{k}=reshape(Y{k},[],1);
end
```

Figure 3-4: Calculation of the position of the origin of the velocity vector

3.1.4 Calculation of the components of the velocity vectors for the previously calculated positions

Firstly, the row vector that contains the magnitude of the velocity depending on the distance from the pillar is converted into a matrix in which this vector in column form is repeated for each value of the angle phi. In this way we have a magnitude of the velocity for each position previously calculated (obviously the values of the rows of this matrix – values of the magnitude of the velocity for the same distance and different angle – will be equal, but this will be useful when the sum of velocity components of different pillars is needed).

Then, the values of the x and y components of the velocity are calculated for each element of the previously created matrix. Since the velocity vector (Vtan in Figure 3-5) is always tangent to the radial distance from the center of the pillar and the sense of the flow will be CW, x and y components – Ut and Vt respectively – are calculated in the following way:



```
%u and v components of the tangential velocity (CW rotation)
for i=1:length(vr1)
    for j=1:length(phi)
            Ut(i,j)=Vtan(i,j)*cos(pi/2-phi(j));
            Vt(i,j)=-Vtan(i,j)*sin(pi/2-phi(j));
    end
end

%u and v components of the rotational velocity (CW rotation)
Vr1_ext=repmat(vr1',1,length(phi));
Ur=Ut./Vr1_ext;
Vr=Vt./Vr1_ext;
```

Figure 3-5: Calculation of the components of the velocity vectors for the previously calculated positions

As it can be appreciated in Figure 3-5, the calculations are firstly done for the tangential velocity and after, the velocity components are divided by the radial distance, so the components of the rotational velocity are achieved.

3.1.5 Interpolation of tangential and rotational velocity values and creating a mesh grid

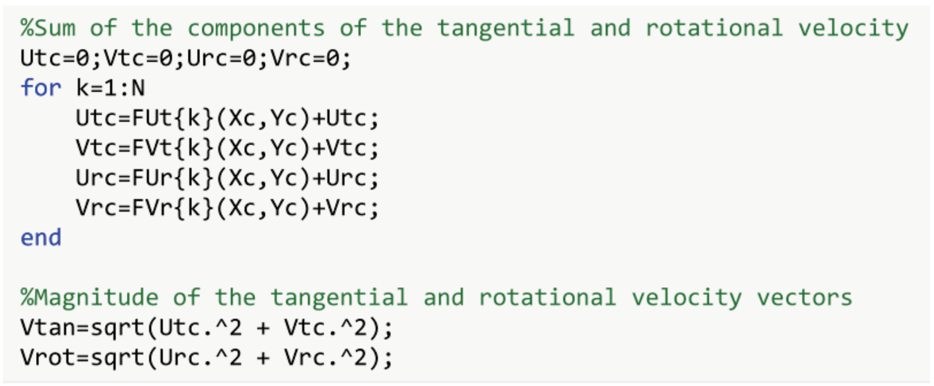
The MATLAB's function *scatteredInterpolant* [6] is used for the linear interpolation of the components of the velocity. This function is descripted in MathWorks' page as: "*Use scatteredInterpolant to perform interpolation on a 2-D or 3-D data set of scattered data. scatteredInterpolant returns the interpolant F for the given data set. You can evaluate F at a set of query points, such as (xq,yq) in 2-D, to produce interpolated values vq = F(xq,yq)*" [6].

Then, a mesh grid is created with the MATLAB's function *meshgrid* [7], where x and y minimum and maximum limits depend on the distance between pillars – a bigger mesh grid is created when a bigger distance between pillars is selected. These values had been chosen so they are big enough to have a proper mesh grid when velocities are summed. Additionally, 100 subdivisions per dimension had been chosen in this code. The number of subdivisions per dimension can be increased so the calculations are more precise, but the code will be slower, and the change will not be noticeable.

3.1.6 Sum of the velocity components in the mesh grid and calculation of the magnitude of the vectors

The value of the velocity (tangential and rotational) components are calculated in each point of the mesh grid for each pillar and then the velocity components of different pillars in the same position of the mesh grid are summed.

When the velocity components are calculated, these are used to calculate the magnitude of the rotational and tangential velocities for every position of the mesh grid.



```
%Sum of the components of the tangential and rotational velocity
Utc=0;Vtc=0;Urc=0;Vrc=0;
for k=1:N
    Utc=FUt{k}(Xc,Yc)+Utc;
    Vtc=FVt{k}(Xc,Yc)+Vtc;
    Urc=FUr{k}(Xc,Yc)+Urc;
    Vrc=FVr{k}(Xc,Yc)+Vrc;
end

%Magnitude of the tangential and rotational velocity vectors
Vtan=sqrt(Utc.^2 + Vtc.^2);
Vrot=sqrt(Urc.^2 + Vrc.^2);
```

Figure 3-6: Sum of the velocity components in the mesh grid and calculation of the magnitude of the vectors

Figure 3-7 is depicted as an example of the correct result of the vector sum. In this figure, (Xc,Yc) are the points created by the mesh grid and N the number of pillars of the motor.

The sum of velocity vectors of different pillars for the same position of the mesh grid can be easily seen in the Figure 3-7. This example shows the motor with two pillars and a distance (D) of 130 µm between them. The colormap is different from the one of the original code so the vectors can be seen more easily.

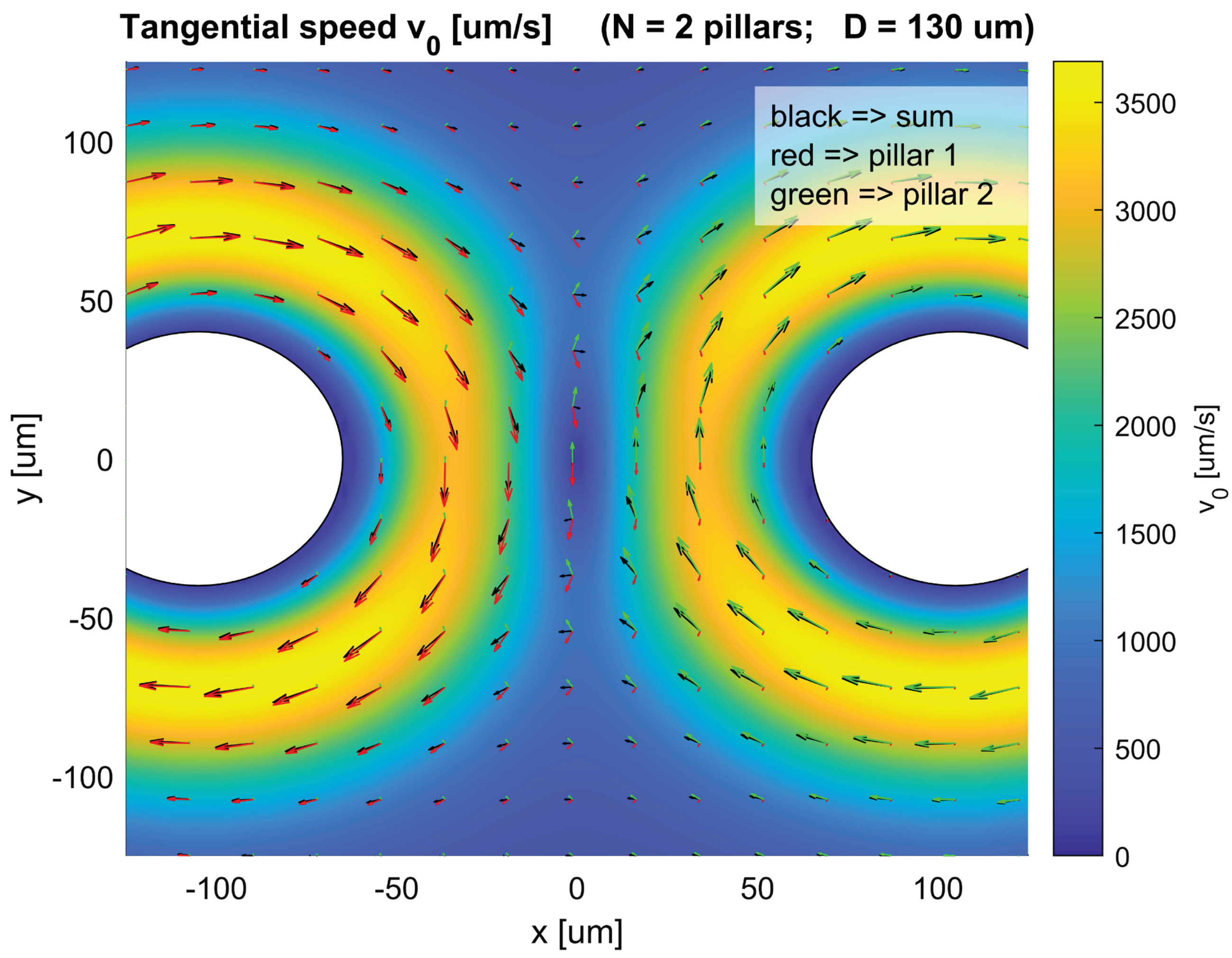


Figure 3-7: Sum of the velocity vectors for 2 pillars – example

3.1.7 Plots of tangential and rotational velocities

The figures representing the rotational and tangential velocities are mainly formed of four different plots based on MATLAB's functions. These are plotted in the next order:

1. *Contourf* function, which is a *"filled 2-D contour plot"* [8] – creates the background colors that represent the magnitude of the velocity.
2. *Quiver* function, which *"plots vectors as arrows at the coordinates specified in each corresponding pair of elements in x and y"* [9].
3. Circles that represent the pillars and had been created through the function *rectangle* that *"creates rectangle with sharp or curved corners"* [10].
4. Annulus that represent the "pot" and has been created through the function *patch* that *"plots one or more filled polygonal regions"* [11] and the function *line* [12] that draws the inner circumference of the annulus.

3.1.8 User Interface

The user interface has been created fully with the MATLAB's function *uicontrol* [13]. It is composed of:

1. One pop up menu for choosing the liquid.
2. One slider bar for choosing the distance between pillars.
3. Five editable text fields for choosing the values of the distance between pillars, the radius of the pillar, the amplitude and frequency of the circular vibrations and the number of pillars.
4. One checkbox for activating or deactivating the "Pot mode".

When some of these values are changed by the user in the UI, one specific callback function is called for updating the variable changed.

3.2 Explanation of the parameter updating functions

The functions created for updating a variable (a, A, D, f, liquid, N and PotOn) are just callback functions that make possible to update the value of the variable modified by the user in the UI.

This type of function works as follows:

1. Declares as global variables the variables that are going to be updated.
2. Reads the variable introduce by the user.
3. Loads necessary variables from the workspace.
4. Updates other variables if needed.
5. Runs the function *RepeatCalculations* that it is a code that repeats the calculations for the new parameter introduced by the user.
6. Updates the UI if needed.

4 PROBLEMS OF THE CODE AND POSSIBLE IMPROVEMENTS

The main problem of this code it is that the simulations are not accurate enough when the distance of the pillars is too small for some specific parameters of the motor. This is because the sum of the velocities of different pillars is done in the complete mesh grid regardless of which are the positions of the pillars. In other words, the circular flux of the fluid around each pillar interacts with the other circular fluxes caused by the other pillars but not with the pillars themselves. Since the flux does not interact with them, some velocities will appear inside the pillars and that is obviously incorrect.

If the pillars are separated the enough distance for the selected parameters, this effect will be neglectable. This is because the velocity of the stream will be small at the distance where it should interact with the pillar.

The distance at which that effect starts to be neglectable it is different for every alternative configuration of the motor set by user in the UI. Despite that, it is simple for the user to identify if the simulation works well or not: the flux must be circular around each pillar - excluding when there are plotted so many pillars that there is not space between them so a circular flux can appear around each pillar. When the pillars overlap – the "pot" is formatted – the velocities in the surface of the "pot" must be zero or at least, small to consider valid the results. Even more, the same must happen in the surface of the pillars when the "pot" is not created.

Some non-null velocities appear inside the pillars also because of the interpolation, but this velocities are very small and neglectable.

The white circles are always plotted above the *quiver* (plot of the arrows in the figure) so the small velocities that appear inside the pillars - that obviously should not exist - cannot be seen because the white circle that represents the pillar is above them. This velocities are not erased in the code because it is much more efficient for it just to hide them below the pillars than setting that velocities to zero. The same happens when "Pot mode" is selected. Just to represent graphically this problem, the pillars had been plotted below the vector field (*quiver*) for the following two simulations:

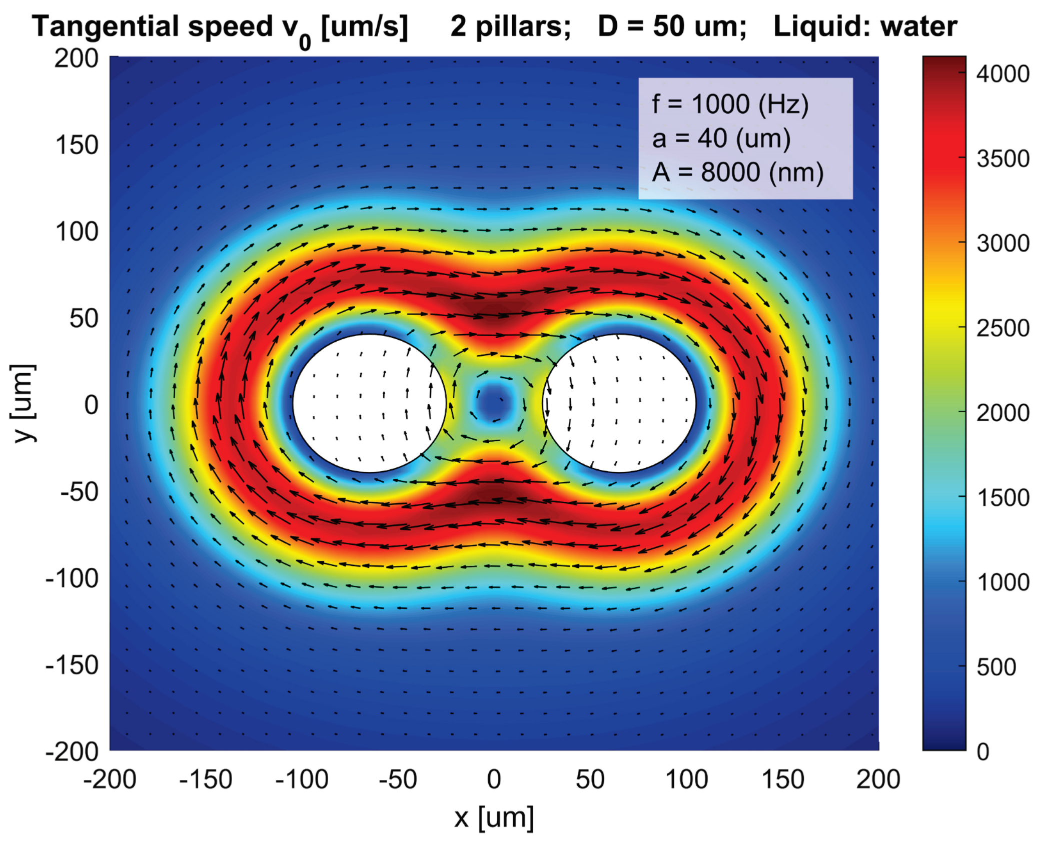


Figure 4-1: Problem representation – not neglectable

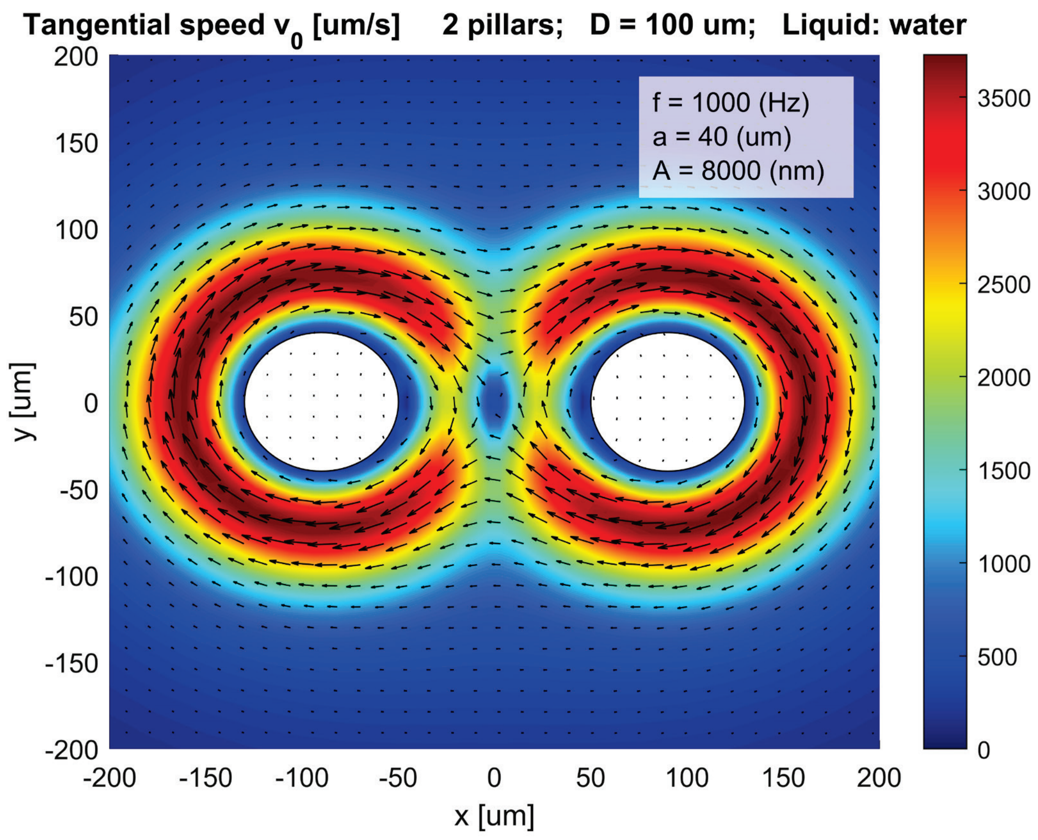


Figure 4-2: Problem representation – neglectable

As it can be seen in Figure 4-1, a circular flux is not present around each pillar, so this shows that the distance between the pillars is too small to obtain an accurate simulation. It can also be seen that big velocities appear inside the pillars, which it is not neglectable and shows that this simulation is not precise.

On the other hand, Figure 4-2 clearly shows a circular flux around each pillar and the velocities inside of the pillars are very small and neglectable. Because of that, it can be said that this is a valid simulation.

Known this problem, these are the possible improvements suggested for the presented code:

- Set to zero all the velocities that are inside any pillar. This is not implemented in the original code because it is inefficient in this case, is better just to hide that velocities below the pillars. These can be useful only if more accurate numerical results are needed. This will obtain better numerical results but will not fix the problem previously exposed.
- Increase the number of calculations done per dimension. This will lead to more precise results, but the simulation will be slower. This is not necessary for the purpose of the original code because more precise calculations will not lead into noticeably better graphic representations of the velocity distribution of the circular flux. However, this will not fix the problem that has been previously exposed.
- Simulate the impact of the fluid with the pillars so more accurate simulations are achieved and even small distances between pillars and low frequencies would be simulated precisely.
- If more accurate results are needed for the pot simulations, it could probably be necessary to develop a mathematical model specific for the rotational flow inside the pot due to the circular vibrations.

5 SIMULATIONS

Different significant simulations will be shown so the response of the microfluid rotational motor can be displayed for different configurations of 1, 2, 3, 6 and 50 pillars.

5.1 1 pillar (f=1kHz, a=40um, A=8um, Water)

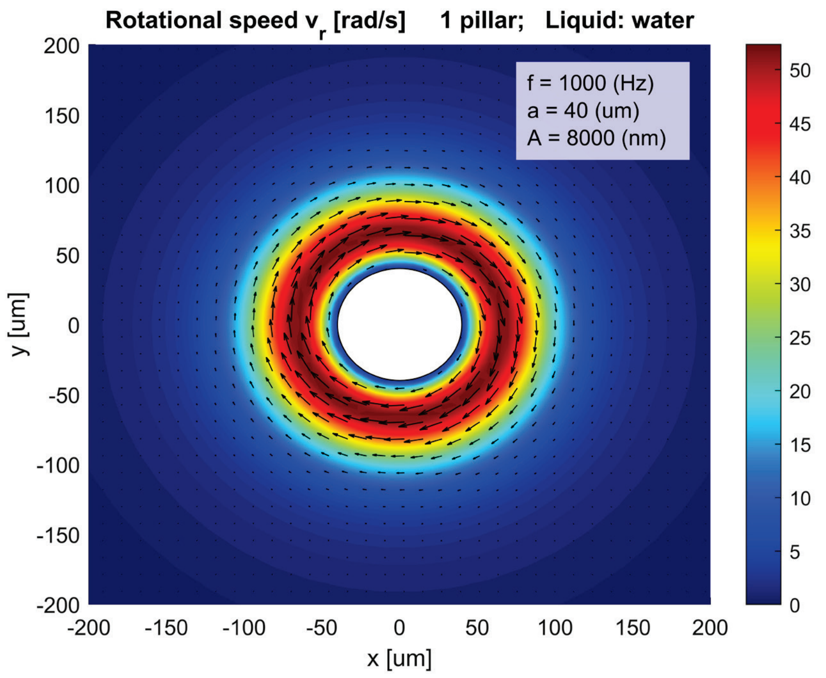


Figure 5-1: Rotational speed distribution - One pillar

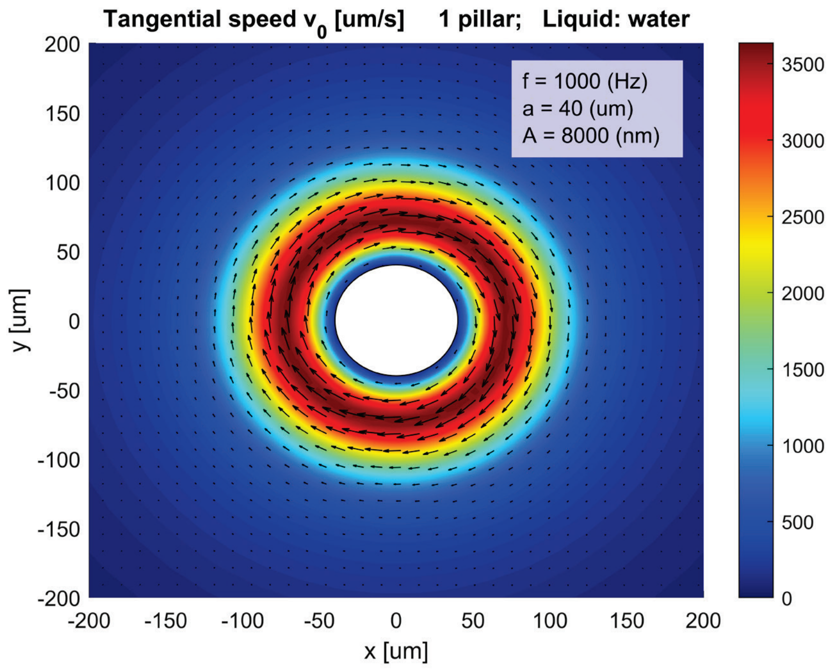


Figure 5-2: Tangential speed distribution - One pillar

These figures show graphically the results of the mathematical model of the microfluidic rotational motor.

5.2 2 pillars (D=120um, f=1kHz, a=40um, A=8um, Water)

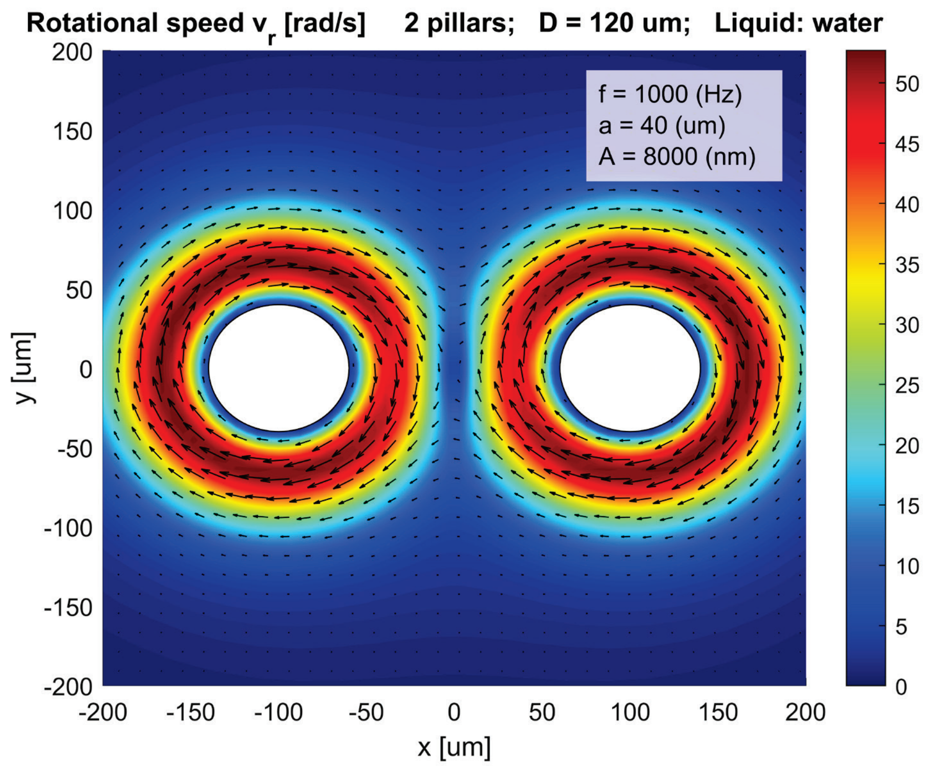


Figure 5-3: Rotational speed distribution - Two pillars

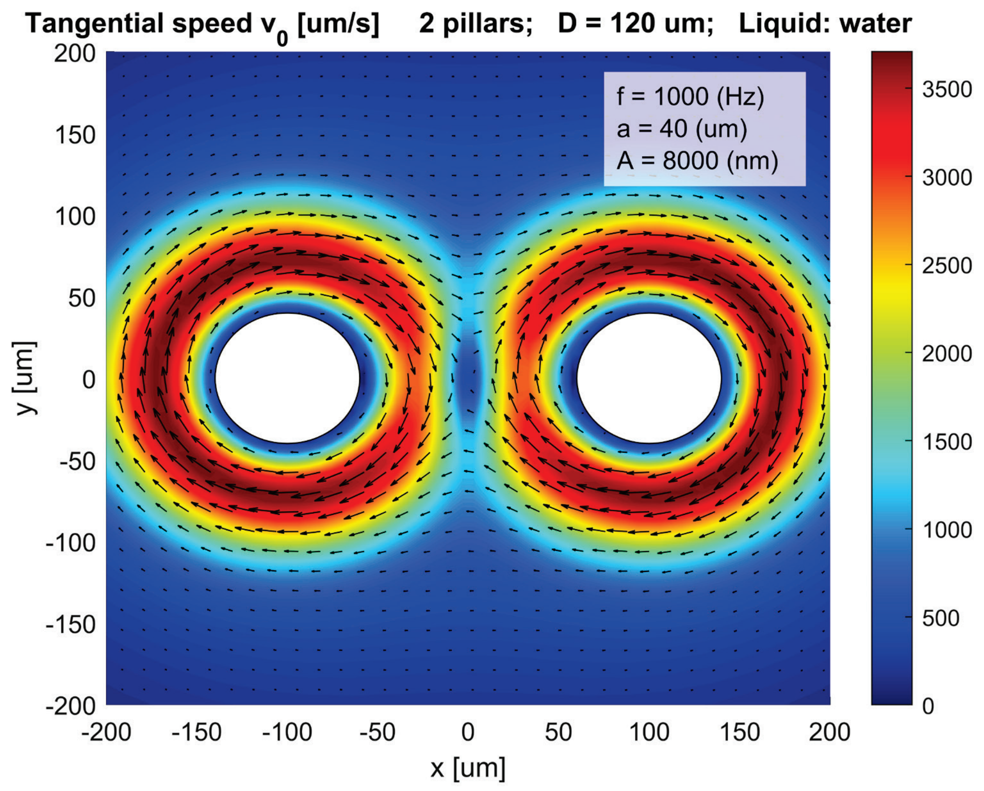


Figure 5-4: Tangential speed distribution - Two pillars

5.3 3 pillars (D=120 um, f=1kHz, a=40um, A=8um, Water)

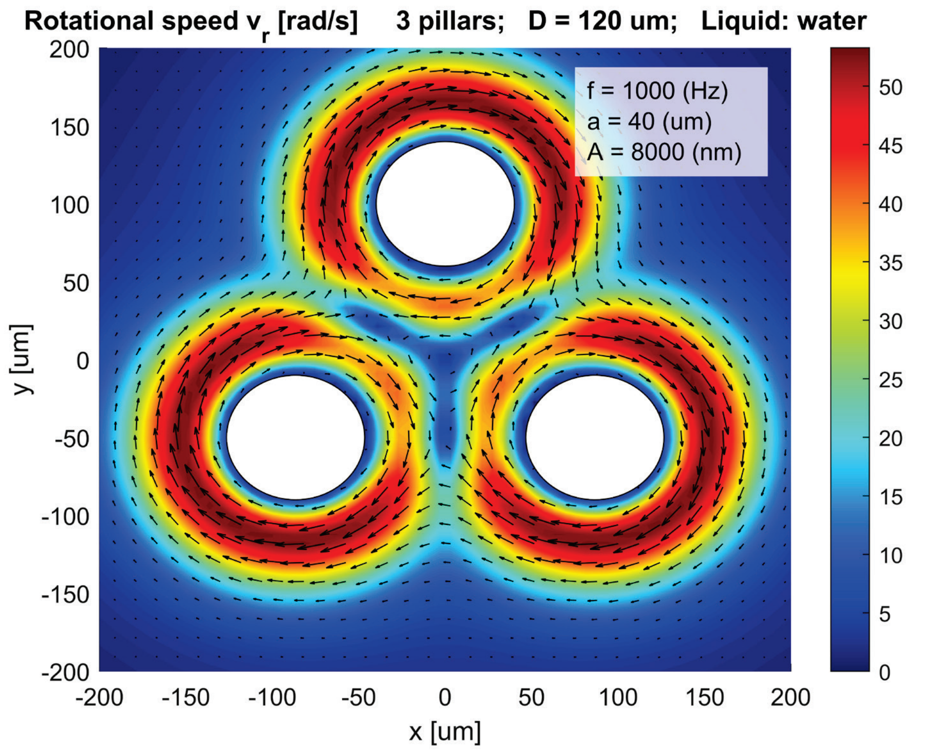


Figure 5-5: Rotational speed distribution - Three pillars

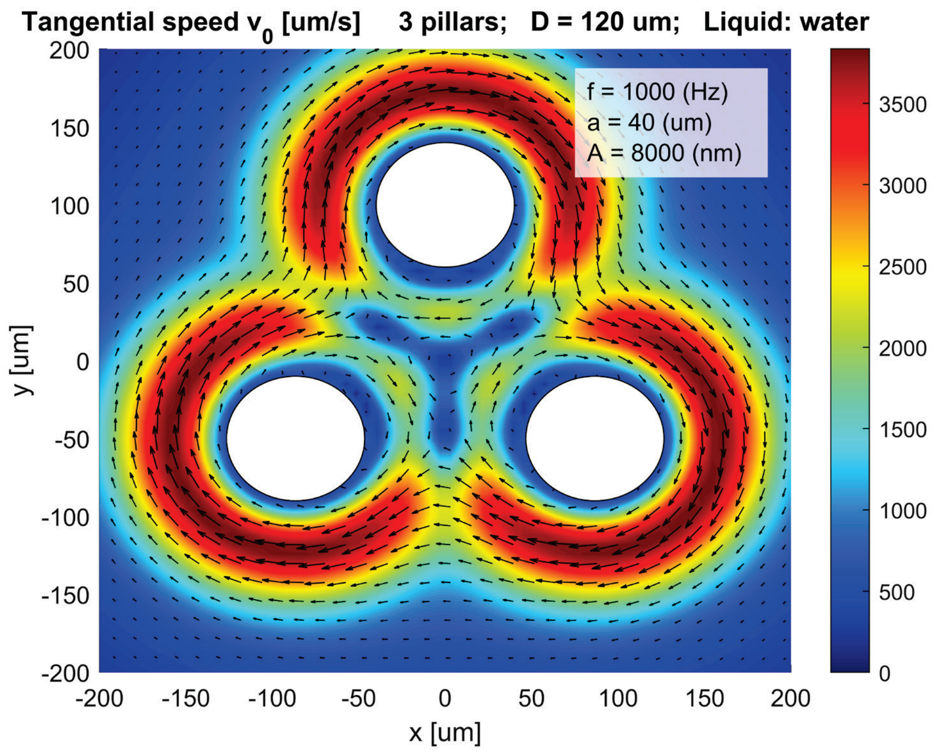


Figure 5-6: Tangential speed distribution - Three pillars

5.4 Comments of simulations for 2 and 3 pillars

In the simulations for 2 pillars and 3 pillars it can be observed that the velocity is much smaller in the middle part of the distance between pillars. This is because the velocity components are cancelled due to the presence of the other pillars.

As expected, the rotational velocity and the tangential velocity vectors only differ in its magnitude.

It can also be shown that the maximum velocities increase slightly when the number of pillars is increased, but also some low velocity zones are created. Because of this not-uniform velocity distribution, these configurations are not a good choice for the design of the motor.

Nevertheless, these results show that the code simulates the velocity distribution correctly.

5.5 6 pillars – not accurate result (D=120 um, f=1kHz, a=40um, A=8um, Water)

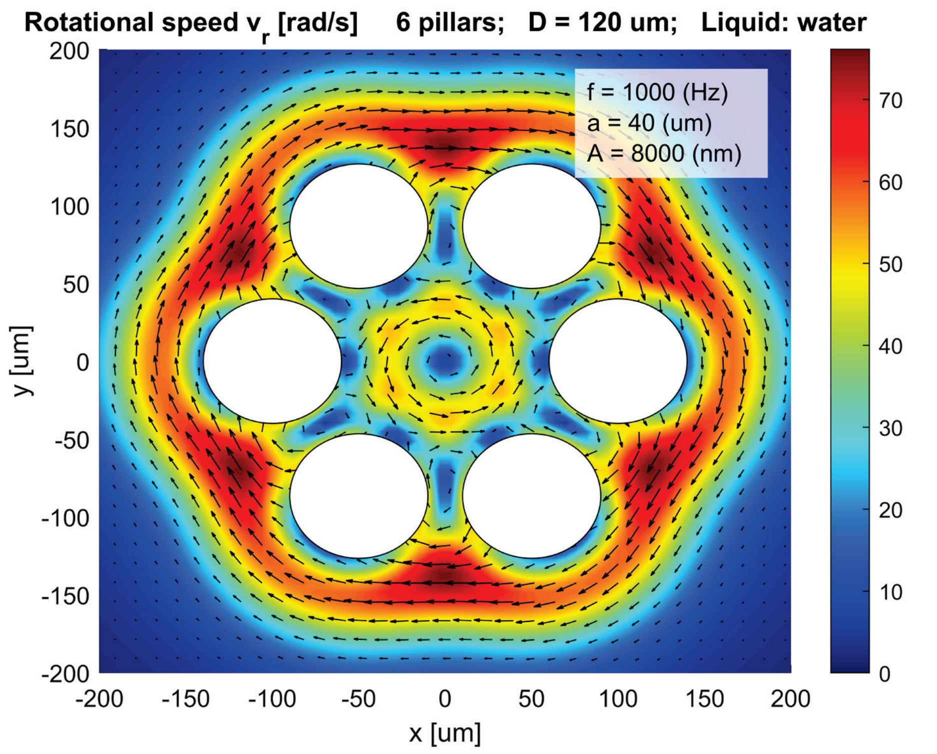


Figure 5-7: Rotational speed distribution. Not accurate - Six pillars

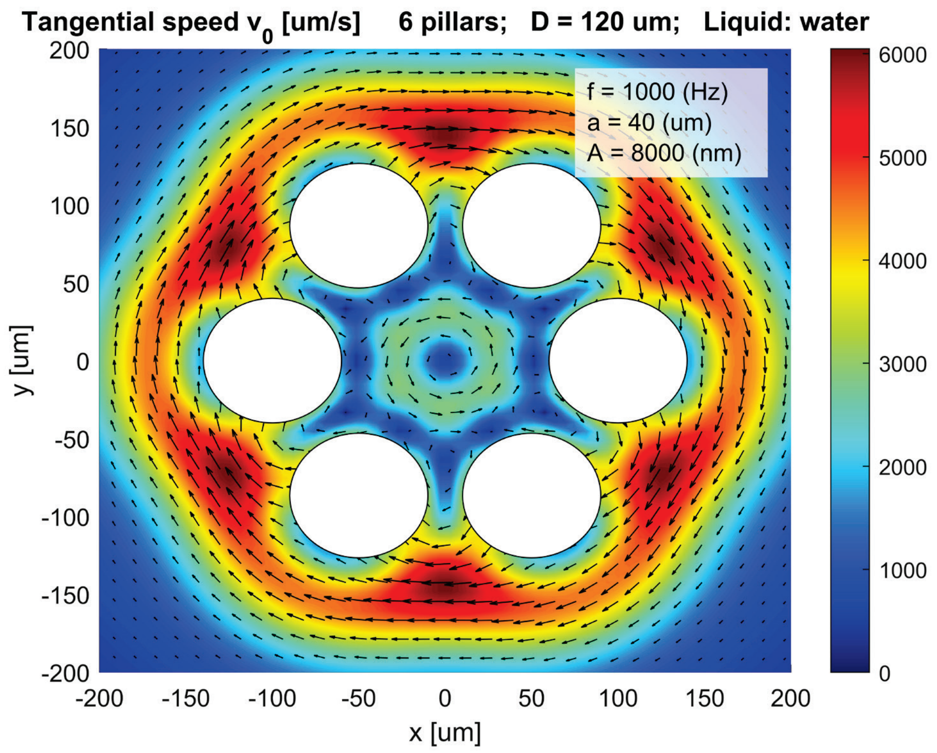


Figure 5-8: Tangential speed distribution. Not accurate - Six pillars

5.6 6 pillars (D=220 um, f=1kHz, a=40um, A=8um, Water)

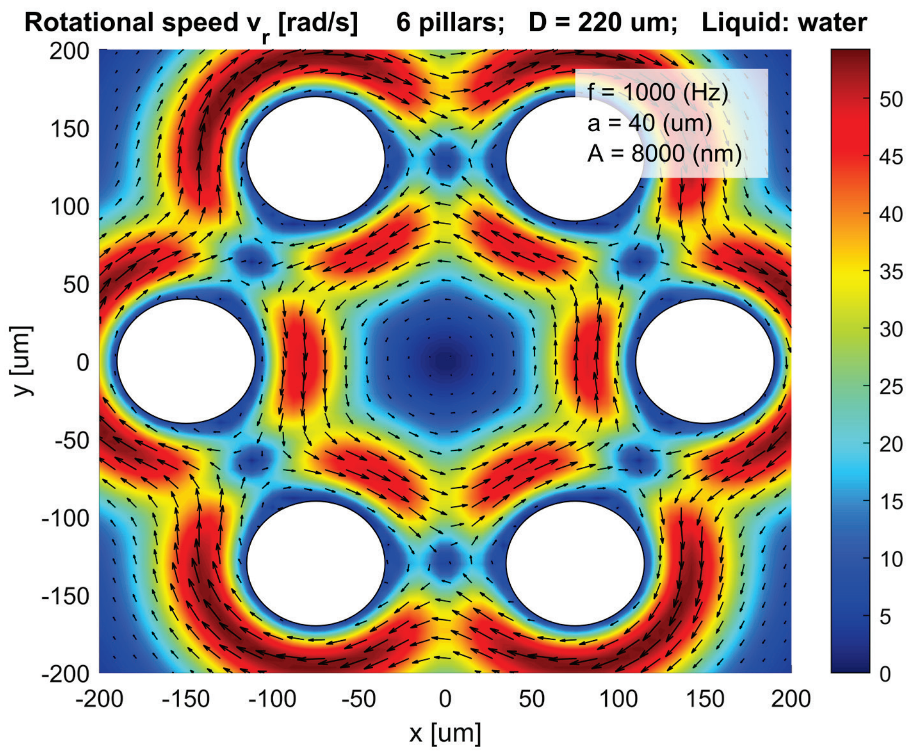


Figure 5-9: Rotational speed distribution - Six pillars

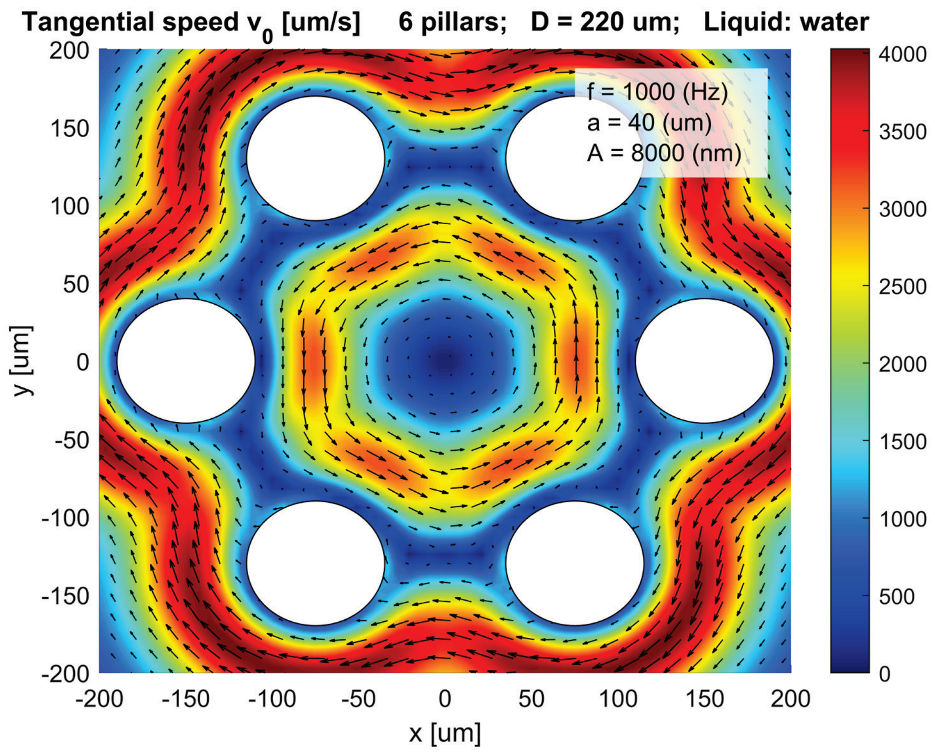


Figure 5-10: Tangential speed distribution - Six pillars

5.7 Comments of simulations for 6 pillars

It can be clearly seen in Figure 5-7 and in Figure 5-8 that the simulated velocity distribution is not correct. This is because the velocity in the surface of the pillar should be zero or at least very small to consider the simulation valid, and in this case the velocity is high in the surface of the pillar.

The reason of this error is explained in PROBLEMS OF THE CODE AND POSSIBLE IMPROVEMENTS section.

For showing a valid simulation of the motor with 6 pillars, the distance between pillars had been increased 100 µm in Figure 5-9 and Figure 5-10.

These last two figures show a more precise simulation. It can be extracted the same conclusions as in the simulations for 2 pillars and 3 pillars . Additionally, in this simulation, it can be seen a hexagon form velocity distribution between the pillars. This is a satisfactory result because it shows that if the number of pillars is infinite (a “pot”) a circular velocity distribution will be obtained inside the “pot”.

It is interesting to notice that the sense of the flow of the fluid in the zone between the pillars (CCW) is opposite to the sense of flow outside that zone and to the sense of flow of the liquid with a unique pillar (CW).

5.8 Pot1 (50 pillars) (D=120 um, f=1kHz, a=40um, A=8um, Water)

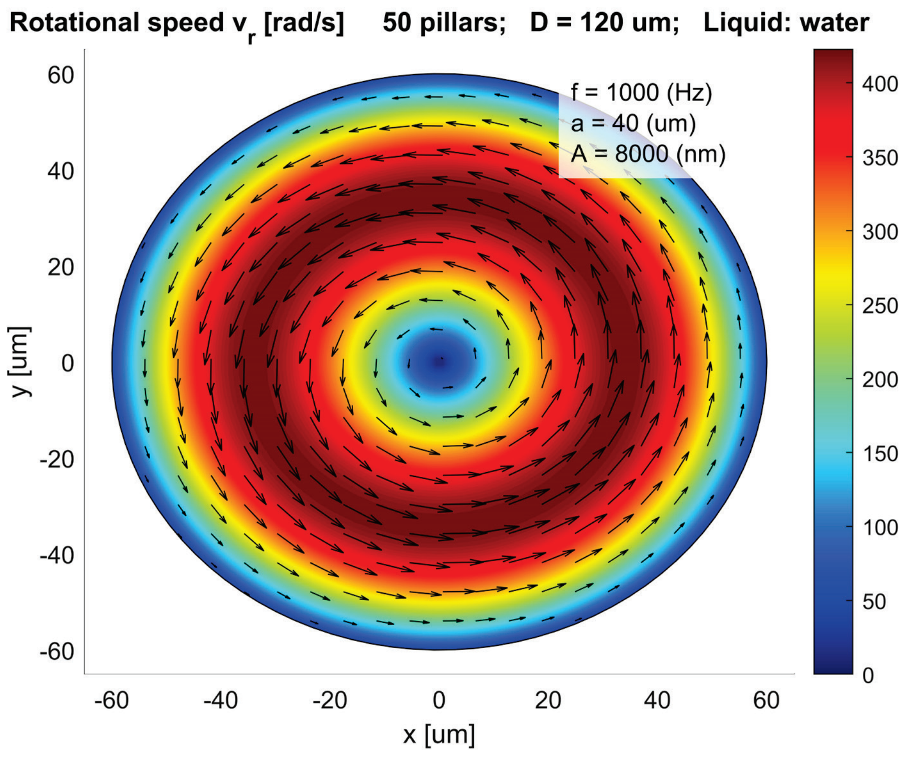


Figure 5-11: Rotational speed distribution - Pot1 (50 pillars)

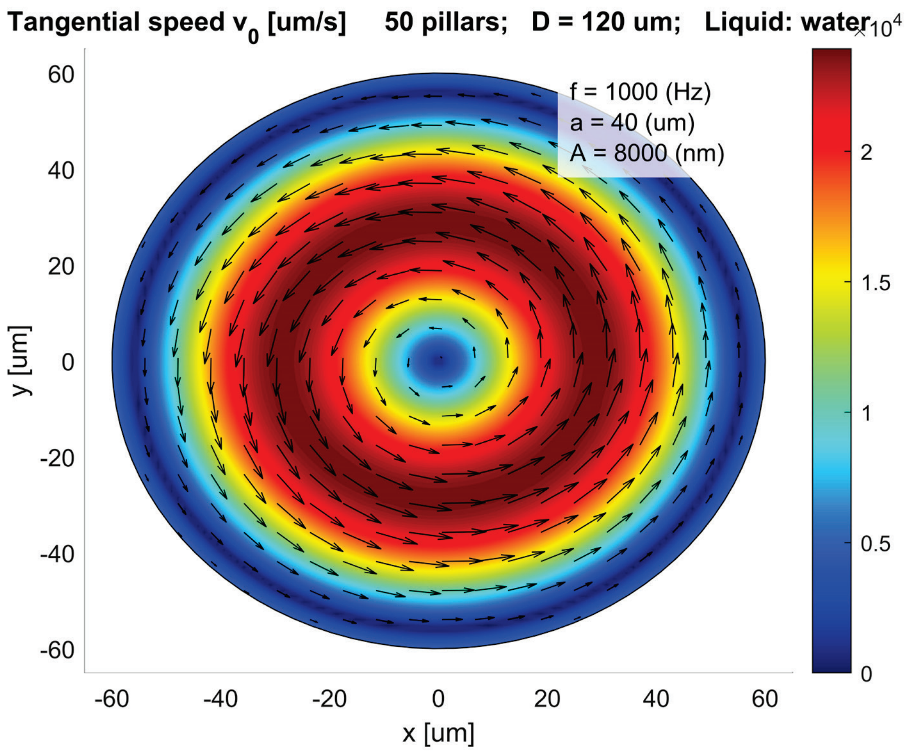


Figure 5-12: Tangential speed distribution - Pot1 (50 pillars)

5.9 Pot2 (50 pillars) (D=200 um, f=1kHz, a=40um, A=8um, Water)

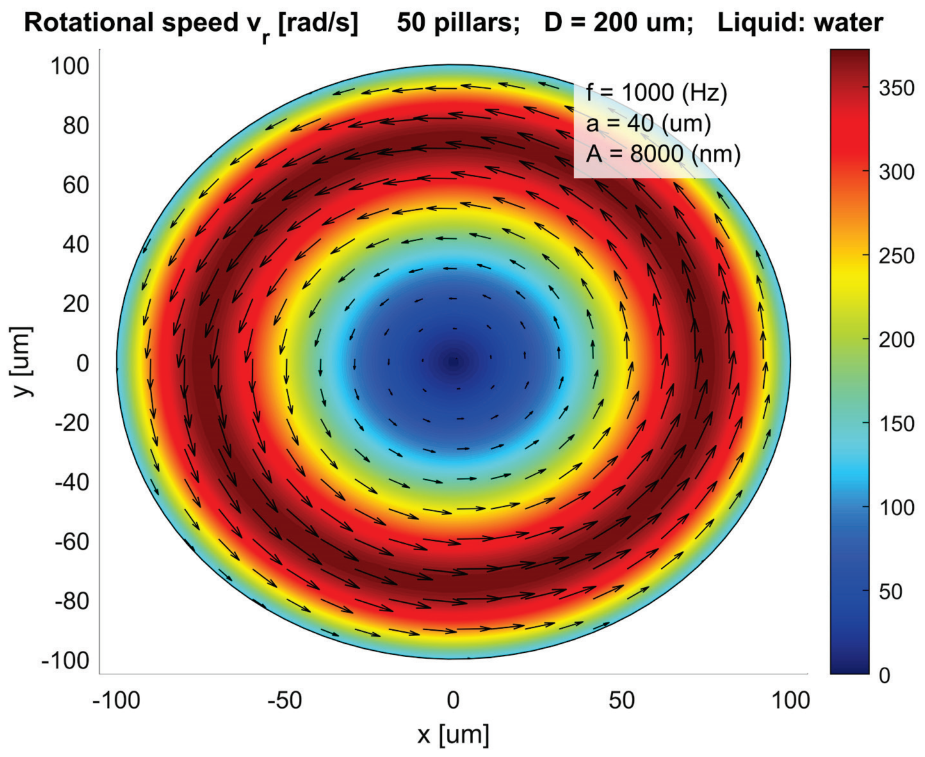


Figure 5-13: Rotational speed distribution - Pot2 (50 pillars)

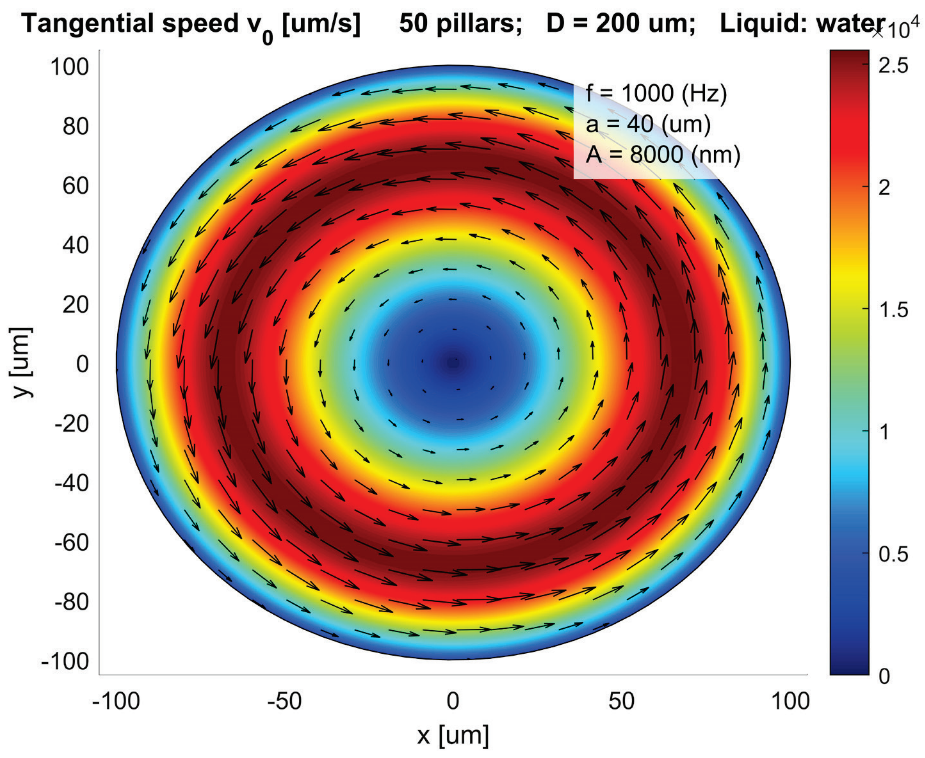


Figure 5-14: Tangential speed distribution - Pot2 (50 pillars)

5.10 Pot3 (50 pillars) (D=120 um, f=1kHz, a=40um, A=192nm, Water)

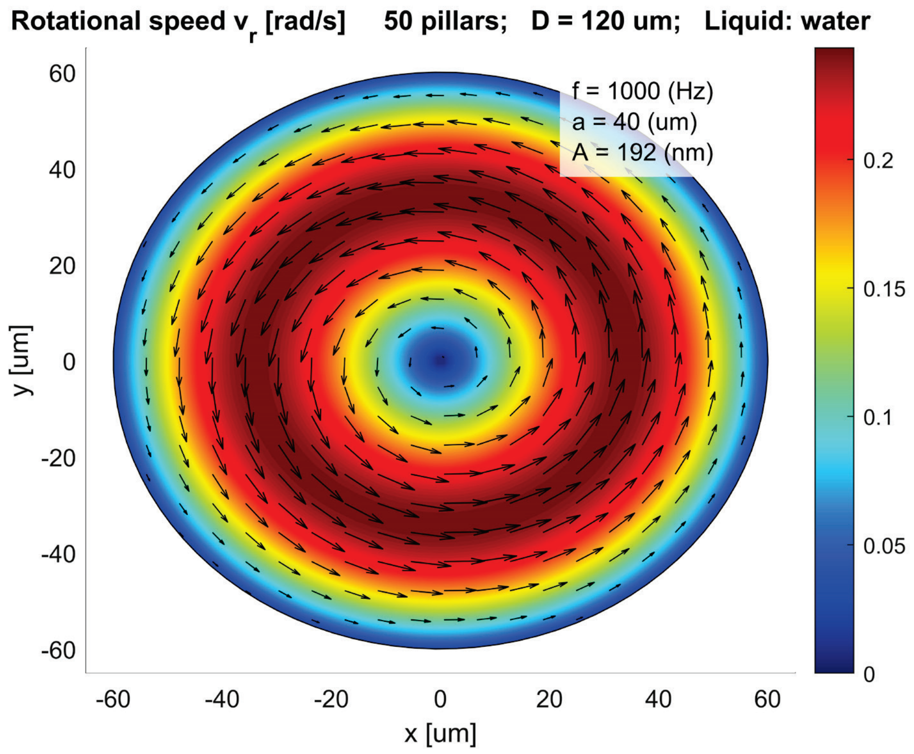


Figure 5-15: Rotational speed distribution - Pot3 (50 pillars)

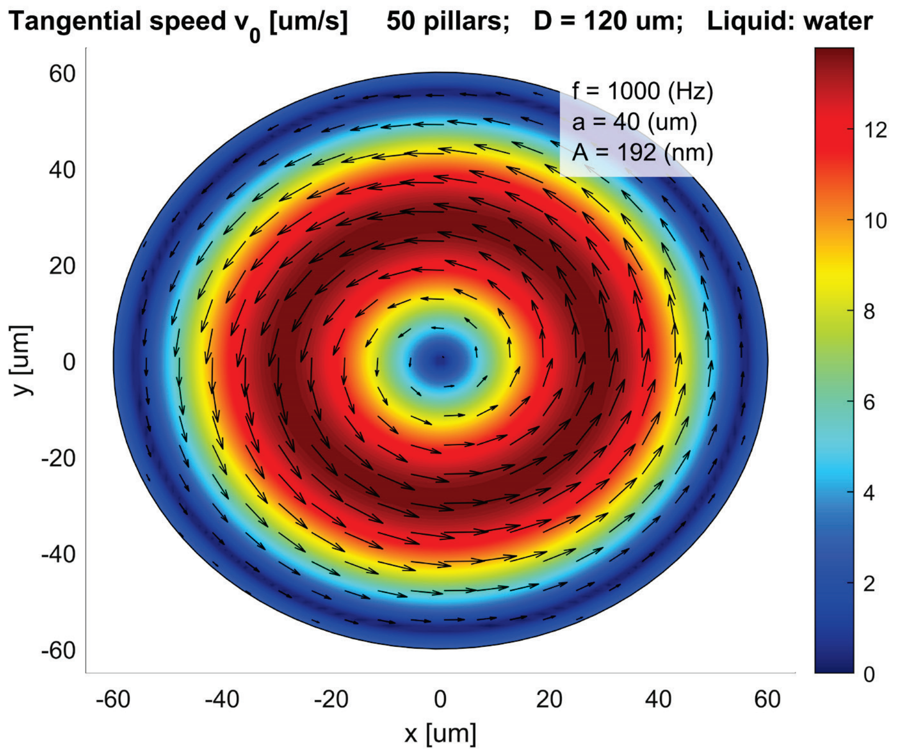


Figure 5-16: Tangential speed distribution - Pot3 (50 pillars)

5.11 Pot4 (50 pillars) (D=150 um, f=800Hz, a=40um, A=8um, Water)

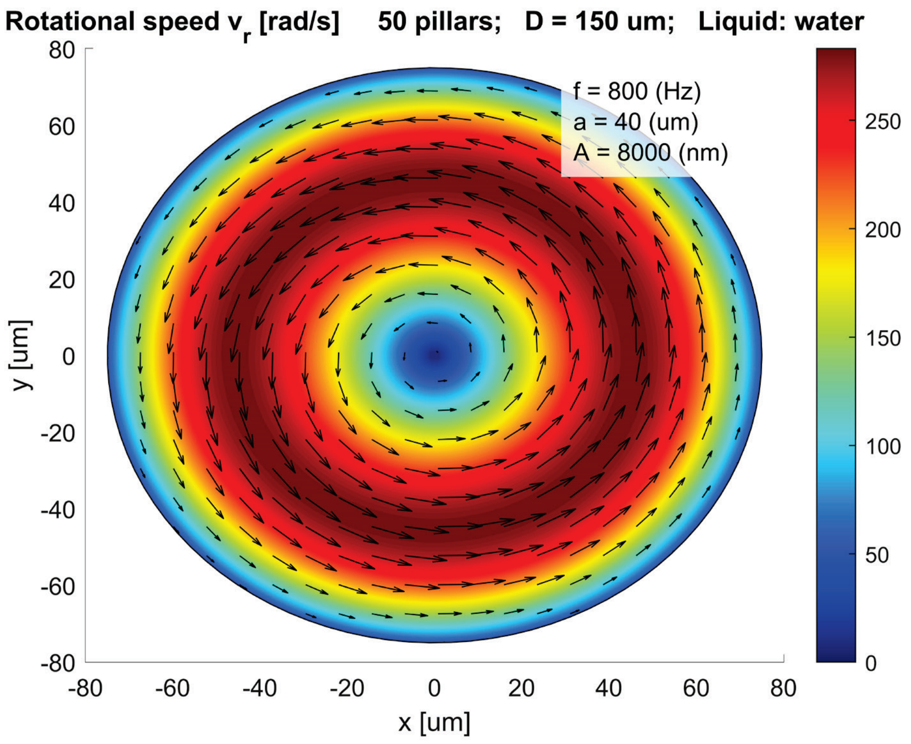


Figure 5-17: Rotational speed distribution - Pot4 (50 pillars)

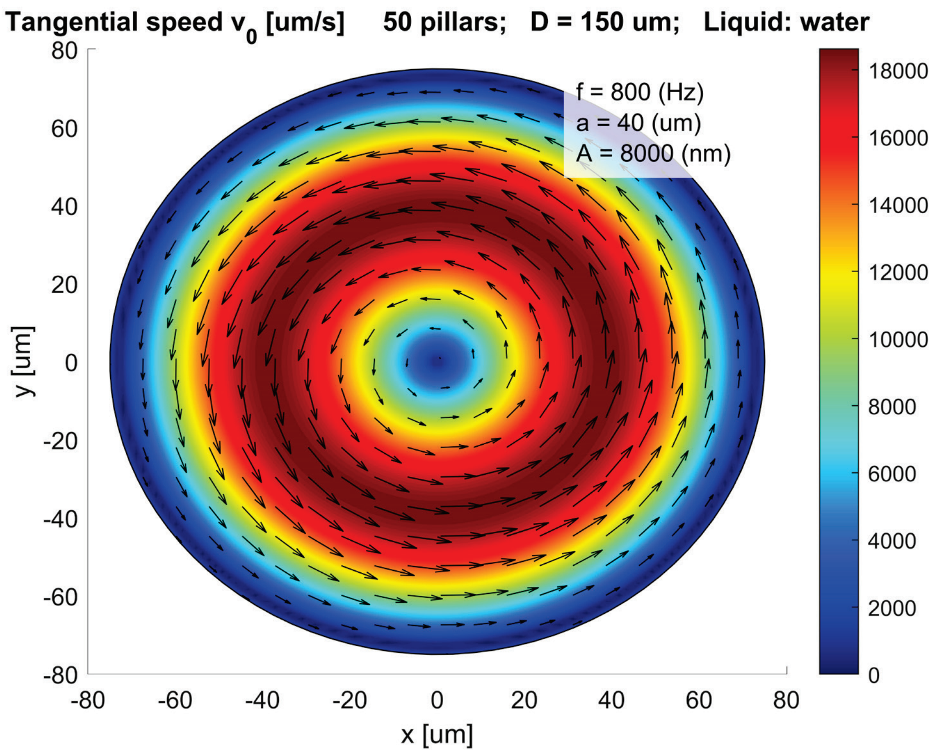


Figure 5-18: Tangential speed distribution - Pot4 (50 pillars)

5.12 Pot5 (50 pillars) (D=120 um, f=1kHz, a=40um, A=8um, Hg)

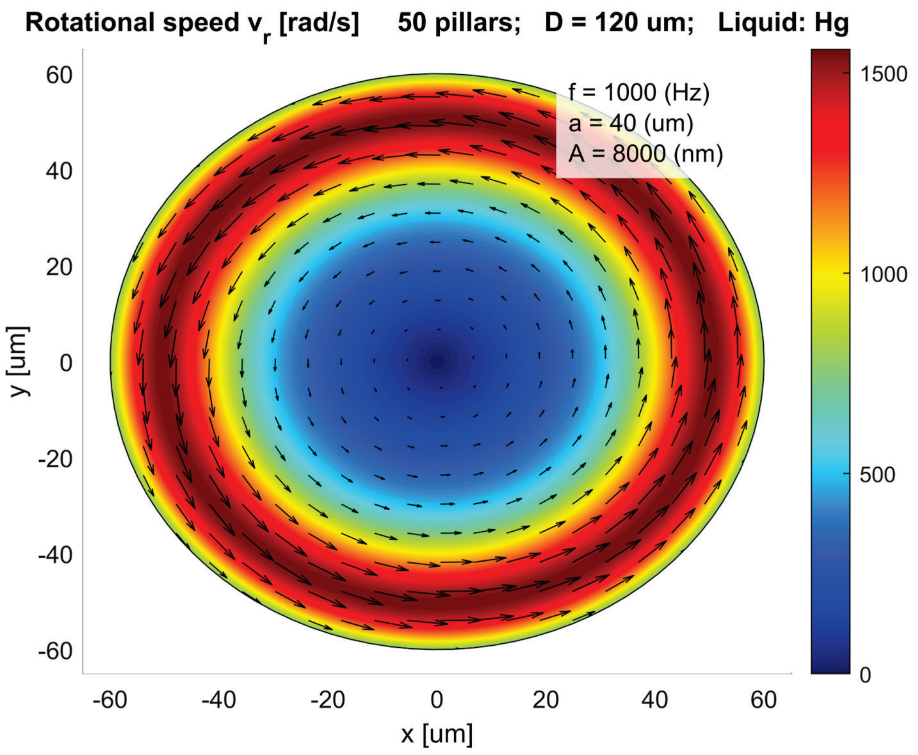


Figure 5-19: Rotational speed distribution - Pot5 (50 pillars)

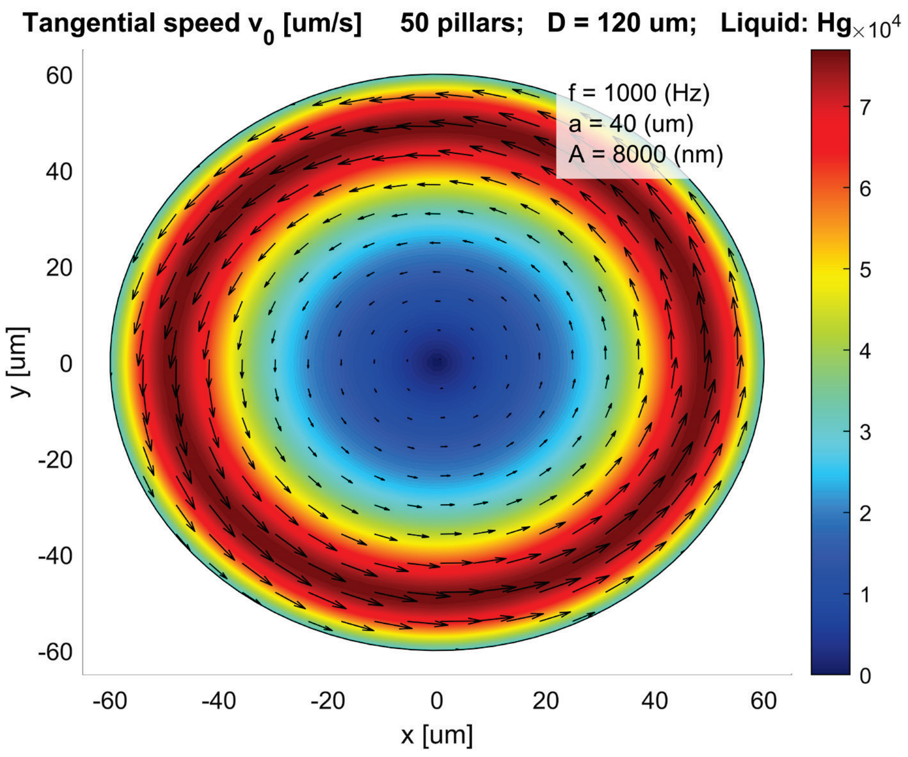


Figure 5-20: Tangential speed distribution - Pot5 (50 pillars)

5.13 Comments of simulations for the “pot” (50 pillars)

The velocity distribution simulation in the “pot” has been done considering a big number of pillars – 50 pillars – so the results can be as accurate as possible.

It is worth noticing that when simulating the pot shaped motor, the diameter of the pot is equal to the distance between pillars. Even more, it is interesting to notice that if a big enough number of pillars is chosen, the radius (a) of each pillar does not have any influence on the simulation of the pot shaped motor.

It has been shown 5 different configurations of the motor with the “pot”. The first one showing the shape of the velocity distribution and the others showing the impact of changing each parameter:

- Pot1:

  It shows a circular flow inside the “pot” like the one created with a unique pillar but in the opposite sense (CCW) and with higher velocity values. The velocity is null – actually, is nearly null – in the middle point and in the surface of the “pot”. It is a satisfactory result.

- Pot2 – Change in the distance between pillars:

  Basically, from a certain distance that depends on the parameters of the motor, the bigger the distance between pillars, the bigger the central low velocity circular shape zone and the smaller the maximum velocity.

- Pot3 – Change in the amplitude of the circular vibrations:

  It affects only the magnitude of the velocity. The bigger the amplitude, the bigger the magnitude of the velocity.

- Pot4 – Change in the frequency of the circular vibrations:

  Increasing the frequency implies increasing the velocity and getting closer to the surface of the “pot” the highest velocity circular zone.

  * In this simulation the distance between pillars has been increased to 150 µm because with a distance of 120 µm and the chosen parameters, the results obtained were not precise due to the problem explained in PROBLEMS OF THE CODE AND POSSIBLE IMPROVEMENTS section.

- Pot5 – Change in the liquid:

The mercury compared to the water archives bigger velocities and gets closer to the surface of the "pot" the highest velocity circular zone.

It can be observed that the velocity is not null in the surface of the "pot" like it should. This shows that the simulation is not totally accurate, but the results are good enough for getting a qualitative conclusion.

6 CONCLUSIONS

It has been developed a valid MATLAB code for the qualitative representation of the velocity distribution of the microfluidic rotational motor studied.

This code confirms theoretically that the “pot” is a good option to include in the motor since it obtains similar results to the motor with a unique pillar but with higher velocity and the opposite sense of the flow. The code can also be used as a tool for testing which would be the best parameters for experimenting with this motor in the laboratory.

As it has been demonstrated previously in this thesis, the code is not totally accurate and when the distance between pillars is too small (relative to the parameters of the motor) the results can´t be taken as valid.

Nevertheless, it provides good qualitative results and it is an easy code to work with thanks to the user interface. More accurate results need the developing of a mathematical model specific for the rotational flow inside the pot due to the circular vibrations, or another more accurate method of simulation of the velocity distribution. However, the results obtained with the code presented in this thesis are valid as a qualitative approximation of the velocity distribution of this microfluidic rotational motor.

6.1 Effects of parameter alteration in the pot shaped motor

- Distance between pillars:

Basically, from a certain distance that depends on the parameters of the motor, the bigger the distance between pillars, the bigger the central low velocity circular shape zone and the smaller the maximum velocity.

- Amplitude of the circular vibrations:

It affects only the magnitude of the velocity. The bigger the amplitude, the bigger the magnitude of the velocity.

- Frequency of the circular vibrations:

Increasing the frequency implies increasing the velocity and getting closer to the surface of the “pot” the highest velocity circular zone.

- Liquid:

The mercury compared to the water archives bigger velocities and gets closer to the surface of the “pot” the highest velocity circular zone.

7 BIBLIOGRAPHY

[1] B. B. a. R. Š. Suzana Uran, "A Microfluidic Rotational Motor Driven by Circular Vibrations," *Micromachines,* 2019.

[2] S. S. a. F. A. Takeshi Hayakawa, "On-chip 3D rotation of oocyte based on a vibration-induced local whirling flow," *Nature,* 2015.

[3] S. S. T. F. Y. Y. a. F. A. Takeshi Hayakawa, "A Single Cell Extraction Chip Using Vibration-Induced Whirling Flow and a Thermo-Responsive Gel Pattern," *Micromachines,* 2014.

[4] MathWorks, "MATLAB Documentation - nsidedpoly," 2020. [Online]. Available: https://es.mathworks.com/help/matlab/ref/nsidedpoly.html. [Accessed 02 08 2020].

[5] M. E. Lemonis, "Calcresource," 24 04 2020. [Online]. Available: https://calcresource.com/geom-ngon.html. [Accessed 02 08 2020].

[6] MathWorks, "MATLAB Documentation - scatteredinterpolant," 2020. [Online]. Available: https://es.mathworks.com/help/matlab/ref/scatteredinterpolant.html. [Accessed 02 08 2020].

[7] MathWorks, "Matlab documentation - meshgrid," 2020. [Online]. Available: https://es.mathworks.com/help/matlab/ref/meshgrid.html?lang=en. [Accessed 02 08 2020].

[8] MathWorks, "Matlab Documentation - contourf," 2020. [Online]. Available: https://es.mathworks.com/help/matlab/ref/contourf.htm. [Accessed 02 08 2020].

[9] MathWorks, "Matlab Documentation - quiver," 2020. [Online]. Available: https://es.mathworks.com/help/matlab/ref/quiver.html?lang=en. [Accessed 02 08 2020].

[10] MathWorks, "Matlab Documentation - rectangle," 2020. [Online]. Available: https://es.mathworks.com/help/matlab/ref/rectangle.html. [Accessed 02 08 2020].

[11] MathWorks, "Matlab Documentation - patch," 2020. [Online]. Available: https://es.mathworks.com/help/matlab/ref/patch.html. [Accessed 02 08 2020].

[12] MathWorks, "Matlab Documentation - line," 2020. [Online]. Available: https://es.mathworks.com/help/matlab/ref/line.html?s_tid=srchtitlel. [Accessed 02 08 2020].

[13] MathWorks, "Matlab Documentation - uicontrol," 2020. [Online]. Available: https://es.mathworks.com/help/matlab/ref/uicontrol.html. [Accessed 02 08 2020].

8 APPENDIX 1: MAIN FILE

MAIN FILE

Simulation for different number of pillars of a microfluidic rotational motor driven by circular vibrations.

Authorship

```
% Code created by Iker Ruiz de Esquide.
% Code from the function 'SimulationOnePillar' created by Prof. Dr. Riko Safaric.
```

User information

```
%-----------------IMPORTANT INFORMATION FOR THE USER-----------------------
%   - CHECK THE OPTION "Pot mode ON" IN THE USER INTERFACE WHEN
%     THE SIMULATION OF A POT IS SEARCHED - WHEN PILLARS OVERLAP - SO THE
%     SCALE CAN BE ADJUSTED PROPERLY. THE BIGGER THE NUMBER OF PILLARS, THE
%     BETTER SIMULATION OF THE POT.
%
%     *If a big number of pillars is going to be selected to simulate the
%     pot, it is better to check first "Pot mode ON" and select later
%     the number of pillars. In this way, we avoid calculating twice the
%     simulation for that big number of pillars. This happens because when
%     checking the option "Pot mode ON" the simulation for the
%     configuration in that moment is recalculated.
%
%   - UNCHECK "Pot mode ON" IF THE PILLARS DO NOT OVERLAP.
%
%   - There are not maximun limits for the parameters in the user
%   interface, a very big value of the number of pillars can lead to a
%   very slow code execution.
%--------------------------------------------------------------------------
%
%Variables plotted in the graphic simulation and in the UI:
%   D => Distance between the exterior part of the pillars - Diameter of
%        the circumcircle of a regular polygon formated by the vertices
%        located in the central part of each pillar minus two times the
%        radius of the pillar.
%   f => Frequency of circular vibrations
%   a => Radius of the pillar
%   A => Amplitude of circular vibrations
%   Liquid => Liquid used for the simulation - Changes the kinematic
%             viscosity used.
%   Number of pillars
%
%Important variable names in the code:
%   --------For the simulation of the motor for one pillar--------
%   vr1 => Radial distance to the vector from the center of the pillar
%   vv => Tangential speeed vector depending on radial distance
%
%   --------For the simulation of the motor for several pillars--------
%   Utc,Vtc => x and y components of the tangential velocity for the
%              positions defined by the mesgrid
%   Urc,Vrc => x and y components of the rotational velocity for the
%              positions defined by the mesgrid
%   Vrot => magnitude of the rotational velocity vector for the positions
%           defined by the mesgrid
%   Vtan => magnitude of the tangential velocity vector for the positions
%           defined by the mesgrid
```

Initial configuration and simulation of the motor with one pillar

```
clc; clear; close all;

%Declaration of global variables
global N;global D;global D1;global f;global ni;global Liquid;global a;
global A;global a_plot;global A_plot;global PotOn; global jj;

%Variables for the simulation of one pillar:
f=1000; %Frequency (change in the UI)
a=0.000040; %Radius of the pillar/s (change in the UI)
ni=1.0e-6; %Kinematic viscosity of water (change in the UI)
Liquid='water'; %Water is selected in the simulation of one pillar (change
                                                                  %in the UI)
A=0.000008; %Amplitude of circular vibrations (change in the UI)

N=1; %Number of pillars (change in the UI)
D=200; %Distance D between the exterior part of the pillars (change in
                                                                 %the UI)

dr=0.000001; %Differential radial distance in the simulation of one pillar
%Number of calculations in the simulation of one pillar. More calculations
if D<300                             %are needed if the distance is bigger.
    jj=600;
elseif D<1000
    jj=600*3;
else
    jj=600*6;
end

[vr1,vv]=SimulationOnePillar(jj,dr,f,a,ni,A); %Simulates the vibrational
                                              %motor for one pillar

a_plot=a*10^6; %Change units of the raidus from m to um
A_plot=A*10^9; %Change units of the raidus from m to nm

D1=D+2*a_plot; %Distance D1 between the center of the pillars (change in
                                                                 %the UI)

n=50; %Number of vectors/circle before interpolation
subdiv=100; %Number of subdivisions per dimension of the meshgrid

PotOn=0; %The scale is not adjusted to the simulation of a pot
         %(pot <-> overlap of pillars)(change in the UI)
```

Calculation of the postion of the pillar/s

```
%Coordinates of the position of the pillars
if N>2
    s=D1*sin(pi/N); %Relation between the side of the regular polygon and
                    %the diameter of its circumcircle
    pgon=nsidedpoly(N,'Center',[0 0],'SideLength',s); %creates a polygon
                                                       %defined by 2-D vertices
    xp=pgon.Vertices(:,1); %x-coordiantes of the vertices of the polygon
    yp=pgon.Vertices(:,2); %y-coordinates of the vertices of the polygon
elseif N==2
    xp(1)=-D1/2;yp(1)=0;
    xp(2)=D1/2;yp(2)=0;
elseif N==1
    xp=0;yp=0;
end
```

Calculation of the position of the velocity vectors

```
phi=0+360/n:360/n:360; %Values for angle phi in degrees (angle between
                       %x-axis and the radial distance to the vector)
phi=phi*pi/180; %Convert phi to radians

%Preallocation of variables (increase efficiency)
X=cell(length(vr1),length(phi));
Y=cell(length(vr1),length(phi));
Xcol=cell(length(vr1),length(phi));
Ycol=cell(length(vr1),length(phi));

%Coordiantes of the velocity vectors
for k=1:N
    for i=1:length(vr1)
        for j=1:length(phi)
            X{k}(i,j)=xp(k)+vr1(i)*cos(phi(j)); %x-coordiante of the
                                                %velocity vector
            Y{k}(i,j)=yp(k)+vr1(i)*sin(phi(j)); %y-coordinate of the
                                                %velocity vector
        end
    end
    %Reshape matrices in a column vector
    Xcol{k}=reshape(X{k},[],1);
    Ycol{k}=reshape(Y{k},[],1);
end
```

Calculation of the components of the velocity vectors for the previously calculated positions

```
Vtan=repmat(vv',1,length(phi)); %Tangential speeed vector depending on
                                %radial distance and angle phi.
                                %Rows depend on radial distance.
                                %Colums depend on phi angle.

%Preallocation of variables (increase efficiency)
Ut=zeros(length(vr1),length(phi));
Vt=zeros(length(vr1),length(phi));

%u and v components of the tangential velocity (CW rotation)
for i=1:length(vr1)
    for j=1:length(phi)
            Ut(i,j)=Vtan(i,j)*cos(pi/2-phi(j));
            Vt(i,j)=-Vtan(i,j)*sin(pi/2-phi(j));
    end
end

%u and v components of the rotational velocity (CW rotation)
Vr1_ext=repmat(vr1',1,length(phi));
Ur=Ut./Vr1_ext;
Vr=Vt./Vr1_ext;
```

Interpolation of tangential and rotational velocity values and creating a meshgrid

```
%Reshape matrices in a column vector
Ut=reshape(Ut,[],1);
Vt=reshape(Vt,[],1);
Ur=reshape(Ur,[],1);
Vr=reshape(Vr,[],1);

%Preallocation of variables (increase efficiency)
FUt=cell(length(Ut),1);
FVt=cell(length(Vt),1);
FUr=cell(length(Ur),1);
FVr=cell(length(Vr),1);

%Interpolation of the velcocity components (rotational and tangential
%velocity)
for k=1:N
    FUt{k}=scatteredInterpolant(Xcol{k},Ycol{k},Ut,'linear','none');
    FVt{k}=scatteredInterpolant(Xcol{k},Ycol{k},Vt,'linear','none');
    FUr{k}=scatteredInterpolant(Xcol{k},Ycol{k},Ur,'linear','none');
    FVr{k}=scatteredInterpolant(Xcol{k},Ycol{k},Vr,'linear','none');
end

%Meshgrid creation
if PotOn==1 %meshgrid created only for the inner part of the pot (when
                                                    %pillars overlap)
    [Xc,Yc]=meshgrid(linspace(-D/2,D/2,subdiv),linspace(-D/2,D/2,subdiv));
elseif PotOn==0 %meshgrid created for a bigger zone (when pillars do not
                %overlap)- bigger for bigger distances
    if D<300
        low=-300; high=300;
    elseif D<500
        low=-300*2; high=300*2;
    else
        low=-300*3; high=300*3;
    end
    [Xc,Yc]=meshgrid(linspace(low,high,subdiv),linspace(low,high,subdiv));
end
```

Sum of the velocity components in the meshgrid and calculation of the magnitude of the vectors

```
%Sum of the components of the tangential and rotational velocity
Utc=0;Vtc=0;Urc=0;Vrc=0;
for k=1:N
    Utc=FUt{k}(Xc,Yc)+Utc;
    Vtc=FVt{k}(Xc,Yc)+Vtc;
    Urc=FUr{k}(Xc,Yc)+Urc;
    Vrc=FVr{k}(Xc,Yc)+Vrc;
end

%Magnitude of the tangential and rotational velocity vectors
Vtan=sqrt(Utc.^2 + Vtc.^2);
Vrot=sqrt(Urc.^2 + Vrc.^2);
```

Plots of tangential and rotational velocities

```
%Plot configuration of the rotational velocity
figure(1);
clf
hold on
linkdata on
grid off
%Axes limits - depends on the "Pot mode ON" checkbox
if PotOn==1
    xlim([-D/2-5 D/2+5])
    ylim([-D/2-5 D/2+5])
elseif PotOn==0 && N==1
    xlim([-200 200])
    ylim([-200 200])
elseif PotOn==0 && N>1
    xlim([-(D+2*a_plot) D+2*a_plot])
    ylim([-(D+2*a_plot) D+2*a_plot])
end
%Labels and title
xlabel('x [um]')
ylabel('y [um]')
ylabel(colorbar, 'v_r [rad/s]')
%Title configuration
```

```
if N>1
    title(['Rotational speed v_r [rad/s]     ',num2str(N), ...
           ' pillars;   D = ',num2str(D),' um;   Liquid: ',Liquid])
elseif N==1
    title(['Rotational speed v_r [rad/s]     1 pillar;   Liquid: ',Liquid])
end
%Configuration of the annotation
str={['f = ',num2str(f),' (Hz)'];['a = ',num2str(a_plot),' (um)']; ...
     ['A = ',num2str(A_plot),' (nm)']};
an=annotation('textbox',[0.58 0.8 0.1 0.1],'String',str);
set(an,'BackgroundColor',[1 1 1],'edgecolor','none');
set(an,'FaceAlpha',0.8);

%Configuration of the contourf plot
contourf(Xc,Yc,Vrot,100,'edgecolor','none') %Filled 2-D contour plot
colorbar
colormap jet
MAX=max(max(Vrot));
caxis([0,MAX]) %Colormap maximun limit adjusted to the maximun value of the
               %velocity

%Configuration of the quiver plot
%Number of vectors plotted per dimension - depends on the state of the
if PotOn==1                                      %checkbox "Pot mode ON"
    N_v=20;
elseif PotOn==0
    N_v=50;
end
[lim1,lim2]=size(Xc);
N_v1=round(lim1/N_v);
quiver(Xc(1:N_v1:lim1,1:N_v1:lim2),Yc(1:N_v1:lim1,1:N_v1:lim2), ...
       Urc(1:N_v1:lim1,1:N_v1:lim2),Vrc(1:N_v1:lim1,1:N_v1:lim2),1,'k');
       %Plot of the vector field

%Plotting the pillars or the pot depending on the user
if PotOn==1
    %Plot of the annulus that represents the pot
    %Boundaries of the annulus
    alpha=linspace(0,02*pi);
    rin = D/2; rout = 3*D/2;
    xin = rin*cos(alpha); xout = rout*cos(alpha);
    yin = rin*sin(alpha); yout = rout*sin(alpha);
    %Plot of the filled polygonal region that forms the annulus and the inner
    %circunference
    patch([xout,xin],[yout,yin],'w','linestyle','none');
    line(xin,yin,'color','k');
elseif PotOn==0
    for k=1:N
        rectangle('Position',[xp(k)-a_plot yp(k)-a_plot 2*a_plot 2*a_plot], ...
        'Curvature',[1,1],'FaceColor',[1 1 1]);
    end
end
hold off
```

```
%Plot configuration of the tangential velocity
figure(2);
clf
hold on
linkdata on
grid off
%Axes limits - depends on the "Pot mode ON" checkbox
if PotOn==1
    xlim([-D/2-5 D/2+5])
    ylim([-D/2-5 D/2+5])
elseif PotOn==0 && N==1
    xlim([-200 200])
    ylim([-200 200])
elseif PotOn==0 && N>1
    xlim([-(D+2*a_plot) D+2*a_plot])
    ylim([-(D+2*a_plot) D+2*a_plot])
end
%Labels and title
xlabel('x [um]')
ylabel('y [um]')
ylabel(colorbar, 'v_0 [um/s]')
%Title configuration
if N>1
    title(['Tangential speed v_0 [um/s]     ',num2str(N), ...
          ' pillars;   D = ',num2str(D),' um;   Liquid: ',Liquid])
elseif N==1
    title(['Tangential speed v_0 [um/s]     1 pillar;   Liquid: ',Liquid])
end
%Configuration of the annotation
str={['f = ',num2str(f),' (Hz)'];['a = ',num2str(a_plot),' (um)']; ...
     ['A = ',num2str(A_plot),' (nm)']};
an=annotation('textbox',[0.58 0.8 0.1 0.1],'String',str);
set(an,'BackgroundColor',[1 1 1],'edgecolor','none');
set(an,'FaceAlpha',0.8);

%Configuration of the contourf plot
contourf(Xc,Yc,Vtan,100,'edgecolor','none') %Filled 2-D contour plot
colorbar
colormap jet
MAX=max(max(Vtan));
caxis([0,MAX]) %Colormap maximun limit adjusted to the maximun value of the
               %velocity

%Configuration of the quiver plot
quiver(Xc(1:N_v1:lim1,1:N_v1:lim2),Yc(1:N_v1:lim1,1:N_v1:lim2), ...
       Urc(1:N_v1:lim1,1:N_v1:lim2),Vrc(1:N_v1:lim1,1:N_v1:lim2),1,'k');
       %Plot of the vector field

%Plotting the pillars or the pot depending on the user
if PotOn==1
    %Plot of the annulus that represents the pot
    %Plot of the filled polygonal region that forms the annulus and the inner
    %circunference
    patch([xout,xin],[yout,yin],'w','linestyle','none');
    line(xin,yin,'color','k');
elseif PotOn==0
    for k=1:N
        rectangle('Position',[xp(k)-a_plot yp(k)-a_plot 2*a_plot 2*a_plot], ...
        'Curvature',[1,1],'FaceColor',[1 1 1]);
    end
end
hold off
```

User Interface

```
%User interface window configuration
f3=figure('name','UI','Position',[20 200 311 376]); clf; linkdata on
if N>1
    %Configuration of the slide bar and editable text field of variable D
    uicontrol(f3,'Style', 'slider','Min',0,'Max',400, ...
              'Value',D,'Position', [88 250 190 10],'Callback',{@updateD});
    uicontrol(f3,'Style','text','Position',[33 240 40 22],'String', ...
              'D (um): ');
    uicontrol(f3,'Style','text','Position',[78 220 35 22],'String','0');
    uicontrol(f3,'Style','text','Position',[250 220 42 22],'String','400');
    uicontrol(f3,'Style', 'edit','Position', [150 188 100 22],...
              'Callback',{@updateD1});
    uicontrol(f3,'Style','text','Position',[69 188 66 22],'String', ...
              'D (um): ');
end
%Configuration of the editable text field of variable N
uicontrol(f3,'Style','edit','Position',[150 288 100 22], ...
          'Callback',{@updateN});
uicontrol('Style','text','Position',[69 288 56 30], ...
          'String','Number of pillars:');
%Configuration of the editable text field of variable f
uicontrol(f3,'Style','edit','Position',[150 141 100 22], ...
          'Callback',{@updatef});
uicontrol('Style','text','Position',[69 141 66 22],'String','f (Hz): ');
%Configuration of the editable text field of variable a
uicontrol(f3,'Style','edit','Position',[150 94 100 22], ...
          'Callback',{@updatea});
uicontrol('Style','text','Position',[69 94 66 22],'String','a (um): ');
%Configuration of the editable text field menu of variable A
uicontrol(f3,'Style','edit','Position',[150 52 100 22], ...
          'Callback',{@updateAA});
uicontrol('Style','text','Position',[69 52 66 22],'String','A (nm): ');
%Configuration of the editable Pop-Up menu of variable Liquid
uicontrol(f3,'Style','popupmenu','Position',[150 330 100 22], ...
          'String',{'Water','Mercury','Galinstan'}, ...
          'Callback',{@updateLiquid});
uicontrol('Style','text','Position',[69 330 66 22],'String','Liquid: ');
%Configuration of the checkbox of variable PotOn
uicontrol(f3,'Style','checkbox','Position',[150 15 100 22], ...
          'Callback',{@updatePotOn});
uicontrol('Style','text','Position',[50 10 90 22],'String','Pot mode ON: ');
```

9 APPENDIX 2: SIMULATION ONE PILLAR

Function: SimulationOnePillar

Code created by Prof. Dr. Riko Safaric.

The resolution of the mathematical model of the microfluidic rotational motor with a unique pillar is done.

Resolution of the mathematical model

```
function [vr1,vv]=SimulationOnePillar(j,dr,f,a,ni,A)
format shortEng
%format compact
%[msg, id] = lastwarn;
%warning('off', id)
%j=70;
%dr=0.000001;
k=1;
Psistr=0;
dPsidr=0;
%ni=0.15e-4;%air
%ni=1.0e-6; %water
%ni=0.114e-6; %Hg
%ni=0.215e-6; %Ga
%f=1000;%frequency
%f=500;%frequency
%a=0.000040; % radius of coloumn
%A=0.000000058;amplitude of vibration in case of pzt vibrator 200 x 200 um
%A=3*0.000000064; %amplitude of vibration in case of pzt vibrator 1 x 1 x 0.5 mm
%A=0.000008;%amplitude of circular vibrations
eps=sqrt(1i*2*pi*f/ni);
epsz=sqrt(-1i*2*pi*f/ni);
r=a;
H01epsa=besselh(0,1,a*eps);
C=besselh(2,1,a*eps)./H01epsa;
Cconj=conj(besselh(2,1,a*eps)./H01epsa);
roxxm1=@(x) 2*pi^3*f^3*A^2/ni^2./x.*(2*besselh(0,1,x*eps)/H01epsa+ ...
            2*conj(besselh(0,1,x*eps)/H01epsa)-2*a^2./x.^2* ...
            C.*besselh(2,2,x*epsz)/conj(H01epsa)-2*a^2./x.^2.* ...
            Cconj.*besselh(2,1,x*eps)/H01epsa-4*besselh(0,1,x*eps)/ ...
            H01epsa.*conj(besselh(0,1,x*eps)/H01epsa)+ ...
            4*besselh(2,1,x*eps)/H01epsa.*conj(besselh(2,1,x*eps)/H01epsa));
roxx=@(x) 2*pi^3*f^3*A^2.*x/ni^2.*(2*besselh(0,1,x*eps)/H01epsa+ ...
          2*conj(besselh(0,1,x*eps)/H01epsa)-2*a^2./x.^2* ...
          C.*besselh(2,2,x*epsz)/conj(H01epsa)-2*a^2./x.^2.* ...
          Cconj.*besselh(2,1,x*eps)/H01epsa-4*besselh(0,1,x*eps)/...
          H01epsa.*conj(besselh(0,1,x*eps)/H01epsa)+ ...
          4*besselh(2,1,x*eps)/H01epsa.*conj(besselh(2,1,x*eps)/H01epsa));
%roxxc=@(x) x.*(2*besselh(0,1,x*eps)/H01epsa +2*conj(besselh(0,1,x*eps)/
            ... H01epsa)-2*a^2./x.^2*C.*conj(besselh(2,1,x*eps)/H01epsa)-
            ... 2*a^2./x.^2.*Cconj.*besselh(2,1,x*eps)/H01epsa-
            ... 4*besselh(0,1,x*eps)/H01epsa.*
            ... conj(besselh(0,1,x*eps)/H01epsa)+
            ... 4*besselh(2,1,x*eps)/H01epsa.*conj(besselh(2,1,x*eps)/H01epsa));
roxx3=@(x) 2*pi^3*f^3*A^2.*x.^3/ni^2.*(2*besselh(0,1,x*eps)/H01epsa+ ...
           2*conj(besselh(0,1,x*eps)/H01epsa)-2*a^2./x.^2* ...
```

```
            C.*besselh(2,2,x*epsz)/conj(H01epsa)-2*a^2./x.^2.* ...
            Cconj.*besselh(2,1,x*eps)/H01epsa-4*besselh(0,1,x*eps)/ ...
            H01epsa.*conj(besselh(0,1,x*eps)/H01epsa)+ ...
            4*besselh(2,1,x*eps)/H01epsa.*conj(besselh(2,1,x*eps)/H01epsa));
roxx5=@(x) 2*pi^3*f^3*A^2.*x.^5/ni^2.*(2*besselh(0,1,x*eps)/H01epsa+ ...
            2*conj(besselh(0,1,x*eps)/H01epsa)-2*a^2./x.^2* ...
            C.*besselh(2,2,x*epsz)/conj(H01epsa)-2*a^2./x.^2.* ...
            Cconj.*besselh(2,1,x*eps)/H01epsa-4*besselh(0,1,x*eps)/ ...
            H01epsa.*conj(besselh(0,1,x*eps)/H01epsa)+ ...
            4*besselh(2,1,x*eps)/H01epsa.*conj(besselh(2,1,x*eps)/H01epsa));
c1x3=-1/48*integral(roxx3,a,inf);
c1=-1/48*integral(roxxm1,a,inf);
%c1=-1/48*integral(roxxm1,a,0.000055,'RelTol',1.3e-13,'AbsTol',1.3e-13)
c2=1/16*integral(roxx,a,inf);
c3=a^4/16*integral(roxxm1,a,inf)-a^2/8*integral(roxx,a,inf);
c4=-a^6/24*integral(roxxm1,a,inf)+a^4/16*integral(roxx,a,inf);
%disp(['     k               r               Psi               dPsi/dr'])
vk=[1];
vr=[a];
vr1=[a];
vpsi=[0];
vv=[0];  % um/s
while k <j
%disp([k,r,Psistr,dPsidr])
k=k+1;
r=r+dr;
Psistrkm1=Psistr;
Psistr=abs(r^4*((1/48)*integral(roxxm1,a,r)+c1)+r^2* ...
       ((-1)/16*integral(roxx,a,r)+c2)+(1/16*integral(roxx3,a,r)+c3)+ ...
       r^(-2)*(((-1)/48)*integral(roxx5,a,r)+c4));
%dPsidr=abs((Psistr-Psistrkm1)/dr);
dPsidr=(Psistr-Psistrkm1)/dr;
vk=[vk,k];
vr=[vr,1e6*(r-a)];
%vr=[vr,1e6*r];
vr1=[vr1,1e6*r];
vpsi=[vpsi,Psistr];
vv=[vv,1e6*dPsidr];
end
k=1;
vr2=vv./vr1;
vv2=0;
while k <j
vv2=vv2+vr2(k+1);
k=k+1;
end
vk2=vv2/j;
k=1;
vk3=[vk2];
while k <j
vk3=[vk2,vk3];
k=k+1;
end
```

Plots of tangential and rotational speed against distance from the pillar

```
% ---------------------------NOT PLOTTED-----------------------------------
% figure(4)
% plot(vr,vv)
% grid
% xlabel('r [um]')
% ylabel('v_0 [um/s]')
% title('Tangential speed v_0 against distance from the pillar')
%
% figure(5)
% plot(vr,vv./vr1,'b', vr,vk3,'r')
% grid
% legend('v_r','average v_r');
% xlabel('r [\mum]')
% ylabel('v_r [rd/s]')
% title('Rotational speed v_r against distance from the pillar')
end
```

10 APPENDIX 3: UPDATE D

Function: updateD

Parameter D is updated through the slide bar.

Declaration of global variables

```
function []=updateD(h,~)
global D;global D1;global jj;
```

Read the distance between pillars introduced in the UI

```
D=get(h,'Value');
```

Load variables from base workspace

```
f=evalin('base','f');
a=evalin('base','a');
a_plot=evalin('base','a_plot');
ni=evalin('base','ni');
Liquid=evalin('base','Liquid');
A=evalin('base','A');
A_plot=evalin('base','A_plot');
N=evalin('base','N');
phi=evalin('base','phi');
PotOn=evalin('base','PotOn');
subdiv=evalin('base','subdiv');
dr=evalin('base','dr');
```

Update variables

```
D1=D+2*a_plot;
%Number of calculations in the simulation of one pillar. More calculations
%are needed if the distance is bigger.
if D<300
    jj=600;
elseif D<1000
    jj=600*3;
else
    jj=600*6;
end
```

Repeat calculations for the updated parameters

```
RepeatCalculations(A,A_plot,f,a,a_plot,ni,Liquid,dr,jj,N,D,D1,phi,PotOn,subdiv)
```

User Interface

```
%User interface window configuration
f3=figure(3);
%Configuration of the editable text field of variable D
uicontrol(f3,'Style', 'edit','Position', [150 188 100 22],...
          'Callback',{@updateD1});
end
```

11 APPENDIX 4: UPDATE D1

Function: updateD1

Parameter D is updated through the text box.

Declaration of global variables

```
function []=updateD1(h,~)
global D;global D1;global jj;
```

Read the distance between pillars introduced in the UI

```
D=str2double(get(h,'String'));
```

Load variables from base workspace

```
f=evalin('base','f');
a=evalin('base','a');
a_plot=evalin('base','a_plot');
ni=evalin('base','ni');
Liquid=evalin('base','Liquid');
A=evalin('base','A');
A_plot=evalin('base','A_plot');
N=evalin('base','N');
phi=evalin('base','phi');
PotOn=evalin('base','PotOn');
subdiv=evalin('base','subdiv');
dr=evalin('base','dr');
```

Update variables

```
D1=D+2*a_plot;
%Number of calculations in the simulation of one pillar. More calculations
%are needed if the distance is bigger.
if D<300
    jj=600;
elseif D<1000
    jj=500*3;
else
    jj=600*6;
end
```

Repeat calculations for the updated parameters

```
RepeatCalculations(A,A_plot,f,a,a_plot,ni,Liquid,dr,jj,N,D,D1,phi,PotOn,subdiv)
```

User Interface

```
%User interface window configuration
f3=figure(3);
%Configuration of the slide bar
uicontrol(f3,'Style', 'slider','Min',0,'Max',400, ...
          'Value',D,'Position', [88 250 190 10],'Callback',{@updateD});
end
```

12 APPENDIX 5: UPDATE N

Function: updateN

Parameter N is updated.

Declaration of global variables

```
function []=updateN(h,~)
global N;
```

Read the number of pillars introduced in the UI

```
N=str2double(get(h,'String'));
```

Load variables from base workspace

```
f=evalin('base','f');
a=evalin('base','a');
a_plot=evalin('base','a_plot');
ni=evalin('base','ni');
Liquid=evalin('base','Liquid');
A=evalin('base','A');
A_plot=evalin('base','A_plot');
dr=evalin('base','dr');
jj=evalin('base','jj');
D=evalin('base','D');
D1=evalin('base','D1');
phi=evalin('base','phi');
PotOn=evalin('base','PotOn');
subdiv=evalin('base','subdiv');
```

Repeat calculations for the updated parameters

```
RepeatCalculations(A,A_plot,f,a,a_plot,ni,Liquid,dr,jj,N,D,D1,phi,PotOn,subdiv)
```

User Interface

```
% User interface window configuration
% Deletes the option of changing distance between pillars if there is only one
% pillar and add the option if there is more than one pillar
f3=figure(3);
if N==1
    close(f3);
    f3=figure('name','UI','Position',[20 200 311 376]);
    linkdata on
    %Configuration of the editable text field of variable N
uicontrol(f3,'Style','edit','Position',[150 288 100 22], ...
          'Callback',{@updateN});
uicontrol('Style','text','Position',[69 288 56 30], ...
          'String','Number of pillars:');
%Configuration of the editable text field of variable f
uicontrol(f3,'Style','edit','Position',[150 141 100 22], ...
          'Callback',{@updatef});
```

```
uicontrol('Style','text','Position',[69 141 66 22],'String','f (Hz): ');
%Configuration of the editable text field of variable a
uicontrol(f3,'Style','edit','Position',[150 94 100 22], ...
          'Callback',{@updatea});
uicontrol('Style','text','Position',[69 94 66 22],'String','a (um): ');
%Configuration of the editable text field menu of variable A
uicontrol(f3,'Style','edit','Position',[150 52 100 22], ...
          'Callback',{@updateAA});
uicontrol('Style','text','Position',[69 52 66 22],'String','A (nm): ');
%Configuration of the editable Pop-Up menu of variable Liquid
uicontrol(f3,'Style','popupmenu','Position',[150 330 100 22], ...
          'String',{'Water','Mercury','Galinstan'}, ...
          'Callback',{@updateLiquid});
uicontrol('Style','text','Position',[69 330 66 22],'String','Liquid: ');
%Configuration of the checkbox of variable PotOn
uicontrol(f3,'Style','checkbox','Position',[150 15 100 22], ...
          'Callback',{@updatePotOn});
uicontrol('Style','text','Position',[50 10 90 22],'String','POT SCALE ON: ');
elseif N>1
    %Configuration of the slide bar and editable text field of variable D
    uicontrol(f3,'Style', 'slider','Min',0,'Max',400, ...
              'Value',D,'Position', [88 250 190 10],'Callback',{@updateD});
    uicontrol(f3,'Style','text','Position',[33 240 40 22],'String', ...
              'D (um): ');
    uicontrol(f3,'Style','text','Position',[78 220 35 22],'String','0');
    uicontrol(f3,'Style','text','Position',[250 220 42 22],'String','400');
    uicontrol(f3,'Style', 'edit','Position', [150 188 100 22],...
              'Callback',{@updateD1});
    uicontrol(f3,'Style','text','Position',[69 188 66 22],'String', ...
              'D (um): ');
end
end
```

13 APPENDIX 6: UPDATE A

Function: updatea

Parameter a is updated.

Declaration of global variables

```
function []=updatea(h,~)
global a;global a_plot;global D;global D1;
```

Read the radius of the pillar introduced in the UI

```
a_plot=str2double(get(h,'String'));
a=a_plot*10^-6; %Change units of the raidus from um to m
```

Load variables from base workspace

```
f=evalin('base','f');
ni=evalin('base','ni');
Liquid=evalin('base','Liquid');
A=evalin('base','A');
A_plot=evalin('base','A_plot');
dr=evalin('base','dr');
jj=evalin('base','jj');
N=evalin('base','N');
phi=evalin('base','phi');
PotOn=evalin('base','PotOn');
subdiv=evalin('base','subdiv');
```

Update variables

```
D1=D+2*a_plot;
```

Repeat calculations for the updated parameters

```
RepeatCalculations(A,A_plot,f,a,a_plot,ni,Liquid,dr,jj,N,D,D1,phi,PotOn,subdiv)
end
```

14 APPENDIX 7: UPDATE AA

Function: updateAA

Parameter A is updated.

Declaration of global variables

```
function []=updateAA(h,~)
global A;global A_plot;
```

Read the amplitude of the circular vibrations introduced in the UI

```
A_plot=str2double(get(h,'String'));
A=A_plot*10^-9; %Change units of the raidus from nm to m
```

Load variables from base workspace

```
f=evalin('base','f');
a=evalin('base','a');
a_plot=evalin('base','a_plot');
ni=evalin('base','ni');
Liquid=evalin('base','Liquid');
dr=evalin('base','dr');
jj=evalin('base','jj');
N=evalin('base','N');
D=evalin('base','D');
D1=evalin('base','D1');
phi=evalin('base','phi');
PotOn=evalin('base','PotOn');
subdiv=evalin('base','subdiv');
```

Repeat calculations for the updated parameters

```
RepeatCalculations(A,A_plot,f,a,a_plot,ni,Liquid,dr,jj,N,D,D1,phi,PotOn,subdiv)
end
```

15 APPENDIX 8: UPDATE F

Function: updatef

Parameter f is updated.

Declaration of global variables

```
function []=updatef(h,~)
global f;
```

Read the frequency introduced in the UI

```
f=str2double(get(h,'String'));
```

Load variables from base workspace

```
a=evalin('base','a');
a_plot=evalin('base','a_plot');
ni=evalin('base','ni');
Liquid=evalin('base','Liquid');
A=evalin('base','A');
A_plot=evalin('base','A_plot');
dr=evalin('base','dr');
jj=evalin('base','jj');
N=evalin('base','N');
D=evalin('base','D');
D1=evalin('base','D1');
phi=evalin('base','phi');
PotOn=evalin('base','PotOn');
subdiv=evalin('base','subdiv');
```

Repeat calculations for the updated parameters

```
RepeatCalculations(A,A_plot,f,a,a_plot,ni,Liquid,dr,jj,N,D,D1,phi,PotOn,subdiv)
end
```

16 APPENDIX 9: UPDATE LIQUID

Function: updateLiquid

Parameter Liquid is updated.

Declaration of global variables

```
function []=updateLiquid(h,~)
global ni;global Liquid;
```

Read the liquid introduced in the UI and asign the corresponding kinematic viscosity

```
h = get(h,'value');
if h==1
    ni=1.0e-6;
    Liquid='water';
elseif h==2
    ni=0.114e-6;
    Liquid='Hg';
elseif h==3
    ni=0.215e-6;
    Liquid='Ga';
end
```

Load variables from base workspace

```
f=evalin('base','f');
a=evalin('base','a');
a_plot=evalin('base','a_plot');
A=evalin('base','A');
A_plot=evalin('base','A_plot');
dr=evalin('base','dr');
jj=evalin('base','jj');
N=evalin('base','N');
D=evalin('base','D');
D1=evalin('base','D1');
phi=evalin('base','phi');
PotOn=evalin('base','PotOn');
subdiv=evalin('base','subdiv');
```

Repeat calculations for the updated parameters

```
RepeatCalculations(A,A_plot,f,a,a_plot,ni,Liquid,dr,jj,N,D,D1,phi,PotOn,subdiv)
end
```

17 APPENDIX 10: UPDATE POT ON

Function: updatePotOn

Parameter PotOn is updated.

Declaration of global variables

```
function []=updatePotOn(h,~)
global PotOn;
```

Read the mode of the checkbox in the UI

```
PotOn=get(h,'value');
```

Load variables from base workspace

```
f=evalin('base','f');
a=evalin('base','a');
a_plot=evalin('base','a_plot');
ni=evalin('base','ni');
Liquid=evalin('base','Liquid');
A=evalin('base','A');
A_plot=evalin('base','A_plot');
dr=evalin('base','dr');
jj=evalin('base','jj');
N=evalin('base','N');
D=evalin('base','D');
D1=evalin('base','D1');
phi=evalin('base','phi');
subdiv=evalin('base','subdiv');
```

Repeat calculations for the updated parameters

```
RepeatCalculations(A,A_plot,f,a,a_plot,ni,Liquid,dr,jj,N,D,D1,phi,PotOn,subdiv)
end
```

18 APPENDIX 11: REPEAT CALCULATIONS

Function: RepeatCalculations

Calculations are repeated for the updated parameters.

Simulation of the vibrational motor for one pillar

```
function []=RepeatCalculations(A,A_plot,f,a,a_plot,ni,Liquid,dr,jj,N,D,D1,phi,PotOn,subdiv)
[vr1,vv]=SimulationOnePillar(jj,dr,f,a,ni,A);
```

Calculation of the postion of the pillar/s

```
%Coordinates of the position of the pillars
if N>2
    s=D1*sin(pi/N); %Relation between the side of the regular polygon and
                    %the diameter of its circumcircle
    pgon=nsidedpoly(N,'Center',[0 0],'SideLength',s); %creates a polygon
                                                      %defined by 2-D vertices
    xp=pgon.Vertices(:,1); %x-coordiantes of the vertices of the polygon
    yp=pgon.Vertices(:,2); %y-coordinates of the vertices of the polygon
elseif N==2
    xp(1)=-D1/2;yp(1)=0;
    xp(2)=D1/2;yp(2)=0;
elseif N==1
    xp=0;yp=0;
end
```

Calculation of the position of the velocity vectors

```
%Preallocation of variables (increase efficiency)
X=cell(length(vr1),length(phi));
Y=cell(length(vr1),length(phi));
Xcol=cell(length(vr1),length(phi));
Ycol=cell(length(vr1),length(phi));

%Coordiantes of the velocity vectors
for k=1:N
    for i=1:length(vr1)
        for j=1:length(phi)
            X{k}(i,j)=xp(k)+vr1(i)*cos(phi(j)); %x-coordiante of the
                                                %velocity vector
            Y{k}(i,j)=yp(k)+vr1(i)*sin(phi(j)); %y-coordinate of the
                                                %velocity vector
        end
    end
    %Reshape matrices in a column vector
    Xcol{k}=reshape(X{k},[],1);
    Ycol{k}=reshape(Y{k},[],1);
end
```

Calculation of the components of the velocity vectors for the previously calculated positions

```
Vtan=repmat(vv',1,length(phi)); %Tangential speeed vector depending on
                                %radial distance and angle phi.
                                %Rows depend on radial distance.
                                %Colums depend on phi angle.

%Preallocation of variables (increase efficiency)
Ut=zeros(length(vr1),length(phi));
Vt=zeros(length(vr1),length(phi));

%u and v components of the tangential velocity (CW rotation)
for i=1:length(vr1)
    for j=1:length(phi)
            Ut(i,j)=Vtan(i,j)*cos(pi/2-phi(j));
            Vt(i,j)=-Vtan(i,j)*sin(pi/2-phi(j));
    end
end

%u and v components of the rotational velocity (CW rotation)
Vr1_ext=repmat(vr1',1,length(phi));
Ur=Ut./Vr1_ext;
Vr=Vt./Vr1_ext;
```

Interpolation of tangential and rotational velocity values and creating a meshgrid

```
%Reshape matrices in a column vector
Ut=reshape(Ut,[],1);
Vt=reshape(Vt,[],1);
Ur=reshape(Ur,[],1);
Vr=reshape(Vr,[],1);

%Preallocation of variables (increase efficiency)
FUt=cell(length(Ut),1);
FVt=cell(length(Vt),1);
FUr=cell(length(Ur),1);
FVr=cell(length(Vr),1);

%Interpolation of the velcocity components (rotational and tangential
%velocity)
for k=1:N
    FUt{k}=scatteredInterpolant(Xcol{k},Ycol{k},Ut,'linear','none');
    FVt{k}=scatteredInterpolant(Xcol{k},Ycol{k},Vt,'linear','none');
    FUr{k}=scatteredInterpolant(Xcol{k},Ycol{k},Ur,'linear','none');
    FVr{k}=scatteredInterpolant(Xcol{k},Ycol{k},Vr,'linear','none');
end

%Meshgrid creation
if PotOn==1 %meshgrid created only for the inner part of the pot (when
                                                     %pillars overlap)
    [Xc,Yc]=meshgrid(linspace(-D/2,D/2,subdiv),linspace(-D/2,D/2,subdiv));
elseif PotOn==0 %meshgrid created for a bigger zone (when pillars do not
                %overlap)- bigger for bigger distances
    if D<300
        low=-300; high=300;
    elseif D<500
        low=-300*2; high=300*2;
    else
        low=-300*3; high=300*3;
    end
    [Xc,Yc]=meshgrid(linspace(low,high,subdiv),linspace(low,high,subdiv));
end
```

Sum of the velocity components in the meshgrid and calculation of the magnitude of the vectors

```
%Sum of the components of the tangential and rotational velocity
Utc=0;Vtc=0;Urc=0;Vrc=0;
for k=1:N
    Utc=FUt{k}(Xc,Yc)+Utc;
    Vtc=FVt{k}(Xc,Yc)+Vtc;
    Urc=FUr{k}(Xc,Yc)+Urc;
    Vrc=FVr{k}(Xc,Yc)+Vrc;
end

%Magnitude of the tangential and rotational velocity vectors
Vtan=sqrt(Utc.^2 + Vtc.^2);
Vrot=sqrt(Urc.^2 + Vrc.^2);
```

Plots of tangential and rotational velocities

```
%Plot configuration of the rotational velocity
figure(1);
clf
hold on
linkdata on
grid off
%Axes limits - depends on the "Pot mode ON" checkbox
if PotOn==1
    xlim([-D/2-5 D/2+5])
    ylim([-D/2-5 D/2+5])
elseif PotOn==0 && N==1
    xlim([-200 200])
    ylim([-200 200])
elseif PotOn==0 && N>1
    xlim([-(D+2*a_plot) D+2*a_plot])
    ylim([-(D+2*a_plot) D+2*a_plot])
end
%Labels and title
xlabel('x [um]')
ylabel('y [um]')
ylabel(colorbar, 'v_r [rad/s]')
%Title configuration
```

```
if N>1
    title(['Rotational speed v_r [rad/s]     ',num2str(N), ...
           ' pillars;   D = ',num2str(D),' um;   Liquid: ',Liquid])
elseif N==1
    title(['Rotational speed v_r [rad/s]     1 pillar;   Liquid: ',Liquid])
end
%Configuration of the annotation
str={['f = ',num2str(f),' (Hz)'];['a = ',num2str(a_plot),' (um)']; ...
     ['A = ',num2str(A_plot),' (nm)']};
an=annotation('textbox',[0.58 0.8 0.1 0.1],'String',str);
set(an,'BackgroundColor',[1 1 1],'edgecolor','none');
set(an,'FaceAlpha',0.8);

%Configuration of the contourf plot
contourf(Xc,Yc,Vrot,100,'edgecolor','none') %Filled 2-D contour plot
colorbar
colormap jet
MAX=max(max(Vrot));
caxis([0,MAX]) %Colormap maximun limit adjusted to the maximun value of the
               %velocity

%Configuration of the quiver plot
%Number of vectors plotted per dimension - depends on the state of the
if PotOn==1                                       %checkbox "Pot mode ON"
    N_v=20;
elseif PotOn==0
    N_v=50;
end
[lim1,lim2]=size(Xc);
N_v1=round(lim1/N_v);
quiver(Xc(1:N_v1:lim1,1:N_v1:lim2),Yc(1:N_v1:lim1,1:N_v1:lim2), ...
       Urc(1:N_v1:lim1,1:N_v1:lim2),Vrc(1:N_v1:lim1,1:N_v1:lim2),1,'k');
       %Plot of the vector field

%Plotting the pillars or the pot depending on the user
if PotOn==1
    %Plot of the annulus that represents the pot
    %Boundaries of the annulus
    alpha=linspace(0,02*pi);
    rin = D/2; rout = 3*D/2;
    xin = rin*cos(alpha); xout = rout*cos(alpha);
    yin = rin*sin(alpha); yout = rout*sin(alpha);
    %Plot of the filled polygonal region that forms the annulus and the inner
    %circunference
    patch([xout,xin],[yout,yin],'w','linestyle','none');
    line(xin,yin,'color','k');
elseif PotOn==0
    for k=1:N
        rectangle('Position',[xp(k)-a_plot yp(k)-a_plot 2*a_plot 2*a_plot], ...
        'Curvature',[1,1],'FaceColor',[1 1 1]);
    end
end

hold off
```

```
%Plot configuration of the tangential velocity
figure(2);
clf
hold on
linkdata on
grid off
%Axes limits - depends on the "Pot mode ON" checkbox
if PotOn==1
    xlim([-D/2-5 D/2+5])
    ylim([-D/2-5 D/2+5])
elseif PotOn==0 && N==1
    xlim([-200 200])
    ylim([-200 200])
elseif PotOn==0 && N>1
    xlim([-(D+2*a_plot) D+2*a_plot])
    ylim([-(D+2*a_plot) D+2*a_plot])
end
%Labels and title
xlabel('x [um]')
ylabel('y [um]')
ylabel(colorbar, 'v_0 [um/s]')
%Title configuration
if N>1
    title(['Tangential speed v_0 [um/s]     ',num2str(N), ...
           ' pillars;   D = ',num2str(D),' um;   Liquid: ',Liquid])
elseif N==1
    title(['Tangential speed v_0 [um/s]     1 pillar;   Liquid: ',Liquid])
end
%Configuration of the annotation
str={['f = ',num2str(f),' (Hz)'];['a = ',num2str(a_plot),' (um)']; ...
     ['A = ',num2str(A_plot),' (nm)']};
an=annotation('textbox',[0.58 0.8 0.1 0.1],'String',str);
set(an,'BackgroundColor',[1 1 1],'edgecolor','none');
set(an,'FaceAlpha',0.8);

%Configuration of the contourf plot
contourf(Xc,Yc,Vtan,100,'edgecolor','none') %Filled 2-D contour plot
colorbar
colormap jet
MAX=max(max(Vtan));
caxis([0,MAX]) %Colormap maximun limit adjusted to the maximun value of the
               %velocity

%Configuration of the quiver plot
quiver(Xc(1:N_v1:lim1,1:N_v1:lim2),Yc(1:N_v1:lim1,1:N_v1:lim2), ...
       Urc(1:N_v1:lim1,1:N_v1:lim2),Vrc(1:N_v1:lim1,1:N_v1:lim2),1,'k');
       %Plot of the vector field

%Plotting the pillars or the pot depending on the user
if PotOn==1
    %Plot of the annulus that represents the pot
    %Plot of the filled polygonal region that forms the annulus and the inner
    %circunference
    patch([xout,xin],[yout,yin],'w','linestyle','none');
    line(xin,yin,'color','k');
elseif PotOn==0
    for k=1:N
        rectangle('Position',[xp(k)-a_plot yp(k)-a_plot 2*a_plot 2*a_plot], ...
        'Curvature',[1,1],'FaceColor',[1 1 1]);
    end
end

hold off
end
```