Building and testing our Modular Multilayer Halbach Cylinder Magnet

Théophile Bompard, Etienne Bout, Baptiste Brunaux, Mathieu De Bettignies, Thibault De Coninck, Tristan Deleplanque, Louise Deleporte, Mats Goossens, Alexandre Grimond, Evie Hyenne, Ulysse Mao, Pauline Masson, Alice Maurice, Martin Rodriguez, Lila Soumphonphakdy, Louise Waeterloos

Lycée de la Croix Blanche – Bondues, France

April 12, 2023

I - Why we want to come to CERN or DESY ?

For young scientists, the Beamline for Schools competition represents a wonderful chance to see a great experiment in action. As students who are willing to become engineers or do research, we think that it is a great opportunity to see a real-life project and experiment. Moreover, we know that the LHC of the CERN is the largest and most powerful particle accelerator of the world so we would be honoured to go there. This is why our French team wants to apply for this contest, as we love and want to learn more about particle physics research. We are indeed extremely motivated to see our experiment grow in this research centre. Finally, going to one of the laboratories would be an opportunity to meet talented physicists and science enthusiasts in general, but also to rekindle the enthusiasm for science in our establishment.

II - Experiment proposal

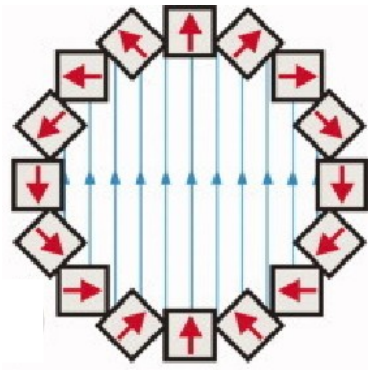
Scientists have made significant discoveries in particle physics, including the existence of subatomic particles such as protons, neutrons, and electrons, as well as the discovery of forces that govern the interaction between these particles. One of the main gears used in this field is the dipole magnet. Their use in particle physics dates to the early 1900s when physicists discovered that charged particles could be deflected by a magnetic field. They are used to steer charged particles along a specific path.

Our experiment aims to test and characterize the Modular Multilayer Halbach Cylinder Magnet that we have designed and want to build. Such a permanent magnet will make it possible to equip all test beam areas where it is difficult to install electromagnets. It is for example the case in the CERN T9 test beam area. We hope that it also will contribute to the development of new and more advanced particle detection technologies, which are essential for a wide range of scientific and technological applications.

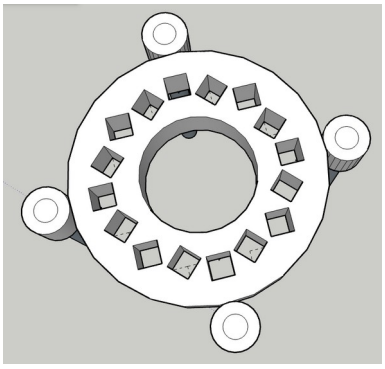
1) Our magnetic dipole

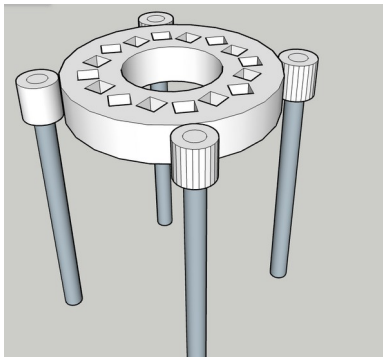
The Halbach array system is an arrangement of magnets that achieves a concentrated magnetic field on one side and a weak or zero magnetic field on the other side. Thus, this arrangement of the magnets makes it possible to produce a stronger and more homogeneous magnetic field than it could be obtained with a traditional arrangement of magnets. The Halbach array takes its name from the German physician Klaus Halbach who proposed this configuration in 1980. The Halbach array is used in many applications, such as electric motors, generators, energy storage devices, magnetic sensors, and particle accelerators. It is also a technique widely used in experimental physics to produce homogeneous and stable magnetic fields in particle and condensed matter physics experiments.

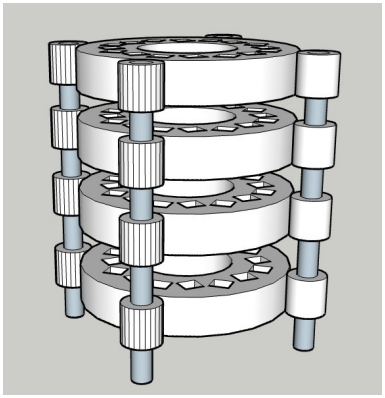
For our proposal, we decided to create a **cylinder halbach** array. This structure allows us, following the arrangement of the magnets, to build magnetic multipoles. In order to deflect the particle beam, we want to build a dipole with a uniform magnetic field. One layer of the cylinder is composed of 16 cubic N52 NdFeB magnets with a 1 cm side. The picture below shows the organisation of the 16 magnets :



We hope to obtain a magnetic field of a few tenths of a tesla with this Halbach configuration. To compensate for this low value, we decided to build a **multilayer** magnet. The length of the magnet will increase the deflection of the beam. And we want it **modular** so that it will be an adaptable tool for future experiments.
We finally got the following design for our **Modular Multilayer Halbach Cylinder Magnet** :







Inner circle diameter is 5 cm in order for the beam to pass. The different parts will be printed with a 3d printer.

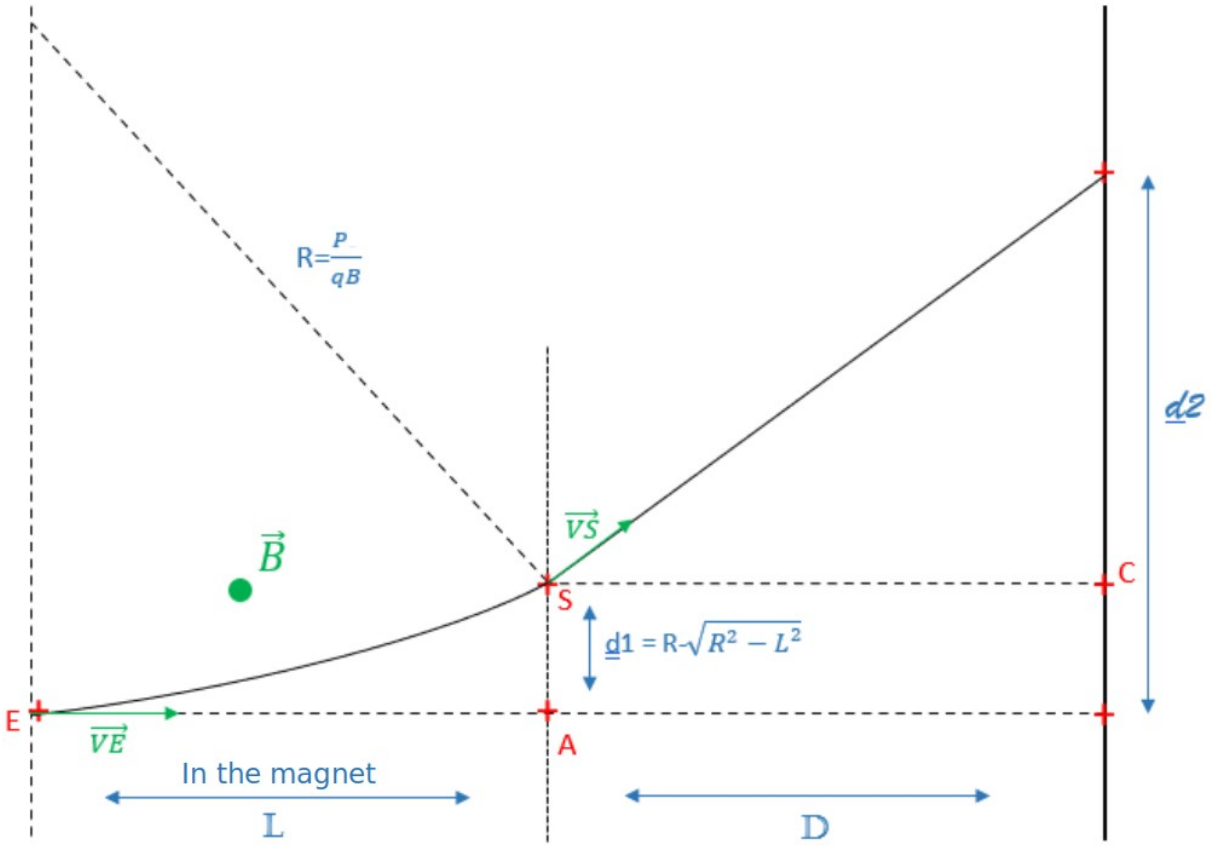
2) Theoretical study of the trajectory

To study the trajectories of particles through our magnet, we begin with the expression of the magnetic Lorentz force acting on particles in a uniform magnetic field :

$$\vec{F} = q\,\vec{v} \wedge \vec{B} \text{ and } F = q \times v \times B \text{ if } \vec{v} \text{ and } \vec{B} \text{ are orthogonal}$$

with q the charge of the particle, $\vec{v}$ its velocity and $\vec{B}$ the magnetic field.

In order to make our calculations, we can map the experiment with the following diagram :



Thanks to Newton's second law and the Frenet basis, we can show that the norm of the velocity is constant, given that the particle's motion is plane. Moreover, the trajectory of a particle in the dipole is circular. Therefore, the acceleration is centripetal. So we can express the bending radius as follow :

$$R = \frac{m\,v}{q\,B} = \frac{p}{q\,B}$$

p being the momentum of the particle, m its rest mass.

Then, we can calculate the horizontal distance d_1 between the initial position of the particle and its position when it leaves the magnet :

$$d_1 = R - \sqrt{(R^2 - L^2)}$$

with L : the length of the magnet.

We can finally calculate the horizontal distance d_2 of the particle at a distance D after the magnet exit :

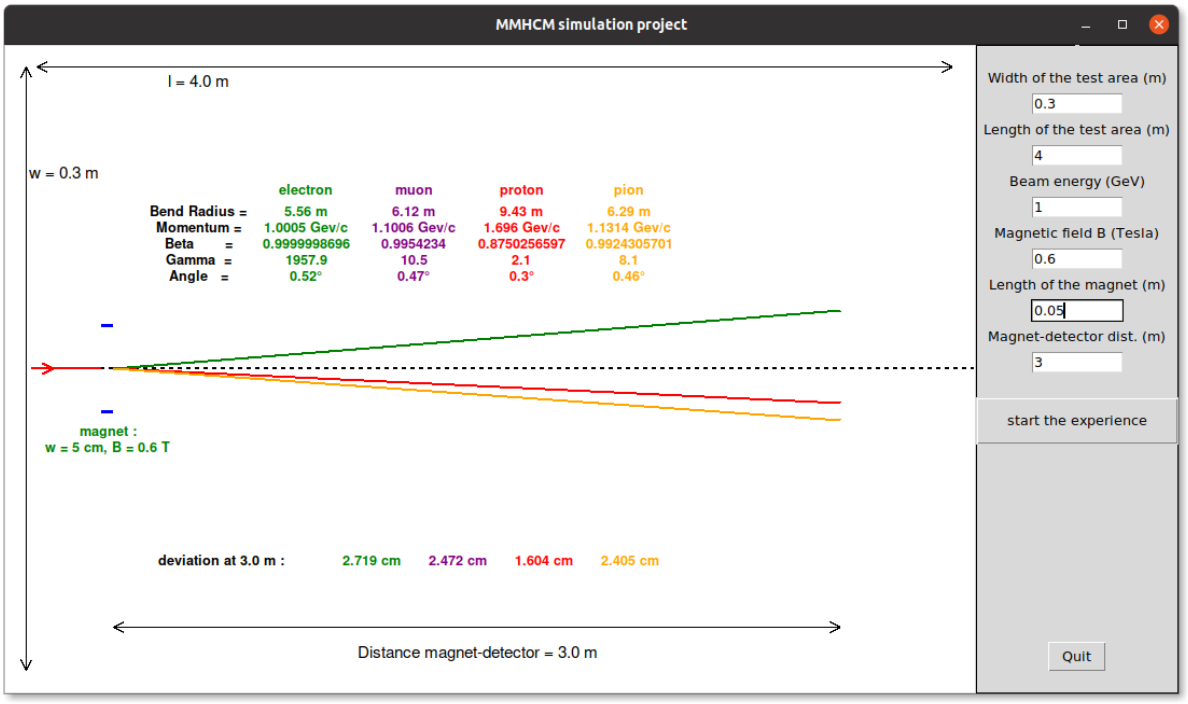
$$d_2 = D \times \tan\left(\arcsin\left(\frac{L}{R}\right)\right) + d_1$$

With D the distance between the magnet and the detector.

In the LHC or DESY experiments, the particles are relativistic. To deal with it, we just have to take the relativistic momentum $p = \gamma m v$ where γ is the Lorentz factor $\gamma = \frac{1}{\sqrt{1-\frac{v^2}{c^2}}}$ that depends of the beam energy.

Consequently, we can determine the position of the particle when it reaches the detector thanks to its mass, its charge, the magnetic field created by our magnet, the length of the magnet, the distance between the magnet and the detector and the beam energy.

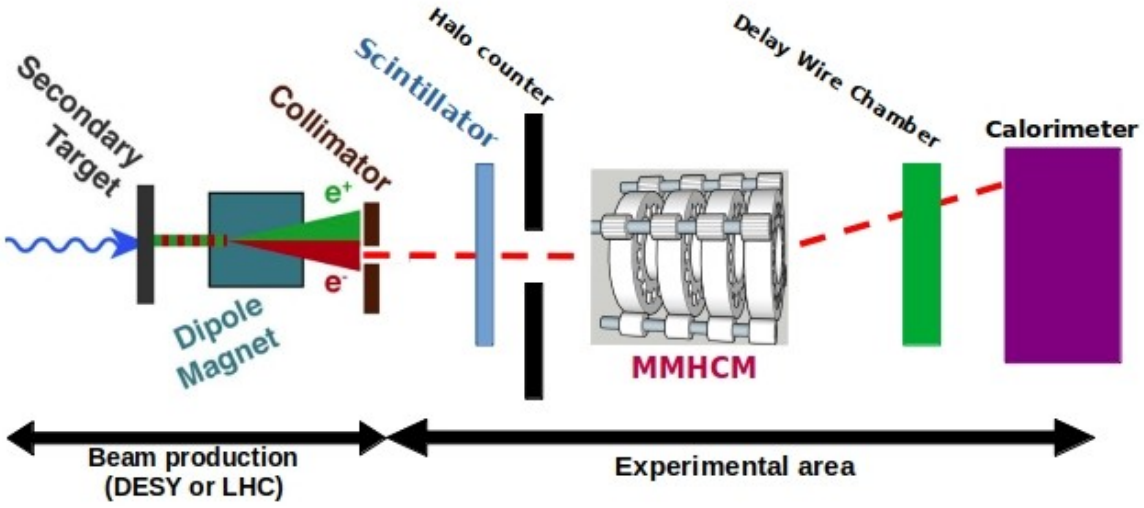
In order to automate our calculations, we have developed a Python simulation (Appendix). We'll then be able to compare the experimental results with theory. Here is an output example of the simulation :



3) Experimental setup

The accelerator produce a beam of particles (electron, positron, proton, pion…) and the beamline is collimated to select a specific momentum. In the experimental area, the beamline passes through a plastic scintillating slab and a halo counter. The opening will be set to the experimentally-determined divergence of a beam (with no targets) by that distance. This allows us to record any unlikely particles which scattered significantly on the scintillator.

Particles pass through our MMHCM and their paths are bent. Subsequently, they pass through a Delay Wire Chamber oriented orthogonal to the beamline direction. The impact point on the DWC gives the position of the particles.
Finally, they impact the calorimeter so their final energy can be measured.
Here is the outline of our experience :



Thanks to this experimental protocol, we will be able to study precisely the characteristics of our MMHCM and study how the different particles are deflected. We will also look at the influence of the placement of the different layers.

III - What we hope to take away

This project allowed us to take an interest in a new approach to physics, which is not covered in our traditional school programs.
The work and the writing of this proposal allowed us to acquire new knowledge and new skills. In addition, we got involved in real research work, creating genuine group cohesion.
We therefore hope to be able to conduct our experiment in one of the laboratories to test our magnet. We believe we can make discoveries that can be used by the scientific community.

IV - Acknowledgements

We would like to thank the CERN and particularly (particule-arly) all the team dedicated to the BL4S project, Margherita Boselli and Markus Joos for the exchanges, the presentation and the answer to our questions.
Our thanks go also to Mr. Arnaud Gniady, our math teacher and at partial time, personal physicist. He proposed us to participate to the contest, created the team, carry the project and gives a lot of his time in order to permit us to develop the ideas of MMHCM.
Special thanks to our school's headmaster, Mr. Mazars and his direction team that supported us and help us to get all the materials needed for the realization of the project.

V - Appendix

Here is the python simulation we use to calculate and represent the beam deflection :

```
#### we import libreries ###########################################################
from tkinter import *
from math import *
#####################################################################################

def start():            ####execute when you click "start the experience"
    l = 900
    w = 600

    wtrue = float(width.get())              ####Get form values
    testzone = float(length.get())
    B = float(fieldB.get())
    magnet0 = float(l_magnet.get())
    magnet = l * float(l_magnet.get()) / testzone
    nrg = float(energy.get())
    detector0 = float(distance_detect.get())
    detector = l * float(distance_detect.get()) / testzone

    can.delete('all')
    can.create_line(25, w / 2, l, w / 2, width=2, fill='black', dash=(4, 4))            ####initial drawing
    can.create_line(25, w / 2, l / 10, w / 2, width=2, fill='red')
    can.create_line(l / 20, w / 2, l / 20 - 10, w / 2 - 5, width=2, fill='red')
    can.create_line(l / 20, w / 2, l / 20 - 10, w / 2 + 5, width=2, fill='red')

    can.create_line(30, 20, l - 20, 20, width=1, fill='black')              ####write the total length of the test zone
    can.create_line(30, 20, 40, 15, width=1, fill='black')
    can.create_line(30, 20, 40, 25, width=1, fill='black')
    can.create_line(l - 20, 20, l - 30, 25, width=1, fill='black')
    can.create_line(l - 20, 20, l - 30, 15, width=1, fill='black')
    can.create_text(2 * l / 10, 35, text="l = {} m".format(testzone), fill="black", font=('Helvetica 12'))

    can.create_line(20, 20, 20, w - 20, width=1, fill='black')              ####write the total width of the test zone
    can.create_line(20, 20, 25, 30, width=1, fill='black')
    can.create_line(20, 20, 15, 30, width=1, fill='black')
    can.create_line(20, w - 20, 25, w - 30, width=1, fill='black')
    can.create_line(20, w - 20, 15, w - 30, width=1, fill='black')
    can.create_text(55, 2 * w / 10, text="w = {} m".format(wtrue), fill="black", font=('Helvetica 12'))

    can.create_line(l / 10, w / 2 - 40, l / 10 + magnet, w / 2 - 40, width=3, fill='blue')                  ####draw magnet
    can.create_line(l / 10, w / 2 + 40, l / 10 + magnet, w / 2 + 40, width=3, fill='blue')
    can.create_text(l / 10 + magnet / 2, w / 2 + 60, text="magnet :", fill="green", font=('Helvetica 10 bold'))
    can.create_text(l / 10 + magnet / 2, w / 2 + 75, text="w = {} cm, B = {} T".format(int(magnet0 * 100), B),
                    fill="green", font=('Helvetica 10 bold'))

    c = 2.99792458 * 10 ** 8                ####Constants
    q = 1.602 * 10 ** (-19)

    me = 0.511 * 10 ** (-3)                 ####4 particles masses
    mmu = 0.10566
    mp = 0.938272
    mpion = 0.140

    gammae = 1 + nrg / me                   ####gamma (relativity)
    gammamu = 1 + nrg / mmu
    gammap = 1 + nrg / mp
    gammapion = 1 + nrg / mpion

    pe = me * sqrt(gammae ** 2 - 1)         ####momentum
    pmu = mmu * sqrt(gammamu ** 2 - 1)
    pp = mp * sqrt(gammap ** 2 - 1)
    ppion = mpion * sqrt(gammapion ** 2 - 1)

    Re = ((pe / c) * q * 10 ** 9) / (q * B)         ####bending radius
    Rmu = ((pmu / c) * q * 10 ** 9) / (q * B)
    Rp = ((pp / c) * q * 10 ** 9) / (q * B)
    Rpion = ((ppion / c) * q * 10 ** 9) / (q * B)

    betae = sqrt(1 - 1 / gammae ** 2)       ####beta (relativity)
    betamu = sqrt(1 - 1 / gammamu ** 2)
    betap = sqrt(1 - 1 / gammap ** 2)
    betapion = sqrt(1 - 1 / gammapion ** 2)

    ys0e = Re - sqrt(Re**2-magnet0**2)  ####y value at the exit of the magnet
    ys0mu = Rmu - sqrt(Rmu**2-magnet0**2)
    ys0p = Rp - sqrt(Rp**2-magnet0**2)
    ys0pion = Rpion - sqrt(Rpion**2-magnet0**2)

    yse = w * ys0e / wtrue                  ####y value at the exit of the magnet (pixel)
    ysmu = w * ys0mu / wtrue
    ysp = w * ys0p / wtrue
```

```
yspion = w * ys0pion / wtrue

d0e = detector0 * tan(asin(magnet0 / Re)) + ys0e          ####y value at the detector
d0mu = detector0 * tan(asin(magnet0 / Rmu)) + ys0mu
d0p = detector0 * tan(asin(magnet0 / Rp)) + ys0p
d0pion = detector0 * tan(asin(magnet0 / Rpion)) + ys0pion

de = w * d0e / wtrue                    ####y value at the detector (pixel)
dmu = w * d0mu / wtrue
dp = w * d0p / wtrue
dpion = w * d0pion / wtrue

anglee = asin(magnet0 / Re) * 180 / pi                    ####trajectory angle value at the exit of the magnet
anglemu = asin(magnet0 / Rmu) * 180 / pi
anglep = asin(magnet0 / Rp) * 180 / pi
anglepion = asin(magnet0 / Rpion) * 180 / pi


####Draw trajectories
can.create_line(l / 10 + magnet, w / 2 - yse, l / 10 + magnet + detector, w / 2 - de, width=2,
            fill='green')
can.create_line(l / 10 + magnet, w / 2 + ysp, l / 10 + magnet + detector, w / 2 + dp, width=2, fill='red')
can.create_line(l / 10 + magnet, w / 2 + yspion, l / 10 + magnet + detector, w / 2 + dpion, width=2, fill='orange')

####Write magnet-detector distance
can.create_line(l / 10 + magnet, 9 * w / 10, l / 10 + magnet + detector, 9 * w / 10, width=1,
            fill='black')
can.create_line(l / 10 + magnet, 9 * w / 10, l / 10 + magnet + 10, 9 * w / 10 - 5, width=1, fill='black')
can.create_line(l / 10 + magnet, 9 * w / 10, l / 10 + magnet + 10, 9 * w / 10 + 5, width=1, fill='black')
can.create_line(l / 10 + magnet + detector, 9 * w / 10, l / 10 + magnet + detector - 10, 9 * w / 10 - 5, width=1,
            fill='black')
can.create_line(l / 10 + magnet + detector, 9 * w / 10, l / 10 + magnet + detector - 10, 9 * w / 10 + 5, width=1,
            fill='black')
can.create_text(l / 10 + magnet + detector / 2, 9 * w / 10 + 25,
            text="Distance magnet-detector = {} m".format(detector0), fill="black", font=('Helvetica 12'))

####Write the deviation at the detector
can.create_text(l / 10 + magnet + 100, 8 * w / 10, text="deviation at {} m :".format(round(detector0, 2)),
            fill="black", font=('Helvetica 10 bold'))
can.create_text(l / 10 + magnet + w * 0.02 / wtrue + 200, 8 * w / 10, text="{} cm".format(round(100 * d0e, 3)),
            fill="green", font=('Helvetica 10 bold'))
can.create_text(l / 10 + magnet + w * 0.02 / wtrue + 280, 8 * w / 10, text="{} cm".format(round(100 * d0mu, 3)),
            fill="purple", font=('Helvetica 10 bold'))
can.create_text(l / 10 + magnet + w * 0.02 / wtrue + 360, 8 * w / 10, text="{} cm".format(round(100 * d0p, 3)),
            fill="red", font=('Helvetica 10 bold'))
can.create_text(l / 10 + magnet + w * 0.02 / wtrue + 440, 8 * w / 10, text="{} cm".format(round(100 * d0pion, 3)),
            fill="orange", font=('Helvetica 10 bold'))

####Write all results
can.create_text(2 * l / 10 + 100, 2.6 * w / 10 - 20, text="electron", fill="green",
            font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 200, 2.6 * w / 10 - 20, text="muon", fill="purple", font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 300, 2.6 * w / 10 - 20, text="proton", fill="red", font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 400, 2.6 * w / 10 - 20, text="pion", fill="orange", font=('Helvetica 10 bold'))

can.create_text(2 * l / 10, 2.6 * w / 10, text="Bend Radius =", fill="black", font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 100, 2.6 * w / 10, text="{} m".format(round(Re, 2)), fill="green",
            font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 200, 2.6 * w / 10, text="{} m".format(round(Rmu, 2)), fill="purple",
            font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 300, 2.6 * w / 10, text="{} m".format(round(Rp, 2)), fill="red",
            font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 400, 2.6 * w / 10, text="{} m".format(round(Rpion, 2)), fill="orange",
            font=('Helvetica 10 bold'))

can.create_text(2 * l / 10, 2.6 * w / 10 + 15, text="Momentum =", fill="black", font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 100, 2.6 * w / 10 + 15, text="{} Gev/c".format(round(pe, 4)), fill="green",
            font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 200, 2.6 * w / 10 + 15, text="{} Gev/c".format(round(pmu, 4)), fill="purple",
            font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 300, 2.6 * w / 10 + 15, text="{} Gev/c".format(round(pp, 4)), fill="red",
            font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 400, 2.6 * w / 10 + 15, text="{} Gev/c".format(round(ppion, 4)), fill="orange",
            font=('Helvetica 10 bold'))

can.create_text(2 * l / 10, 2.6 * w / 10 + 30, text="Beta    =", fill="black", font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 100, 2.6 * w / 10 + 30, text="{}".format(round(betae, 10)), fill="green",
            font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 200, 2.6 * w / 10 + 30, text="{}".format(round(betamu, 10)), fill="purple",
            font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 300, 2.6 * w / 10 + 30, text="{}".format(round(betap, 10)), fill="red",
            font=('Helvetica 10 bold'))
can.create_text(2 * l / 10 + 400, 2.6 * w / 10 + 30, text="{}".format(round(betapion, 10)), fill="orange",
            font=('Helvetica 10 bold'))

can.create_text(2 * l / 10, 2.6 * w / 10 + 45, text="Gamma  =", fill="black", font=('Helvetica 10 bold'))
```

```
    can.create_text(2 * l / 10 + 100, 2.6 * w / 10 + 45, text="{}".format(round(gammae, 1)), fill="green",
                    font=('Helvetica 10 bold'))
    can.create_text(2 * l / 10 + 200, 2.6 * w / 10 + 45, text="{}".format(round(gammamu, 1)), fill="purple",
                    font=('Helvetica 10 bold'))
    can.create_text(2 * l / 10 + 300, 2.6 * w / 10 + 45, text="{}".format(round(gammap, 1)), fill="red",
                    font=('Helvetica 10 bold'))
    can.create_text(2 * l / 10 + 400, 2.6 * w / 10 + 45, text="{}".format(round(gammapion, 1)), fill="orange",
                    font=('Helvetica 10 bold'))

    can.create_text(2 * l / 10, 2.6 * w / 10 + 60, text="Angle   =", fill="black", font=('Helvetica 10 bold'))
    can.create_text(2 * l / 10 + 100, 2.6 * w / 10 + 60, text="{}°".format(round(anglee, 2)), fill="green",
                    font=('Helvetica 10 bold'))
    can.create_text(2 * l / 10 + 200, 2.6 * w / 10 + 60, text="{}°".format(round(anglemu, 2)), fill="purple",
                    font=('Helvetica 10 bold'))
    can.create_text(2 * l / 10 + 300, 2.6 * w / 10 + 60, text="{}°".format(round(anglep, 2)), fill="red",
                    font=('Helvetica 10 bold'))
    can.create_text(2 * l / 10 + 400, 2.6 * w / 10 + 60, text="{}°".format(round(anglepion, 2)), fill="orange",
                    font=('Helvetica 10 bold'))
################################################################################

def leave():
    window.destroy()  # close the window !
################################################################################

#### Creation of the window ##############################
# Main window with its title
window = Tk()
window.title("MMHCM simulation project")

# the two zones of the window : the "canvas".
l = 900                 # size of the drawing canvas
w = 600
can = Canvas(window, bg='white', height=w, width=l)                  # creation of the left canvas where we draw
can.grid(row=1, column=0)

can2 = Canvas(window)                                  # creation of the right canvas where the buttons and input fields are
can2.grid(row=1, column=1, sticky='ns')
################################################################################

#### Creation of buttons and fields ##########################
# input fields
Label(can2, text="").pack(side=TOP)
Label(can2, text="Width of the test area (m)").pack(pady=2, side=TOP)
width = Entry(can2, width=10)
width.pack(pady=2, side=TOP)
width.insert(0, 0.8)

Label(can2, text="Length of the test area (m)").pack(pady=2, side=TOP)
length = Entry(can2, width=10)
length.pack(pady=2, side=TOP)
length.insert(0, 2)

Label(can2, text="Beam energy (GeV)").pack(pady=2, side=TOP)
energy = Entry(can2, width=10)
energy.pack(pady=2, side=TOP)
energy.insert(0, 2.75)

Label(can2, text="Magnetic field B (Tesla)").pack(pady=2, side=TOP)
fieldB = Entry(can2, width=10)
fieldB.pack(pady=2, side=TOP)
fieldB.insert(0, 1.75)

Label(can2, text="Length of the magnet (m)").pack(pady=2, side=TOP)
l_magnet = Entry(can2, width=10)
l_magnet.pack(pady=2, side=TOP)
l_magnet.insert(0, 0.1)

Label(can2, text="Magnet-detector dist. (m)").pack(pady=2, side=TOP)
distance_detect = Entry(can2, width=10)
distance_detect.pack(pady=2, side=TOP)
distance_detect.insert(0, 1)

# initialize button
l = Button(can2, text='start the experience', height=2, width=20, relief=GROOVE, activebackground="dark green",
           activeforeground="white", command=start)
l.pack(pady=20, side=TOP)

Button(can2, text='Quit', command=leave).pack(pady=20, side=BOTTOM)                  # exit button

can.create_line(25, w / 2, l, w / 2, width=2, fill='black', dash=(4, 4))                  ####initial drawing before launch
can.create_line(25, w / 2, l / 10, w / 2, width=2, fill='red')
can.create_line(l / 20, w / 2, l / 20 - 10, w / 2 - 5, width=2, fill='red')
can.create_line(l / 20, w / 2, l / 20 - 10, w / 2 + 5, width=2, fill='red')

window.mainloop()                          ####window loop
```