École Publique d'Ingénieurs en 3 ans

Report

DEVELOPMENT OF SIMULATIONS OF THE BEHAVIOUR OF AN ULTRACOLD NEUTRON (UCN) DECELERATOR PROTOTYPE

the 21. August 2020,

Version 1.1

Thomas GUILBAUD
2nd Year Student
tguilbaud@ecole.ensicaen.fr

School's tutor : Gilles BAN
Laboratory tutor : Dieter RIES

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

JG|U
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

www.ensicaen.fr

Acknowledgements

I would like to acknowledge my tutor and professor Gilles Ban for putting me in contact with the Johannes Gutenberg Universität. I acknowledge my laboratory tutor Dieter Ries who trusted me during my internship. I acknowledge my coworker Christian Schmidt for helping me to develop the simulations and the new features. I acknowledge the τSPECT's team for its professionalism and pleasant atmosphere. I would like also to acknowledge Snježana Teljega from the PRISMA+ Welcome Services and Internship for the administrative follow-up.

Contents

List of Figures

NEUTRON'S LIFETIME

1. The Johannes Gutenberg Universität

The Johannes Gutenberg Universität (JGU) is one of the largest universities in Germany with more than 30,000 students from around the world. Located in Mainz in Rheinland-Pfalz, the JGU is a partner with international renowned institutes such as Max Planck Institutes or Helmholtz Institute. The research area of the university are diverse and mainly in particle physics, materials sciences or medicine [1].

1.1. The TRIGA Reactor

The JGU's campus is the home of a research nuclear reactor of the TRIGA type. This reactor is a non-power nuclear reactor designed by General Atomics. It is used by industries and laboratories for the production of radioisotopes, for treatment of tumours and fundamental research. The TRIGA reactors can be operated in a range of power between 0.1 and 16 MW and can reach a peak power of 22 GW in short pulses [2].

The JGU's TRIGA reactor either runs continuously at 100 kW power or in a pulsed mode where pulse up to 250 MW is produced for 30 ms. The pulsed mode is used for ultracold neutron (UCN) production [3]. The source uses a moderator composed of solid deuterium (sD_2) to provide a source of ultracold neutrons that can be used for different experiments [4][5]. The UCN source has been upgraded to increase the available ultracold neutron density in 2017 [6]. One of these experiments is the τSPECT experiment connected on the port D of the reactor.

1.2. The τSPECT Experiment

The τSPECT experiment tries to determine the neutron lifetime by storing the ultracold neutrons produced by the TRIGA reactor. Indeed, the measurement of this value allows to test the standard model of particle physics. Knowing this value also determines the ratio of hydrogen and helium in the early age of the universe.

There is currently a debate on the neutron's mean lifetime value because of the divergence of results found with different methods. The experiment is continuously optimised to increase the precision of the measurement under the second [7].

2. State of Art

2.1. The neutron lifetime

The ultracold neutrons (UCN) are neutrons that are in a particular range of energy. Their kinetic energy is a few hundreds of nano electron-volts (neV) which corresponds to a velocity of a couple of meter per second. When there is a great number of ultracold neutrons, the ensemble behaves like an ideal gas and the kinetic energy distribution can be approximate with a Boltzmann distribution. In this case, the temperature of the gas is a few millikelvins [8]. At these energies, the UCN can be stored in a container with a suitable material, like stainless steel or with a magnetic gradient or in a gravitational trap. Thus, measurements and experiments can be performed on the stored free neutrons.

The neutron is not stable when it is a standalone particle. It decays with a mean lifetime of around 880.0 ± 0.9s [9] and follows a beta decay.

$$n^0 \rightarrow p^+ + e^- + \bar{\nu}_e \tag{1}$$

The neutron decays into a proton and an electron that can be easily detected due to their electric charge. However, the neutron's mean lifetime is not yet fixed because of the different uncertainty due to the method of measurement. Thus, several experiments are being carried, such as the τSPECT experiment to measure the neutron lifetime precisely.

There exist two main ways to measure the mean lifetime of the neutron called the beam method and the bottle method.

2.1.1. The beam method

The beam method consists of creating a flux of cold neutrons (CN) that pass through a chamber. Then, there is a count of neutrons' beta decay inside the volume of the chamber. With this method, the mean lifetime of the neutron is around 888.0 ± 2.0s [9][10].

2.1.2. The bottle method

The storage method uses ultracold neutrons, which are contained in a vessel of a few dozen litres [6]. The walls are coated with a highly reflective material such as stainless steel or an alloy with ^{58}Ni [8][11]. Magnetic fields can be used instead of physical walls if the UCN are polarized. Compared to the beam method, the lifetime is around 879.4 ± 0.6s [9][10].

With this method, there is a risk that the neutrons with relatively high kinetic energies escape the vessel. Also, the maximum density of UCN in the vessel limits the quantity of UCN stored. Therefore, it is essential to have a maximum of ultracold neutrons inside the container, with the lowest kinetic energy to optimise the impermeability and reduce the uncertainty on the results.

2.2. Deceleration of ultracold neutrons using a piston

The τSPECT experiment wants to use a moving mirror in a long tube to Doppler-decelerate even more the ultracold neutrons at the entry of the TRIGA source. Previous calculations and simulations have been made. However, the program was too complicated, and it became entirely unusable for the team to tune the speed of the moving part to have an optimised piston. Additionally, the *Kassiopeia* framework doesn't include features or commands to move surfaces or assemblies during a simulation.

The primary mission of my internship was to improve the architecture of the current project by adding the possibilities for anyone to include in their simulations any moving part. Then, to reproduce the piston with a configuration file that can be easily modified by anyone to improve the simulations.

2.3. *Kassiopeia*, an extensible tracking particle framework

During this internship, the primary tool to simulate the behaviour of the UCN was *Kassiopeia* [12]. This framework can track particles in complex geometries with electric and magnetic fields for Monte Carlo simulations. It includes three-dimensional visualisation tools of the geometry and the track of the simulated particles. In addition, many parameters can be taken into accounts, such as the spin or the bremsstrahlung of charged particles.

Kassiopeia was first created for the KATRIN experiment to counter a lack in the simulations softwares available that don't allow the simulation of particles tracking and the generation of complex electromagnetic fields. The framework was made to be fully adaptable and extensible. It is coded in C++, which is a modern and powerful object-oriented coding language [12].

To use this framework, a user has to copy, install and build the code from the GitHub repository of the project on his machine [13]. Then, with a text editor, the user creates a configuration file in XML code following the syntax of *Kassiopeia*. The basic structure of these files can be found in the appendix A. Finally, the user executes the configuration file with a terminal prompt via a command line. Examples are given to test and experiment with the framework.

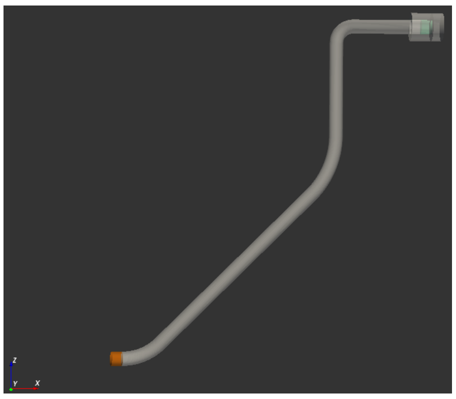
3. The simulation of ultracold neutrons in *Kassiopeia*

3.1. The beamline of the τSPECT Experiment

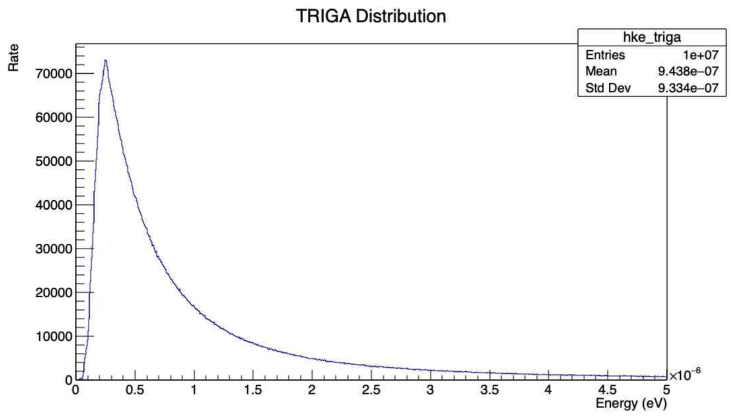
The first task of the internship was to simulate the behaviour of the ultracold neutrons inside the beamline of the experiment. It is to get the spectrum before and after this component and see if the piston improves the density of UCN in the experiment. This beamline is an essential part of the experiment because it guides the neutrons from the TRIGA source to the τSPECT experiment.

3.1.1. First model of the beamline

The first of all tasks before a simulation is to reproduce the geometry of the beamline. The *Kassiopeia* framework provides a set of basic shapes that can be assembled to construct more complex geometries [14]. An example is given in the figure 1a.



(a) Visualization of the beamline geometry with the package VTK in Kassiopeia.



(b) Spectrum of the TRIGA source of ultracold neutrons at the beginning of the beamline.

Figure 1. Configuration of the simulation of UCN in the beamline.

Then, the configuration file needs to generate the ultracold neutrons. The τSPECT experiment has access to an ultracold neutron source provided by the TRIGA reactor mentioned earlier. The spectrum of this source has to be implemented to reproduce reality. This has been made according to the chart of an article [8] provided by my tutor Dr Ries. The spectrum was created in a ROOT file [15] that can be used by *Kassiopeia* during the simulation. The spectral distribution is shown in figure 1b.

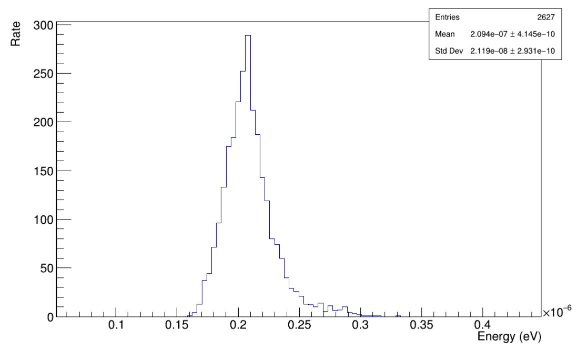
Also, the simulation doesn't need to run every neutron at each energy. Indeed, the UCN must have minimum kinetic energy to reach the top of the beamline due to the effect of the gravity that cannot be neglected for these particles. The energy can be found with the following formula [8] :

$$E = mg\Delta z \tag{2}$$

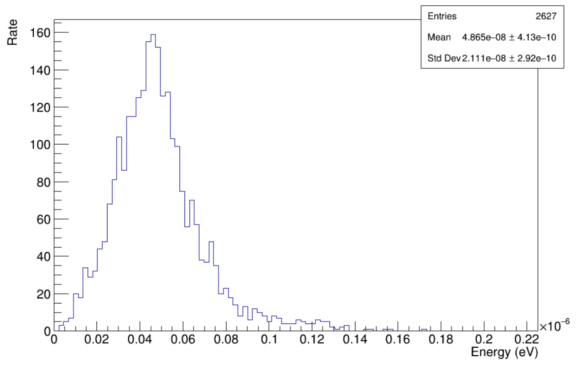
Here Δz is the height of the end of the beamline which is roughly 1.60m. So the neutrons must have a minimum kinetic energy of 146 neV to reach the entrance of the τSPECT experiment.

The key task of the simulation is to include the reflection of UCN on the inner surfaces of the beamline. *Kassiopeia* has an implemented command for this interaction [16]. An example of configuration code for stainless steel [8][11] can be found in the appendix B.

After setting other parameters, the simulation can be run properly. The spectrum of the UCN at the end of the beamline is shown in the figure 2b. By comparing this spectrum with the initial one (figure 1b), the neutrons which are produced from a full range spectrum are now in a narrow kinetic energy range between 0 neV and 140 neV. These boundaries can be explained from the previous configuration. The lower limit is linked with the minimum kinetic energy mentioned earlier to reach the top of the beamline due to gravity. The upper limit is linked with the Fermi potential of the material that composes the beamline, which is 208 neV for the stainless steel. If the neutrons have an energy higher than this potential, it means that the neutrons have even more probability of crossing the surface and leaving the beamline. Hence, the spectrum of the initial UCN that have reached the top of the beamline is represented in figure 2a.



(a) Initial UCN spectrum.



(b) Final UCN spectrum.

Figure 2. Spectrum of the neutrons that have succeeded to reach the top of the beamline.

Another trend can also be deduced from the histograms of the figure 2, the maximum probability in the spectrum has shifted to lower energy. In this case, from 209.4 neV to 48.7 neV. This phenomenon is again due to the gravity which slows down the neutrons when they go up to the end of the guide. In the end, the neutrons become slower which means a better containing during the experiment.

3.1.2. Adding slits to improve the model

According to the number of neutrons tracked during the simulation, only 0.0047% have reached the top of the beamline, which is very low. With this reference rate, the beamline can be modified to be more realistic by including slits between parts that compose the beamline. These slits induce a loss of neutrons. So, a command to terminate the track is linked to these slits which are represented by a skinny cylinder in the simulation.

As a result, the number of neutrons which have reached the top of the beamline has dropped off to 0.00093%. This decrease was expected because the configuration file includes a command to loose neutrons during the guidance.

3.1.3. Possible improvements

To improve these rates, the inner surface of the beamline can be coated with a ^{58}Ni alloy to increase the Fermi potential from 208 neV to 358 neV [6]. Another way can be to modify the shape of the beamline to include less bended part and less slits.

3.2. A moving mirror to decelerate the ultracold neutrons

After a month of simulations of the beamline and an excellent spectrum at the end, I focused on a moving mirror. This part has to slow down the neutrons from the TRIGA source when they bounce on its surface. It must have the motion of a simple piston that slides horizontally.

3.2.1. The interaction of UCN with a moving surface

The first step was to understand how an ultracold neutron behaves when it bounces on a fixed surface in the source code of *Kassiopeia*. As mentioned earlier, *Kassiopeia* already has a specific module to include the reflection interaction of ultracold neutrons which is called *ksint_surface_UCN* [16] (Appendix B).

To include a moving part (surface, space or assembly), a new component has to be designed and implemented in the source code of *Kassiopeia*. The first solution considered was to create a clone of the previous command and include the effect of a moving part. The figure 3 represents the implementation of the ultracold neutron's interaction on a surface with the command *ksint_surface_UCN* and an additional vector $\vec{p}_m$ in blue for the moving part. A more detailed explanation can be found in the corresponding paper [16].

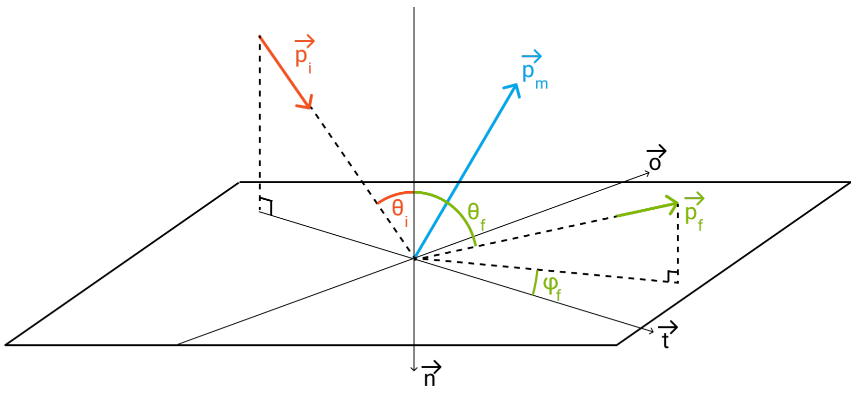


Figure 3. Drawing of the reflection of UCN on surfaces coded in the Kassiopeia (not on scale). The red part is the initial momentum of the neutron. The green part represents the final momentum of the neutron after the interaction with the surface. The blue part is the momentum of the surface which is zero for a fix surface.

To take into account the velocity of the moving part, the code subtracts two times the momentum of the part (represented in blue on the figure 3) from the initial momentum. The subtraction can be seen as a change of frame of reference. If the particle and the moving piece go in the opposite direction, it is equivalent to a particle which has more velocity relative to a fixed wall. The factor two stems from the conservation laws of momentum and energy. Due to the massive difference of scale between a neutron and a human scale moving piece, the part has an infinite mass compared to the neutron. The velocity of the neutron can be given with the following formula for a one dimension elastic collision :

$$v_f = -v_i + 2v_m \tag{3}$$

The source code can be modified by adding a new component called *ksint_moving_surface_UCN*, which is based on the previous one. The command to include this interaction is in the appendix C.

A simple test can be made to check if the particle gains or loses the correct amount of energy due to the velocity of the surface. According to the equation (3), if a particle with a velocity of 5 m/s interacts with a moving surface with a velocity of -0.05 m/s, this leads to a final velocity of -5.1 m/s. For its kinetic energy, this means a change from 130.7 neV to 135.9 neV. The figure 4 shows the result of the simulation with the configuration presented above.

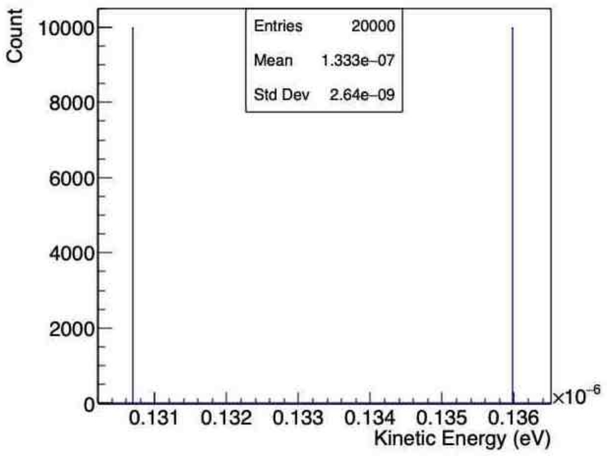


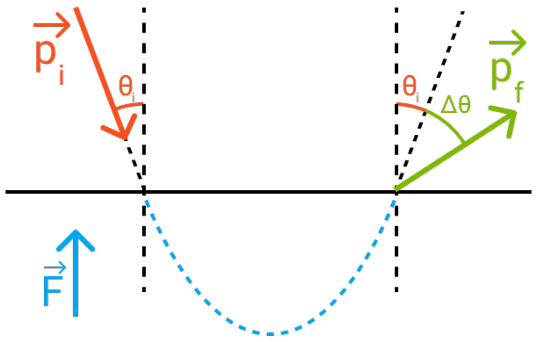
Figure 4. Histogram of the changing kinetic energy of a particle reflected by a moving surface at a speed of 5 cm/s.

As expected, the change of energy has been correctly done during the simulation, and the energy given by the spectrum is exactly the same as calculated.

However, even if the first results are excellent, the main problem is to displace the objects that compose the geometry during the simulation. This change of coordinates is much more complicated than designing a new component and needs to look deeper into the code. So it has been decided to use another simpler approach that can be tested to simulate moving parts.

3.2.2. A force field to reproduce a collision

Instead of using surfaces, the interaction would be through the action of a virtual constant force field. The figure 5a explains the principle of using a virtual force field to reproduce an elastic collision.



(a) Schema of a collision reproduced with a force field (not on scale).

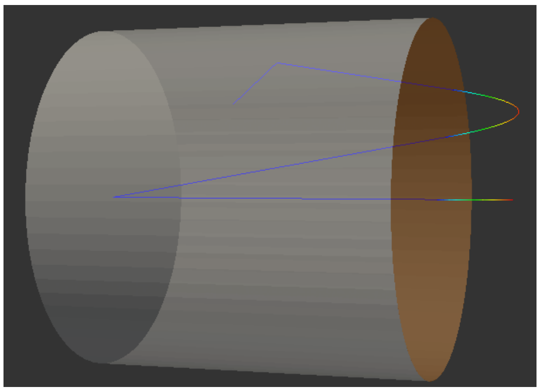

(b) Example of simulation with a reflection using weak force field.

Figure 5. Illustration of the reproduction of an elastic collision using a force field.

As the trajectory in the constant force field is a parabola, the incident and outgoing angles are the same. By applying a stronger force field, the width and height of the parabola (in blue) decrease until it becomes nearly invisible. This trend converges into an elastic collision. Then, the outgoing angle is modified by following the models used in the previous component. Additionally, if the surface moves up when the particle is below, it automatically rests more steps inside the force field. Thus, the particle gains more energy which is the behaviour expected of this component. The same principle can be applied when the moving part goes down. Also, the outgoing angle changes a little bit, but it was considered irrelevant to apply a correction. The figure 5b shows a test simulation of a weak force field that has difficulties in changing the direction of the particle. The UCN reflection effect is not included yet to simplify the process of implementation.

To include this new effect, a new component has to be designed. This component will be placed in the *trajectory* component that manage the time scale, various force field and methods of calculation during the track of a particle (see Appendix A). To avoid a global force field acting in the whole simulation, the component needs to include a switch-mode to activate or deactivate the force field if the particle is below or above the virtual surface. This is the same effect as a step function. It leads to calculating the distance between the particle and the virtual surface.

To begin with, the surface origin coordinate O'_0 must be defined at the beginning of the simulation (with O the absolute origin). Then, the configuration has the same parameters as the previous designed component *ksint_moving_surface_UCN* with the direction angles, the equation of motion and the time range. The equation (4) allows to calculate the distance between the surface and the particle at the point M where the vector $\vec{n}$ is the normal to the surface (same direction as the vector $\vec{F}$) and $r(t)$ is the time-dependent function of motion along the normal axis.

$$d = \overrightarrow{O'_0M} \cdot \vec{n} - r(t) \tag{4}$$

In the end, if the value d is positive, it means the particle is above the virtual surface, and if it is negative, the particle is below. Before adding the UCN interaction surface effect, several tests have to be made to be sure that this method works for simple case like non-moving surface and constant speed motion.

The first test is similar to the previous one presented above with the check of the correct exchange of energy for a simple bounce. The error rate for this test is under the percent so that further tests can be performed. One of these is the volume decrease in the container. The neutron is placed inside, and during the simulation, this neutron exchanges energy with the virtual moving surface and begins to gain kinetic energy. A gain of energy also means an increase in the temperature, which is the same behaviour as the compression of an ideal gas in a piston.

During the simulation, the neutron becomes faster due to the gain of energy and escapes from the trap. This effect can be seen on the figure 10 in the Appendix D. The energy of the ultracold neutron reaches the Fermi potential that composes the material of the vessel at around 7.5 seconds. The particle has now enough energy to pass through the material and escape, which was expected.

Nevertheless, when the simulation goes for a significant number of steps during the tracking, the evolution of the kinetic energy of the particle seems to increase. To compare, another simulation has been made with only fix reflective surfaces, and the change of energy was insignificant.

Also, there is a lot of vertical dots during the interaction with the virtual force field in the previous figure 10. This is because of the field which slows the particle and increases his velocity during a few steps in the simulation, just like the free fall of a ball. A solution would be to increase the force to limit the number of these dots, but this result in an instability of the simulation. Indeed, the ultracold neutron gains too much energy and breaks the conservation law of energy.

Other ways have been explored, such as applying a change of velocity by calculating the force needed to change the direction of the velocity of the particle before and after it crosses the virtual surface without success. To counter this lack, I made the decision to return to the first idea to move the current geometry during the simulation.

3.2.3. Move entire geometry during a simulation

The first task was to identify the main files where the simulation computes the displacement and decides of the termination or if the particle interacts with a specific surface. After that, a new folder called *Motions* has been added to the *Kassiopeia* project that will focus on the geometry's motions.

To move the geometries to another place, the *Kassiopeia* code provides various classes to generate and manipulate geometries shapes during the configuration phase before the simulation. One particular method will be useful and it is *Transform* inside the *KGSurface* class. This method allows to displace and rotate a surface in any direction. So, to move the surface during each step of the simulation, the code has to calculate the displacement between steps.

A new component *ksmotion_surface_translation* link with a new class has been created to move the surfaces along a linear path. It reuses the same parameters as the component *ksint_moving_-surface_UCN* without the UCN reflection parameters and effects. It also stores the initial positions of the surfaces after the initialisation ($\overrightarrow{OO'_0}$). A function *ExecuteMotion* is called for each displacement of surface. This function must have the current time of the simulation to be synchronised with the whole simulation and the interaction moving surface. To calculate the displacement, the program uses the equation (5) with $\vec{r}(t)$ the new distance along the path and $\overrightarrow{OO'}(t-dt)$ the position of the surface before the displacement.

$$\vec{d}_{t-dt \to t} = \vec{r}(t) - \overrightarrow{OO'}(t-dt) + \overrightarrow{OO'_0} \tag{5}$$

The strength of this formula is that it will replace the surfaces at their initial position at the beginning of a new particle tracking. If multiple surfaces are composing a space or an assembly, the program calculates the displacement for each surface because of the different initial coordinates.

After building the modified framework correctly, several tests have been made to check again if the behaviour was correct. The first one is a simple bounce reflection, and it gives the same result as on the figure 4, an elastic collision.

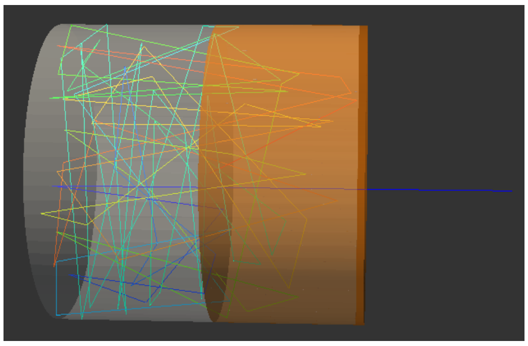


Figure 6. UCN reflection on a moving cylinder.

The second test was a volume decrease inside the container. The figure 6 illustrates this compression. This compression shows the translation of the moving part (orange) from right to left. The trajectory of the neutron during the simulation can be seen with colour that represent the kinetic energy from the lowest (red) to the highest (blue) value during the tracking. It is clear that some of the bounces made by the neutron in the orange cylinder are in the volume where the moving part was during the simulation. While the volume decreases, the ultracold neutron gains more and more energy until it exits the vessel (blue line), which was expected.

The same simulation has been made with the reverse process. In the end, the neutron loses energy and becomes slower which finally validates the implementation of a moving part in *Kassiopeia* [17]. A code sample is given in the Appendix E.

3.2.4. The τSPECT Experiment's piston

After building the new *Kassiopeia* framework, the piston of the τSPECT experiment can be modelled. The geometry is based on the configuration of the actual piston. The implementation can be seen in the figure 7.

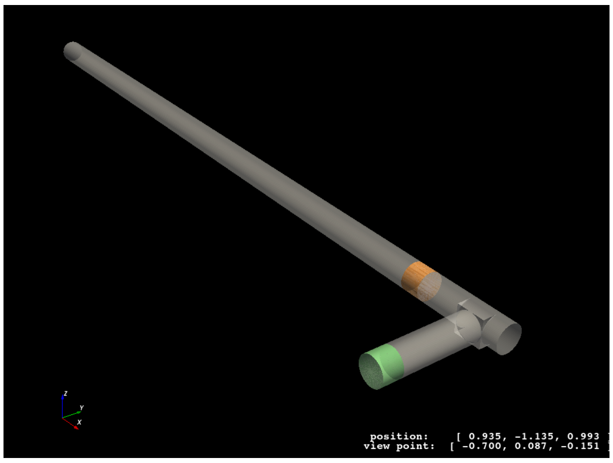
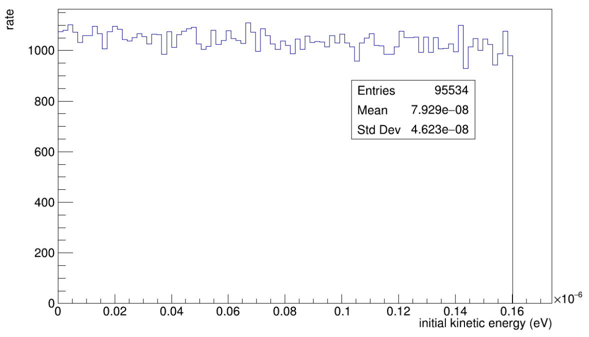
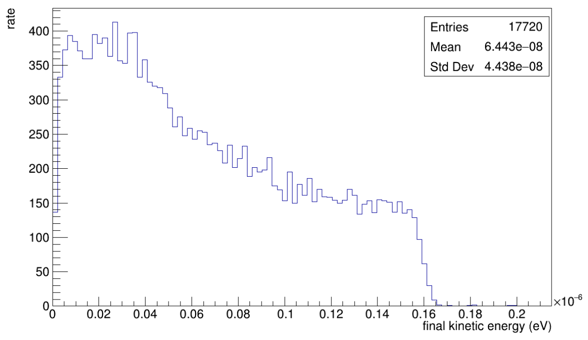

position: [0.935, -1.135, 0.993]
view point: [-0.700, 0.087, -0.151]

Figure 7. Piston of the τSPECT Experiment.

The following simulation generates a uniform spectrum to evaluate how the neutrons behave on the piston. The range of this spectrum is [0; 160] neV which is the same as the spectrum at the end of the beamline (figure 2b). The moving part (orange) moves at a constant speed backwards to decelerate the neutrons (from right to left). The speed of the moving geometry is half the velocity of the maximum probability of the spectrum at the end of the beamline (figure 2b). The one-half factor is due to the equation (3) to have the most efficient deceleration of the UCN. The program counts the neutrons that have reached the output which is represented by the green cylinder on the figure 7. This simulation made on the Mogon Cluster of the JGU university gives the spectra shown on the figure 8.



(a) Initial uniform kinetic energy.



(b) Final kinetic energy at the output detector.

Figure 8. Spectrum of the neutrons before and after the simulation inside the τSPECT piston.

As shown, the spectrum (figure 8a) has shifted to the lower energy as the relative amount of UCN at these energies has increased (figure 8b). Also the rate is different because a terminator has been placed at the position of the source to avoid an exit of the piston.

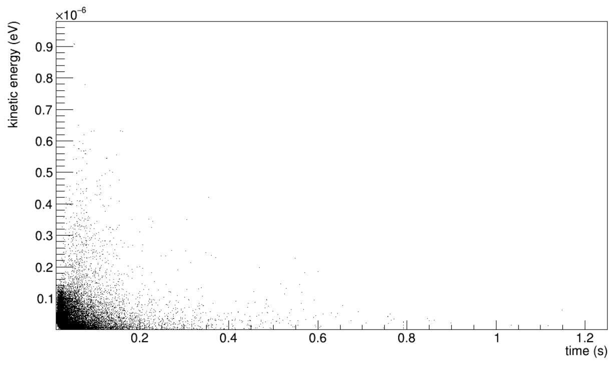


Figure 9. Sample of UCN's energy reflected by the τSPECT's piston at a speed of 1.4752 m/s.

The figure 9 is a sample of the kinetic energy of the neutrons that bounce on the moving part. The speed of the piston has been correctly chosen to nearly stop the UCN according to the huge amount of dots concentrated in low energy nearly at the beginning of the simulation.

To improve the deceleration for long simulations, the speed of the moving part has to be high at the beginning and then slows down continuously. This behaviour improves the deceleration of the neutrons when they have a little amount of kinetic energy. The last improvement of this geometry was to include a moving shutter at the entry of the piston and to integrate the TRIGA source spectrum.

This new feature implemented in the *Kassiopeia* source code can be extended due to the high flexibility of C++ and the architecture of the framework. Indeed, this first motion is along a linear path, but it is possible to add parts that can go through a specific path or rotate part on-axis. It could also be a combination of these motions.

4. Conclusion

To conclude, the behaviour of a moving mirror has been successfully implemented into *Kassiopeia*. It is easy to use and stable for the simulation of UCN. The piston component will be a crucial part for the τSPECT experiment as it will increase the efficiency of the beamline and consequently increase the density of UCN inside. The moving part features can be easily extended for other experiments that include moving pieces. They offer the possibility to investigate new experiments with dynamic components. Finally, these codes can be reused and easily modified thanks to the comments and the documentation provided for the team.

Appendix

A *Kassiopeia* XML structure

Kassiopeia is a framework that must be build by the computer. A simple way is to install the program on an Unix machine like Linux.

Before running a simulation, the user has to write a configuration file. *Kassiopeia* provides basic and complex examples of configuration such as a dipole trap or a photomultiplier. These configuration files have to follow the same structure :

```
<messages>  <!-- specification of the messaging configuration here -->
</messages>

<geometry>  <!-- specification of the geometry and geometry extensions here -->
</geometry>

<kassiopeia>  <!-- specification of the simulation elements and parameters here -->
    <electrostatic_field/>
    <magnetic_field/>
    <generator/> <!-- generate the particle at the beginning of the simulation -->
    <trajectory/> <!-- define the movement and physical laws applied to track the particles -->
    <navigation/> <!-- define the state of a particle when cross a surface or enter a space -->
    <terminator/> <!-- define the conditions to end the tracking of a particle -->
    <interaction/> <!-- define the interactions between the particles and the geometry -->
    <writer/> <!-- where to write the results of the simulation -->
    <output/> <!-- define the data to store in the output files -->
    <structure/> <!-- define links between particles and the goemetry
                    (interactions, terminators, writers, ...) -->
    <simulation/>
</kassiopeia>

<vtk_window>  <!-- optional specification of the VTK window display here (requires VTK) -->
</vtk_window>

<root_window>  <!-- optional specification of a ROOT display window here (requires ROOT)
</root_window>
```

B Implementation of the interaction between a UCN and a surface

The *Kassiopeia* framework includes a component to have the ultracold neutrons reflection on surfaces [16]. Below is an example of an implementation for the stainless steel that composes the τSPECT's beamline [8][11].

```
<ksint_surface_UCN name="int_surface_UCN"
    eta="2.e-4" alpha="5.e-6"
    real_optical_potential="2.08e-7" correlation_length="20.e-9"
/>
```

The *eta* parameter is a loss per bounce constant. The *alpha* is a constant that represents the depolarisation probability. The *real_optical_potential* is the Fermi potential in eV of the surface for ultracold neutrons which is approximately the limit energy to pass through the material. For this implementation, we have chosen a slightly higher Fermi potential (190 neV $\rightarrow$ 208 neV) to have better statistics on the results. And the *correlation_length* is the roughness of the material in meters.

C New interaction between a UCN and a moving surface

```
<ksint_moving_surface_UCN name=''int_moving_surface_UCN''
    eta="2.e-4" alpha="5.e-6"
    real_optical_potential="2.08e-7" correlation_length="20.e-9"
    theta="90." phi="180."
    value_formula="0.05*x" value_min="0." value_max="1."/>
```

This component includes the same parameters as the previous one. For the first implementation, the motion is restricted along a linear path. The user has to give the direction of the motion with the spherical angles *theta* and *phi* which follow the classical convention. Then, a function of the motion has to be given to *value_formula*. This function must be a time-dependent function and must follow the ROOT's syntax to be correctly implemented [18]. For example, a surface moves with a constant speed of 0.05 m/s has a motion equation of $r(t) = 0.05t$ which is equivalent to "0.05*x". The "x" letter indicates the variable, which is the time in second. The two next parameters are the time limits of the motion. Indeed, the user can give a range of action of his moving part, which is, in this case between 0 and 1 second. Even if the actual surface cannot move, a particle interacting with a surface which has this command will have the same effect.

D Simulation graphs of the force field method

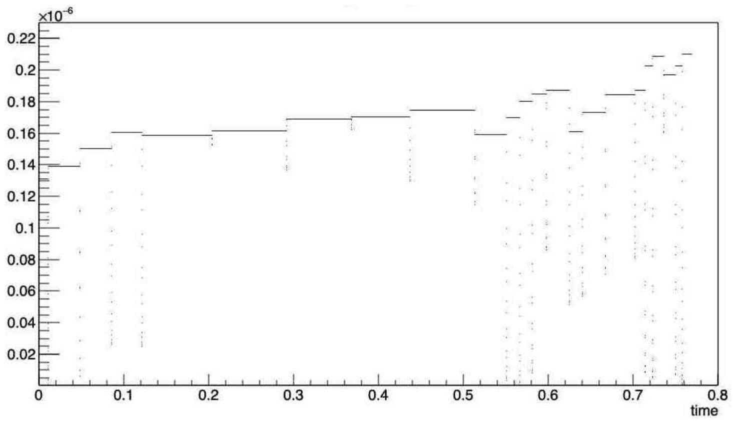


Figure 10. Evolution of the kinetic energy during a decrease of the volume at a speed of 10 cm/s.

E Implementation of a moving part in XML configuration file

A sample example of implementation of a moving surface can be found below. This example is extract from the configuration file *PistonSimulation.xml* in the *Kassiopeia* bank of examples [19]. Only remain the most important part of the code that treats the motion of a moving part.

In this example, the geometry is composed of a *moving_mirror* which is a closed cylinder and a *fix_mirror* which is a cylinder open at one end. This two parts compose the *piston* which is in a *world* space.

```
<kassiopeia>
    <!-- Motions -->
    <define name="equation_motion" value="0.05*x"/> <!-- time-dependent position function -->
    <define name="time_start_motion" value="0."/> <!-- Time (s) -->
    <define name="time_end_motion" value="1."/> <!-- Time (s) -->
    <define name="theta_motion" value="90."/> <!-- Angle (°) -->
    <define name="phi_motion" value="180."/> <!-- Angle (°) -->

    <ksmotion_surface_translation name="motion_surface_translation"
        surfaces="world/piston/moving_mirror/#"
        theta="[theta_motion]" phi="[phi_motion]"
        value_formula="[equation_motion]"
        value_min="[time_start_motion]" value_max="[time_end_motion]"/>

    <!-- Interactions -->
    <ksint_surface_UCN name="int_surface_UCN"
        eta="2.e-4" alpha="5.e-6" real_optical_potential="2.08e-7"
        correlation_length="20.e-9"/>

    <ksint_moving_surface_UCN name="int_moving_surface_UCN"
        eta="2.e-4" alpha="5.e-6"
        real_optical_potential="2.08e-7"
        correlation_length="20.e-9"
        theta="[theta_motion]" phi="[phi_motion]"
        value_formula="[equation_motion]"
        value_min="[time_start_motion]" value_max="[time_end_motion]"/>

    <!-- Structure -->
    <ksgeo_space name="space_world" spaces="world">
        <command parent="root_surface_motion" field="add_surface_motion"
            child="motion_surface_translation"/>

        <!-- Surface reflection of the UCN on fix mirror -->
        <geo_surface name="surface_reflection" surfaces="world/piston/fix_mirror/#">
            <command parent="root_surface_interaction" field="set_surface_interaction"
                child="int_surface_UCN"/>
        </geo_surface>

        <!-- Surface reflection of the UCN on moving mirror -->
        <geo_surface name="moving_surface_reflection" surfaces="world/piston/moving_mirror/#">
            <command parent="root_surface_interaction" field="set_surface_interaction"
                child="int_moving_surface_UCN"/>
        </geo_surface>
    </ksgeo_space>
</kassiopeia>
```

BIBLIOGRAPHY

[1] About Johannes Gutenberg University. https://university.uni-mainz.de/ (2020)

[2] About the General Atomics reactor TRIGA. https://www.ga.com/triga/ (2020)

[3] Specifications of the Mainzer TRIGA reactor. https://www.nuclear-chemistry.uni-mainz.de/reactor/specifications/ (July 2018)

[4] B. Lauss, D. Ries, et al., Comparison of ultracold neutron sources for fundamental physics measurements, Phys. Rev. C 95, 045503, (2017)

[5] Frei, A. et al. First production of ultracold neutrons with a solid deuterium source at the pulsed reactor TRIGA Mainz. The European Physical Journal A. 34. 119-127. 10.1140/epja/i2007-10494-2. (2007)

[6] J. Kahlenberg et al. Upgrade of the ultracold neutron source at the pulsed reactor TRIGA Mainz. arXiv:1706.07795, (2017)

[7] Presentation of the τSPECT Experiment. https://www.prisma.uni-mainz.de/facilities/triga-reactor-and-neutron-source/spect-experiment/ (January 2020)

[8] I. Altarev et al. Neutron velocity distribution from a superthermal solid 2H2 ultracold neutron source. The European Physical Journal A, 37: 9-14 (July 2008)

[9] A.P. Serebrov, A.K. Fomin. New evaluation of neutron lifetime from UCN storage experiments and beam experiments. arXiv:1104.4238, (2011)

[10] W. Wei. A New Neutron Lifetime Experiment with Cold Neutron Beam Decay in Liquid Helium. arXiv:2001.10041, (2020)

[11] F. Atchison et al. Diffuse reflection of ultracold neutrons from low-roughness surfaces. The European Physical Journal A, 44(1):23–29, (Apr 2010)

[12] Daniel Furse et al. Kassiopeia: a modern, extensible C++ particle tracking package. New J. Phys. 19 053012, (2007)

[13] Repository of the *Kassiopeia* Project folder, https://github.com/KATRIN-Experiment/Kassiopeia, (June 2020)

[14] Kassiopeia's KGeoBag, http://katrin-experiment.github.io/Kassiopeia/kgeobag_basic.html (2016)

[15] ROOT: analyzing petabytes of data, scientifically, https://root.cern/ (2020)

[16] Z. Bogorad. Ultracold Neutron Storage Simulation Using the Kassiopeia Software Package. Massachusetts Institute of Technology 2019, (June 2019)

[17] Repository of the modified *Kassiopeia* Project folder, https://github.com/Blawker/Kassiopeia, (July 2020)

[18] ROOT Formula syntax, https://root.cern.ch/doc/master/classTFormula.html, (2020)

[19] Source code of the *PistonSimulation.xml*, https://github.com/Blawker/Kassiopeia/blob/master /Kassiopeia/XML/Examples/PistonSimulation.xml, (July 2020)

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Ce stage a pour mission de développer des simulations du comportement des neutrons ultrafroids dans des guides et des pistons pour l'expérience τSPECT à Mayence en Allemagne. Le logiciel de simulation utilisé est *Kassiopeia* qui est un code open-source écrit en C++ et extensible. Une partie de ce stage consiste à étendre les possibilités de ce logiciel en incluant des parties mobiles de toute forme dans le code source. Le but est d'optimiser la vitesse du piston pour ralentir au maximum les neutrons ultrafroids avant qu'ils entrent dans l'enceinte de mesure. Ce piston permettra d'avoir des meilleures mesures du temps de vie du neutron qui n'est actuellement pas fixé par la communauté scientifique.

This internship aims to develop simulations of the behaviour of ultra-cold neutrons in guides and pistons for the τ SPECT experiment in Mainz, Germany. The simulation software used is *Kassiopeia*, which is an open-source code written in C ++ and extensible. Part of this internship was to extend the possibilities of this framework by including moving parts of any form inside the source code. The aim is to optimise the speed of the piston to decelerate the ultra-cold neutrons as much as possible before they enter the measurement chamber. This piston will make it possible to have better measurements of the neutron lifetime, which is currently not set by the scientific community.