POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Validation and Characterization of an Extended Range Neutron Detector: Design of a Control Network and Software for Multi-Device Management

Tesi di Laurea Magistrale in
Nuclear Engineering - Ingegneria Nucleare

Author: **Margherita Morriello**

Student ID: 10831560
Advisor: Prof. Stefano Luigi Maria Giulini Castiglione Agosteo
Co-advisors: Ing. Michele Ferrarini, Dott. Francesco Bonforte
Academic Year: 2023-2024

Abstract

The National Center for Oncological Hadrontherapy (CNAO) is equipped with a synchrotron capable of accelerating protons and carbon ions for hadrontherapy. Currently, CNAO is expanding to include a Boron Neutron Capture Therapy (BNCT) facility and a proton therapy unit with a rotating gantry. Additionally, the synchrotron will be upgraded to accelerate He, Li, O, and Fe ions for therapeutic and experimental purposes. Secondary neutrons, generated by the interactions between the primary beam and machine components, contribute to ambient radiation doses, with additional radiation losses possible during beam injection, extraction, or target interactions. To continuously monitor the neutron field, CNAO's radiation protection team developed a cost-effective REM counter, utilizing a Microstructured Semiconductor Neutron Detector (MSND) within a plastic moderator with cadmium and lead inserts. The radiation protection team requested the development of a system capable of managing multiple detectors and providing a thorough characterization of the detector's performance. This thesis focuses on the development of a network of REM counters, managed through centralized software, aimed at controlling and managing ambient dose equivalent measurements throughout the facility. A network was established, connecting multiple detectors to a single computer, enabling real-time data visualization, network checks, measurement control, data retrieval from Raspberry Pi devices, and database creation for storing and displaying dose information. Additionally, data from the Czech Metrology Institute (CMI) were analyzed for calibration coefficient calculations and assess linearity, and data from the CERN-EU High-Energy Reference Field (CERF) were examined to verify the dose response in mixed fields. Additional tests were conducted at CNAO, where the detectors were validated in fields generated by irradiating phantoms in the treatment room, with comparisons to the WENDI detector. The network and software were initially validated using an Am-Be neutron source, and comparisons, along with Monte Carlo simulations using FLUKA, confirmed the system's accuracy and reliability. Subsequent testing in the XPR research room further validated the system's robustness and long-term durability.

Keywords: CNAO, Neutron Fields, Neutron Detectors, Enviromental Dosimetry, Raspberry Pi

Abstract in lingua italiana

Il Centro Nazionale di Adroterapia Oncologica (CNAO) è dotato di un sincrotrone in grado di accelerare protoni e ioni di carbonio per applicazioni di adroterapia. Attualmente, il CNAO si sta espandendo per includere una struttura di Boron Neutron Capture Therapy (BNCT) e un'unità di protonterapia con gantry rotante. Inoltre, il sincrotrone sarà potenziato per accelerare ioni di He, Li, O e Fe, a scopi terapeutici e sperimentali. Neutroni secondari, generati dall'interazione tra il fascio primario e i componenti della macchina, contribuiscono ai livelli di dose ambientale di radiazione, con ulteriori perdite radiative che possono verificarsi durante i processi di iniezione, estrazione o interazione con i bersagli. Per monitorare continuamente il campo di neutroni, il team di radioprotezione del CNAO ha sviluppato un contatore REM economico, che utilizza un Microstructured Semiconductor Neutron Detector (MSND) all'interno di un moderatore di plastica con inserti di cadmio e piombo. Il team di radioprotezione ha richiesto lo sviluppo di un sistema in grado di gestire più rivelatori e di fornire una caratterizzazione completa delle prestazioni del rivelatore. Questa tesi si concentra sullo sviluppo di una rete di contatori REM, gestiti tramite un software centralizzato, per il controllo e la gestione delle misure di Equivalente di Dose Ambientale in tutta la struttura. È stata creata una rete, collegando più rivelatori a un unico computer, consentendo la visualizzazione in tempo reale dei dati, la verifica della rete, il controllo delle misurazioni, il recupero dei dati dai dispositivi Raspberry Pi e la creazione di un database per l'archiviazione e la visualizzazione delle informazioni sulla dose. Inoltre, i dati del Czech Metrology Institute (CMI) sono stati analizzati per calcolare il coefficiente di calibrazione e verificare la linearità, mentre i dati del CERN-EU High-Energy Reference Field (CERF) sono stati esaminati per verificare la risposta in dose in campi misti. Ulteriori test sono stati condotti al CNAO, dove i rivelatori sono stati validati in campi generati irradiando fantocci nella sala di trattamento, confrontando i risultati con il rivelatore WENDI. La rete e il software sono stati inizialmente validati utilizzando una sorgente di neutroni Am-Be, e i confronti con le simulazioni Monte Carlo tramite FLUKA hanno confermato l'accuratezza e l'affidabilità del sistema. Ulteriori test nella sala di ricerca XPR hanno poi validato la robustezza e la durabilità a lungo termine del sistema.

Parole chiave: CNAO, Campi Neutronici, Rivelatori di Neutroni, Dosimetria Ambientale, Raspberry Pi

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

Introduction **1**

1 Radiation Protection at CNAO **3**

1.1 Hadron Therapy . . . 3

1.1.1 CNAO Synchrotron and Protontherapy with Rotating Gantry . . . 7

1.2 BNCT . . . 9

1.3 Radiation Protection at CNAO . . . 12

1.3.1 Nuclear Reactions at Hadron Therapy Energies . . . 13

1.3.2 Neutron fields at CNAO . . . 14

1.4 Mechanisms for neutron detection . . . 17

1.4.1 Nuclear Reactions for Thermal Neutron Detectors . . . 18

1.4.2 REM Counters . . . 21

1.5 Extended Rem Counter . . . 25

1.5.1 MSND Detector and Moderator . . . 26

1.5.2 Previous characterization and validation of the Rem counter . . . 28

1.6 Aim of the thesis . . . 31

2 Software Development for Extended Rem Counter Control **33**

2.1 Previous work . . . 34

2.1.1 Previous visualization software . . . 34

2.1.2 Previous counting program . . . 34

2.2 CNAO Network . . . 35

2.2.1 New counting program . . . 36

2.2.2 Advanced software version . . . 37
2.2.3 Network functionality verification . . . 38
2.2.4 Initiating and halting measurements . . . 41
2.2.5 Requesting and retrieving files from the Raspberry Pi devices . . . 47
2.2.6 Creating and displaying the database . . . 49
2.2.7 Automatic Network control . . . 53
2.2.8 Operational pathways of the software . . . 53
2.3 Measurements at CNAO with an Am-Be source for software verification . . 54
2.3.1 Setup and network architecture . . . 54
2.3.2 Measurement Results . . . 57
2.3.3 Comparison with FLUKA simulations . . . 57

3 Characterization and validation of the Rem counter 61
3.1 Czech Metrology Institute facility . . . 61
3.1.1 calibration coefficient evaluation at CMI facility . . . 62
3.1.2 Linearity test of the Rem Counter at CMI facility . . . 63
3.2 CERN-EU high-energy reference field . . . 64
3.2.1 Measurements performed at CERF facility . . . 66
3.2.2 Results at CERF facility . . . 67

4 Experimental Campaigns at CNAO 71
4.1 Measurements performed in treatment room number three . . . 71
4.1.1 Measurement Setup and Geometry . . . 72
4.1.2 Experimental Comparison with WENDI detector results . . . 73
4.2 Measurements performed in the experimental room XPR . . . 76
4.2.1 Measurement Setup and Geometry . . . 77
4.2.2 Experimental Results . . . 79

5 Conclusion and Future Developements 83

Bibliography 87

A Appendix A 91
A.1 Python Code . . . 91
A.1.1 Libraries . . . 91
A.1.2 Pathways of the software . . . 91

A.1.3 Log File Section 93
A.1.4 Ping code 93
A.1.5 Ping status window 94
A.1.6 File Esterno Function Code 95
A.1.7 IP Selector Function Code 96
A.1.8 Filename and Measure Time 97
A.1.9 Start Function Code 98
A.1.10 Stop Function Code 100
A.1.11 Connecting to Raspberry Pi Function Code 101
A.1.12 File Request Function Code 102
A.1.13 Update DB 103
A.1.14 Show Data 106
A.1.15 Server Raspberry Function Code 119
A.2 C Code 120
A.2.1 Counting Program 120
A.2.2 Converting Program 123

B Appendix B **125**
B.1 Collected data during measurements in treatment room three 126

C Appendix C **133**
C.1 The Monte Carlo code 133
C.1.1 The Monte Carlo code FLUKA 133

List of Figures **135**

List of Tables **139**

Introduction

Radiation therapy is a medical treatment that uses ionizing radiation to control or destroy malignant cells, commonly as part of cancer therapy. Ionizing radiation consists of subatomic particles or electromagnetic waves with sufficient energy to ionize atoms or molecules by dislodging electrons. This type of radiation can be generated through nuclear reactions, radioactive decay, or by accelerating charged particles. Artificial sources of ionizing radiation include devices such as nuclear reactors, particle accelerators, and x-ray machines. Ionizing radiation works by damaging the DNA of cancer cells, ultimately leading to their destruction. To minimize harm to surrounding healthy tissues, such as the skin or organs that the radiation must pass through to target the tumor, precisely shaped radiation beams are used. These beams are directed from different angles, converging at the tumor site. This approach allows for a higher radiation dose to be delivered to the tumor while limiting exposure to the surrounding healthy tissues. Hadrontherapy is a specialized form of radiotherapy used for the treatment of tumors that are often inoperable or resistant to conventional radiotherapy. Unlike conventional radiotherapy, which relies on x-rays or electrons, hadrontherapy uses protons and carbon ions. These particles, known as hadrons, are heavier and carry more energy than electrons, making them even more effective at destroying cancer cells. For the production of these particles, accelerators are typically used. However, during the operation of high-energy accelerators, radiation can interact with the surrounding environment, potentially causing activation of components and infrastructure. The CNAO in Pavia is one of only six centers worldwide that performs hadrontherapy using both protons and carbon ions, accelerated by a synchrotron. In recent years, CNAO has initiated an expansion project to install a Boron Neutron Capture Therapy (BNCT) facility. BNCT is a type of cancer treatment based on the (n, α) reaction with Boron-10. Additionally, the center is preparing to enhance its capabilities by adding a proton therapy unit equipped with a rotating gantry. Furthermore, the existing synchrotron will undergo an upgrade to accelerate helium, lithium, oxygen, and iron ions, both for therapeutic and experimental purposes. The radiation protection group at CNAO is responsible for ensuring that the dose received by the public and personnel stays below regulatory limits. To achieve this, the radiation protection team at CNAO con-

ducts regular measurements using well-established neutron detectors. Detecting neutrons typically relies on indirect methods, as neutrons, being neutral, do not interact directly with electrons in matter. There are two main types of neutron interactions with matter. Neutrons can scatter off a nucleus, transferring some of their kinetic energy to it, which may result in ionization if sufficient energy is transferred. Alternatively, neutrons can trigger nuclear reactions that produce prompt, energetic charged particles such as protons and alpha particles. These charged particles generate a measurable current as they lose energy inside the detector's active volume. Most neutron detectors combine a target material designed to facilitate this conversion with a conventional radiation detector, such as semiconductor detectors or proportional counters. However, these types of detectors are often expensive and difficult to repair. In anticipation of the planned expansions, CNAO's radiation protection team has designed a new REM counter that is both cost-effective and easy to maintain. This REM counter incorporates a Microstructured Semiconductor Neutron Detector (MSND), housed within a plastic moderator enhanced with cadmium and lead inserts to broaden its energy detection range. For comprehensive radiation surveillance throughout the entire facility, multiple detectors will be strategically placed at key locations both inside and outside CNAO. These placements are chosen to ensure optimal coverage of high-risk areas where neutrons might be present, such as treatment rooms, accelerator spaces, and to monitor the entire facility. To facilitate seamless control and data management of this distributed network of detectors, the development of a dedicated software platform and communication network became essential. This thesis focuses on the design and implementation of a system that connects all detectors to a central server, allowing for streamlined control, real-time data acquisition, and continuous monitoring of radiation levels. The network enables remote control of each detector, allowing operators to start or stop measurements, retrieve data logs, and store dose readings in a centralized database. Moreover, the custom-built software provides real-time data visualization, enabling users to track integral neutron counts and total doses.

This thesis is dedicated to the characterization and validation of the detector developed at CNAO, ensuring its proper functionality and reliability. A key aspect of this work involves the analysis of data collected at both the CMI and CERF facilities, which allowed for a thorough evaluation of the detector's performance in different operational conditions. The thesis also covers the development of the software that interfaces with the detector, outlining each stage of the development process. Furthermore, a detailed analysis of the system's real-time performance in a live environment is provided, highlighting its strengths and identifying areas for potential improvement. This comprehensive approach ensures a robust assessment of the detector's capabilities and its readiness for practical application.

1 | Radiation Protection at CNAO

The National Centre for Oncological Hadrontherapy (CNAO), located in Pavia, Italy, is a hospital and research center dedicated to hadron therapy. Notably, it is one of only six centers in the world that uses both protons (p) and carbon ions (C^{6+}) for hadron therapy purposes [1]. In recent years several hadron therapy centres are in operation worldwide and some are being constructed or planned. Since the 1990s, the number of patients treated with protons has reached 290,000 and grows by more than 35,000 per year. Carbon ion therapy has been used for about 40,000 patients to date [1]. Since 2011 when the patient treatments started , CNAO has treated over 4,000 patients, averaging between 500 and 700 annually [2]. CNAO is preparing to enhance its facilities by adding a proton therapy unit equipped with a rotating gantry and a Boron Neutron Capture Therapy (BNCT) facility. Additionally, the current synchrotron will expand its capabilities to accelerate helium, lithium, oxygen, and iron ions for both therapeutic and experimental purposes.

1.1. Hadron Therapy

Hadron therapy is a form of radiotherapy used for the treatment and care of tumors that are often inoperable or resistant to traditional radiotherapy treatments. Unlike traditional radiotherapy, which uses X-rays or electrons, hadron therapy involves the use of protons and carbon ions. These particles have the advantage of being heavier and more energetic than electrons, making them even more effective at destroying cancer cells. This kind of therapy is a highly precise form of treatment in which hadrons are accelerated to high energies using a powerful particle accelerator. When the cancer cells are hit, the DNA in their nuclei is deeply damaged. The cells die, and the immune system eliminates them. More specifically, the overall aim of radiation therapy is to maximize absorbed dose within the tumor to achieve local control and minimize the dose and cell kill within normal tissues to decrease treatment-related acute and late morbidity. This concept is termed the “therapeutic ratio". From a physical perspective, the absorbed dose is the mean energy imparted in a volume of mass, ∂m, defined as:

$$D = \frac{\partial \overline{E}}{\partial m} \tag{1.1}$$

The unit of absorbed dose is J/kg, and its special name is Gray (Gy). The therapeutic

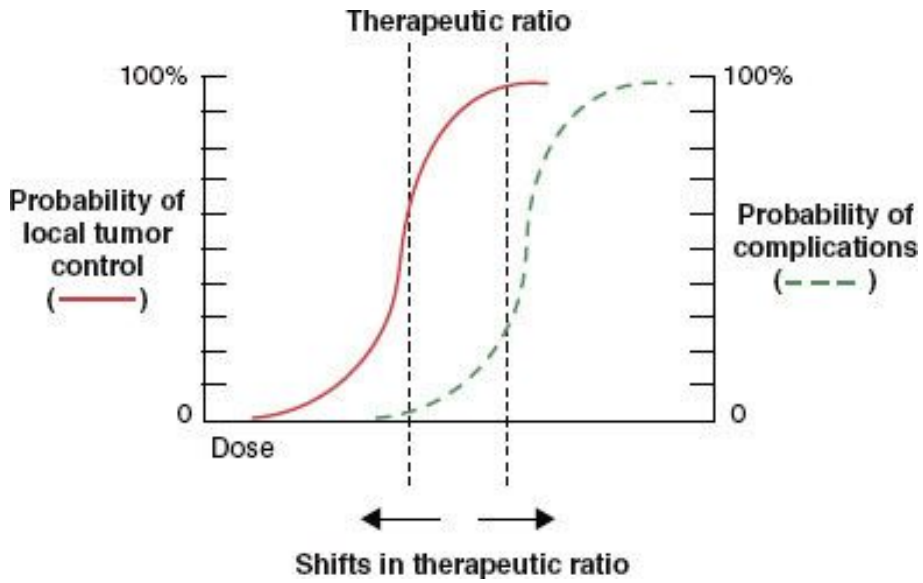

Figure 1.1: Dose-effect curves for tumour local control and complication insurgence.

ratio [3] is defined based on the dose-effect curves illustrated in Figure 1.1. These curves plot the probability of tumor control and the probability of complication onset against the dose. Ideally, the curve representing local tumor control should be at lower dose values (i.e., to the left) compared to the curve representing complication onset. This arrangement ensures effective treatment. The optimal dose, according to the radiotherapy principle, maximizes the probability of local control while minimizing the risk of complications. The therapeutic ratio is calculated as the ratio between the dose that induces complications in 50% of the treated patients and the dose that achieves local control in 50% of the treated patients. Enhancing a radiotherapy technique for a specific pathology should aim to increase this therapeutic ratio compared to the current state of art. The high ballistic precision of hadron therapy is one of the key reasons why the therapeutic ratio for certain tumors, such as those located near highly radiosensitive tissues or organs is improved compared to treatments with conventional X-ray therapy. From a physical perspective, this behavior can be explained by the different energy deposition patterns of various particles. The energy loss of a charged particle in a target can be expressed as a function of the target's density (ρ), the target charge (Z),the atomic number of the target (A), the incident ion charge (z), and the velocity of the incident ion (v), as follows:

$$\frac{dE}{dx} \propto \rho \frac{z^2}{v^2} \frac{Z}{A} \tag{1.2}$$

Electron beams, X-rays of different energies, and protons penetrate human tissue in different ways, as showen in Figure 1.2. Electrons travel a very short distance and are useful only for areas near the skin. X-rays penetrate more deeply, but the absorbed dose in the tissue exhibits an exponential decay with increasing thickness. In contrast, for protons and heavier ions, the dose increases with depth until it reaches the Bragg peak, which occurs just before the end of their path. Beyond this peak, the dose drops to zero (for protons) or nearly zero (for heavier ions). The advantage is a reduced energy deposition in the healthy tissue surrounding the target, resulting in less collateral damage. Thanks

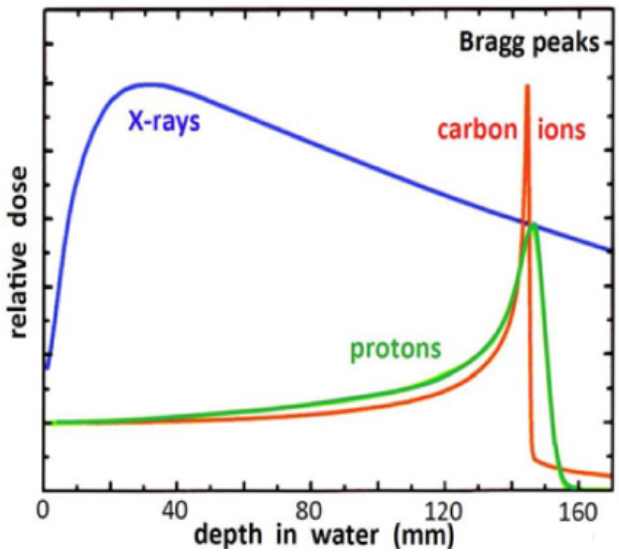

Figure 1.2: Dose as function of the depth in water.

to this behavior, there is a highly selective energy release, intensely targeting only tumor cells while sparing healthy tissues. To cover the entire tumor, a superposition of Bragg peaks with varying heights and depths is necessary, resulting in a Spread Out Bragg Peak (SOBP). Moreover, carbon ions are more precise than protons because, due to the larger mass, they have a reduced longitudinal range variation (straggling) and a smaller lateral scattering thus a sharper lateral penumbra. In addition to its high precision and preservation of healthy tissues, radiobiological studies [4] suggest that hadron therapy has the ability to treat radioresistant tumors. The energy release mechanism of hadrons causes a large number of breaks in the chemical bonds in biological macromolecules, particularly in

DNA. While DNA has the ability to repair itself, if the damage is extensive, the cell's repair mechanisms become less effective, leading to inactivation or cell death. In traditional radiotherapy, DNA damage is modest, but with carbon ion therapy, the high number of breaks can effectively target tumors that are resistant to traditional therapy. Carbon ions have a different action on the cells of the traversed organs with respect to protons or X-rays. At the end of their range the energy deposition per unit length (the Linear Energy Transfer or LET) is producing a larger number of irreparable double strand DNA breaks with a much higher probability of cell killing [5]. This action is usually expressed in terms of the Relative Biological Effectiveness (RBE) defined as:

$$RBE = \frac{D_{ref}}{D} \tag{1.3}$$

where D_{ref} is the photon dose and D the dose of the particle radiation leading to the same biological effect. The RBE of carbon ions on the tumour can be as high as 3, thus allowing the delivery of a higher dose in the tumour target with respect to photons and protons, while keeping the same dose in the surrounding normal tissues, or the same dose in the tumour and reduced doses and damages to the normal tissues. Another quantity well known in literature [6], is the so-called Oxygen Enhancement Ratio (OER):

$$OER = \frac{D}{D_0} \tag{1.4}$$

where D is the absorbed dose required to induce a given biological effect in the tissue under study (e.g., the tissue at its real oxygen concentration) and D_0 is the dose which would be required to generate the same effect if the tissue were completely oxygenated in air at standard pressure. The oxygen effect, quantified by OER, decreases with rising LET, suggesting a clinical advantage of high-LET radiotherapy with heavy ion beams, such as carbon therapy which has an OER of approximately 1,compared with low-LET photon and proton irradiation [7]. This property enhances the effectiveness of ion therapy in treating hypoxia-related radio-resistant tumors, enabling the eradication of tumors that are resistant to both X-ray and proton therapy.

1.1.1. CNAO Synchrotron and Protontherapy with Rotating Gantry

The construction of the CNAO facility has been completed at the beginning of 2010, Figure 1.3. Inside the building the following different functional areas are present:

- the outpatient service;
- the imaging service (diagnostics in images and nuclear medicine);
- the therapeutic service (hadrontherapy);
- the administrative departments and the offices of personnel;
- technological areas, including electrical power plants, thermomechanical and special plants for the operation of the building;
- the technical areas at the service of high technology (synchrotron).

The synchrotron Figure 1.4, located in a 1600-square-metre bunker in the heart of the CNAO, has the shape of a 25-metre diameter and 80-metre circumference ring and is isolated from the rest of the structure with shielding for radiation in concrete in the range from 2 to 6 metres.

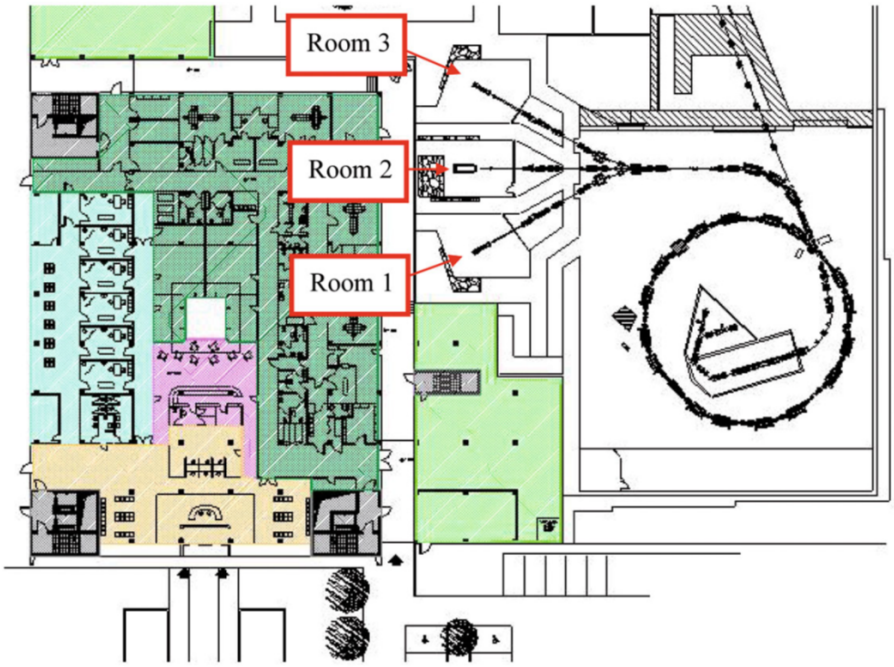

Figure 1.3: The layout of the underground floor.

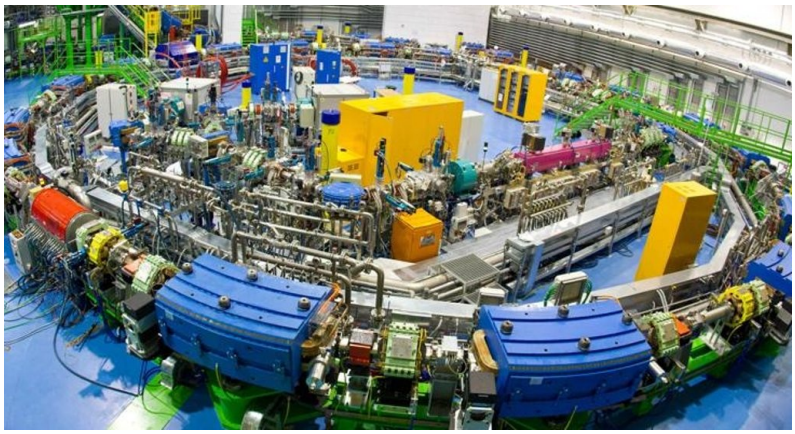

Figure 1.4: The synchrotron.

In two different areas inside the ring there are two identical-sources-type ECR (Electron Cyclotron Resonance) are operating, one using H_2 and the other CO_2 gas [8] . Simply by changing the gas used to produce plasma and optimizing the settings of certain parameters, such as RF power and the potential of extraction electrodes, several kinds of beams can be produced. In this way each source is able to compensate the possible absence of the other. To generate helium, lithium, oxygen, and iron ions (new ions), a third ECR source called AISHa (Advanced Ion Source for hadron therapy) will be installed [9]. These beams are sent to the radiofrequency quadrupole (RFQ) that provides a suitable beam to be pre-accelerated by the linear accelerator (LINAC). It therefore accelerates, longitudinally bunches and transversally adjusts the beam. The energy at the entrance to the RFQ is 8 keV/u while the energy output is 400 keV/u. The LINAC structure is of interdigital (IH) type that accelerates the beam from 400 keV/u to 7 MeV/u in one 3,8 m long tank, [10]. The pre-accelerated beam is sent to the synchrotron, where it is further accelerated to reach kinetic energies of up to 250 MeV for protons and up to 400 MeV/u for carbon ions. In Figure 1.5 the synchrotron layout with the sources and the treatment rooms is shown. After acceleration, the beam can be directed to three treatment rooms (TR1, TR2, and TR3) as well as one experimental room (XPR).TR1 and TR3 a fixed horizontal beam lines are installed, while in TR2 has both horizontal and vertical beam lines. Finally, XPR features a fixed beam line with four isocenters. In the near future, a proton therapy facility composed by a proton accelerator (Hitachi) serving the new rotatig gantry will be installed at CNAO, Figure 1.7. The gantry allows the beam to move around the patient, reducing patient positioning times on one hand, and increasing therapeutic solutions on the other. This will ensures independent productivity in terms of patient throughput and it is a back-up solution for patient treatments in case of maintenance or

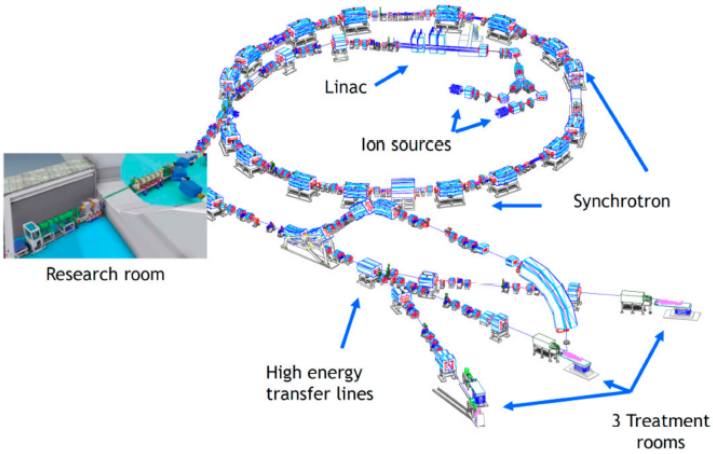

Figure 1.5: Model of the compact layout of the CNAO synchrotron.

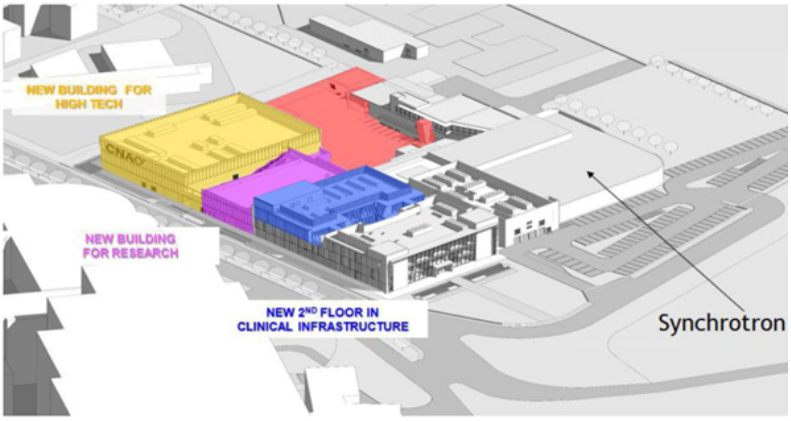

Figure 1.6: Model of the new buildings (in colours) in construction at CNAO.

downtime of the existing synchrotron.

1.2. BNCT

Boron neutron capture therapy (BNCT) is a kind of hadrontherapy treatment consisting in the selective absorption of ^{10}B loaded molecules inside tumour cells and in the subsequent irradiation with neutron beams which should be thermal having an energy of 0.025 eV at the depth where the tumour is seated. This energy level triggers the neutron capture reactions by boron, resulting in the following:

$$n +^{10} \mathrm{Li} \rightarrow \begin{cases} ^{7}\mathrm{Li} + \alpha + 2.79\,\mathrm{MeV} & (6\%) \\ ^{7}\mathrm{Li}^{*} + \alpha + 2.31\,\mathrm{MeV} & (94\%) \end{cases} \tag{1.5}$$

Figure 1.7: Model of the proton therapy technology proposed by Hitachi.

The ^{10}B neutron capture reaction can proceed through two different ways. In the first case, an α particle and a recoiling (^{7}Li) nucleus in the ground state are emitted with a Q-value of 2.79 MeV. In most cases (94%) boron undergoes decay, resulting in the emission of an α particle and a recoiling ^{7}Li nucleus in an excited state,with an initial kinetic energy of 1.47 MeV (α) and 0.84 MeV (^{7}Li), respectively, while 0.48 MeV are transported by the ^{7}Li de-excitation gamma ray. The range in tissue of a 1.47 MeV α-particle and of a 0.84 MeV ^{7}Li ion is 8 μm and 4 μm, respectively. Since the average dimension of a human cell is about 10 μm, all the kinetic energy of the high LET particles is deposited in the cell loaded with the boron isotope. Hence, if a boron loaded molecule is absorbed in the nucleus of a tumour cell, the probability of induction of non reparable damage in the DNA is high. The advantages of this kind of therapy are given by its selectivity and by the fact

Beam Quality Component	**Definition**	**Reference Value**
Therapeutic epithermal flux	ϕ_{epi}	$> 5 \times 10^{8}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$
Thermal to epithermal flux ratio	$\frac{\phi_{th}}{\phi_{epi}}$	< 0.05
Beam directionality	$\frac{J}{\phi_{epi}}$	> 0.7
Fast neutron dose per unit epithermal fluence	$\frac{D_H}{\int \phi_{epi}(t)\,dt}$	$< 7 \times 10^{-13}\,\mathrm{Gy\,cm}^2$
Gamma dose per unit epithermal fluence	$\frac{D_\gamma}{\int \phi_{epi}(t)\,dt}$	$< 2 \times 10^{-13}\,\mathrm{Gy\,cm}^2$

Table 1.1: Reference neutron beam quality factors. [11]

that the dose is deposited in about 12 μm by high LET particles. The main challenge

is represented by the boron carrier: the development of pharmaceuticals for delivering boron to tumor cells while minimizing its concentration in healthy cells and in the blood remains a significant area of ongoing research. Moreover, the thermal and epithermal neutron fields required for irradiation are very complex to be set up by minimizing the other components of the radiation field (photons and fast neutrons). Specifically the beam must meet quality standards as outlined in Table 1.1. In the table, Φ_{epi} and Φ_{th} represent the epithermal and thermal neutron flux, respectively. J indicates the neutron current, D_H represents the fast neutron dose, and D_γ stands for the gamma dose. In response to this challenge, there has been an increasing focus in recent years on developing accelerators suitable for installation in hospitals or facilities such as CNAO, specifically designed for clinical treatments. CNAO will host a BNCT facility based on a tandem accelerator that generates neutrons through the endothermic reaction:

$$^{7}\mathrm{Li} + p \rightarrow^{8} \mathrm{Be} + n \quad \text{- } 1.65\,\mathrm{MeV} \tag{1.6}$$

in which a proton impinges on a ^{7}Li and a ^{10}Be and a neutron are produced. It should be stressed that neutrons can also be generated via the ^{7}Li(p,n) ^{10}Be at energies higher than about 2.4 MeV. For this reaction channel, ^{7}Be is generated at its 0.478 MeV excited level. Therefore, 0.478 MeV gamma rays are also generated from the de-excitation of ^{7}Be. The cross section for the reaction in reported in Figure 1.8. For lithium targets the proton

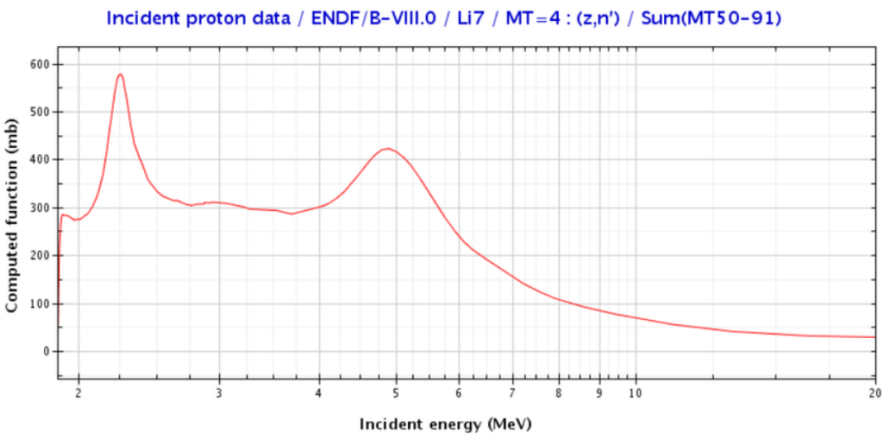

Figure 1.8: Cross section for ^{7}Li(p,n) ^{10}Be reaction.

bombarding energy is usually set up at 2.5 MeV for maximizing the neutron yield. The acceleration energy is slightly higher than the one corresponding to the maximum cross section (2.3 MeV), since ionization energy losses of the projectile inside the thick target are accounted for. The BNCT facility at CNAO will be capable of accelerating protons to

2.5 MeV with a maximum current of approximately 10 mA. The ideal irradiation neutron field should be characterized by the maximum thermal fluence rate at the depth where the boron-loaded tumour is seated. To address the ideal irradiation neutron field and achieve the desired energy, a Beam Shaping Assembly (BSA) is typically used, Figure 1.9. The

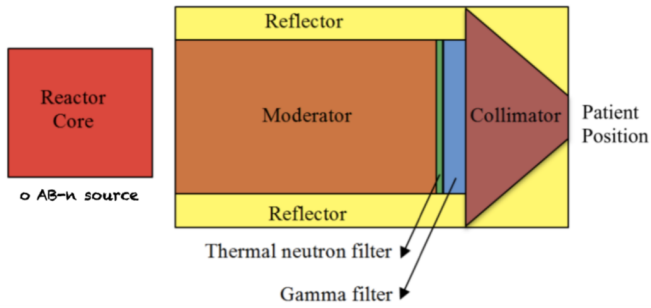

Figure 1.9: Beam Shaping Assembly design.

main elements of a BSA are:

- moderator: made by MgF_2, AlF_3 or CaF in order to slow-down neutrons
- reflector: made by Pb in order to maximize the neutron fluence in the region of interest;
- thermal neutron filter made by Cd;
- gamma filter made by Pb in order to minimize photons generated by the interaction of neutrons with the moderator and the reflector;
- collimator in order to design the final beam shape for the treatment.

1.3. Radiation Protection at CNAO

As previously mentioned, the CNAO expansion project involves the installation of a new proton therapy facility with a rotating gantry and a BNCT facility. The design and development of these new facilities require extensive studies to ensure compliance with current radiation protection regulations [12]. These research activities are carried out in collaboration with several universities and focus on various aspects, such as evaluating the radiation physics of the secondary fields generated within the synchrotron room, determining the maximum permissible workloads for the new ions, and designing the shielding for the BNCT facility. Additionally, research has been conducted on the characterization of an extended-range REM counter for neutron dosimetry and its application in the BNCT field. In all these applications, the primary radiation protection challenge

is the generation of secondary neutrons, which can interact with matter and induce secondary activation in materials. Due to the complexity of the physical processes involved, Monte Carlo methods are frequently employed for radiation protection assessments. The following sections introduce key concepts of neutron physics and detection, which will be referenced throughout the thesis. This is followed by a description of the research background relevant to this thesis and the objectives of the work.

1.3.1. Nuclear Reactions at Hadron Therapy Energies

If energetic particles from an accelerator are allowed to fall upon bulk matter, there is the possibility of a nuclear reaction taking place. A typical and compact way to write a nuclear reaction is:

$$X(a,b)Y \tag{1.7}$$

where a is the accelerated projectile, X is the target and Y and b are the reaction products. Given the high energies involved in particle accelerators used for hadron therapy, one prevalent type of reaction is spallation.

Spallation reactions

Spallation reactions are usually defined as the processes by which an incident hadron of relatively high energy interacts with a nuclear target and expels from the latter several particles, mainly nucleons, but also, with a lesser yield, composites and pions Figure 1.10. According to literature [13], spallation reaction are described in terms of a two-step model. In the first step, the incident high-energy particle enters the target nucleus and interacts with some of the individual nucleons in the target in what is known as an intranuclear cascade. In this process, a direct reaction takes place, resulting in the ejection of a few nucleons from the target nucleus. If the incident energy is high enough (300 MeV), pions will be created and some of these too may be emitted along with the nucleons. The residual target nucleus, which now has fewer nucleons and is in an excited state, undergoes de-excitation in the second step. During the de-excitation the residual nucleus loses its excess energy by emitting single nucleons and small clusters of nucleons through the usual evaporation process evaporation takes place when the excited nucleus relaxes by emitting low-energy (<20 MeV) neutrons, protons, alpha particles, etc., with the majority of the particles being neutrons. Additionally, secondaries emitted from a spallation reaction can trigger a series of subsequent spallation events, resulting in a

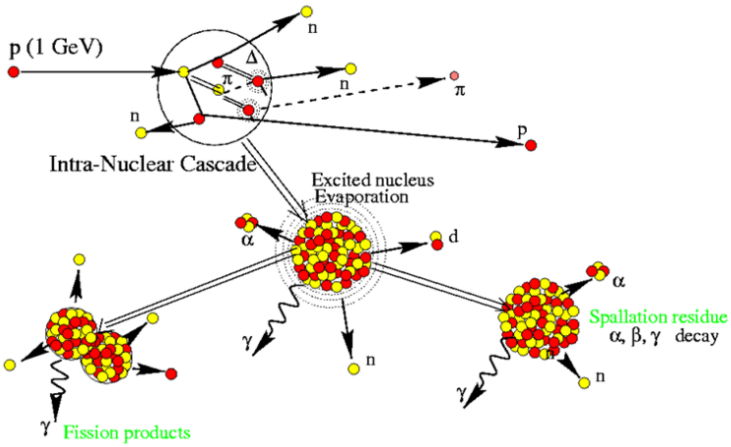

Figure 1.10: Mechanism of spallation reaction.

phenomenon known as hadronic cascade.

1.3.2. Neutron fields at CNAO

In radiation therapy, neutrons are usually secondary particles produced by the interaction of primary beam particles with materials in the beam path. Moreover, since these fields are diffused in all directions and not confined to the primary beam direction, they can cause additional interactions within the patient's body and result in undue dose exposure. Hence, one of the main challenges in radiation protection for high-energy particle accelerators is understanding the radiation fields generated. Investigating these fields requires addressing mixed radiation environments.

Beam losses produce a mixed high-energy secondary field, primarily consisting of neutrons up to hundreds of MeV and high-energy photons. In practical scenarios, assessing these radiation fields through simple calculations is difficult due to the complexity of the physical processes involved in each particle interaction. To achieve accurate results, the radiation protection group at CNAO uses the Monte Carlo code FLUKA to develop a highly precise model of the facility. This model includes the current synchrotron, the synchrotron with a rotating gantry, and the BNCT facility. The implemented geometry in FLUKA is depicted in Figure 1.11. The model takes into account the total masses of the accelerator magnets to understand their influence on the radiation field and their shielding effect. The primary sources of dose outside the shielding are secondary neutrons generated through the interaction of the primary beam with machine components such as magnets and the vacuum chamber, as well as losses resulting from accidental injection and beam extraction, and losses on targets, including patients and experimental targets. For the

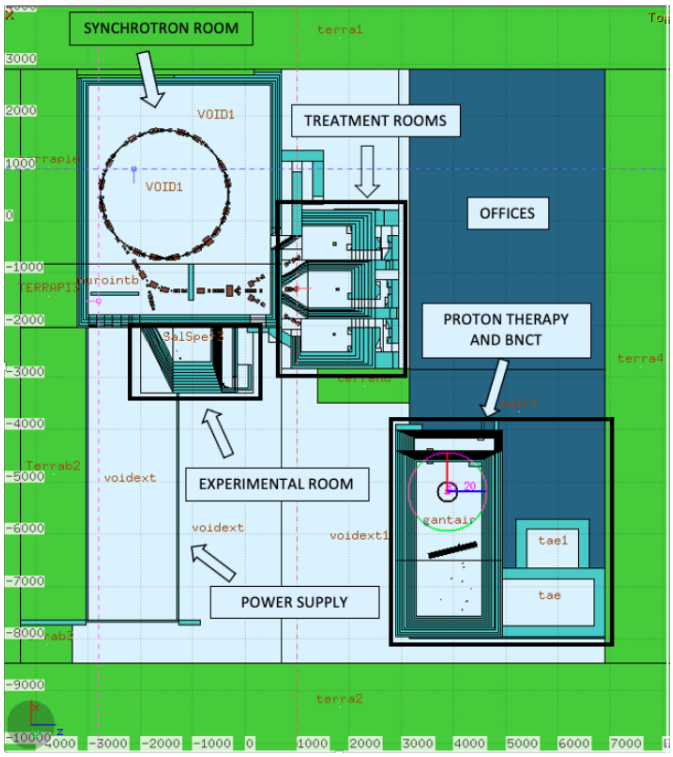

Figure 1.11: Geometry of CNAO implemented in FLUKA.

existing synchrotron, a beam loss model has been implemented [14]. The model is based on the assumption that approximately 50% of the beam losses are caused by treatments or experiments, while the remaining 50% are attributed to accidental losses. Additionally, 29 loss points were identified and integrated into the simulations. The locations of these loss points are depicted in Figure 1.12.

Ion Beam	**Energy** $[MeV/u]$
^{4}He	250
^{7}Li	306
^{12}C	400
^{16}O	400
^{56}Fe	306

Table 1.2: List of ions with corresponding energies used in the simulations.

The studies [15] conducted for the ions listed in Table 1.2, using the beam loss model above-mentioned, consider two different geometries: one including all machine elements and another with only the vacuum chamber. In both scenarios, the secondary field inside the synchrotron room is primarily composed of neutrons and photons. The spectra

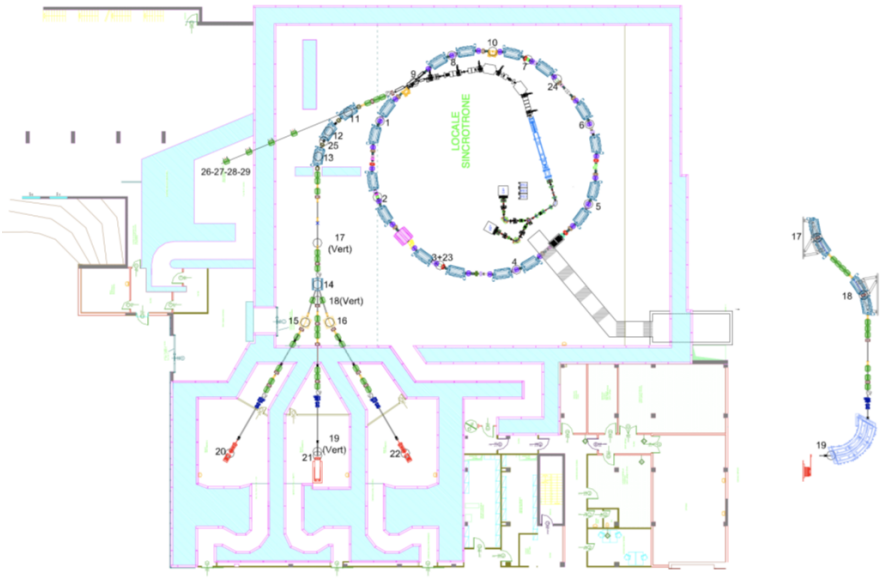

Figure 1.12: Beam loss map of the center for points from 1 to 29.

of these secondary particles are illustrated in Figure 1.13 and Figure 1.14, showing the neutron and photon spectra inside the shielding when the primary beam consists of carbon ions. The presence of magnets attenuates the high-energy peak of neutrons centered around 100 MeV, while their influence on the photon spectrum is minimal. Neutron spectrum are characterized by three peaks: spallation peak around 100 MeV, evaporation peak around 1 MeV and thermal peak around 0.025 eV. Photon spectra show peaks due to (n, γ) reactions on ^{1}H (2.2 MeV), ^{24}Mg (3.7 MeV), ^{23}Na (4.7 MeV), ^{28}Si (6.3 MeV), ^{27}Al (7.6 MeV), ^{40}Ca (6.3 MeV), $^{56}Fe(7.6MeV)$ in concrete. The secondary field is similar for all beams and without magnets it is composed by 65% from neutrons, 30% from photons, and 5% from heavy ions; with magnets the heavy ion component can be neglected and the field is composed only by neutrons and photons.
Another issue that CNAO has tackled is defining the workloads for the new ions listed in Table 1.2, which will be accelerated in the existing synchrotron without alterations to the shielding. Typically, a first assessment of the shielding can be performed by using Monte Carlo simulations, in which neutron yields and the ambient dose equivalent, H*(10), spatial distribution are obtained. A model has been developed [16] using a Monte Carlo approach to determine H*(10) values at various concrete depths generated by secondary radiation. Simulations are performed for primary beams in 1.2. For facilities already accelerating carbon ions, an equivalent carbon ratio is introduced to estimate the ion currents required to achieve the same H*(10) as a carbon ion at a given point. Numerical results are obtained by simulating an ion beam impinging on thick iron or ICRU (International Commission on Radiation Units and Measurements) tissue, representing magnets

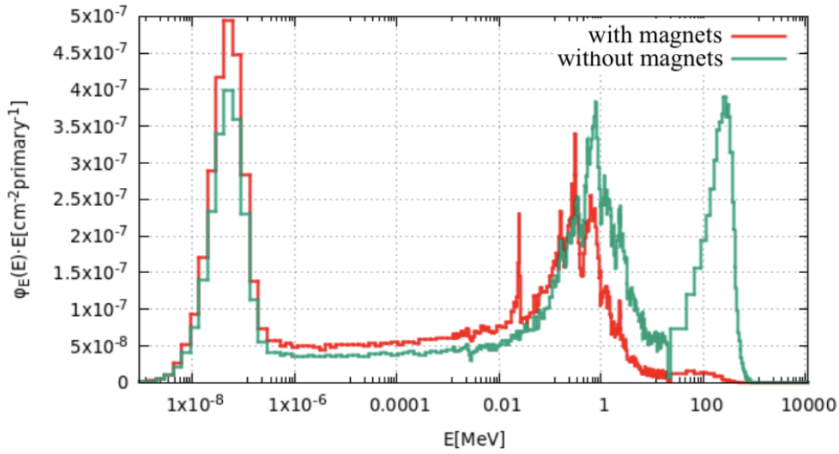

Figure 1.13: Neutron spectra generated inside the synchrotron room with and without magnets.

and patients, respectively. The secondary radiation is then transported through a substantial concrete shielding, where H*(10) values are calculated as a function of concrete thickness.

In the context of BNCT, studies were conducted to accurately assess the composition of the shielding [17]. Typically, shielding in hadron therapy centers is designed with a focus on high-energy neutrons (>1 MeV) and utilizes concrete walls. However, for BNCT, the secondary radiation field primarily consists of low-energy neutrons (<100 keV), making it impractical to use the same shielding material. From other studies [18], it has been observed that incorporating an initial layer of concrete enriched with 1% boron offers significant benefits in meeting both radiation protection regulations and the desired workload. This is because boron can substantially reduce the thermal neutron scattering component within the treatment room.

1.4. Mechanisms for neutron detection

As mentioned earlier, a significant challenge in high-energy applications is the generation of secondary neutrons. An important aspect of this challenge is the detection and monitoring of these neutrons. Generally, mechanisms for detecting neutrons in matter rely on indirect methods.Neutrons are electrically neutral particles and do not interact directly with electrons in matter. However, two basic types of neutron interactions with matter are available. First, neutrons can undergo scattering with a nucleus, where some of their kinetic energy is transferred to the nucleus. If sufficient energy is transferred, the recoiling nucleus may ionize the material surrounding the point of interaction. The second type

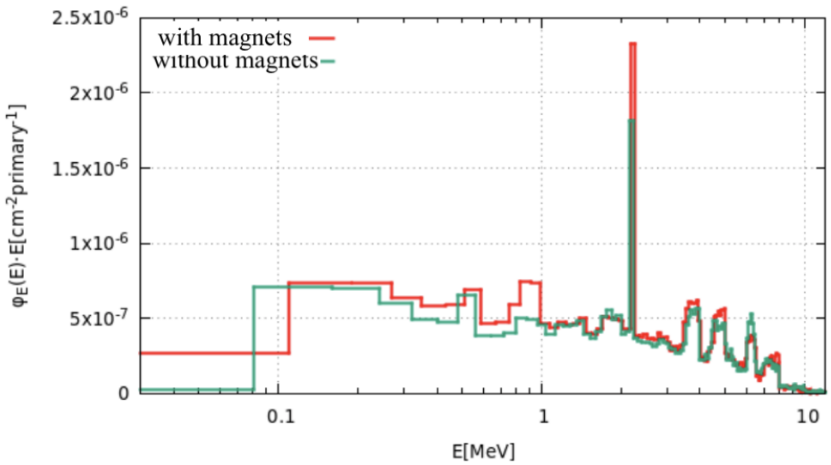

Figure 1.14: Photon spectra generated inside the synchrotron room with and without magnets.

of interaction involves nuclear reactions. Neutrons can induce nuclear reactions within certain materials, producing energetic charged particles, such as protons or alpha particles. These charged particles are promptly released and can be detected, making this mechanism a common method for neutron detection. The measurable current is caused by ions losing energy inside the active volume of the detector. So in general every type of neutron detector involves the combination of a target material designed to carry out this conversion together with one of the conventional radiation detectors as semiconductors detectors and proportional counters. Developing a neutron detector is challenging due to the wide range of neutron energies, spanning from meV to GeV. Moreover, the probabilities of various interactions occurring in the target material are strongly dependent on the neutron energy, which is why it is useful to distinguish between thermal neutrons (with energies around 0.025 eV), epithermal neutrons (with energies up to 0.1 MeV), and fast neutrons (with energies up to 20 MeV). Generally, the cross sections are higher for thermal neutrons, which is why neutrons are sometimes slowed down using moderators such as polyethylene or paraffin. In the following section, the key nuclear reactions used for thermal neutron detection are introduced.

1.4.1. Nuclear Reactions for Thermal Neutron Detectors

In searching for nuclear reactions that might be useful in neutron detection, several factors must be considered. First, the cross section for the reaction must be as large as possible so that efficient detectors can be built with small dimensions. For the same reason, the target nuclide should either be of high isotopic abundance in the natural element, or

alternatively, an economic source of artificially enriched samples should be available for detector fabrication. In many applications, intense fields of gamma rays are also found with neutrons and the choice of reaction bears on the ability to discriminate against these gamma rays in the detection process. Of principal importance here is the Q-value of the reaction that determines the energy liberated in the reaction following neutron capture. The higher the Q-value, the greater is the energy given to the reaction products. The distance traveled by the reaction products following their formation also has important consequences in detector design. If we are to capture the full kinetic energy of these products, the detector must be designed with an active volume that is large enough to fully stop the particles. However, the wall effect should also be considered: if the reaction occurs near the wall, not all of the kinetic energy can be captured. This can affect the efficiency and accuracy of energy detection.
The cross sections of the most popular reactions for the conversion of slow neutrons into directly detectable particles are are reported in Figure 1.15.

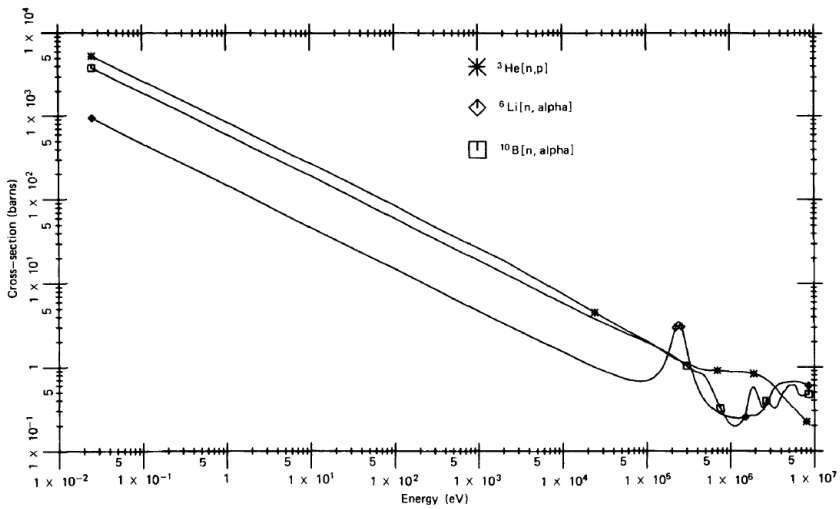

Figure 1.15: Cross section versus neutron energy for some reactions of interest in neutron detection. [19]

The ^{10}B(n,α) Reaction

One of the most popular reaction for the conversion of slow neutrons into directly detectable particles is the ^{10}B(n,α) reaction. The reaction can be written as:

$$n +^{10} \mathrm{Li} \rightarrow \begin{cases} ^{7}\mathrm{Li} + \alpha + 2.79\,\mathrm{MeV} & \text{(ground state)} \\ ^{7}\mathrm{Li}^{*} + \alpha + 2.31\,\mathrm{MeV} & \text{(exited state)} \end{cases} \tag{1.8}$$

When thermal neutrons (0.025 eV) are used to induce the reaction, about 94% of all reactions lead to the excited state and only 6% directly to the ground state. In either case, the Q-value of the reaction is very large (2.310 or 2.792 MeV) compared with the incoming energy of the slow neutron. Additionally, due to the negligible incoming linear momentum, the reaction products must exhibit a net momentum of essentially zero. As a result, the two reaction products must be emitted in exactly opposite directions, with the reaction energy always being equally distributed between them. The individual energies of the alpha particle and lithium nucleus are respectively 1.47 MeV and 0.84 MeV.

The ^{6}Li(n,α) Reaction

Another common reaction for the detection of slow neutrons is the (n, α) reaction in ^{6}Li. In this reaction, the process proceeds directly to the ground state of the product and can be written simply as:

$$n +^6 \text{Li} \rightarrow^3 \text{H} + \alpha + 4.78 \text{ MeV} \tag{1.9}$$

In this reaction too, the particles are emitted in opposite directions, with ^{3}H nucleus receiving 2.73 MeV and the α particle receiving 2.05 MeV. The thermal neutron cross section for this reaction is 940 barns. Figure 1.15 shows that the cross section remains below that for the ^{10}B reaction until the resonance region ($>$ 100 keV). Although the lower cross section is generally a disadvantage, it is partially offset by the higher Q-value, resulting in greater energy being imparted to the reaction products.

The ^{3}He(n,p) Reaction

The gas ^{3}He is also widely used as a neutron detection medium through the reaction:

$$n +^3 \text{He} \rightarrow^3 \text{H} + p + 0.764 \text{ MeV} \tag{1.10}$$

The products of the reaction are emitted with energies of 0.573 MeV fot the proton and 0.191 MeV for ^{3}H. The thermal neutron cross section for this reaction is 5330 barns, significantly higher than that for the boron reaction, and it also decreases with a $\frac{1}{\nu}$ energy dependence. Although ^{3}He is commercially available, its relatively high cost can be a limiting factor in some applications.

1.4.2. REM Counters

REM counters are devices used to assess the ambient dose equivalent $H^*(10)$. A rem counter consists of a detector with a high efficiency to thermal neutrons placed inside a moderator designed in such a way that the response function of the instrument reproduces sufficiently well the curve of the conversion coefficients from neutron fluence to $H^*(10)$ over a wide energy range. In other words, at the center the fluence of thermal neutrons is proportional to the ambient dose equivalent. Mathematically the ambient dose equivalent, can be expressed as:

$$H^*(10) = \int_{E_{\min}}^{E_{\max}} h^*(10)(E)\phi(E)\, dE \tag{1.11}$$

and it's unit is the Sievert [Sv]. $H^*(10)$ coefficients are reported in 1.16 in unit of $pSv{*}cm^2$ and are function of the incident neutron energy from E_{min} to E_{max},while $\Phi(E)$ m^{-2} represent the neutron fluence at energy E and $h^*(10)$ is the fluence to ambient dose equivalent conversion function. Due to the complexity of solving the integral 1.11, H*(10)

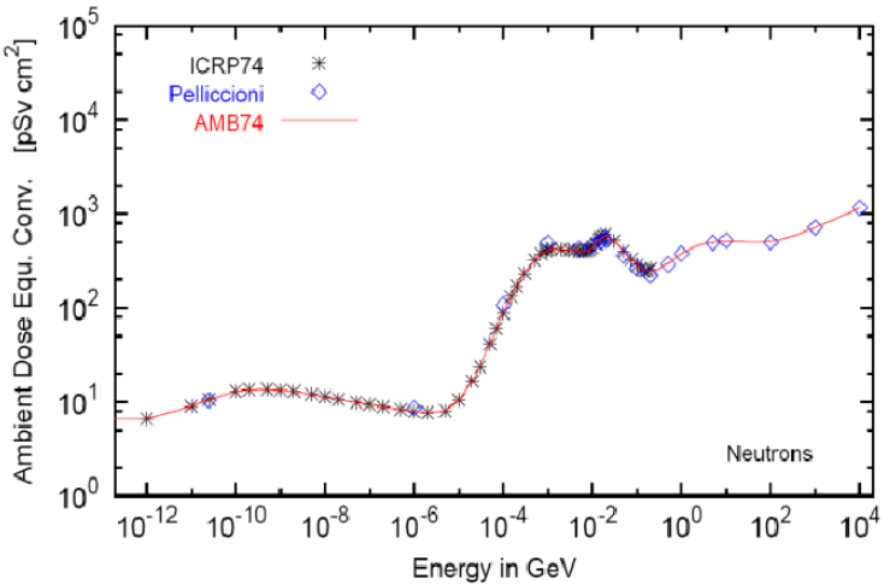

Figure 1.16: Conversion coefficient from fluence to ambient dose equivalent values for neutrons. [20]

can be calculated through the sum:

$$H^*(10) = \sum_E h^*(10)(E)\phi(E) \tag{1.12}$$

The counts measured by the rem counter can be expressed as:

$$Counts = \sum_E R(E)\phi(E) \tag{1.13}$$

with R(E) the response function of the rem counter as function of neutron energy. This function is calculated by the rem counter manufacturer mostly with Monte Carlo codes. From 1.12 and 1.13 it is straightforward to assess that:

$$H^*(10) \propto Counts \tag{1.14}$$

The proportionality constant can be calculated by a calibration coefficient and verified experimentally in a reference field with a Am-Be or ^{252}Cf a calibration source or at accredited facilities. In Figure 1.17 the response of the most widely used commercial

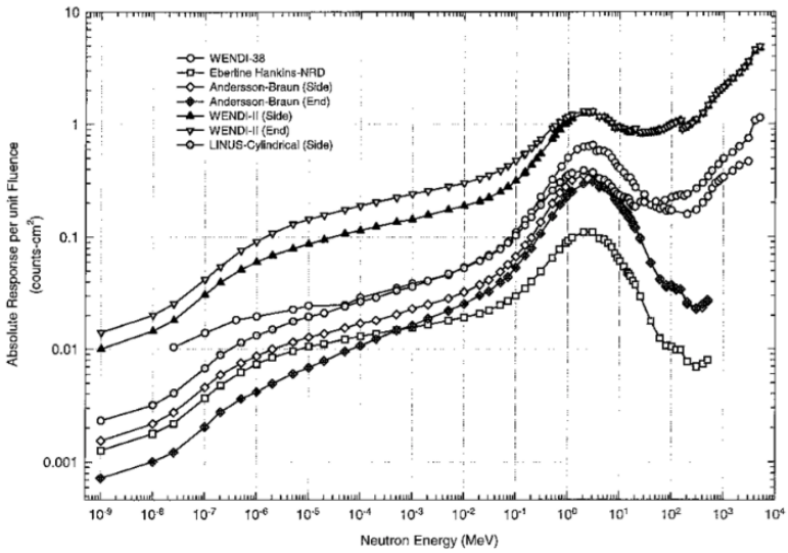

Figure 1.17: Absolute response functions (counts per unit fluence) for the following rem meters: WENDI-38, Hankins-NRD, Andersson Braun, Wendi-II , and cylindrical LINUS. [21]

rem counters as function of neutron energy, expressed in counts $\cdot$ cm^{-2}, is reported. A notable discrepancy exists between the Andersson-Braun counter and the The Wide Energy Neutron Detection Instrument (Wendi) and Long Interval Neutron Survey-meter (Linus) responses at high energies (>10 MeV). The Wendi and Linus counters, being extended-range rem detectors, provide improved detection capabilities for high-energy neutrons due to their moderator design. Typically, these moderators are constructed from a polyethylene sphere with added cadmium and lead inserts to enhance neutron detection at high energies. In the following the most popular rem counters are introduced. The

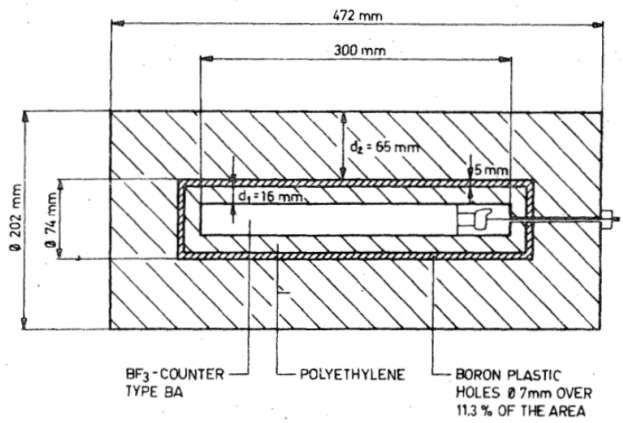

Figure 1.18: Design of Andersson-Braun rem counter.

Andersson-Braun REM counter [22] is presented in Figure 1.18. A BF_3 proportional counter is placed in the center of two polyethylene shells with a 5 mm boron plastic insert (200 mg/cm^2). As it can be seen from Figure 1.17 the Andersson-Braun rem counter is able to detect neutrons up to about 15 MeV [23]. At higher energies the response decreases and the instrument underestimates the ambient dose. The WENDI [21] makes use of both neutron generation and absorption to contour the detector response function,the design is showed in Figure 1.19. Tungsten or tungsten carbide (WC) powder is added to a polyethylene moderator with the expressed purpose of generating spallation neutrons in tungsten nuclei and thus enhance the high-energy response of the meter beyond 8 MeV. Tungsten's absorption resonance structure below several keV was also found to be useful in contouring the meter's response function. This design placed a BF_3 counter in the center of a spherical moderator assembly. A borated silicone rubber absorber covered an inner polyethylene sphere to control the meter's response at intermediate energies. A second generation design (WENDI-II) further extended the high-energy response beyond 20 MeV, increased sensitivity. A ^{3}He counter tube is located in the center of a cylindrical polyethylene moderator assembly. Tungsten powder surrounds the counter tube at an inner radius of 4 cm and performs the double duty of neutron generation above 8 MeV

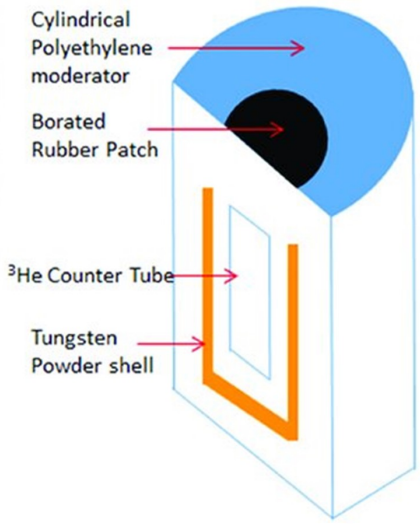

Figure 1.19: Design of the Wendi-II rem counter.

and absorption below several keV. An advancement of the last mentioned REM counters

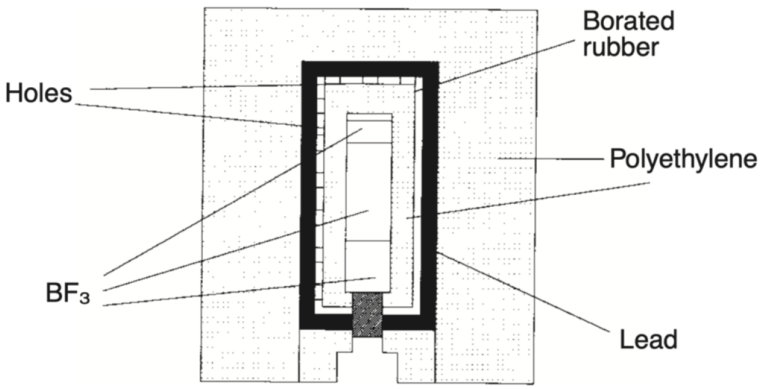

Figure 1.20: Longitudinal cross section of the cylindrical version of the rem counter LINUS.

is the Linus [24] which, historically, was the first instrument responding to high-energy neutrons and whose geometry is reported in Figure 1.20. In this detector, a 1 cm thick lead layer is positioned between the boron doped plastic attenuator and the outer polyethylene. The effect of this additional material is to detect neutrons with energy greater than about 10 MeV via the evaporation neutrons produced in inelastic scattering reactions, that can be subsequently moderated by the polyethylene and then detected by BF_3 proportional counter located at the center of the REM counter. No significant effect is produced on neutrons with energy below about 10 MeV so that in this energy interval the response of the monitor is the same as that of conventional moderator instruments. This upgraded

design allows the detection of neutrons up to 1 GeV.

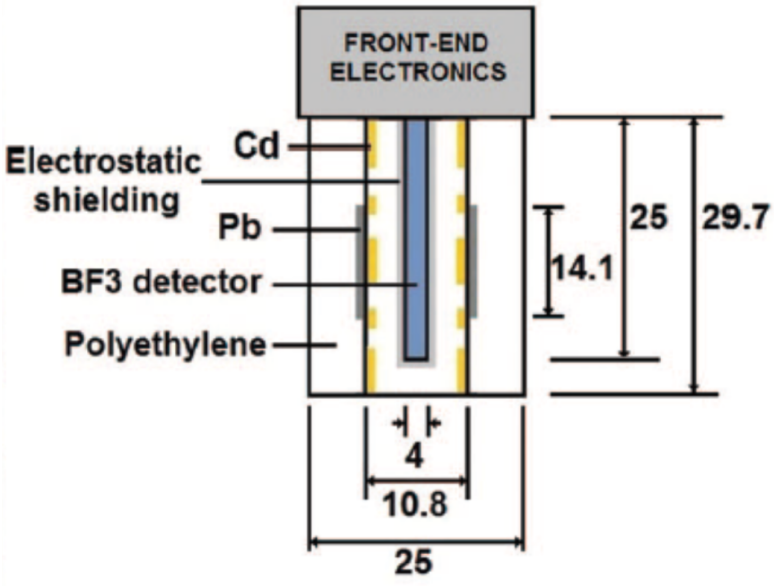

Figure 1.21: Design of the LUPIN REM counter.

The LUPIN design is reported in Figure 1.21. This detector employs a different type of moderator with similar characteristics to the previous ones. The main differences are the design of the lead and the addition of cadmium inserts over the borated plastic. The substantial improvements of this device is its high sensitivity and the ability to response and its linear response up to nSv due to its logarithmic electronics [25].

1.5. Extended Rem Counter

The CNAO expansion project includes the addition of a new proton synchrotron with a rotating gantry and a BNCT facility. Additionally, the existing synchrotron, which currently accelerates protons and carbon ions, will be upgraded to also accelerate helium, lithium, oxygen, and iron ions. In preparation for this, several studies have been conducted in recent years to evaluate the responses of different detectors in a simulated mixed field, similar to the one expected for future BNCT treatments, as well as for facility monitoring. In particular, the Radiation Protection Group at CNAO has proposed a new detector that is cost-effective, easy to repair, and capable of continuous monitoring. The proposed REM counter features a Microstructured Semiconductor Neutron Detector (MSND) housed within a plastic-based moderator that includes cadmium and lead inserts to extend its response range. The MSND interfaces with a Raspberry Pi Model 4 via General Purpose Input/Output (GPIO) pins for data visualization and collection.

The Radiation Protection Group has already developed C and Python code for data acquisition and visualization on the Raspberry Pi, which logs each interaction in a text file, recording both the data and the timestamp of each count. A series of devices will be placed throughout the entire CNAO facility to monitor radiation dose. To ensure simple and effective management of these devices, the Radiation Protection Group requested the development of software for their control and monitoring. As part of the present thesis work, new Python visualization software has been created. This software enables remote control of multiple counters connected through a switch, streamlining the monitoring process.

1.5.1. MSND Detector and Moderator

The DOMINO neutron detector is made of a Microstructured Semiconductor Neutron Detector (MSND) technology [26], see Figure 1.22. MSND were first developed at the Kansas State University SMART Laboratory to address the limitations of traditional planar coated diode detectors while maintaining key advantages such as compactness, low power consumption, and cost-effectiveness. MSNDs are fabricated by etching microstructures into a semiconductor substrate, forming a pn junction within these etched features, and subsequently backfilling them with neutron-reactive material [27]. This design allows for an increased amount of neutron-reactive material in close proximity to the semiconductor, enhancing the likelihood of detecting reaction products.

MSNDs are notable for their high neutron detection efficiency, outstanding neutron-to-

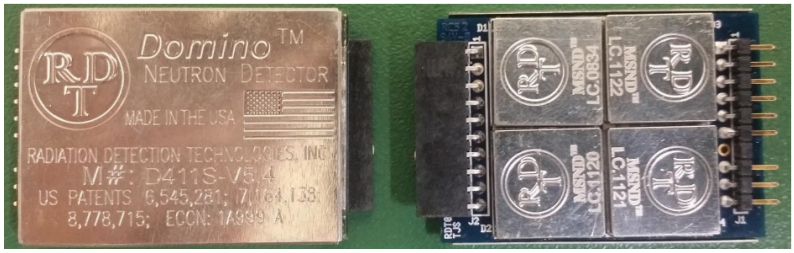

Figure 1.22: Internal structure of a DOMINO detector.

gamma-ray selectivity ($1 : 10^7$), low operational power requirements, and lower manufacturing costs compared to ^{3}He detectors. The deep cavities etched into the silicon (Si) substrate are backfilled with neutron-reactive material, typically ^{6}LiF. When neutrons are absorbed by the ^{6}LiF, they undergo a nuclear reaction that produces energetic charged particles, 1.5.1. These particles then intersect the Si fins, allowing their detection by the Si diode structure (see Figure 1.23).

Sensitive area: 4 cm^2	Pile-up limit< 33 kcps
Mass: 9.5 g	Dead time: 150 μs
Input voltage: 2.9 - 5.5 V	Sensibility to ambient light: No
Output voltage: 2.7 V	Temperature sensor: Yes
Power consumption: 290 μA	Temperature range: -40 °C to 55 °C
Pulse output amplitude: 5 - 50 μs	Detection efficiency: 30% @ 20 °C

Table 1.3: DOMINO V5.4 features. [28]

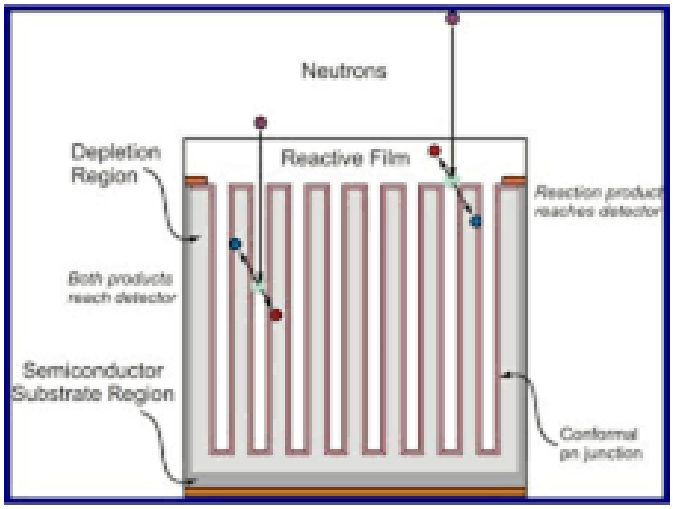

Figure 1.23: Deep microstructures backfilled with neutron-reactive material enhance both neutron absorption efficiency and reaction product detection efficiency in MSND®s.

$$n +^6 \text{Li} \rightarrow^3 \text{H} + \alpha + 4.78 \text{ MeV} \tag{1.15}$$

State-of-the-art neutron detectors like MSND offer a viable alternative to ^{3}He-based detectors, delivering high thermal-neutron detection efficiency with significantly reduced volume, mass, voltage, and power requirements compared to gas or liquid detectors. The active region of the MSND comprises a diode measuring 4 cm^2 x 0.5 μm. The sensor is equipped with onboard electronics, including a pre-amplifier, shaping amplifier, discriminator, analog-to-digital converter, and temperature sensor. Additionally, the system features a voltage regulator to produce an output signal, typically a square wave with a pulse width ranging from 5 to 50 μs. The main features of the MSND are provided by the manufacturer, as shown in Table 1.3.
The DOMINO thermal neutron detector is centrally located within a spherical moderator, held in place by a polyethylene plug. As detailed in [29] and depicted in Figure 1.24,

this moderator has a 25 cm diameter and consists of two polyethylene shells separated by a lead layer, with strategically placed cadmium buttons to enhance the extended range response.

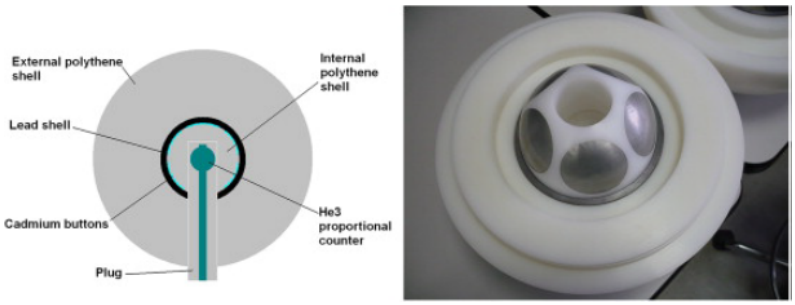

Figure 1.24: Design of the moderator.

1.5.2. Previous characterization and validation of the Rem counter

One of the initial studies on the REM counter [30] involved using a diode connected to a Raspberry Pi. A Python algorithm was developed to register a count whenever an event was detected by the diode. When the detector is active, it creates a text file that logs the counts along with their corresponding date and time. In the aforementioned study, the diode was compared with other detectors, such as the LUPIN described in 1.4.1. It was observed that the intensity of the examined field led to saturation issues in nearly all the detectors considered. This saturation problem arises because these detectors were originally designed for weaker fields, where high sensitivity is required. For this application, reducing sensitivity could be beneficial to avoid saturation. The analysis revealed that in intense fields, REM counters with diodes reached saturation, with this issue being more pronounced when paired with a Raspberry Pi compared to a driver board.
A further study was conducted on the REM counter,[18]. Specifically, the MSND is connected to the Raspberry Pi, which supplies power and facilitates data transmission via the General Purpose Input/Output (GPIO) pins. These GPIO pins operate in binary states (on and off) at a voltage of 3.3 V, whereas the MSND outputs a 5 V square wave signal with frequencies of up to 66 kHz. The primary objective of the first part of the study was to evaluate the Raspberry Pi system's ability to accurately measure square wave signals up to 66 kHz. In the experiments, an Arduino was employed as a square wave generator, and the frequencies of the Arduino signals were measured using a digital oscilloscope, the Picoscope 6. To assess the Raspberry Pi's response, a Python or C script was utilized to update a counter whenever the GPIO pin was in the 'on' state. However, the set frequencies do not precisely match the measured values. This discrepancy is at-

tributed to latency in the Arduino's delay command. This latency is more prominent at lower frequencies and remains constant above 5 kHz, but as the frequency increases, the deviation from the set frequency becomes noticeable.
Additionally, a new counting program was developed in C, given that C offers greater speed compared to Python. This program checks the state of the diode every second and writes the data to an external text file, including the count, date, and time information. Testing the program demonstrated a 99.9% efficiency in detecting square waves, even beyond the maximum frequency of 90 kHz. Moreover, visualization software was developed using the Python interface Tkinter, which will be described in detail in the following section.
In the same study, the calibration coefficient was determined experimentally at the Czech Metrology Institute (CMI) in Prague. Furthermore, to assess the detector's response in high-energy mixed fields, it was irradiated at the CERN facility. The results are also presented in [31].
At CMI radioactive sources are employed for calibration, including those designed for measuring environmental neutron dose equivalents. CMI upholds the primary standard (ECM 440-2/97-003) for the spectral response of neutron fluxes using Am-Be and ^{252}Cf sources. These measurements cover dose rates ranging from 30 $\mu Sv/h$ to 20 mSv/h. A calibration coefficient of 1.25 ± 0.06 $nSv/count$ was estimated using an Am-Be source. This calibration coefficient was then used to perform the linearity test of the instrument and to assess the ambient dose equivalent response as a function of neutron energy. The linearity test revealed a saturation dose rate of 2 mSv/h, as shown in Figure 1.25. The

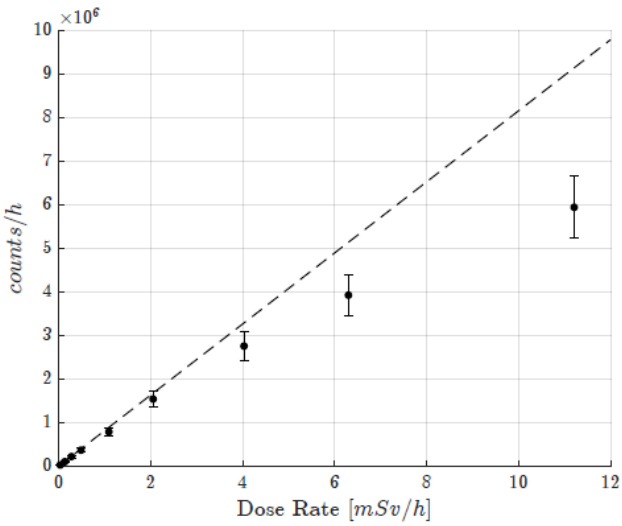

Figure 1.25: Plot of the linearity test of the rem counter.

calibration coefficient was then used to calculate the response function as a function of

neutron energy, ranging from 1 meV to 1 GeV, as shown in Figure 1.26.

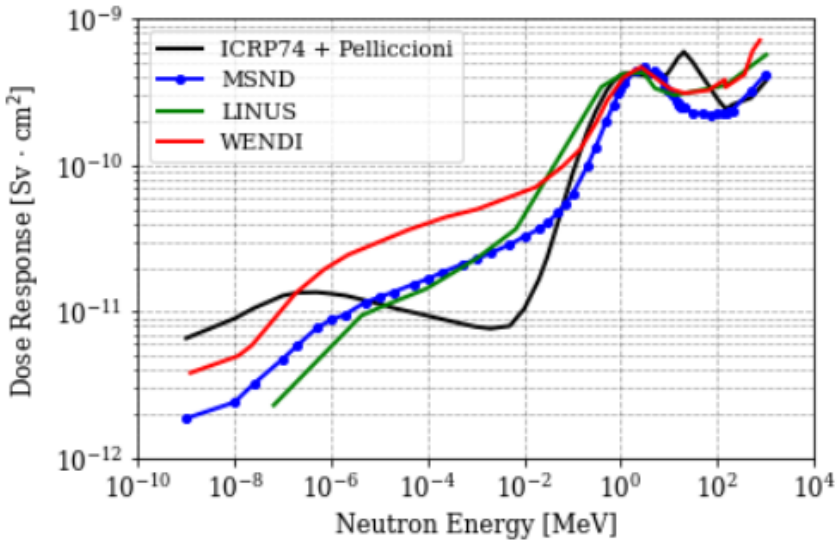

Figure 1.26: Response function of the detector, expressed in $Sv \cdot cm^2$, as a function of the incident neutron energy. The black line indicates the fluence to ambient dose equivalent coefficients calculated by ICRP and Pelliccioni [32], the blue line represents the MSND rem counter results, and the green and red lines respectively indicate the LINUS and WENDI dose responses.

Further measurements were conducted at the CERF facility to validate the calibration coefficient of the rem counter in complex mixed fields up to a few $GeVs$. These measurements were performed using 100 GBq Am-Be sources, yielding dose rates of 80 $\mu Sv/h$ and 200 $\mu Sv/h$. The data revealed a mean underestimation of the ambient dose. To investigate this underestimation, the Monte Carlo model of the rem counter was employed. Specifically, simulations irradiating the detector with Am-Be and CERF's neutron spectra in reference positions were performed. The ratio between the results obtained with Am-Be and CERF's spectra was compared. At the end, is was observed a great agreement between the experimental measurements performed at CERF's facility and the rem counter model, demonstrating excellent consistency despite the inevitable geometry approximations introduced by the Monte Carlo model of the detector.
At the end, measurements with the extended-range rem counter were conducted within the synchrotron room at CNAO. These experimental results were then compared with Monte Carlo simulations. Given the positive outcomes from the experimental campaigns at CERF and CNAO, particularly in mixed high-rate fields, there is ongoing consideration of utilizing the rem counter both for ambient dose evaluation and beam loss monitoring at CNAO. The positive results led the radioprotection team at CNAO to the idea of de-

veloping a network of detectors.
As part of this thesis, the network was implemented, and dedicated software was developed to control it. This system enables simple and direct monitoring of the dose throughout the entire facility. Furthermore, additional field verifications were conducted at CMI, and the collected data were analyzed to reassess the detector's linearity and calibration coefficients. Additionally, the robustness of the detector was evaluated in mixed radiation fields at CERF.

1.6. Aim of the thesis

The primary goal of this thesis is to develop the network and software for the remote control of multiple Extended Range REM Counters developed by the CNAO Radiation Protection Group, as well as to verify the proper functioning of the detector in mixed fields. Another key objective was to evaluate the linearity and calibration coefficients of the detector, and to assess its performance in both mixed radiation fields and real-world environments.
CNAO opted to create its own REM counter to deliver a device that is cost-effective, easy to repair, and capable of continuous monitoring. The instrument consists of a solid-state thermal neutron semiconductor detector integrated with a Raspberry Pi, housed within a plastic-based moderator featuring cadmium and lead insertions to extend its response range. Notably, the REM counter operates without requiring high voltage and can be fully managed by the Raspberry Pi, eliminating the need for an electronic chain that includes amplification, a single-channel analyzer, and a counter. These detectors will be strategically positioned throughout the facility to monitor neutron doses and will be employed in the new BNCT-dedicated structure. This thesis has developed the network and software that will enable the simultaneous control of the detectors installed in the structure. This system will facilitate easy and effective management of the entire network of detectors to be installed in the CNAO structure, providing real-time feedback on the functionality of both the network and the detectors. The software has been validated through measurements conducted at the CNAO, utilizing a americium-beryllium source. The proper functioning of the entire network developed in this thesis was observed. The next chapter will offer a detailed description of the neutron dosimeter and the software implementation.
In addition to the development of software that will allow for the control of measurements throughout the CNAO, this thesis includes the analysis of data for the characterization of individual detectors. Specifically, data from the Czech Metrology Institute (CMI) were analyzed to verify the detector's linearity and calibration coefficient, while measurements at the CERN-EU High-Energy Reference Field (CERF) facility assessed detector perfor-

mance in high-energy fields. The final chapter outlines the measurements conducted at CNAO to achieve key objectives. These measurements were designed to verify the proper functioning of the detectors, as well as the network and control software developed in this thesis, within the typical experimental fields of the facility. Additionally, the system was tested for continuous operation over a 4-day period, successfully confirming its reliability for long-term monitoring of neutron dose levels throughout the facility and demonstrating its capability for sustained use.

2 | Software Development for Extended Rem Counter Control

This chapter provides a detailed description of the network and software developed in this thesis, specifically designed to control the detector described in 1.4.2. The development of both the network and software was driven by the necessity to improve radiation monitoring capabilities within the facility, ensuring precise control and safety during operations. The detector plays a crucial role in monitoring radiation levels in real-time, offering enhanced oversight and protection for both patients and staff. The software and network infrastructure were developed to offer a robust and scalable solution for managing the detector's data acquisition and control processes. By providing centralized control and advanced diagnostic capabilities, this system addresses the critical need for reliable radioprotection in an eviroment like the CNAO.

In the following, Section 2.1 presents the software and the counting program previously developed by the radiation protection group at CNAO. The subsequent section describes the work carried out in this thesis, specifically the creation of the network and the control software.

Finally, the measurements conducted to verify the proper functioning of the entire system are presented. A preliminary test was performed to evaluate the operation of the system, which includes both the network and the software. This test involved measuring the dose from an Am-Be source, using the calibration coefficients derived from the data analysis at CMI, which will be discussed in the following chapter. The measured dose was then compared with the dose calculated using FLUKA, providing confirmation of both the robustness of the network system and the accuracy of the detector's measurements.

2.1. Previous work

As described in Section 1.5.1, several studies have already been conducted to develop a control system for initiating measurements on the detector developed at CNAO, outlined in 1.4.2. In particular, a Graphical User Interface (GUI) was created, allowing users to start and stop measurements on each individual device. The counting program, which records the measurements, is also briefly described.

2.1.1. Previous visualization software

For the implementation of the REM counter software, the radioprotection group developed a Python visualization tool to streamline the input of measurement parameters [18], Figure 2.1. In the previous GUI, displayed on a 3.5-inch touchscreen attached to the

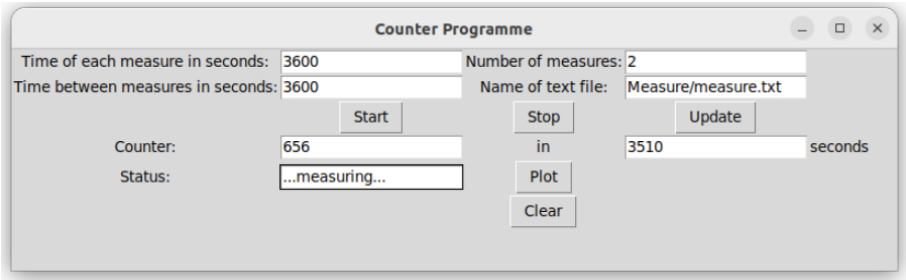

Figure 2.1: View of the old counting interface written with the python interface Tkinter.

Raspberry Pi, users could configure various measurement parameters. These included setting the measurement time in seconds, the number of measurements to perform, the interval between measurements, and specifying the name and location of the data file for the current measurements. The user can initiate the measurement using the "Start" button, update the counter and time with the "Update" button, plot the data using the "Plot" button, and reset the measurement with the "Clear" button.
This system allows for the control of individual devices but does not provide comprehensive management of the entire network. Therefore, a new software solution has been requested and developed to address this need.

2.1.2. Previous counting program

The radioprotection group also developed a C program for neutron counting. Each time a neutron interacts with the detector, the program increments a counter, enabling the acquisition of integral counts over time. The program records the counts, along with the precise date and time of each event, to a text file at one-second intervals. This continuous

logging allows for detailed monitoring of neutron interactions, providing a reliable dataset for further analysis of radiation levels and detector performance.

2.2. CNAO Network

As previously mentioned, the CNAO facility will soon be expanded with a new structure dedicated to BNCT and new ions will be accelerated. In light of this new facility and the need to monitor synchrotron losses, multiple devices will be placed at various strategic locations. The points where the detectors and switches will be placed are shown in Figure 2.2. These locations were selected based on the hypothesized loss points discussed in the study presented in 1.3.1, while also taking into account technical constraints and criticalities. Additional devices will also be installed on the roof, at ground level, on the first floor, and around the structure. To manage all the REM counters placed throughout the

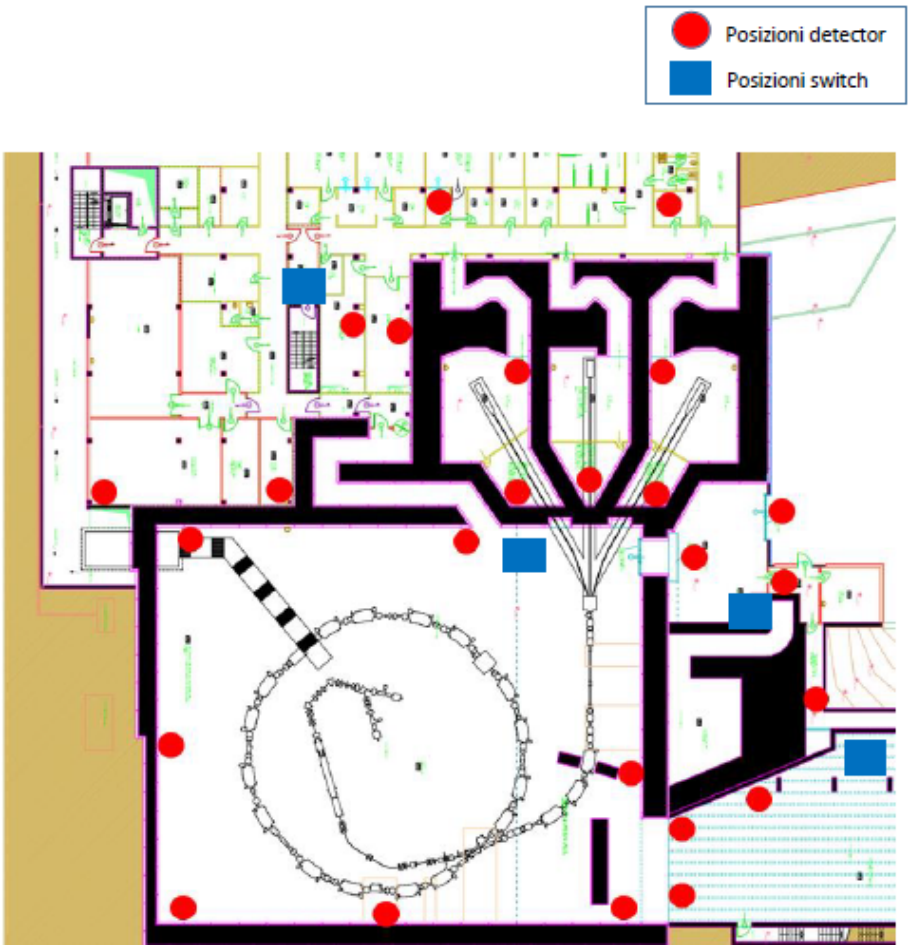

Figure 2.2: Scheduled locations for the REM counters and switches on the basement level.

facility in a simple and effective manner, a system was needed to facilitate their control.

Therefore, in addition to the placement of the REM counters, Figure 2.2 also illustrates the positioning of several switches for the creation of a network. A network switch is a device that enables communication between two or more IT devices, such as Raspberry Pis. By connecting multiple IT devices, a communication network is established, creating a Local Area Network (LAN) [33], which enables data transmission to and from multiple devices. Devices connected to a LAN communicate with each other using Ethernet over TCP/IP. Transmission Control Protocol (TCP) [34] is a standard protocol on the that ensures the reliable transmission of data between devices on a network. It defines how to establish and maintain a network conversation by which applications can exchange datas, ensuring that information is transmitted accurately and in the correct sequence.
This type of system allows for the control of multiple devices (slaves) through a single computer (master). To manage the network, specialized software is required. Therefore, the radioprotection group has also requested the development of a software to facilitate comprehensive control over the entire system. This software will integrate with the network to ensure effective monitoring and management of the REM counters and associated devices. More details will be provided in the next paragraph.

2.2.1. New counting program

In this work, the previous counting code described in Section 2.1.1 was modified, and specifically, the file format was changed from a text file to a binary file. One of the primary advantages of using a binary file is its reduced memory footprint. For example, in a text file, numerical data such as timestamps and counts are stored as human-readable characters, specifically in ASCII format, which can take up significantly more space. Each character in a text file requires at least one byte of storage, and often multiple bytes are required to represent a single number. In contrast, binary files store data in a more compact form, representing numbers directly in their binary format. This can lead to a considerable reduction in file size, especially when dealing with large datasets. As a general rule, binary files are more efficient for storage, as they avoid the overhead of converting data to and from text representations, thus minimizing memory usage and improving performance. Additionally, the new code includes a step that saves the created files into a specific directory (`File_da_inviare_pc`) on the Raspberry Pi. This organization facilitates operations that can be performed with the software, which will be described in detail shortly. The code is presented in Appendix A.2.1.
To read the data, a C program was also developed in this work to convert the binary file back into a text file, as described in Appendix A.2.2. An example of the generated text file is provided in Figure 2.11.

The C program can be executed directly from the Raspberry Pi, either using the old graphical interface or controlled through the new software that will be described in the following sections. Through this software, it is possible to initiate the counting program directly from a PC by specifying variables such as the measurement time and the desired name for the output file.

2.2.2. Advanced software version

The new visualization software, displayed in Figure 2.3, has been developed using Python, specifically utilizing the PyQt5 library, A.1.1. PyQt5 was selected due to its superior capabilities when compared to Tkinter. It provides a more advanced and feature-rich environment for GUI development, offering a wide range of widgets, tools, and customization options. This makes it particularly well-suited for developing complex and sophisticated applications, where flexibility and a modern user interface are essential. The final and complete code can be find in Appendix A.
The software facilitates the management of multiple Raspberry Pi devices from a single computer, provided they are connected to the same network via a switch. Each Raspberry Pi is assigned a unique IP address, which identifies each device individually. Consequently, each computer will be connected to a series of IP addresses, the number of which depends on the ports available on the switch. The software offers the following capabilities:

- Network functionality verification;
- Initiating and halting measurements;
- Requesting and retrieving files from the Raspberry Pi devices;
- Creating and displaying the database where the collected data is stored.

These features are linked to specific buttons, the detailed description will be provided in the following sections. Additionally, the status of the operation can be monitored through the status indicators shown on the GUI. Users can also click the information icons next to each button for further details on its functionality, an example is shown in Figure 2.4. Furthermore, the software also enables monitoring of the system's performance. Each time an operation is performed, the request and its result are logged in a text file called `log_file`, which also records the date and time of the request. Further details are provided in the Appendix A.1.3. The following paragraphs will provide a detailed description of the buttons and associated functions available on the GUI.

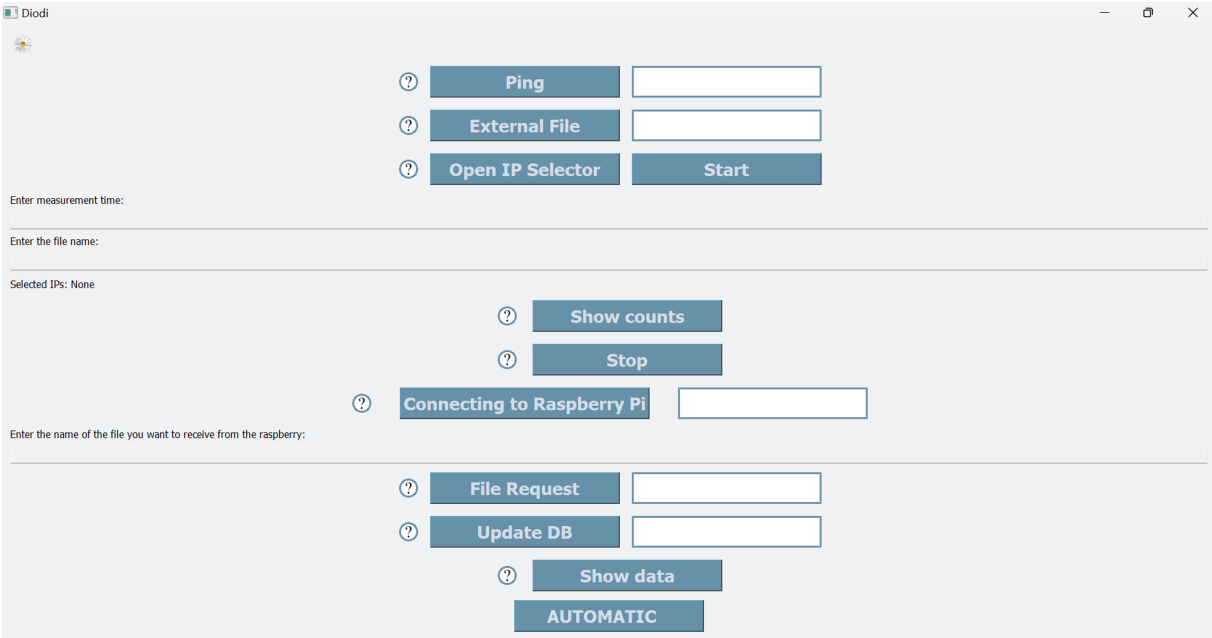

Figure 2.3: View of the new interface written with python.

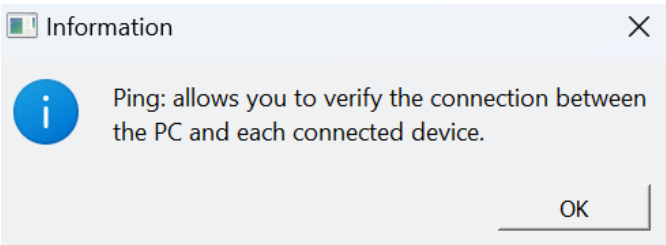

Figure 2.4: Example of the information button.

2.2.3. Network functionality verification

This section describes the buttons and their associated functions used to verify the correct connection of the network and the proper presence of a file named `File_Esterno`, which is essential for the correct functioning of the software. A "ping" function has been developed to ensure continuous monitoring of the connection status between the Raspberry Pi devices and the PC. This functionality is essential, as the detectors will be used to monitor dose levels; thus, the ability to verify connection and system integrity in real time is crucial. The implementation of the ping function allows for the prompt identification of any disruptions or communication issues, facilitating timely intervention and maintaining the reliability of the monitoring system. For monitoring purposes, a function has been created to verify the presence and non-emptiness of a file named "File_Esterno." This file can be modified by the user whenever a new detector needs to be added or replaced, characterized by a different IP address and calibration coefficient. It is essential to ensure

that the data within this file are not deleted, as will be explained later, since this file is critical for the proper functioning of the entire system.

Ping

The ping function is an integral part of network diagnostics, particularly in settings where multiple devices must be monitored and their connectivity verified. This function is designed to ping a series of IP addresses, record the results, and provide real-time feedback through the GUI.

The function takes two key inputs:

- A list of IP addresses (host) that the function will attempt to ping.
- The number of ping requests to be sent to each IP address (`ping_count`), specifies how many ping requests will be sent to each IP address, typically set higher than one for increased reliability.

As the function begins executing, it initializes a flag variable to zero. This variable will later be used to determine whether any of the ping operations fail. Additionally, a global variable, test, is prepared to store the status of each single device. The function logs the initiation of the ping process by writing a message to the log file.

Ping execution:

- The function enters a loop, iterating through each IP address in the provided list. For each IP address, it constructs and executes a ping command using the Popen method from Python's subprocess module. This command sends the specified number of ping requests to the target IP address and captures the output.
- The captured output is read line by line, and the function looks for the presence of a specific keyword, "TTL" (Time To Live) [35], which indicates a successful ping response. If the "TTL" keyword is found in the output, the function considers the ping attempt successful. If the keyword is not found, the function logs a "Failed Ping" message, indicating that the ping attempt was unsuccessful. Additionally, the flag variable is set to one to indicate that at least one ping operation failed.
- As the function processes each IP address, it opens a new window on the GUI that displays the status of each ping operation in real-time, Figure 2.5. Users can immediately see which devices are successfully responding to the ping requests and which are not. This visual feedback is critical for identifying the specific device that may be experiencing connectivity issues.
- After all the IP addresses have been pinged, the function evaluates the flag variable to determine the overall result. If any ping operation fails, the test variable is set

to "A Ping failed." Conversely, if all pings are successful, the test variable is set to "All the Pings are successful." This status is then displayed on the GUI in a status rectangle adjacent to the ping button, providing immediate visual feedback. This real-time status update is particularly beneficial when monitoring a large number of devices, as it allows users to swiftly identify any connectivity issues across the network without needing to manually check each individual result.

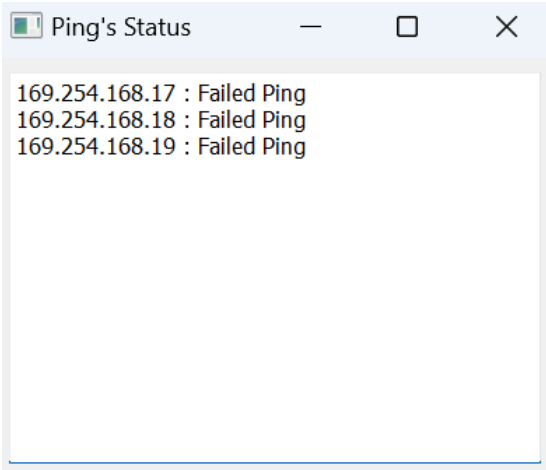

Figure 2.5: Example of the ping window when no devices are properly connected.

The ping function is crucial in ensuring the reliability of network connections. By verifying the connectivity of multiple devices, it ensures that all components within a network are reachable and functioning properly. This is especially important in environments with distributed devices. The code implementing this functionality is provided in Appendix A.1.4 and A.1.5.

File Esterno

The purpose of this function is to ensure that a file named `File Esterno` is present on the PC used to control all the devices, A.1.6. This file is crucial as it contains essential information required for constructing the database, which will be detailed further. Specifically, it includes the device's IP address, its position within the CNAO, and the calibration coefficient for the associated diode, Figure 2.6. In particular, the list of IP addresses (host) is sourced from this file. This is important because if a device fails and needs to be replaced, it is sufficient to update the External File with the new IP address, location, and calibration coefficient.

The function performs the following checks:

- File Presence: It ensures that the specified file exists within the given directory.
- File Content: It verifies that the file is not empty.

The result of the check is displayed on the GUI in the status rectangle near the "File Esterno" button. Moreover, the outcome is also recorded in the log file. Specifically, the function will inform the user if:

- The file is not present in the directory.
- The file is present but empty.
- The file is present and not empty.

This functionality is crucial, as the absence or emptiness of the file could lead to issues in all functions, since the list of detectors is contained within the file. So, ensuring that this file is correctly located and populated is essential for the proper functioning of the subsequent operations.

```
169.254.168.17,Treatment_Room,1.20
169.254.168.18,Synchrotron_Room,1.19
169.254.168.19,Roof,1.25
```

Figure 2.6: Example of the "File Esterno": from left: IP address, the Raspberry Pi's position and the calibration coefficient.

2.2.4. Initiating and halting measurements

In this section, the essential components of the GUI for measurement will be described. The radiation protection group requested a system that would enable users to easily start and stop measurements across all devices, as well as provide real-time visualization of the data collected. This is particularly important given the numerous devices that will be positioned throughout the structure, making it essential to have a system that facilitates their management. In response to the request from the radiation protection group for a system that enables users to easily start and stop measurements across all devices while providing real-time visualization of the data collected, the interface has been designed accordingly. As previously mentioned, several Raspberry Pi devices will be strategically positioned within the structure, and these can be simultaneously controlled via the software if they are connected to the same switch. This functionality has been implemented, and further details will be provided in the following sections. The GUI offers several critical functionalities:

- Device Selection: Users can easily choose which Raspberry Pi to use for the measurement.

- Measurement Duration: The interface allows users to specify the duration of the measurement.
- File Naming: Users can assign a name to the file that will be created on the Raspberry Pi during the measurement.
- Measurement Initialization: Once the device and settings are selected, the user can easily start the measurement process with a simple command.
- Real-Time Data Visualization: To enhance monitoring, the interface includes real-time visualization of the data being collected.
- Measurement Termination: The GUI also provides an option to stop the measurement at any time.

These features are designed to make the measurement process as simple and efficient as possible, allowing for effective control and real-time monitoring of multiple Raspberry Pi devices within the network. In a facilty of the size of CNAO, these functions are essential, as the detectors will be positioned at various locations. It is important to be able to start and stop measurements from a single PC and to visualize the data in real time, ensuring comprehensive oversight of the monitoring system.

IP Selector

As previously mentioned, the software allows control over a certain number of devices depending on the inputs of the switch. To provide greater control and flexibility, the ability to select the specific device on which to perform measurements was requested. To address this need, the IPSelector class was developed, playing a crucial role within the application. The IPSelector class was specifically designed to simplify the process of selecting IP addresses. It provides a dialog window that enables users to easily choose the IP addresses of the devices they wish to monitor. When this window is opened, it offers a clear and straightforward interface, allowing users to select one or multiple IP addresses from a list. When this dialog window, created by the IPSelector class, is opened, users are presented with a straightforward interface. A clear instruction at the top asks them to choose the IP addresses they wish to use. Below this, a list of predefined IP addresses is displayed, from which users can select one or more by simply clicking on them, Figure 2.7. This multi-selection capability is essential in situations where multiple devices, such as Raspberry Pi units, need to be managed at the same time.

The interface also includes a "Get Selected IPs" button. Once users have made their selections, clicking this button retrieves the chosen IP addresses and prepares them for

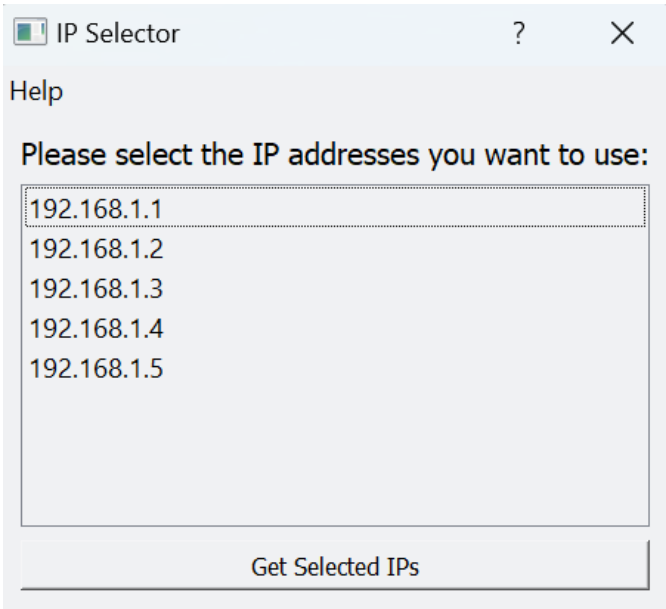

Figure 2.7: Ip Selector list.

further actions within the application. The selected IP addresses are then passed to the Start function, which will be described in the following section, to initiate the measurement process. Additionally, after the selection, the chosen IP addresses can be displayed on the GUI. Additionally, the class includes a help menu, accessible from the top of the dialog window, Figure 2.8. This feature ensures that if users need assistance, they can easily access helpful information about how to use the interface, enhancing the overall user experience.

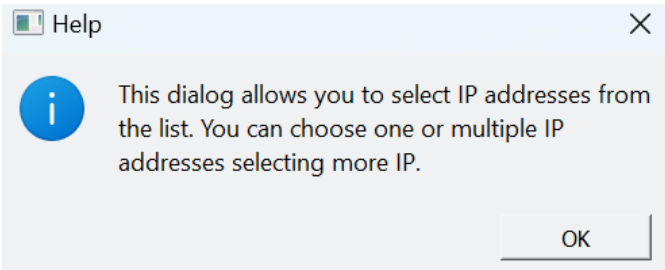

Figure 2.8: Help message displayed to the user in the IP Selector window.

By organizing this functionality within its own class, the IPSelector not only provides a clear, efficient, and supportive user interface but also contributes to the overall maintainability and modularity of the application's codebase. This modular approach is particularly beneficial in software development, as it allows each component to be developed, tested, and maintained independently, leading to a more robust and flexible application. More technical details are provided in the Appendix A.1.7.

Filename and Measure Time

Once the IP addresses have been selected, the user can also specify the measurement duration in the section labeled "Enter Measutement time" and the file name in the section labeled "Enter the file name" for storing the counts on the Raspberry Pi, Appendix A.1.8. This is achieved by directly entering the desired values into the GUI. These parameters are then passed to the Start function, which, as will be detailed shortly, initializes the measurement process.

Given the various applications of the detector, it is essential and beneficial to allow users to input the measurement time. This flexibility enables the customization of measurement intervals according to specific experimental requirements. Additionally, being able to specify the file name aids in organizing the data effectively, facilitating easier retrieval and analysis in the future.

Start

The Start button is connected to the `send_to_c_program` function, Appendix A.1.9, which is designed to facilitate the remote execution of the counting program (as detailed in Section 2.1.1) on multiple machines by leveraging the Secure Shell (SSH) protocol. SSH is a network protocol that enables secure communication between two networked devices, allowing for the remote execution of commands and file transfer over an encrypted channel. The function takes three parameters: `"max_count"`, `"filename"`, and `"selected_ips"`. These parameters are used as inputs for the C program and specify the machines on which the program will be executed. Specifically, `"max_count"` corresponds to the measurement duration, and "filename" refers to the file name, both provided by the user through the GUI. The `"selected_ips"` parameter contains the IP addresses that the user has previously selected using the appropriate button.

The function begins by setting up the SSH client using the "paramiko" library, which is a Python implementation of the SSH protocol. The SSH client is initialized with default parameters, with the SSH port set to 22, the standard port for SSH communications. The function also specifies the credentials required for the SSH connection: the username (pi) and the password (Fondazione09). For each IP address, the SSH client is configured to automatically add the host key to the list of known hosts if it is not already present. The function then attempts to connect to the remote machine using the provided credentials. Once the SSH connection is established, the function constructs a command string that navigates to the Desktop directory on the remote machine and executes the C program (‘./cc‘) with `"max_count"` and `"filename"` as command-line arguments. This command

is executed on the remote machine using the `exec_command` method provided by the paramiko library. The function captures both the standard output (stdout) and standard error (stderr) streams resulting from the command execution. The standard output and standard uncertainty from the remote execution are decoded and processed. The function appends a summary of the operation, including the IP address and relevant output, to the log file.
Finally, regardless of the outcome of the command execution, the function ensures that the SSH connection is properly closed by invoking the "close method" on the SSH client. This step is critical for releasing network resources and maintaining the stability of the application.

Show Counts
Clicking the "Show Counts" button initializes a function named `automatica_show_counts`. This function is designed to automatically request measurement data files from Raspberry Pi devices at regular intervals, for example, every 10 seconds. The files, which are created on the Raspberry Pi when the measurement begins, contain the results of the measurement process. The function retrieves these files and displays the associated counts for each IP address in real time on a designated window, as illustrated in Figure 2.9. To

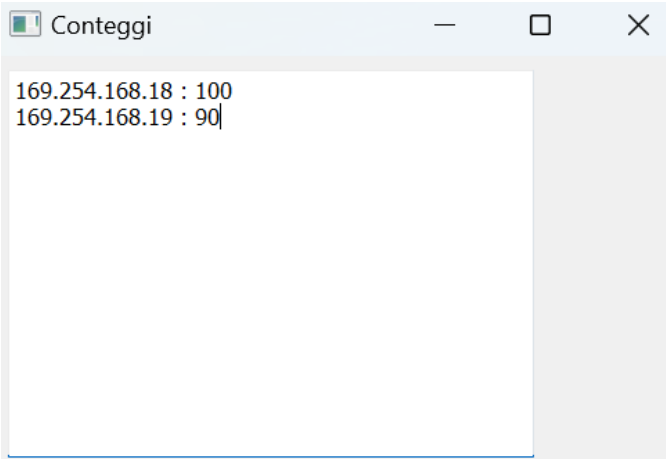

Figure 2.9: Example of the show counts function.

start, the function sets up an SSH client using the paramiko library. For each IP address in the `selected_ips` list, the function attempts to connect to the device and execute a remote command. This command runs a Python script (`Server_Raspberry.py`) on the Raspberry Pi, which is responsible for managing the server-side operations necessary for data collection.
After executing the remote command, the function establishes a socket connection to the Raspberry Pi on a specified port (12346), as it is done in the `file_request_raspberry`

function. This connection is used to transfer the data file from the Raspberry Pi to the local machine. Throughout the process, the function incorporates error-handling mechanisms to manage any issues that arise during SSH connections, command execution, or data retrieval. If an error occurs, a descriptive message is logged and appended to the `log_file` string to provide feedback on the issue.
After saving the data files, the function defines a helper method, `read_last_line`, to read the last line of each file. This method is used to extract the most recent measurement data, which is crucial for providing up-to-date information. The function iterates over all the files in the `SAVE_PATH` directory. It parses the IP address from the filename and extracts the count from the last line of data. The extracted measurement counts are then displayed in the application's user interface. The function appends these counts to a text widget, offering a real-time view of the measurement results from each Raspberry Pi device.
In summary, the `automatica_show_counts` function provides an automated and efficient way to manage and display measurement data from multiple Raspberry Pi devices. By leveraging SSH and socket communication, it ensures that commands are executed remotely, data is transferred securely, and results are presented in real time.

Stop

The Stop button is also linked to a function called `stop_c_program`, Appendix A.1.10. This function operates in a manner similar to `send_to_c_program`, utilizing the SSH protocol to communicate with the devices on which the C program responsible for measurement has been initiated. However, in the case of `stop_c_program`, the function specifically sends a command to terminate the running process on the remote devices.
Just as with `send_to_c_program`, the `stop_c_program` function begins by establishing a secure SSH connection with each of the target devices using the 'paramiko' library. It authenticates using predefined credentials and navigates to the appropriate directory on the remote machine. Instead of initiating a new process, however, the function executes a command that stops the currently running C program by sending a termination signal to the process.
If the Stop button is not clicked, the C program will automatically terminate after the measurement duration specified by the user through the GUI. The Stop button is provided as an option for the user to manually interrupt the measurement process before the set time elapses. This method allows the user to safely conclude the program at any point, ensuring that the system can be halted securely and without risk to the ongoing process.

2.2.5. Requesting and retrieving files from the Raspberry Pi devices

As previously mentioned, the system developed in this thesis was requested to facilitate easy dose monitoring throughout the CNAO structure. The requirement was not only for a system to initialize measurements but also for one that allows for data storage and the creation of a database to enable straightforward visualization and archiving of the collected data. To address this need, functions have been developed to transfer files from the Raspberry Pi devices to the PC, consolidating them into a single folder. Initially, it is essential to establish a secure connection between the two devices to ensure the safe exchange of files. Once this connection is established, users can request the files generated on the Raspberry Pi, which are then transferred to the PC where the software is being utilized. This process facilitates the creation of the database, as will be described later, and enables the analysis of the data without the need to manually copy files from the Raspberry Pi. For ease of visualization and analysis, the database is subsequently converted into an Excel file, allowing for simplified data analysis.

Connecting to Raspberry Pi

The `connessione_a_raspberry`, Appendix A.1.11, is a function designed to establish SSH connections between a the master computer and multiple Raspberry Pi devices. The function leverages the paramiko library to automate and streamline these interactions. Also in this case, the function operates using port 22 and, to authenticate the connection, it employs the default Raspberry Pi username "pi" and the specified password "Fondazione09". These credentials are vital for accessing the Raspberry Pi's command-line interface securely via SSH.

Then the function systematically processes a list of IP addresses provided in the host parameter. For each IP address, it attempts to establish an SSH connection by invoking the connect method of the paramiko.SSHClient instance. Upon successfully establishing the SSH connection, the system remotely executes a specified command on the Raspberry Pi. Specifically, it utilizes the `exec_command` method of the SSH client to run a Python script, `Server_Raspberry.py`, which is located in the Desktop directory of the Raspberry Pi.

At the end, the output of the command, including both standard output and any errors, are logged in the `log_file` on the master device for subsequent analysis. After the execution of the remote command, the function ensures the SSH connection is properly terminated, thereby maintaining security and optimizing resource usage.

Server Raspberry

The script implements a fundamental TCP server designed to manage file transfers over a network. This server enables communication with client devices, allowing them to request and receive specific files. The complete code for the server is detailed in the Appendix A.1.15.

Configured to listen for incoming connections on a designated port, the server is prepared to handle file transfer requests. Upon establishing a connection with a client, the server processes the request as follows:

- Request Handling: If the server receives a valid request, it proceeds to retrieve the requested file. The file is sent to the client in manageable chunks of 1024 bytes, ensuring efficient and complete data transfer.
- Error Management: The server is designed to handle various errors that might arise during the file transfer process, such as file access issues or network problems. It provides appropriate feedback to the client in such scenarios, helping to manage and mitigate potential issues effectively.

Specifically, the files are retrieved from a designated directory, File_da_inviare_pc, which is the folder where the counting program saves the files created for measurements. Overall, the server facilitates robust and reliable file transfers, addressing potential errors and ensuring that files are delivered accurately to client devices.

File Request

Firstly, when measurements are taken, a specific name is assigned to the file, which is then stored in a designated folder on the Raspberry Pi's Desktop. Consequently, all created files will be present within this folder. To facilitate the selection of files for retrieval from the Raspberry Pi, a feature has been implemented allowing users to choose the file they wish to request. Specifically, this can be done through the GUI by entering the name of the desired file in the section labeled: "Enter the name of the file you wish to receive from the Raspberry Pi".

The `file_request_raspberry` function is responsible for requesting a file from one or more Raspberry Pi devices and saving it locally. This process involves establishing a network connection, sending the file request, and managing the file transfer.

The function iterates through a list of host addresses provided in the HOSTS parameter. For each host, it establishes a TCP connection using the socket library on port 12346, which corresponds to the port used by the `Server_Raspberry` code running on the Raspberry Pi.

Once a connection is successfully established, the function sends the name of the requested

file to the server. This file name is obtained from a text input field in the GUI. The function then waits for the server's response. If the server responds with anything other than "OK," an error message is logged, and the function moves on to the next host. If the server responds with "OK," the function begins receiving the file data.
To ensure unique file names, the function generates a local file path that includes the current timestamp and the host address. An example is given in Figure 2.10. This approach

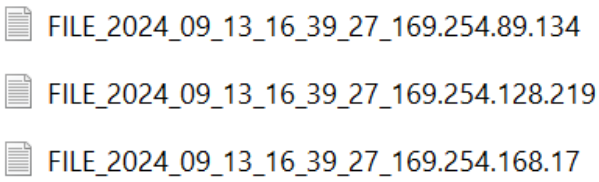

Figure 2.10: Example of the naming convention used for the files.

not only prevents files from being overwritten but also allows for easy identification of the file corresponding to a measurement conducted on a specific day by a particular Raspberry Pi. This is particularly useful for pinpointing the exact location within the CNAO facility where the measurement was taken. The file is then opened in binary write mode, and the function receives the file data in chunks of 1024 bytes. These chunks are written to the file until the transfer is complete.
Upon successfully receiving and saving the file, a confirmation message is printed and logged. If an error occurs during the connection or file transfer process, the error message is displayed in the designated area next to the button in the GUI. Otherwise, the successful receipt of the file is also indicated in this same area, providing immediate feedback to the user.
The code is reported in Appendix A.1.12

2.2.6. Creating and displaying the database

Once the files have been retrieved, they are saved in a designated directory. At this stage, the database can be created using the Update Database button, and subsequently, it can be reviewed using the Show Data button. The creation of a database is a fundamental step for not only storing the data but also for providing a mechanism to efficiently conduct searches within files that may contain a substantial number of rows, depending on the measurement interval. Each file originating from the Raspberry Pi contains data points recorded every ten seconds, including the timestamp, date, and integral counts. This process is illustrated in Figure 2.11.

```
13-09-2024,12:06:10,7
13-09-2024,12:06:20,18
13-09-2024,12:06:30,28
13-09-2024,12:06:40,40
```

Figure 2.11: Example of a file generated by the counting program and received by the master PC.

Update DB

As previously mentioned, the Update DB button enables the creation of a database that stores essential details, including the IP address of each Raspberry Pi, its position, the date and time of the measurement, the integral count number, and the calibration coefficient. When the Update DB button is clicked, the function detailed in Appendix A.1.13 is executed. This function ensures that all relevant data from the Raspberry Pi's files are systematically organized and stored in the database, facilitating easy access and further analysis.

The `db_update()` function is crucial for maintaining an accurate and up-to-date database by systematically incorporating the latest data collected from various Raspberry Pi devices. This function operates within the framework of SQL (Structured Query Language) [36], which is the standard language used for managing and manipulating relational databases. SQL provides a powerful set of tools for creating, reading, updating, and deleting data within a database, ensuring that complex data operations are handled efficiently and reliably.

The function begins by establishing a connection to the SQLite database, a lightweight, file-based database management system commonly used for applications requiring an embedded database engine without the need for a separate server process. SQLite is particularly suited for scenarios where simplicity, ease of use, and low resource consumption are priorities. To avoid conflicts with previous data, the function first executes an SQL DROP TABLE statement to remove any existing table named MyDataBase, thus providing a clean slate for new data. This is followed by an SQL CREATE TABLE statement, which defines the structure of a new table designed to store the required data. The table includes columns for storing the IP address of each Raspberry Pi, its position within the structure, the date and time at which the data was recorded, the measurement counts, a calibration coefficient, and a derived value termed "dose," which is calculated as the product of the measurement counts and the calibration coefficient. Next, the function accesses a specified directory where files containing data from the Raspberry Pi devices

are stored. It iterates through each file in this directory, utilizing Python's file handling capabilities to open and read both the data files from the Raspberry Pi and a reference file named `file_esterno.txt`, which contains additional details such as the IP address, position, and calibration coefficient for each Raspberry Pi, as outlined in the preceding section. As the function processes each data file, it extracts relevant information, including the date, time, and measurement counts. Using the IP address derived from the file name, it cross-references the corresponding position and calibration details from the reference file. This step is crucial for correctly linking the calibration coefficient stored in the reference file to the file from the Raspberry Pi, as the directory `File_da_Raspberry` will contain files from all Raspberry Pis connected to the same switch. This information is then consolidated, and the "dose" is calculated. Once the data is processed, the function executes an SQL INSERT INTO statement for each record, inserting the processed data into the newly created table within the database. The INSERT INTO statement is a fundamental SQL command that allows the insertion of new records into a specified table, ensuring that all relevant information is systematically stored. This process is repeated for each file in the directory, ensuring that all the most recent data is accurately recorded. After completing the data insertion for all files, the function commits the changes to the database using an SQL COMMIT command, which finalizes the transaction and ensures that all modifications are saved. A confirmation message is then logged to indicate the successful insertion of data. Finally, the function emits a signal summarizing the outcome of the operation and closes the database connection using the conn.close() method. Closing the connection is crucial for freeing up resources and maintaining the stability of the application. An example is shown in Figure 2.12. In this case, the database is viewed using a specialized software tool for viewing SQL databases called SQL Browser for SQLite.

In summary, the `db_update()` function leverages the capabilities of SQL within the SQLite database management system to automate the updating of the database with fresh data from multiple Raspberry Pi devices. By systematically organizing and storing all relevant information, the function ensures that the data is readily available for subsequent analysis, making it a vital component of the overall data management process.

Show Data

The `show_data()` function is a component for data management and visualization within the application, Appendix A.1.14. Its primary role is to facilitate the extraction of data from a SQLite database, convert this data into an Excel format, and subsequently open the file for user review.

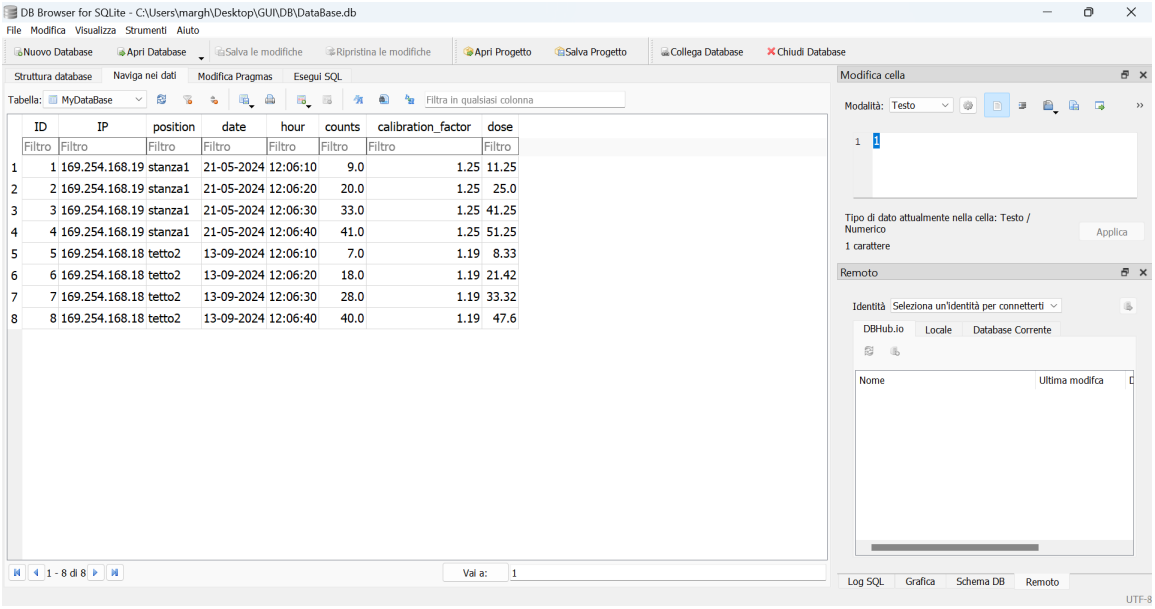

Figure 2.12: Example of a Database created with SQL.

The function operates as follows: The function initiates by establishing a connection to the SQLite database using the path specified by DB_path. The connection to the database allows the function to execute SQL queries and interact with the stored data. Once the connection is established, the function constructs and executes an SQL query to retrieve all records from the table named MyDataBase. All columns and rows of data are extracted from the table. The retrieved data is then loaded into a Pandas DataFrame. Pandas is a powerful data manipulation and analysis library for Python, which provides data structures and functions needed to work with structured data. Following the data retrieval, the function closes the connection to the SQLite database using conn.close(). This step is crucial to release the database resources and to ensure that no further operations can be performed on the database until a new connection is established. The function proceeds to export the DataFrame to an Excel file. The file is saved at a location specified by `excel_output_path`. An example of the resulting Excel file is shown in Figure 2.13. This figure illustrates how the data from the SQLite database is presented in a structured Excel format, demonstrating the successful conversion and organization of the database records. Finally, the function opens the newly created Excel file. This launches the file with the default application associated with Excel files on the user's system, thereby allowing immediate access and review of the data. In conclusion, the `show_data()` function performs a vital role in data management by extracting data from the SQLite database, converting it into a user-friendly Excel format, and providing immediate access to the data. This functionality is essential for enabling users to efficiently review and analyze the data stored within the database.

A	B	C	D	E	F	G	H
ID	IP	position	date	hour	counts	calibration_factor	dose
1	169.254.168.19	stanza1	21-05-2024	12:06:10	9	1,25	11,25
2	169.254.168.19	stanza1	21-05-2024	12:06:20	20	1,25	25
3	169.254.168.19	stanza1	21-05-2024	12:06:30	33	1,25	41,25
4	169.254.168.19	stanza1	21-05-2024	12:06:40	41	1,25	51,25
5	169.254.168.18	tetto2	13-09-2024	12:06:10	7	1,19	8,33
6	169.254.168.18	tetto2	13-09-2024	12:06:20	18	1,19	21,42
7	169.254.168.18	tetto2	13-09-2024	12:06:30	28	1,19	33,32
8	169.254.168.18	tetto2	13-09-2024	12:06:40	40	1,19	47,6

Figure 2.13: Example of a Database converted in Excel.

2.2.7. Automatic Network control

In the GUI shown in Figure 2.3, in addition to the functionalities previously described, there is a button labeled `AUTOMATIC`. This button automatically executes the functions described in 2.2.2 and 2.2.4 at fixed time intervals. This feature is particularly useful when a constant verification of the network's functionality is required, along with the automated saving of measurement files and database creation, without the need for manual operations. For example, this could be very helpful during long measurements. The results of the operations are then written to a text file named `log_file_automatic`, including the time and date when each check was performed.

2.2.8. Operational pathways of the software

To ensure the proper functioning of the software and the reliable storage of data, several folders have been created on the master PC. These folders are organized within a directory named GUI. This structure is essential for the software to correctly access and save data during operation. Each folder serves a specific purpose, allowing the code to reference the correct file paths when executing tasks such as storing measurement results, logs, and configuration files.

- External_File: This folder contains the external file that includes the IP address, location, and calibration coefficient.
- Files_from_Raspberry: This folder stores files received from the Raspberry Pis.
- DB: This folder contains the .db file for the database as well as the Excel file.
- Log_File: This directory houses the log file that records all operations and results from the software.

All of these elements are illustrated in Figure 2.14, while the details of the pathways can

be found in the Appendix A.1.2. Additionally, the folder contains all graphics used for the graphical user interface.

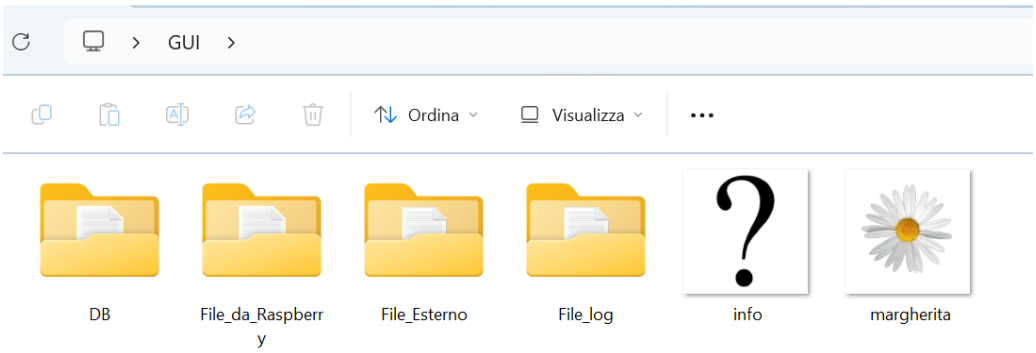

Figure 2.14: Example of the GUI folder structure.

Furthermore, each individual Raspberry Pi is equipped with a dedicated folder to store all generated counting files. These files are stored locally and can be retrieved from their respective folders and sent to the master PC upon request.
This organizational structure has been designed to maintain order and clarity within all file paths, making it easy to manage data flow between the detectors and the PC. By structuring the file directories in this way, modifications can be made efficiently, allowing the software to be easily adaptable and user-friendly. This enhances both the accessibility of the data and the overall flexibility of the system for future adjustments or expansions.

2.3. Measurements at CNAO with an Am-Be source for software verification

To verify the functionality of the software and the accuracy of the detector counts, two measurements were conducted at the CNAO laboratory. The tests were performed using an Am-Be source, three different rem counters, and the WENDI detector. At the end of the experiments,the dose measured by the MSND was compared with the dose from the source. Specifically, both the source and the laboratory were simulated using FLUKA. This comparison validated the proper operation of the network architecture and control software, while also providing additional confirmation of the functionality of the detectors.

2.3.1. Setup and network architecture

The experimental setup involved positioning the Am-Be source at a distance of 100 cm from the detectors, as shown in Figure 2.15. The detectors were carefully arranged to ensure optimal measurement conditions. The complete experimental setup, including the

detectors and the network architecture used for data acquisition and monitoring, is depicted in Figure 2.16.

On the left side of the setup, the master PC, which manages the software responsible for

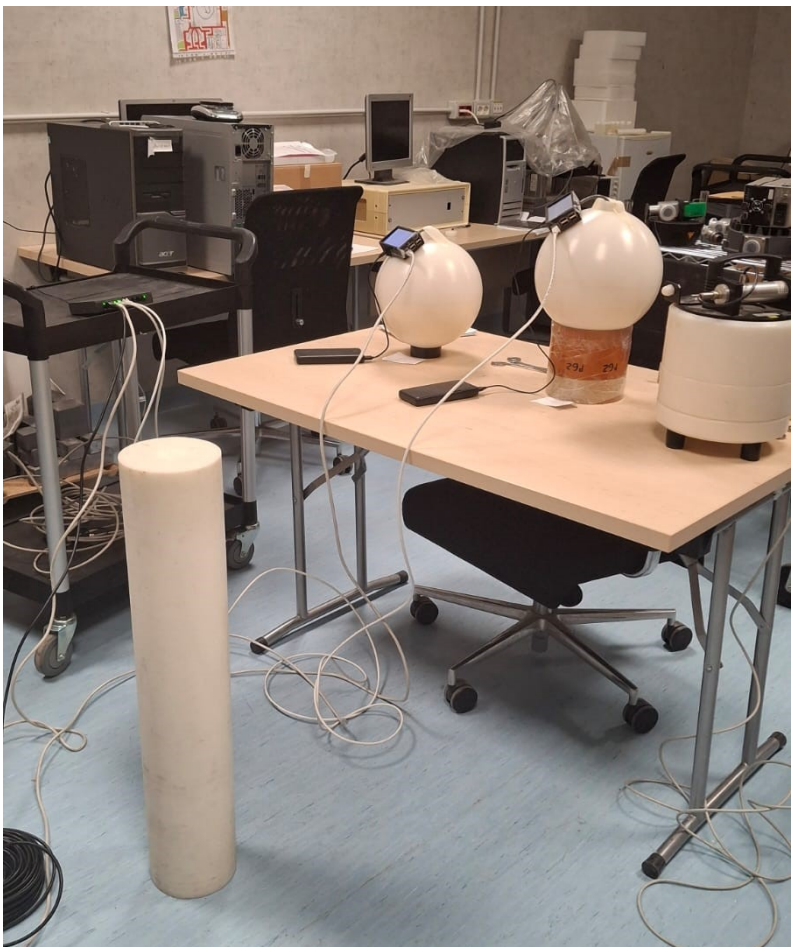

Figure 2.15: Layout of source and detector positions for measurement setup.

controlling the entire network, is visible. This PC is connected via an ethernet cable to a network switch, as illustrated in Figure 2.17. The switch is linked to four REM counters through additional ethernet cables. Out of these four counters, three were correctly connected and used for the measurements, while the fourth was intentionally left unconnected. This allowed for a verification of the software's ability to distinguish between properly connected detectors and those not connected.

The entire system was controlled remotely during the measurements. Several key functionalities of the software were validated:

- Verification during measurements to ensure that all connections remained stable and functioning correctly;
- Starting the measurement on the selected detectors;

Figure 2.16: Network Architecture for detector monitoring and control system.

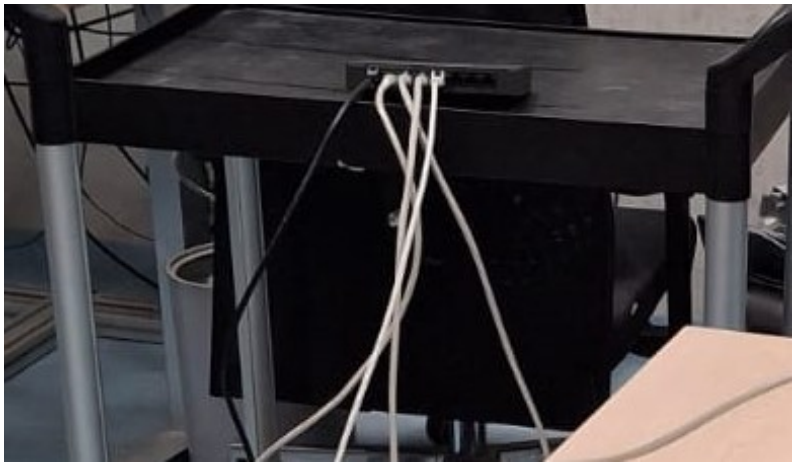

Figure 2.17: Switch connected to the master PC and detectors via ethernet cables.

- Real-time feedback of integral counts;
- Request and reception of files from all Raspberry Pis;
- Correct creation and visualization of the database;
- Creation of a log.txt file with operation results;
- The correct measurement of the dose.

As confirmed by the results, all components are functioning as expected, making the software fully capable of performing real measurements.

2.3.2. Measurement Results

Two one-hour measurements were conducted using three REM counters. The results, including the integral counts and the measured dose, are listed in the Table 2.1. The dose was calculated by multiplying the counts by the calibration coefficient. The calibration coefficient will be discussed in detail in the subsequent chapters in 3.1.
The measurements confirmed the proper functioning of both the software and the network architecture developed in this thesis project. Specifically, all required functionalities were thoroughly tested and verified, ensuring that the entire system is ready for future installation and use throughout the CNAO facility. This system will enable efficient and precise monitoring of all detectors deployed across the facility, facilitating seamless control and data acquisition. The successful validation of the system highlights its reliability and suitability for integration into the operational workflow at CNAO, ensuring enhanced oversight and management of radiation detection across the structure. The results are presented in Table 2.1, where the doses measured by the three detectors during the two experimental runs are detailed.

Meas. Number	IP Address	MSND Int. Counts [counts/h]	Calibration Coefficient [nSv/count]	MSND Dose Rate [μSv/h]
1	169.254.168.17	2336	1.34	3.13
1	169.254.89.134	2667	1.07	2.89
1	169.254.128.219	2872	1.18	3.38
2	169.254.168.17	2355	1.34	3.15
2	169.254.89.134	2659	1.07	2.84
2	169.254.128.219	2822	1.18	3.32

Table 2.1: Results of the measurements conducted in the laboratory.

2.3.3. Comparison with FLUKA simulations

To verify the accuracy of the acquired data, a comparison was made with the dose measured one meter away from the Am-Be source using the FLUKA software, described in Appendix C. Specifically, a representative Am-Be source was considered for the measurements taken, and the laboratory setup was simulated. Concrete walls were modeled, with thicknesses of 30 cm on the right, left and front sides and 1 cm on the back. Taking into account the location where the source was positioned in the laboratory, the dose around the source was simulated. The geometry can be seen in Figure 2.18. From the image, we

can observe the attenuation of the neutron dose in the wall, and it is also evident that scattering occurs. To evaluate the dose at 100 cm from the source using FLUKA, the cylindrical `USERBIN` car was employed, with a height equal to that of the source. The angular range ϕ was divided into 1-degree intervals, and for the radial distance, a division was made for each cm. An angular beam between -45° and 45° was selected to exclude scattering effects. Therefore, the dose was evaluated under these conditions, considering the average dose in the angular range at the height of the source.

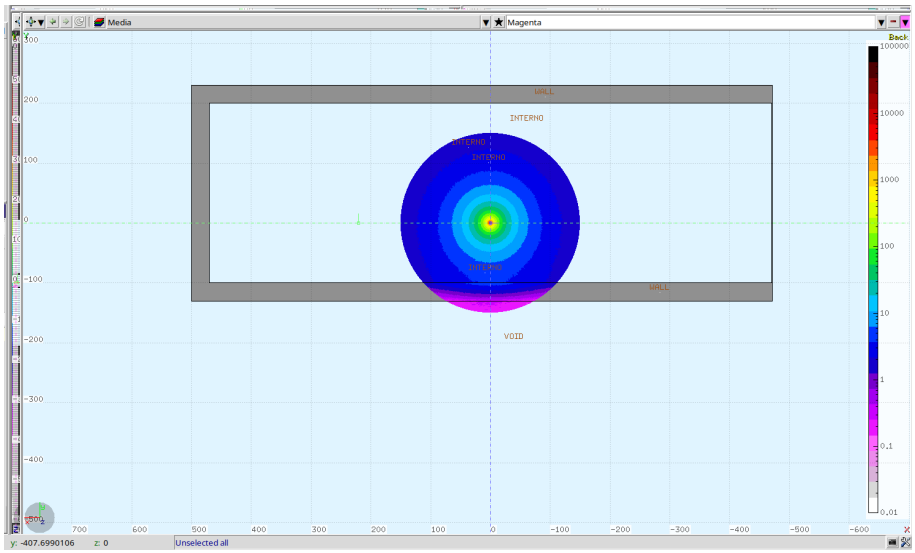

Figure 2.18: FLUKA Simulation of the laboratory with the Am-Be source.

Finally, from the image 2.19, the trend of the ambient dose equivalent as a function of radius can be observed. In particular, at 1 meter from the source, a dose of 2.99 $\mu Sv/h$ was found. The dose was calculated by normalizing the primary value considering the following factors:

- The primary dose refers to the initial dose calculation from FLUKA without any normalization.
- A neutron flux of 2.05 $\cdot 10^5$ neutrons per second was used, representing the emission rate of neutrons from the source.
- A conversion factor of 3600 was applied to convert the time from seconds to hours, as the flux is given per second, but the dose is desired in $\mu Sv/h$.
- A factor of 10^{-6} was used to convert from pSv to (μSv), as the dose was initially provided in pSv.

In conclusion, the recorded doses align closely with the expected value from the source, remaining within a 30% margin of uncertainty. This margin accounts for all potential

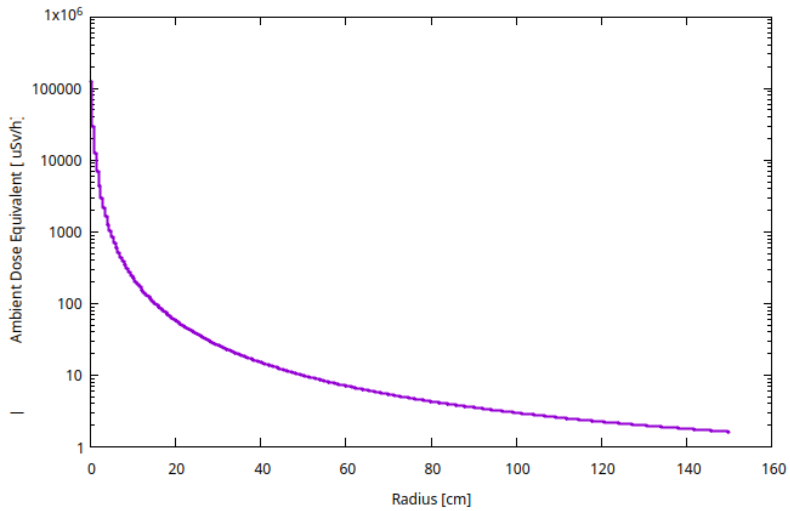

Figure 2.19: Ambient dose equivalent with radius.

sources of uncertainty, including measurement uncertainties, geometrical factors, and the characteristics of the source. Given these uncertainties related to both the simulations and the real-world measurements, the acquired data listed in Table 2.1 are consistent with the dose attributed to the source. This supports the accuracy of the measurements obtained with the MSND. This analysis affirms the reliability of both the detectors and the network, demonstrating their effective performance in accurately measuring radiation doses that correspond with the expected values from the source.

3 | Characterization and validation of the Rem counter

This chapter discusses the experimental campaigns conducted to characterize the REM counter. Measurements with extended-rem counters were performed at the CMI to evaluate the detectors' linearity and calibration, and tests were also conducted at the CERF to verify dose response in mixed radiation fields.
The experiments were planned in collaboration with the CNAO radioprotection team, who carried out the on-site measurements. The collected data was then analyzed and presented in this thesis. As discussed in Section 1.5.1, this process had been previously conducted, but further verification was required due to modifications introduced in the counting code (Section 2.2), which affected the system's behavior. The results of these analyses are detailed in the following sections.

3.1. Czech Metrology Institute facility

The Czech Metrology Institute (CMI), headquartered in Prague, is a leading experimental facility specializing in the calibration of both active and passive instruments, particularly in the field of ionizing radiation. At CMI's Unit of Primary Metrology of Ionizing Radiation, a variety of radioactive sources are utilized for the precise calibration of instruments designed to measure environmental neutron dose equivalents. These sources are crucial for ensuring the accuracy and reliability of measurements across diverse applications, including industrial, scientific, and research-based settings. In collaboration with external institutions, CMI conducts calibration and metrological reliability confirmation for instruments used in radiation protection. These calibrations are based on well-established measurement standards, ensuring compliance with international protocols and high levels of accuracy.
Typically, the calibration process is carried out using a low-rate Am-Be or ^{252}Cf source. The radioactive sources used for instrument calibration at CMI allow for the adjustment and validation of detectors across a broad spectrum of neutron doses, making CMI a key

player in the field of environmental radiation measurement. A picture of the calibration room at CMI is presented in Figure 3.1.

Figure 3.1: Calibration room at CMI.

3.1.1. calibration coefficient evaluation at CMI facility

The calibration coefficient, denoted as k, is used to convert the recorded counts into nSv. The measurements were performed using an Am-Be source with a certified reference dose rate of 36 ± 1.66 μSv/h, with the detectors positioned at a distance of 50 cm from the source. The calibration coefficient for MSNDs was calculated using the following formula:

$$k = \frac{H^*(10)}{\text{counts}} \tag{3.1}$$

where ambient dose equivalent, $H^*(10)$, and the counts are integrated over the measurement time t. A total of eight detectors were calibrated, and the results of these measurements are summarized in Table 3.1. A comparison is also made with the previously evaluated calibration coefficients at CNAO.

Meas. Number	Meas. Time [h]	Ref. Dose [μSv]	MSND Counts	Calibration Coefficient [nSv/count]
1	0.17	6.10	4805 ± 69	1.27 ± 0.06
2	0.21	7.47	4848 ± 69	1.54 ± 0.08
3	0.17	5.99	5194 ± 72	1.15 ± 0.06
4	0.16	5.88	4991 ± 70	1.18 ± 0.06
5	0.16	5.93	4973 ± 70	1.19 ± 0.06
6	0.18	6.61	4945 ± 70	1.34 ± 0.07
7	0.17	6.08	4480 ± 66	1.36 ± 0.07
8	0.18	6.42	5017 ± 70	1.28 ± 0.06

Table 3.1: Results of the measurements and the evaluated calibration coefficients.

Finally, an average calibration coefficient of 1.29 nSv/count was determined, based on a mean count value of 4909 ± 70 and a mean reference dose of 6310 nSv.

3.1.2. Linearity test of the Rem Counter at CMI facility

The detector linearity test was also conducted at CMI. Specifically, the measurements were taken using a single detector. An Am-Be source was used for the measurements, and the detector was positioned at varying distances from the source, ranging from 15 to 150 cm, providing a reference dose between 0.13 mSv/h and 13 mSv/h. For the evaluation of linearity, the counts and the reference dose were integrated over a measurement time t. The results of the measurements are shown in the Figure 3.2 and in the Table 3.2.
In the analysis, both statistical and systematic (Type B) uncertainties were considered. Statistical uncertainties were estimated using Poisson statistics, where the uncertainty on the real response is given by the square root of the measured counts. A A value of 5% was applied to both the real and ideal responses to account for calibration and instrumental type B uncertainties. Additionally, a 5% uncertainty in the dose rate was propagated to the ideal response. The total uncertainty was calculated by adding the statistical and type B variances, ensuring a comprehensive assessment of the measurement uncertainties.

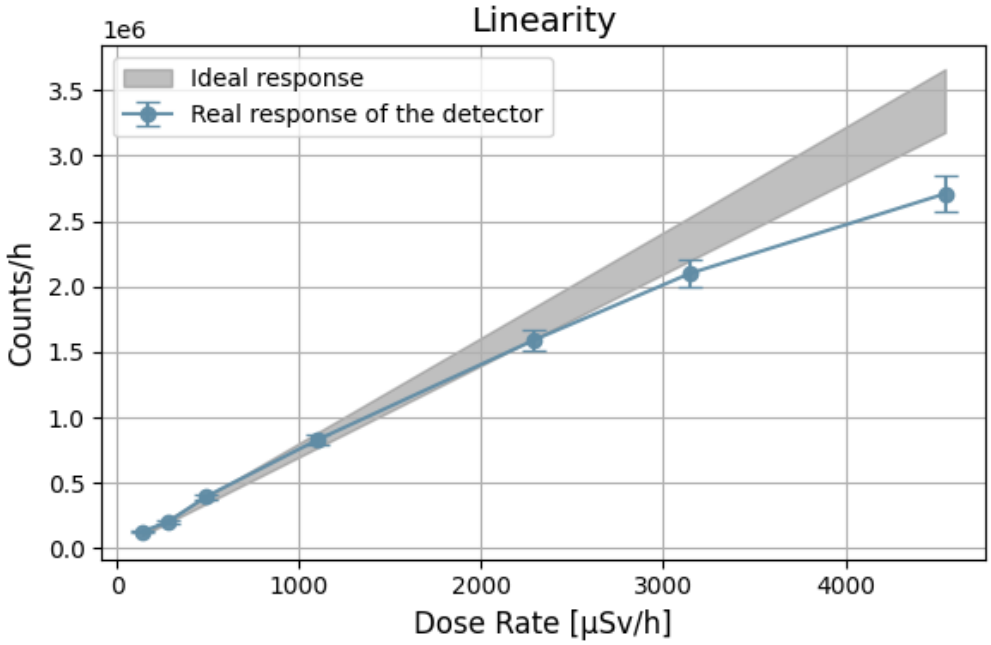

Figure 3.2: Plot of the linearity test of the REM counter; the shaded gray area represents the ideal response with a propagated uncertainty of 7%.

Dose Rate [mSv/h]	MSND Counts [counts/h]
4.54 ± 0.209	$2.70 \cdot 10^6 \pm 1643$
3.14 ± 0.144	$2.09 \cdot 10^6 \pm 1445$
2.29 ± 0.105	$1.59 \cdot 10^6 \pm 1260$
1.10 ± 0.050	$8.29 \cdot 10^5 \pm 398$
0.49 ± 0.022	$3.96 \cdot 10^5 \pm 629$
0.28 ± 0.013	$2.03 \cdot 10^5 \pm 450$
0.13 ± 0.006	$1.29 \cdot 10^5 \pm 359$

Table 3.2: Results of the measurements.

The results demonstrate that linearity is maintained up to approximately 3 mSv/h.

3.2. CERN-EU high-energy reference field

The CERN-EU high-energy Reference Field (CERF) facility was established in the early 1990s to simulate mixed radiation fields similar to those found outside the shielding of

high-energy hadron accelerators,[37]. It is located in the North Experimental Area on CERN's Prévessin site in France and is powered by a secondary beam from the Super Proton Synchrotron (SPS). This beam consists of positively charged hadrons with a momentum of 120 GeV/c, originating from the impact of the primary SPS proton beam (450 GeV/c) on a 30 cm long beryllium target (T4 target). Inside the CERF facility, the secondary beam strikes a cylindrical copper target measuring 50 cm in length and 7 cm in diameter. This target is positioned beneath either an 80 cm thick concrete roof (Figure 3.3, referred to as the concrete top, CT) or a 40 cm thick iron roof (Figure 3.4, referred to as the iron top, IT). The secondary particles generated by the copper target—primarily neutrons and photons, along with a smaller fraction of electrons, protons, pions, and muons—traverse these roof shields, creating relatively uniform radiation fields over two 2x2 m^2 areas, which are situated approximately 90° to the incoming beam direction. Each of these 2x2 m^2 areas is subdivided into 16 squares, each measuring 50x50 cm^2. Each square represents a reference point for radiation exposure, with the nominal measurement positions being at the center of each square, located 25 cm above the roof or 25 cm away from the lateral shielding. For additional flexibility, the copper target can be removed, allowing direct placement of instrumentation or materials in the beam path. In the Figure 3.5, we can see an example of the field generated at three specific locations: one on the concrete top (CT8), another on the concrete side (CS2), and a third on the iron top (IT4).

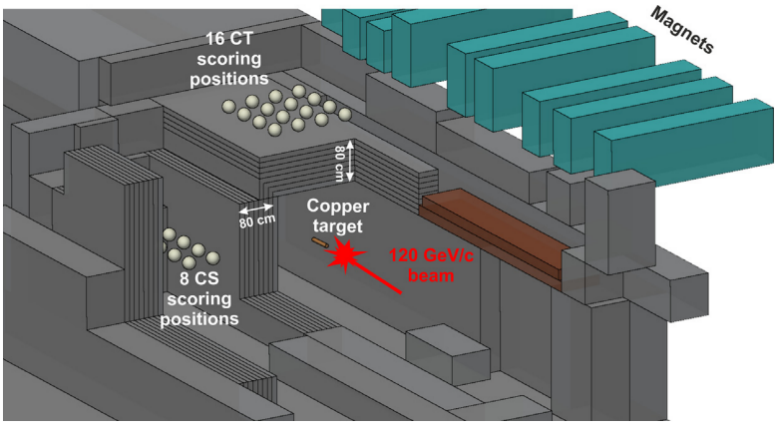

Figure 3.3: Cross sectional view of the irradiation area with the copper target under the concrete roof.

The geometry of the CERF facility was accurately reproduced in FLUKA as showd in Figures 3.3 and 3.4.

During the beam extraction, the intensity remains constant, resulting in stable radiation fields at the designated exposure locations. Beam monitoring is crucial for normalizing

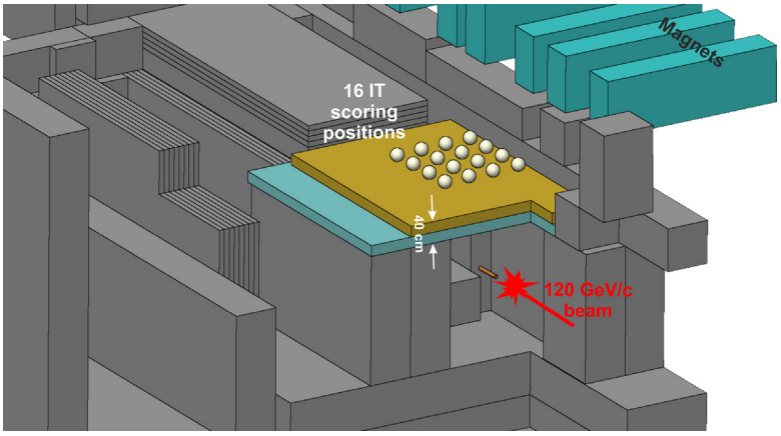

Figure 3.4: Cross sectional view of the irradiation area with the copper target under the iron roof.

all measurements and is accomplished via an air-filled, parallel-plate, transmission-type ionization chamber (IC). Recent comparisons between the IC calibration coefficients and those obtained through FLUKA simulations,modeling the expected charge collected on the IC plates, show excellent agreement. Ambient dose equivalent rates measured range from approximately 5 μSv/h to 250 μSv/h on the concrete roof and side, and from 18 μSv/h to 360 μSv/h on the iron roof. To provide independent monitoring and ensure beam stability, a fixed-position CERN LINUS rem counter is deployed on either the concrete or iron roof. Thanks to detailed characterization of the mixed radiation field, achieved through both Monte Carlo simulations and experimental data, the CERF facility offers an updated set of reference values for radiological quantities relevant to radiation protection and dosimetry . These values, computed using FLUKA simulations, have been validated against the results obtained from LINUS rem counters used to monitor the radiation field, [38].
This allows for the evaluation of the accuracy of the measurements obtained with the rem counters, ensuring a reliable basis for comparison.

3.2.1. Measurements performed at CERF facility

Measurements at the CERF were conducted both on the concrete top, as shown in the Figure 3.6, and on the iron top at several specific locations. In particular, the measured reference dose rates for the different positions are as follows:

- CT (Concrete Top): 5-250 μSv/h;
- IT (Iron Top): 18-360 μSv/h.

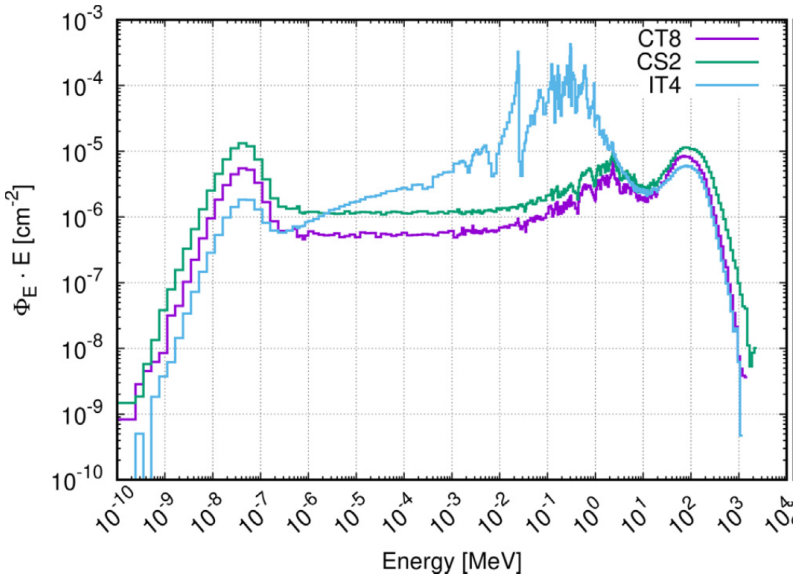

Figure 3.5: Neutron fluence per lethargy unit in the reference positions CT8 (purple), CS2 (green) and IT4 (light blue). The values are normalized per incident beam particle on the copper target.

The reference values on both the concrete top (CT) and iron top (IT) are well below the saturation limit of 3 mSv/h, as measured at CMI, 3.1.1. This confirms that the REM counter can reliably operate at the specified reference positions within the facility without reaching its saturation threshold.

3.2.2. Results at CERF facility

The collected measurement data was analyzed, and a comparison was made between the dose measured by the MSND and the dose values calculated using FLUKA and the IC counts from the CERF radioprotection group. In particular, the dose rate for the REM counters was determined by applying the calibration coefficients measured at CMI to the counts recorded by the MSND. The FLUKA dose rate is calculated by multiplying the reference FLUKA values at a given position by the IC counts provided by the facility team, [38]. The results of the measurements performed on both the concrete top (CT) and iron top (IT) are presented respectively in Tables 3.3 and 3.4.
The data analysis revealed an underestimation by the detector under study compared to the values calculated with FLUKA, showing an average underestimation of 9% for the measurements taken on the concrete top and a 30% underestimation for those on the iron top. This significant underestimation, especially in comparison to the measurements taken on the concrete positions, can be attributed to the different neutron emissions from the iron top. These emissions occur within an energy range where the rem counter is

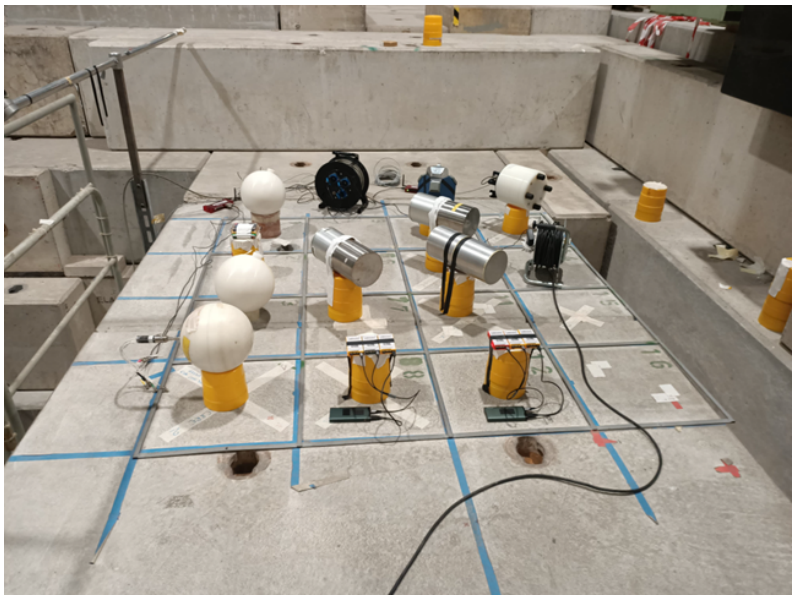

Figure 3.6: Positions available on the concrete top (CT).

known to underestimate the International Commission on Radiological Protection (ICRP) coefficients. This analysis corroborates the experimental data acquired during the previous mission and analyzed in [18].

Meas. Number	Position	Meas. Time [m]	FLUKA $H^*(10)$ [μSv]	MSND $H^*(10)$ [μSv]	Ratio MSND/FLUKA
1	CT1	10	30.45	26.10	0.86
2	CT2	10	35.17	29.49	0.84
3	CT3	69	205.12	201.21	0.98
4	CT4	17	32.07	20.61	0.64
5	CT4	97	103.19	86.48	0.84
6	CT4	60	113.91	95.34	0.84
7	CT5	12	22.18	20.22	0.91
8	CT8	11	18.23	12.52	0.69
9	CT9	6	8.75	7.75	0.89
10	CT11	6	10.30	8.32	0.81
11	CT11	6	8.42	6.87	0.82
12	CT13	69	90.89	84.94	0.93
13	CT13	10	31.43	35.67	1.00
14	CT13	10	34.33	33.85	0.99
15	CT13	69	201.17	191.00	0.95
16	CT13	17	35.18	34.20	0.97
17	CT13	11	16.38	15.34	1.09
18	CT13	6	7.40	8.06	1.09
19	CT13	6	8.17	7.93	0.97
20	CT13	6	6.69	6.06	0.91
21	CT15	6	7.15	6.47	0.91
22	CT15	6	8.74	8.05	0.92
23	CT15	6	7.91	7.83	0.99
24	CT15	11	17.51	16.33	0.93
25	CT15	12	20.31	18.05	0.89
26	CT15	17	37.59	39.71	1.06
27	CT15	69	214.98	189.81	0.88
28	CT15	10	36.68	39.29	1.07
29	CT15	10	33.53	35.671	1.06

Table 3.3: Results of the measurements performed on the concrete top.

Meas. Number	Position	Meas. Time [m]	FLUKA $H^*(10)$ [μSv]	MSND $H^*(10)$ [μSv]	Ratio MSND/FLUKA
1	IT1	9	50.92	37.95	0.75
2	IT2	62	382.20	247.52	0.65
3	IT2	41	266.23	247.51	0.64
4	IT3	9	63.38	39.871	0.63
5	IT4	9	48.10	60.42	0.80
6	IT5	91	624.80	451.66	0.72
7	IT6	9	75.76	47.94	0.63
8	IT7	10	84.31	59.78	0.71
9	IT8	10	80.97	51.02	0.63
10	IT10	10	77.55	50.72	0.65
11	IT11	10	85.64	68.85	0.80
12	IT13	9	29.68	40.51	0.73
13	IT14	9	53.49	36.83	0.69
14	IT15	9	59.12	39.91	0.68
15	IT16	9	55.72	45.91	0.82

Table 3.4: Results of the measurements performed on the iron top.

4 | Experimental Campaigns at CNAO

This chapter examines two distinct sets of measurements conducted at the CNAO. The first measurement was performed in Treatment Room number three, while the second took place in the experimental room (XPR). In the first instance, the objective was to verify and validate the functionality of the individual detectors within a typical treatment field, simulating the presence of a patient using a phantom. These measurements were conducted under standard operational conditions that are representative of the facility. In the second instance, the focus shifted to further validating the functionality of the diodes, with an emphasis on ensuring the proper operation of the network and the control software for managing multiple devices.
The results demonstrated that both the detectors and the system developed in this thesis, which includes the network and control software, are suitable and applicable in a facility like CNAO. In particular, it was confirmed that the monitoring system functions reliably over extended periods, which is essential since the entire system was designed for continuous dose level monitoring.The comprehensive testing and validation processes confirmed the performance and reliability of the system, ensuring it meets the operational requirements necessary for effective radiation monitoring and control in such a specialized environment.

4.1. Measurements performed in treatment room number three

As previously mentioned, the CNAO facility features a synchrotron capable of accelerating protons up to 250 MeV and carbon ions up to 400 MeV/u. This synchrotron delivers the beam to three treatment rooms, where patients undergo therapy, as well as a dedicated research room. The measurements discussed here were conducted in treatment room number three. Specifically, two phantoms were irradiated: one shaped like a head and the other resembling a torso. The neutron field in this case was generated by the interaction

of the proton beam with the phantom, as well as potential interactions of the beam with other elements in the room, as described in Section 1.2. To carry out the measurements, both extended-rem counters and WENDI detectors were used. In this chapter, the performance of these two detectors is compared. As previously discussed, the detectors can be controlled both individually and via software interfaces. In this experiment, we chose to control them individually to comprehensively assess the functionality of each detector.

4.1.1. Measurement Setup and Geometry

The measurements were conducted by irradiating two phantoms, representing a patient's head and torso, with protons at energies ranging from 60 to 230 MeV. These phantoms were positioned on a treatment couch, similar to those used in patient therapies, and exposed to a proton beam delivering a dose of 1 Gy. This setup closely followed an actual treatment protocol, ensuring a realistic simulation of the conditions encountered in clinical treatment rooms. Two WENDI detectors and three extended-range counters, as detailed in this thesis, were strategically positioned at various angles ranging from -90° to 140° around the phantoms at 2 meters, as shown in Figure 4.1. At each angle,

Figure 4.1: Treatment Room three with the head phantom and the positioned detectors.

the phantoms were irradiated for a duration of 1 minute and 30 seconds for the head, and 1 minute and 58 seconds for the torso. The detectors under investigation provide a file that records the integral counts as the measurement progresses. From this file, the

differential counts between one irradiation and the next were obtained, allowing for the subsequent calculation of the dose based on the calibration coefficients described in the previous chapter. The results are presented in the following sections.

4.1.2. Experimental Comparison with WENDI detector results

The collected measurements were analyzed, and a comparison was made between the dose measured by the well-known WENDI detector and the extended-rem counter under study. All data can be found in the Appendix C. Specifically, Tables B.1, B.2, B.3 display the data for the head-shaped phantom, while Table B.4, B.5, B.6 contains the data for the torso-shaped phantom. The dose measurements from the diodes were calculated by multiplying the calibration coefficients by the differential counts obtained between consecutive measurements. By considering the differential dose acquired with the two WENDI detectors, a direct comparison can be made at different angles. In Tables 4.1 and 4.2, which correspond to the measurements for the head-shaped and torso-shaped phantoms, respectively, a concordance is observed between the doses measured by the MSND and the WENDI detectors. This comparison was made by averaging the doses recorded by the various diodes at both negative and positive angles, and correlating these with the doses measured by the WENDI detectors. The results indicate a strong agreement, suggesting that both detection methods provide consistent measurements under the experimental conditions employed. This consistency enhances the reliability of the data obtained during the measurements with the detector under investigation.

Angle	MSND [μSv]	WENDI [μSv]	Ratio
0°	2.347	2.64	1.124
30°	0.200	0.224	1.165
60°	0.097	0.086	0.892
70°	0.087	0.075	0.854
90°	0.059	0.078	0.883
110°	0.060	0.049	0.807
120°	0.046	0.045	0.974
130°	0.046	0.049	1.057
140°	0.050	0.048	0.950

Table 4.1: Results of the measurements on the head phantom.

Angle	MSND [μSv]	WENDI [μSv]	Ratio
0°	30.218	46.25	1.530
30°	1.701	1.735	1.019
45°	0.986	1.19	1.206
60°	0.628	0.742	1.180
90°	0.374	0.413	1.101
120°	0.379	0.335	0.883
140°	0.347	0.339	0.975

Table 4.2: Results of the measurements on the torso phantom.

The dose distribution as a function of angle is shown in Figure 4.2 for the torso phantom and Figure 4.3 for the head phantom.

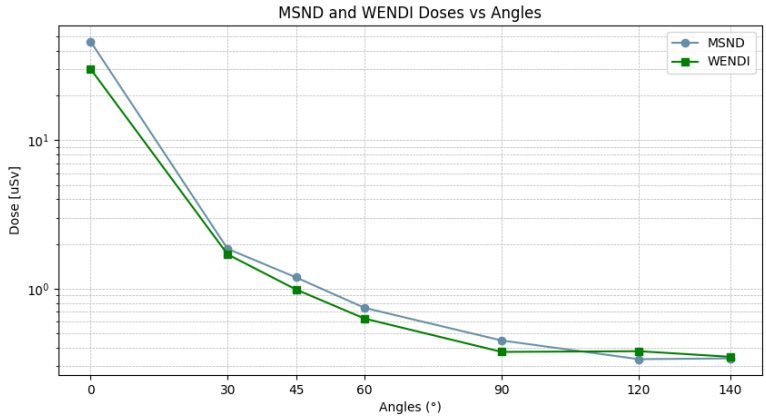

Figure 4.2: Angular dose distribution of MSND and WENDI detectors for torso phantom at 1 Gy proton irradiation.

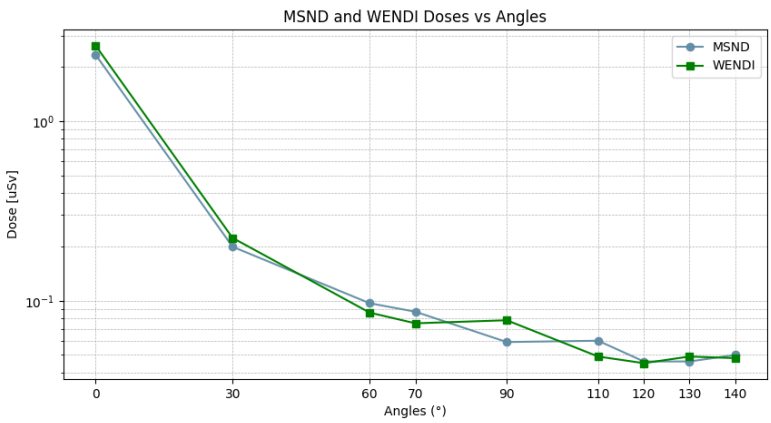

Figure 4.3: Angular dose distribution of MSND and WENDI detectors for head phantom at 1 Gy proton irradiation.

As shown in the figures, the highest neutron dose in both experiments was recorded at 0°, directly in line with the proton beam. This result indicates that the majority of secondary neutrons are produced in the forward direction, closely aligned with the primary proton beam's trajectory. Such a trend aligns with theoretical expectations, where neutron production due to proton interactions is largely concentrated along the beam path. However, the torso phantom exhibited a substantially higher dose at 0° compared to the head phantom. This increased dose can be attributed to the larger volume and cross-sectional area of the torso, leading to a greater number of neutron interactions and, consequently, an amplified forward dose. In both cases, the dose decreased sharply from 0° to 30°, with very low doses observed at angles greater than 30°. This rapid reduction suggests that secondary neutrons are largely confined to the forward direction, with minimal lateral or

rearward scatter. The geometric differences between the head and torso phantoms significantly impacted the measured dose distribution. As observed in Figure 4.3, the head phantom, with a smaller cross-sectional area, likely produced fewer secondary neutrons, resulting in a lower forward dose at 0° compared to the torso phantom. Conversely, as shown in Figure 4.2, the torso phantom, with its larger mass and volume, allowed for more extensive neutron production, leading to an increased forward dose. This geometric influence is a critical factor in understanding the observed differences in dose distribution, despite both phantoms being irradiated with the same primary dose of 1 Gy.
The differences in dose readings between the WENDI and MSND detectors can be primarily attributed to the distinct geometries of the two instruments and the construction of their moderators. Additionally, variations in the materials used in each detector likely contribute to these discrepancies, as heavy charged particles interact differently with different materials. The difference in readings was more pronounced in the torso phantom experiment, which supports the observation that the larger volume of the torso generates a higher neutron flux, especially in the forward direction.

4.2. Measurements performed in the experimental room XPR

The measurements conducted in the experimental room XPR, aimed to verify the functionality of the system developed in this thesis within a real-world setting.
The CNAO experimental room is specifically designed to support research activities; therefore, it is not used for patient treatment. Its current beam line configuration functions as a "general purpose" facility, allowing for a wide range of research applications across various fields. Potential activities include the irradiation of cells, testing of beam monitoring systems, developing in-beam monitoring devices, and conducting studies on radiation hardness. To optimize the use of available space, the beam line within the experimental room can be assembled in various configurations, [39].
Specially, these measurements were carried out in collaboration with the CREMA project. The experiment involved irradiating a crystal that redirected the beam onto a silicon pixelated detector. Consequently, the neutron field generated by the interaction of the proton and carbon ions beam with the system components was measured.
At the end, the results were then compared with the WENDI present in the experimental room, allowing for a comprehensive verification of the entire system.
Therefore, the purpose of the experiment, which was to assess the robustness of the system, was achieved. Additionally, another key objective was to verify the system's

performance over extended periods, as it is intended to be used for continuous dose monitoring. To perform this, several types of measurements were conducted, including three short-duration measurements and one continuous measurement over a 4-day period. In the end, it was confirmed that the system performed reliably even during long periods, which was the most crucial aspect to verify. Specifically, the system continued counting without issue, and through the software, it was possible to retrieve the files, create the database, and view the collected data in Excel, confirming the system's capability for long-term operation.

4.2.1. Measurement Setup and Geometry

As previously described in the earlier section, the experiment involved irradiating a crystal to redirect the beam onto a silicon detector, Figure 4.4. The irradiation took place over

Figure 4.4: Irradiation Setup: on the left the irradiated silicon-based detector.

two consecutive nights, with beam configurations set according to the experimenters' specifications. Both protons and carbon ions were used during the experiment. On the first night, irradiation were performed with protons at 227.2 MeV and carbon ions at 400 MeV/u, while on the second night, only carbon ions at 300 MeV/u were used. In this case, the neutron field is generated by the interaction of the particles with the elements

of the setup and the room. For field monitoring, five MSNDs were employed, connected via the network (see Figure 4.5). The detectors were then controlled using the software

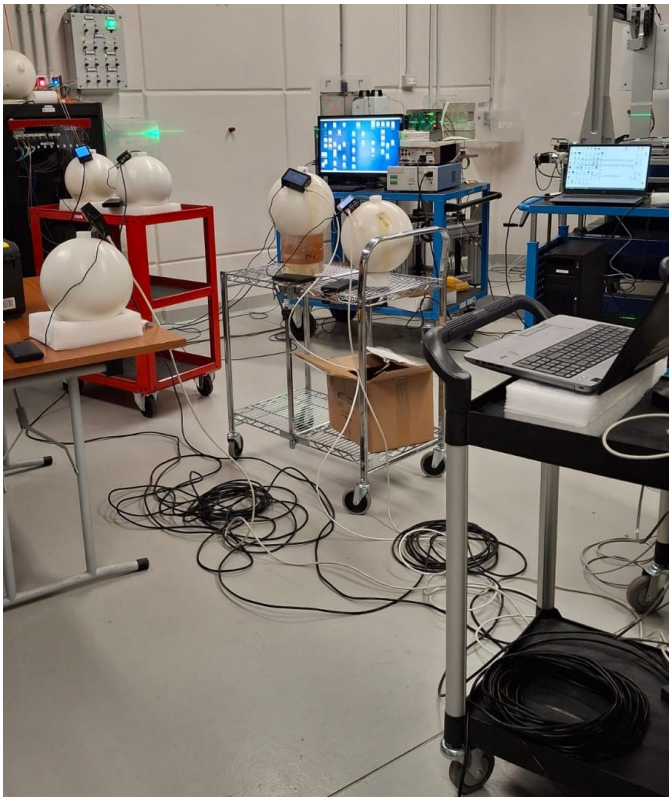

Figure 4.5: Complete retup in the XPR room: an overview of equipment and errangement.

outlined in the second chapter of this thesis. The detectors were positioned at various distances ranging from 2 to 4 meters from the irradiation point, as illustrated in Figure 4.5. During the first night, three distinct measurements were conducted:

- The first lasted 1 hour with protons only.
- The second lasted 1.40 hours with both protons and carbon ions.
- The final measurement lasted 1.45 hours with carbon ions only.

On the second night, a continuous measurement was performed, starting prior to the irradiation and lasting for four days. This extended measurement also allowed for the assessment of the system's reliability over time. Details regarding the results are provided in the following section.

4.2.2. Experimental Results

The aim of the measurements was to assess the system's capability to measure in mixed radiation fields resulting from various irradiations, as well as its performance over extended periods of time.
At the conclusion of the measurements, a comparison was made between the total dose evaluated by the different MSNDs and the WENDI available in the experimental room, providing further validation of the detectors themselves.
The results of the first three measurements conducted during the first night are presented in Tables 4.3, 4.4, 4.5. The tables present the results obtained with the various MSND detectors. In particular, the angle increases from top to bottom. The complete files with all the data are not presented due to their large size.

Device	Counts	Calibration Coeff. [nSv/count]	MSND [μSv]
1231	5114	1.19	6.08566
1469	5593	1.15	6.43195
2329	4297	1.35	5.80095
2394	4214	1.33	5.60462
545	2516	1.26	3.17016
WENDI	-	-	5.600

Table 4.3: Results of the first measurement with protons,with a total measurement duration of 1 hour. The angle increases from top to bottom.

Device	Counts	Calibration Coeff. [nSv/count]	MSND [μSv]
1231	165412	1.19	196.8403
1469	170037	1.15	195.5426
2329	130721	1.35	176.4734
2394	127425	1.33	169.4753
WENDI	-	-	172.338

Table 4.4: Results of the second measurement with protons and carbon ions, with a total measurement duration of 1 hour and 40 minutes. The angle increases from top to bottom.

Device	Counts	Calibration Coeff. [nSv/count]	MSND [μSv]
1231	207994	1.19	247.512
1469	210412	1.15	241.973
2329	164301	1.35	221.806
545	87224	1.26	109.902
WENDI	-	-	190.5

Table 4.5: Results of the third measurement with carbon ions, with a total measurement duration of 1 hour and 45 minutes. The angle increases from top to bottom.

The doses measured with the WENDI detector located in the experimental room are as follows:

- First measurement with protons: 5.600 μSv in 1:00 h
- Second measurement with protons and carbon ions: 172.338 μSv in 1:40 h
- Third measurement with carbon ions: 190.5 μSv in 1:45 h

The WENDI detector is positioned 4.20 meters from the irradiation point at an angle of approximately 90°. In contrast, the other devices were set up at varying angles ranging

from about 100° to 170° and at distances between 2 meters and 4 meters. Due to these variations in angles and distances, a direct comparison between the measurements is not feasible. Moreover, the complexity of the setup and the instruments in the room further complicate the ability to obtain a clear understanding of the radiation field. Nevertheless, the doses measured with the MSND detector are consistent with those obtained from the WENDI detector. It is also evident that the measured dose tends to decrease as the angle increases. It is important to emphasize that the primary objective of these measurements was to validate the network and control software. Ultimately, it has been confirmed that the system is functioning as intended.
The measurements conducted over four days began in the afternoon prior to the initiation of irradiation. Subsequently, the process transitioned to irradiation with various beams of protons and carbon ions, utilizing different particle numbers. Measurements continued in the following days without any further irradiation. The results are presented in the Table 4.6.

Diode Number	Counts	Calibration Coeff. [nSv/count]	MSND [μSv]
1469	545369	1.15	627.174
305	491203	1.18	579.619
2329	420110	1.35	567.148
2394	282848	1.33	376.187
545	205116	1.26	258.446

Table 4.6: Results of the measurements conducted for four days.

The various measurement modalities and durations employed were specifically selected to assess the reliability and functionality of the devices under different conditions. As discussed in previous chapters, the primary objective of the network system and the development of these detectors is not only to conduct short-term measurements but, more importantly, to establish a continuous monitoring system, as the system will be used for ongoing monitoring in a facility that operates daily. This implies that the system must operate effectively over extended periods without encountering issues. The most recent measurements have allowed for the evaluation of the system's durability. The system's ability to measure continuously over an extended period was successfully verified, with particular focus on the functionalities required for data collection. Specifically, after four days of continuous operation, the measurement files were retrieved from each Raspberry

Pi unit using the software under review and then consolidated on the master PC, where a database was created. The data were subsequently visualized and analyzed in Excel, confirming that the system can perform continuous monitoring and that the data can be reliably managed and analyzed over time. These results verify that the system is fully functional and ready for operational use.

5 | Conclusion and Future Developements

This thesis was developed in light of the CNAO's expansion, specifically with the goal of implementing new sources, a new proton gantry, and a new BNCT facility. To support this growth, the radiation protection team wants to initiate a continuous monitoring system across the entire facility to control the ambient neutron equivalent dose. Within this context, the work presented here has two main objectives: first, to develop a network and control software for the neutron detectors designed at CNAO, and second, to characterize and validate the functionality of both the detectors and the networked monitoring system.

To achieve comprehensive neutron monitoring, the design includes positioning detectors at strategic points both inside and outside the facility. A new, cost-effective REM counter was developed by the Radiation Protection team using a Microstructured Semiconductor Neutron Detector (MSND) embedded within a plastic moderator, with cadmium and lead inserts to enhance performance. Detailed descriptions of the neutron fields at CNAO, the various detectors used for neutron detection, and the MSND central to this thesis are provided in Chapter 1.

Chapter 2 then details the network and control software planned for installation across the CNAO facility, alongside verification of the software's functionality and an additional validation of the detector itself. To enable seamless detector management, the network design involved connecting the detectors via Ethernet cables through a switch to a single control PC. This system allows centralized control of multiple devices, enabling users to start and stop measurements as desired. The software also records data every 10 seconds, such as integral counts and measured doses, adjusted by each detector's calibration coefficient, into a database. Further validation of both the detector and software was conducted using an Am-Be source, and FLUKA simulations confirmed the accuracy of the detectors in measuring neutron dose.

In Chapter 3, an experimental campaign conducted at the CMI and the CERF facility is discussed. Data from the CMI experiments enabled the estimation of a calibration

coefficient of 1.29 ± nSv/count with the Am-Be source, while linearity tests identified a saturation dose rate of approximately 3 mSv/h. This result suggests the suitability of this REM counter for beam loss monitoring.
Data from CERF revealed a slight underestimation of dose in high-energy neutron fields, a known behavior in REM counters generally calibrated with Am-Be or ^{252}Cf sources, which peak at 1-10 MeV. Since the CNAO and CERF spectra contain neutrons from thermal energies up to hundreds of MeV, this calibration discrepancy can be attributed to the energy range.

Finally, in Chapter 4, two separate measurements were conducted in different setups at CNAO to fully validate both the detectors and the network.
In the first setup, individual detectors were tested in Treatment Room 3, where neutron fields were generated by irradiating two phantoms with proton beams delivering a dose of 1 Gy following a treatment plan. Measured doses at various angles with the MSND were then compared with measurements from two WENDI detectors, showing consistency across the two methods under these conditions. This agreement supports the reliability of data obtained from the detector under investigation.
The second experiment focused on verifying the entire monitoring system. Conducted in Experimental Room XPR in collaboration with the CREMA project, this test involved irradiating a silicon-based pixel detector with proton and carbon ion beams. The complexity of the field generated in this setup, due to multiple elements in the room, required validation of the software under varying conditions. Three shorter measurements (from 1 hour to 1 hour and 45 minutes) and one four-day measurement were conducted to verify long-term functionality, essential for the continuous monitoring application of the system. Results, while not directly comparable due to complex geometry, aligned closely with data from the WENDI detectors in the experimental room, further supporting the robustness of the monitoring setup. The system's ability to operate continuously over an extended period was tested. After four days of uninterrupted measurement, the data was retrieved from the Raspberry Pi units, stored on the master PC, and organized into a database. The data were then visualized and analyzed using Excel, confirming that the system can perform continuous monitoring and handle data management effectively. This process ensured that the system is fully functional and ready for use.

In conclusion, the results of this work are highly satisfactory, demonstrating the effectiveness of the developed system in monitoring ambient dose levels and beam losses throughout the CNAO facility. The system has been successfully designed to allow for continuous monitoring of multiple detectors simultaneously, even when they are placed in different locations, sometimes far apart within the facility. This flexibility ensures comprehensive

coverage across the facility. The system enables users to easily select the measurement duration, choose which detectors to monitor, and visualize real-time counts, providing immediate feedback. Additionally, it offers a streamlined process to organize, analyze, and display the collected data in a user-friendly and efficient manner, making it accessible for further analysis and decision-making. This approach ensures that the system can operate autonomously for extended periods, providing accurate and timely data for ongoing facility operations and research activities. Future improvements could include adding real-time graphical displays of dose trends to enhance monitoring and visualization. Additionally, in the future, all Ethernet cables could be integrated into the CNAO network, simplifying system installation and reducing the risk of cable clutter and potential disconnections.

Bibliography

[1] PTCOG. Facilities in operation, a.n.d. Accessed: 2024-07-08.

[2] Sandro Rossi. Hadron therapy achievements and challenges: The cnao experience. *Physics*, 4(1):229–257, 2022.

[3] van der Kogel A. Joiner M. *Basic Clinical Radiobiology.* Hodder Arnold, 2009.

[4] Juliette Thariat, Samuel Valable, Carine Laurent, Siamak Haghdoost, Elodie A. Pérès, Myriam Bernaudin, François Sichel, Paul Lesueur, Mathieu Césaire, Edwige Petit, Aurélie E. Ferré, Yannick Saintigny, Sven Skog, Mihaela Tudor, Michael Gérard, Sebastien Thureau, Jean-Louis Habrand, Jacques Balosso, and François Chevalier. Hadrontherapy interactions in molecular and cellular biology. *International Journal of Molecular Sciences*, 21(1), 2020.

[5] Kase Y. Ando K. Biological characteristics of carbon-ion therapy. *Int J Radiat Biol.*, 85(9), 2009.

[6] Skarsgard LD. Palcic B, Brosing JW. Survival measurements at low doses: oxygen enhancement ratio. br j cancer. *Br J Cancer.*, 46(6), 1982.

[7] Eirik Malinen and Åste Søvik. Dose or 'let' painting – what is optimal in particle therapy of hypoxic tumors? *Acta Oncologica*, 54(9):1614–1622, 2015.

[8] S Rossi. The status of cnao. *The European Physical Journal Plus*, 126(8):78, 2011.

[9] L Celona, G Ciavola, S Gammino, L Andò, and D Mascali. Design of the aisha ion source for hadrontherapy facilities. *Proceedings of ECRIS2012, Sydney, Australia Published in Experiments Read more... Monday*, 7:17, 2017.

[10] Bernhard Schlitt, A. Reiter, G. Breitenberger, J. Cavaco, Gianluigi Clemente, M. Hörr, N. Kischnik, C.M. Kleffner, M. Maier, G. Schulz, W. Vinzenz, and Hartmut Vormann. Linac commissioning at the italian hadrontherapy centre cnao. 01 2010.

[11] IAEA. *Advances in boron neutron capture therapy.* International Atomic Energy Agency, 2023.

[12] Gazzetta ufficiale della repubblica italiana. Decreto legislativo 31 luglio 2020, n. 101, 2020.

[13] R. SERBER. Nuclear reactions at high energies. *P H YS ICAL REVI EW*, 72(11):1114–1115, 1947.

[14] Matteo Frosini. Secondary radiation field characterization and neutron dosimetry measurements in a synchrotron for hadrontherapy, 2020.

[15] Stefano Dibartolomeo. Secondary radiation field characterization and neutron dosimetry measurements in a synchrotron for hadrontherapy, 2020.

[16] Francesco Bonforte, Michele Ferrarini, Antonio D'angola, Elio Giroletti, and Daniele Introini. Heavy-ions shielding data for hadrontherapy application with monte carlo methods. *Radiation protection dosimetry*, 199, 07 2023.

[17] Giuseppe Mazzola. Monte carlo approach for dose and activation analysis in an a-bnct facility: case study, 2021.

[18] Francesco Bonforte. Radiation physics and experimental characterization of the radiation fields produced by a heavy ion accelerator for medical applications. 2024.

[19] Glenn F Knoll. *Radiation detection and measurement*. John Wiley & Sons, 2010.

[20] A. Ferrari and M. Pelliccioni. Fluence to Dose Equivalent Conversion Data and Effective Quality Factors for High Energy Neutrons. *Radiation Protection Dosimetry*, 76(4):215–224, 04 1998.

[21] Anthony Beverding Jeffrey H. Kleck William H. Casson Dennis G. Vasilik Richard H. Olsher, Hsiao-Hua Hsu and Robert T. Devine. WENDI: AN IMPROVED NEUTRON REM METER. *Health Physics*, 79(2):170–181, 2000.

[22] I. O. Andersson and J. Braun. A Neutron Rem Counter. *echnical report, Aktiebolaget Atomenergi*, Stockholm(Sweden), 1964.

[23] V Mares, AV Sannikov, and H Schraube. Response functions of the andersson–braun and extended range rem counters for neutron energies from thermal to 10 gev. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 476(1-2):341–346, 2002.

[24] Claudio Birattari, Adolfo Esposito, Arnaud Ferrari, Mario Pelliccioni, Tiziana Rancati, and Marco Silari. The extended range neutron rem counter linus: Overview and latest developments. *Radiation Protection Dosimetry*, 76:135–148, 1998.

[25] M. Caresana; C. Cassell; M. Ferrarini; E. Hohmann; G. P. Manessi; S. Mayer; M. Silari; V. Varoli. A new version of the lupin detector: Improvements and latest experimental verification. *Review of Scientific Instruments*, 85(6), 2014.

[26] Rotunda Scientific Technologies. Microstructured semiconductor neutron detector, 2021.

[27] Steven L Bellinger, Walter J McNeil, and Douglas Scott McGregor. Improved fabrication technique for microstructured solid-state neutron detectors. *MRS Online Proceedings Library (OPL)*, 1164:1164–L06, 2009.

[28] Rotunda Scientific Technologies. Domino, 2021.

[29] S. Agosteo, Marco Caresana, Michele Ferrarini, and Marco Silari. A dual-detector extended range rem-counter. *Radiation Measurements*, 45:1217–1219, 12 2010.

[30] Antonella Talamo. Study of the response of active dosimeters for a bnct radiation field, 2021.

[31] F Bonforte, D Introini, A D'Angola, S Lamorte, S Agosteo, F Pozzi, G Garlaschelli, I De Battista, and M Ferrarini. Characterization of an extended range rem counter based on micro structured neutron detector. In *RAD CONFERENCE PROCEEDINGS*, 2024.

[32] M. Pelliccioni. Overview of Fluence-to-Effective Dose and Fluence-to-Ambient Dose Equivalent Conversion Coefficients for High Energy Radiation Calculated Using the FLUKA Code. *Radiation Protection Dosimetry*, 88(4):279–297, 04 2000.

[33] D.D. Clark, K.T. Pogran, and D.P. Reed. An introduction to local area networks. *Proceedings of the IEEE*, 66(11):1497–1517, 1978.

[34] Kinza Yasar. Transmission control protocol (tcp), 2024.

[35] Eric Rosen, Arun Viswanathan, and Ross Callon. Multiprotocol label switching architecture. Technical report, 2001.

[36] Jim Melton and Alan R Simon. *Understanding the new SQL: a complete guide*. Morgan Kaufmann, 1993.

[37] Marco Silari and Fabio Pozzi. The cern-eu high-energy reference field (cerf) facility: applications and latest developments. In *EPJ Web of Conferences*, volume 153, page 03001. EDP Sciences, 2017.

[38] F Pozzi and M Silari. The cern-eu high-energy reference field (cerf) facility: New fluka reference values of spectral fluences, present and newly proposed operational quanti-

ties. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 979:164477, 2020.

[39] Marco Pullia et al. The Experimental Beam Line at CNAO. In *7th International Particle Accelerator Conference*, page TUPMR038, 2016.

[40] Giuseppe Battistoni, Till Boehlen, Francesco Cerutti, Pik Wai Chin, Luigi Salvatore Esposito, Alberto Fasso, Alfredo Ferrari, Anton Lechner, Anton Empl, Andrea Mairani, et al. Overview of the fluka code. *Annals of Nuclear Energy*, 82:10–18, 2015.

[41] FLUKA Collaboration. Fluka online manual. `http://www.fluka.org/content/manuals/online/INDEX-manual.html`. Accessed: 2023-03-19.

A | Appendix A

A.1. Python Code

A.1.1. Libraries

```
import sys
from PyQt5.QtWidgets import  QDesktopWidget,QSpacerItem,QSizePolicy,
    QAction,QMessageBox, QMenuBar, QDialog, QApplication,QTextEdit,
    QWidget, QMainWindow,QListWidget, QListWidgetItem, QAbstractItemView,
    QPushButton,QLineEdit, QVBoxLayout, QHBoxLayout, QLabel, QFileDialog,
    QTableWidget, QTableWidgetItem, QGridLayout
from PyQt5.QtCore import pyqtSignal,pyqtSlot,QObject,QMetaType,
    QCoreApplication, Qt,QSize
from PyQt5.QtGui import QFont, QPixmap,QPixmap, QIcon
from re import findall
import subprocess
from subprocess import Popen, PIPE
import time
import os
import threading
import sqlite3
from datetime import datetime
import pandas as pd
import socket
import atexit
import paramiko
from pathlib import Path
import csv
import timeit
import concurrent.futures
```

A.1.2. Pathways of the software

```
### PATH ###
Directory_Path_File_Esterno = "/Users/margh/Desktop/GUI/File_Esterno"  #
```

```
    Path to the external file with temperature and position data
Nome_File_Esterno = "file_esterno.txt"
SAVE_PATH = 'C:/Users/margh/Desktop/GUI/File_da_Raspberry/'  # Path to
    save received files and files to be inserted into the DB
SAVE_PATH_SHOW_COUNTS = 'C:/Users/margh/Desktop/GUI/File_show_counts/'
    # Path to save received files for the show counts
Directory_Path_File_Esterno_host = "C:/Users/margh/Desktop/GUI/
    File_Esterno/file_esterno.txt"

### COPIA LA LISTA DEGLI IP DAL FILE ESTERNO ###
try:
    # Apri il file CSV in modalit  lettura
    with open(Directory_Path_File_Esterno_host, 'r') as file:
        reader = csv.reader(file)

        # Crea una lista vuota per memorizzare la prima colonna
        host = []

        # Scorri ogni riga del file
        for riga in reader:
            if riga:
                host.append(riga[0])
except Exception as e:
    print(f"Si    verificato un errore durante la lettura del file: {e}"
        )

# Image paths
path_margherita = 'C:/Users/margh/Desktop/GUI/margherita.png'
info_path = 'C:/Users/margh/Desktop/GUI/info.png'


# Output path for the database
DB_path = 'C:/Users/margh/Desktop/GUI/DB/DataBase.db'
DB_path_automatico = 'C:/Users/margh/Desktop/GUI/DataBase_automativo.db'

# Output path for the Excel file
excel_output_path_automatico = 'C:/Users/margh/Desktop/GUI/
    output_automatico.xlsx'
excel_output_path = 'C:/Users/margh/Desktop/GUI/output.xlsx'

# Global variables
current_time = time.localtime()
current_date = time.strftime("%Y-%m-%d %H:%M:%S", current_time)
test = ""
```

```
test_file = ""

log_file_path_automatica = 'C:/Users/margh/Desktop/GUI/File_log//
    log_automatica.txt'
log_file_automatica = None  # To ensure that the variable is not
    accidentally used or modified before it's initialized with a valid
    value.

log_file_path = 'C:/Users/margh/Desktop/GUI/File_log/log.txt'
log_file = None
```

A.1.3. Log File Section

```

# Opens a file and keeps it open for the entire execution, automatically
    closing it when the GUI is closed
def open_log_file():
    global log_file
    log_file = open(log_file_path, 'a')
    atexit.register(close_file, log_file)  # Ensures the file is closed
        when the program exits

def close_file(file):
    file.close()
    print("File closed successfully.")

#Current date
        current_time = time.localtime()
        current_date = time.strftime("%Y-%m-%d %H:%M:%S", current_time)

        open_log_file()   #file opening
        log_file.write(f" Data : {current_date} \n") #write the current
            date and hour each time the file is opended
```

A.1.4. Ping code

```
### PING CODE ###
    ping_finished = pyqtSignal(str)  # Used to connect the GUI to
        different threads
    def ping(self, host, ping_count):
        flag = 0
        global test
        log_file.write("Executing Ping \n")
```

```
7
8           for ip in host:  # Iterates through all the IPs in the host list
9               data = ""
10              output = Popen(f"ping {ip} -n {ping_count}", stdout=PIPE,
                    encoding="utf-8")  # -c for Linux, -n for Windows,
                    command line
11
12              for line in output.stdout:
13                  data = data + line
14                  ping_test = findall("TTL", data)  # TTL on Raspberry Pi
                        should be 64
15
16              if ping_test:
17                  print(f"{ip} : Successful Ping")
18                  log_file.write(f"{ip} : Successful Ping \n")
19                  self.w.text_edit.append(f"{ip} : Successful Ping")
20
21              else:
22                  print(f"{ip} : Failed Ping")
23                  log_file.write(f"{ip} : Failed Ping \n")
24                  self.w.text_edit.append(f"{ip} : Failed Ping")
25                  flag = 1
26
27          if flag == 1:
28              test = "A Ping failed"
29          else:
30              test = "All the Pings are successful"
31
32          self.ping_finished.emit(test) #To emit the ping's status on the
                GUI
33
34
35  ###PING THREAD###
36  def ping_button_clicked(self):
37      thread2 = threading.Thread(target=self.ping, args= host,ping_count))
38      thread2.start()
39      self.ping_finished.connect(self.update_ping_status)
```

A.1.5. Ping status window

```
1  ### CLASS FOR PING STATUS WINDOW ###
2  class Window_Ping(QMainWindow):
3      def __init__(self):
4          super().__init__()
```

```
        self.setWindowTitle("Ping's Status")
        self.setGeometry(200, 200, 400, 300)

        self.text_edit = QTextEdit(self)
        self.text_edit.setGeometry(10, 10, 380, 280)
###TO OPEN THE PING WINDOW###
def open_new_window(self,test):
    self.w = Window_Ping()
    self.w.show()
```

A.1.6. File Esterno Function Code

```
### FILE ESTERNO ###
 check_file_in_directory_finished = pyqtSignal(str)
    def check_file_in_directory(self, directory, filename):
        global test_file
       # Ottieni la lista dei file nella directory
        files_in_directory = os.listdir(directory)

        # Verifica se il file    presente nella lista dei files
        if filename in files_in_directory:
            file_path = os.path.join(directory, filename)
            # Verifica che il file non    vuoto
            if os.path.getsize(file_path) > 0:
                print(f"Il file '{filename}'    presente nella directory '{directory}' e non    vuoto.\n")
                log_file.write(f"Il file '{filename}'    presente nella directory '{directory}' e non    vuoto.\n")
                test_file = "File presente e non vuoto"
            else:
                print(f"Il file '{filename}'    presente nella directory '{directory}' ma    vuoto.\n")
                log_file.write(f"Il file '{filename}'    presente nella directory '{directory}' ma    vuoto.\n")
                test_file = "File presente e vuoto"
        else:
                print(f"Il file '{filename}'    presente nella directory '{directory}'.\n")
                log_file.write(f"Il file '{filename}' non    presente nella directory '{directory}'.\n")
                test_file = "File non presente"
        self.check_file_in_directory_finished.emit(test_file)
### FILE ESTERNO THREAD ###
def check_file_in_directory_button_clicked(self):
```

```
    thread3 = threading.Thread(target=self.check_file_in_directory, args
        =(Directory_Path_File_Esterno, Nome_File_Esterno))
    thread3.start()
    self.check_file_in_directory_finished.connect(self.
        update_check_file_in_directory_status)
```

A.1.7. IP Selector Function Code

```
### CLASS TO SELECT IP ###
class IPSelector(QDialog):
    ipsSelected = pyqtSignal(list)  # Segnale personalizzato per inviare
        gli IP selezionati

    def __init__(self, parent=None):
        super(IPSelector, self).__init__(parent)
        self.setWindowTitle('IP Selector')
        self.setGeometry(300, 300, 300, 200)
        self.initUI()

    def initUI(self):
        # Layout verticale
        vbox = QVBoxLayout()

        # Messaggio informativo
        info_label = QLabel("Please select the IP addresses you want to
            use:")
        info_label.setFont(QFont("Helvetica", 10))

        # Lista di selezione IP
        self.listWidget = QListWidget()
        self.listWidget.setSelectionMode(QAbstractItemView.
            MultiSelection)

        for ip in host:
            item = QListWidgetItem(ip)
            self.listWidget.addItem(item)

        # Pulsante per ottenere IP selezionati
        btn = QPushButton("Get Selected IPs")
        btn.clicked.connect(self.showSelectedIPs)


        vbox.addWidget(info_label)
        vbox.addWidget(self.listWidget)
```

```
        vbox.addWidget(btn)
        self.setLayout(vbox)

        # Barra men
        menubar = QMenuBar(self)
        help_menu = menubar.addMenu("Help")


        help_action = QAction("About", self)
        help_action.setStatusTip("Show information about this dialog")
        help_action.triggered.connect(self.showHelpMessage)
        help_menu.addAction(help_action)

        vbox.setMenuBar(menubar)

    def showHelpMessage(self):
        QMessageBox.information(self, "Help", "This dialog allows you to
            select IP addresses from the list. You can choose one or
           multiple IP addresses selecting more IP.")

    def showSelectedIPs(self):
        selected_items = self.listWidget.selectedItems()
        selected_ips = [item.text() for item in selected_items]
        self.ipsSelected.emit(selected_ips)  # Emissione del segnale con
            gli IP selezionati
        self.accept()
```

A.1.8. Filename and Measure Time

```
# Sezione input per il tempo di misura e il filename
        input_layout = QVBoxLayout()
        self.label_max_count = QLabel('Enter measurement time:')
        self.line_edit_max_count = QLineEdit()

        self.label_filename = QLabel('Enter the file name:')
        self.line_edit_filename = QLineEdit()

        input_layout.addWidget(self.label_max_count)
        input_layout.addWidget(self.line_edit_max_count)
        input_layout.addWidget(self.label_filename)
        input_layout.addWidget(self.line_edit_filename)

        self.layout.addLayout(input_layout)
```

A.1.9. Start Function Code

```
### START ###
 def send_data(self):
        if not self.selected_ips:
            print("No IPs selected!")
            return
        # Logica per gestire gli IP selezionati
        print("Starting with IPs:", ", ".join(self.selected_ips))

        max_count = self.line_edit_max_count.text()
        filename = self.line_edit_filename.text()
        if not max_count or not filename:
            self.ip_label.setText(f"Selected IPs: {', '.join(self.
               selected_ips)}. \nPlease select a filename and a
               measurement time.")
            return  # Termina la funzione se mancano input
        else:
            self.ip_label.setText(f"Selected IPs: {', '.join(self.
               selected_ips)}.")  # Pulisce il messaggio se i dati sono
               forniti

        print(f'{max_count}')
        print(f'{filename}')
        log_file.write(f"measurement time = {max_count}, Filename = {
           filename}\n")
        thread7 = threading.Thread(target=self.send_to_all_ips, args=(
           max_count, filename,self.selected_ips))
        thread7.start()

    def send_to_ip(self,ip, max_count, filename):
        port = 22  # Porta SSH
        username = "pi"
        password = "Fondazione09"

        ssh_client = paramiko.SSHClient()

        try:
            ssh_client.set_missing_host_key_policy(paramiko.
               AutoAddPolicy())
            ssh_client.connect(hostname=ip, port=port, username=username
               , password=password)
            print(f"Connessione riuscita con {ip}\n")
            log_file.write(f"Connessione riuscita con {ip}\n")

```

```
36            # Esegui il comando remoto
37            command = f"cd Desktop && sudo ./File_c_10 {max_count} {
                filename}"
38
39            stdin, stdout, stderr = ssh_client.exec_command(command)
40            stdout = stdout.read().decode()
41            stderr = stderr.read().decode()
42
43            if stderr:
44                print(f"Errore durante l'esecuzione su {ip}: {stderr}\n"
                    )
45                log_file.write(f"Errore durante l'esecuzione su {ip}: {
                    stderr}\n")
46            else:
47                print(f"Output di prova su {ip}: {stdout}\n")
48                log_file.write(f"Output di prova su {ip}: {stdout}\n")
49
50        except Exception as e:
51            print(f"Errore di connessione con {ip}: {e}\n")
52            log_file.write(f"Errore di connessione con {ip}: {e}\n")
53        finally:
54            ssh_client.close()
55
56    def send_to_all_ips(self, max_count, filename, selected_ips):
57        # Definisci un numero massimo di thread nel pool, quindi quanti
            Raspberry si possono controllare. Cambiare numero se sono
            pi   di 20
58        max_threads = 20
59
60        # Usa ThreadPoolExecutor per gestire i thread
61        with concurrent.futures.ThreadPoolExecutor(max_workers=
            max_threads) as executor:
62            # Mappa l'esecuzione della funzione send_to_ip a ogni
                indirizzo IP
63            futures = [executor.submit(self.send_to_ip, ip, max_count,
                filename) for ip in selected_ips]
64
65
66            for future in concurrent.futures.as_completed(futures):
67                try:
68                    future.result()  # Attende che il thread sia
                        completato
69                except Exception as exc:
70                    print(f'Errore nel thread: {exc}')
```

```
71                      log_file.write(f'Errore nel thread di lancio del
                            comando di start : {exc}\n')
72
73          print("Tutti i comandi sono stati inviati.")
74          log_file.write("Tutti i comandi sono stati inviati agli IP
                selezionati.\n")
```

A.1.10. Stop Function Code

```
1   ### STOP ###
2       def stop_c_program(self, selected_ips):
3           port = 22  # Porta SSH, di solito 22
4           username = "pi"
5           password = "Fondazione09"
6           ssh_client = paramiko.SSHClient()
7           print("Stopping program on Raspberry")
8           log_file.write("Stopping program on Raspberry\n")
9
10          for ip in selected_ips:
11              try:
12                  ssh_client.set_missing_host_key_policy(paramiko.
                        AutoAddPolicy())
13                  ssh_client.connect(hostname=ip, port=port, username=
                        username, password=password)
14                  print("Connessione riuscita")
15                  log_file.write(f"Connessione riuscita con {ip}\n")
16
17                  # Comando per fermare il programma
18                  command = f"cd Desktop && sudo pkill File_c_10"
19
20                  stdin, stdout, stderr = ssh_client.exec_command(command)
21
22                  stdout = stdout.read().decode()
23                  stderr = stderr.read().decode()
24                  if stderr:
25                      print(f"Errore durante il tentativo di fermare il
                            programma: {stderr}")
26                      log_file.write(f"Errore durante il tentativo di
                            fermare il programma su {ip}: {stderr}\n")
27                  else:
28                      print(f"Programma fermato su {ip}: {stdout}")
29                      log_file.write(f"Programma fermato su {ip}: {stdout
                            }\n")
30
```

```
31            except Exception as e:
32                print(f"Errore di connessione: {e}")
33                log_file.write(f"Errore di connessione nello stop del
                     programma: {e}\n")
34            finally:
35                ssh_client.close()
36 ### STOP THREAD ###
37  def stop(self):
38        thread = threading.Thread(target=self.stop_c_program, args=(self
             .selected_ips,))
39        thread.start()
```

A.1.11. Connecting to Raspberry Pi Function Code

```
1 ### CONNECTING TO A RASPBERRY PI ###
2 def connessione_a_raspberry(self, host):
3        port = 22  # Porta SSH, di solito 22
4        username = "pi"
5        password = "Fondazione09"
6        # Creare un'istanza del client SSH
7        ssh_client = paramiko.SSHClient()
8
9        # Aggiungere automaticamente il nuovo host alla lista degli host
             conosciuti
10       ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
11       for ip in host :
12           try:
13               # Connettersi al dispositivo remoto
14               ssh_client.connect(hostname= ip, port=port, username=
                     username, password=password)
15               print(f"Connessione riuscita all'IP :{ip}")
16               log_file.write(f"{ip} : Connessione riuscita \n")
17               # Eseguire un comando remoto
18               stdin, stdout, stderr = ssh_client.exec_command("cd
                     Desktop && python3 Server_aggiornato.py")
19               print("Output del comando:", stdout.read().decode())
20               log_file.write(f"{ip} : Output del comando:", stdout.
                     read().decode())
21               # Chiudere la connessione
22               ssh_client.close()
23           except Exception as e:
24               print(f"Errore di connessione: {e}")
25               log_file.write(f"{ip} : Errore di connessione: {e}")
26 ### CONNECTING TO RASPBERRY THREAD ###
```

```
def connessione_a_raspberry_button_clicked(self):
        thread0 = threading.Thread(target=self.connessione_a_raspberry,
            args=(host,))
        thread0.start()
```

A.1.12. File Request Function Code

```
### FILE REQUEST ###
file_request_raspberry_finished = pyqtSignal(str)
    def file_request_raspberry(self,HOSTS,SAVE_PATH):
        global current_date2
        current_time1 = time.localtime()
        current_date2 = time.strftime("%Y_%m_%d_%H_%M_%S", current_time1
            )
        print("Richiesta file raspberry")
        log_file.write("Richiesta file raspberry\n")
        global output_messages
        output_messages = ""  # Stringa per memorizzare i messaggi di
            output
        PORT = 12346
        try:
            for host in HOSTS:  # Itera su ogni host nella lista HOSTS
                with socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                    as s:
                    s.connect((host, PORT))
                    print(f"Connesso al server {host}:{PORT}")
                    log_file.write(f"Connesso al server {host}:{PORT}\n"
                        )
                    output_messages += f"Connesso al server {host}:{PORT
                        }\n"

                    # Invia il nome del file al server
                    filename = self.line_edit_filename_RP.text()
                    s.sendall(filename.encode())

                    response = s.recv(1024).decode()
                    if response != "OK":
                        print(f"Errore: {response}")
                        log_file.write(f"Errore: {response}\n")
                        output_messages += f"Errore: {response}\n"
                        continue

                    file_path = os.path.join(SAVE_PATH, f'FILE_{
                        current_date2}_{host}.txt')  # Nome del file
```

```
                        quando viene ricevuto
32                    with open(file_path, 'wb') as f:
33                        while True:
34                            data = s.recv(1024)
35                            if not data:
36                                break
37                            f.write(data)
38
39                    print(f'File ricevuto e salvato con successo su {
                          file_path}')
40                    log_file.write(f"File ricevuto e salvato con
                          successo su {file_path}\n")
41                    output_messages += f'File ricevuto e salvato con
                          successo su {file_path}\n'
42        except Exception as e:
43            print(f"Errore durante la richiesta del file da {host}: {e}"
                  )
44            log_file.write(f"Errore durante la richiesta del file da {
                  host}: {e}\n")
45            output_messages += f"Errore durante la richiesta del file da
                  {host}: {e}\n"
46
47        self.file_request_raspberry_finished.emit(output_messages)
48
49 ###FILE REQUEST THREAD ###
50 def file_request_raspberry_button_clicked(self):
51        thread4 = threading.Thread(target=self.file_request_raspberry,
              args=(host,SAVE_PATH))
52        thread4.start()
53        self.file_request_raspberry_finished.connect(self.
              update_file_request_raspberry_status)
```

A.1.13. Update DB

```
1 ### UPDATE DB ####
2  db_update_finished = pyqtSignal(str)
3     def db_update(self):
4         print("Richiesta update db")
5         log_file.write("Richiesta update db\n")
6         global output_messages_db
7         output_messages_db = ""
8         # Connessione al database
9         conn = sqlite3.connect(DB_path)
10        cursor = conn.cursor()
```

```
11          # Elimina la tabella se esiste gi
12          cursor.execute("DROP TABLE IF EXISTS MyDataBase")
13
14          # Creazione della tabella
15          cursor.execute('''CREATE TABLE MyDataBase
16                              (ID INTEGER PRIMARY KEY AUTOINCREMENT, IP TEXT
                                    , position TEXT, date TEXT, hour TEXT,
                                    counts REAL, calibration_factor REAL, dose
                                    REAL)''')
17
18          # Verifica se la tabella    stata creata correttamente
19          cursor.execute("SELECT name FROM sqlite_master WHERE type='table
                ' AND name='MyDataBase'")
20          tabella_esiste = cursor.fetchone()
21
22          if tabella_esiste:
23              output_messages_db += f"La tabella 'MyDataBase'    stata
                    creata correttamente.\n"
24              log_file.write(f"La tabella 'MyDataBase'    stata creata
                    correttamente.\n")
25              print("La tabella 'MyDataBase'    stata creata correttamente
                    .")
26          else:
27              output_messages_db += f"Errore durante la creazione della
                    tabella 'MyDataBase'.\n"
28              log_file.write(f"Errore durante la creazione della tabella '
                    MyDataBase'.\n")
29              print("Errore durante la creazione della tabella 'MyDataBase
                    '.")
30
31          # Elenco dei file di tipo 1
32          Directory_File_Da_Raspberry = r'C:\Users\margh\Desktop\GUI\
                File_da_Raspberry'
33          Directory_File_Esterno = r'C:\Users\margh\Desktop\GUI\
                File_Esterno'
34          Files_Da_Raspberry = [file for file in os.listdir(
                Directory_File_Da_Raspberry) if file.endswith('.txt')]
35
36          # Itera su ciascun file di tipo 1 e inserisci i dati
                corrispondenti nel database
37          for File_Da_Inserire_Nel_DB in Files_Da_Raspberry:
38
39              with open(os.path.join(Directory_File_Da_Raspberry,
                    File_Da_Inserire_Nel_DB), 'r') as file1, open(os.path.
```

```
                join(Directory_File_Esterno,'file_esterno.txt'), 'r') as
                file2:
40               dati2 = {}  # Dizionario per memorizzare i dati dal file
                     di testo
41
42               # Leggi i dati dal secondo file e memorizzali nel
                    dizionario
43               for line in file2:
44                   IP, posizione, taratura = line.strip().split(',')
45                   dati2[IP] = (posizione, taratura)
46
47               #Leggo il nome del file, estraggo dal nome l'ip e lo uso
                     come collegamento con il file esterno
48               IP1 = os.path.basename(File_Da_Inserire_Nel_DB).split('_
                    ')
49               IP=IP1[2].rstrip('.txt')
50               print(F"{IP}")
51
52               # Leggi i dati dal primo file e associa i valori di
                    taratura e posizione utilizzando l'IP
53               for line in file1:
54                   data_str, ora_str, conteggi = line.strip().split(','
                        )
55                   posizione, taratura = dati2.get(IP, ('N/D', 'N/D'))
56
57                   # Converti la stringa data e ora in un oggetto
                        datetime
58                   data_ora_str = f"{data_str} {ora_str}"
59                   data_ora = datetime.strptime(data_ora_str, '%d-%m-%Y
                         %H:%M:%S')
60                   data_str = data_ora.strftime('%d-%m-%Y')
61                   ora_str = data_ora.strftime('%H:%M:%S')
62
63                   #calcolo la dose
64                   dose = float(taratura) * float(conteggi)
65
66                   # Esegui l'inserimento nel database
67                   cursor.execute("INSERT INTO MyDataBase (IP, position
                        , date, hour, counts, calibration_factor, dose)
                        VALUES (?, ?, ?, ?, ?, ?, ?)",
68                                  (IP, posizione, data_str, ora_str,
                                        float(conteggi), float(taratura),
                                        float(dose)))
69           conn.commit()
```

```
            output_messages_db += f"Dati del file '{File_Da_Inserire_Nel_DB}' inseriti con successo nel database.\n"
            print(f"Dati del file '{File_Da_Inserire_Nel_DB}' inseriti con successo nel database.")
            log_file.write(f"Dati del file '{File_Da_Inserire_Nel_DB}' inseriti con successo nel database.\n")
        self.db_update_finished.emit(output_messages_db)
        # Chiusura della connessione
        conn.close()

### DATABASE UPDATE THREAD ###
    def db_update_button_clicked(self):
        thread5 = threading.Thread(target=self.db_update, args=())
        thread5.start()
        self.db_update_finished.connect(self.update_db_update_status)
```

A.1.14. Show Data

```
### SHOW DATA ###
    def show_data(self):

        # Nome della tabella da estrarre
        table_name = 'MyDataBase'

        conn = sqlite3.connect(DB_path)

        # Esegui una query per estrarre i dati dalla tabella
        query = f"SELECT * FROM {table_name}"
        df = pd.read_sql_query(query, conn)

        conn.close()

        df.to_excel(excel_output_path, index=False)

        print(f"I dati sono stati convertiti con successo in {excel_output_path}")
        log_file.write(f"Dati del file '{log_file}' inseriti con successo nel database.\n")
        # Apri il file Excel
        os.startfile(excel_output_path)
        os.system(f"I dati sono stati convertiti con successo in {excel_output_path}")

```

```
### SHOW DATA THREAD ###
    def show_data_button_clicked(self):
        thread6 = threading.Thread(target=self.show_data, args=())
        thread6.start()
```

```
def initUI(self):
        ########PING#########
        self.create_ping_section()

        ########FILE ESTERNO##########
        self.create_file_section()

        #CREATE SECTION FOR IP SELECTION AND START BUTTON
        self.create_ip_and_start_section()

        #SHOW_COUNTS : Mostra i conteggi
        self.create_show_counts_secion()

        #STOP: Esegue lo stop del conteggio
        self.create_stop_section()

        ########CONNESSIONE A RASPBERRY#########
        self.create_raspberry_connection_section()

        #######RICHIESTA FILE RASPBERRY##########
        self.create_file_request_section()

        ########DB UPDATE########
        self.create_db_update_section()

        #SHOW DATA
        self.create_show_data_section()

        ######AUTOMATICA#####
        self.create_automatica_section()


#######CREAZIONE TASTI#########
    def openIPSelector(self):
        self.ip_selector = IPSelector(self)
        self.ip_selector.ipsSelected.connect(self.displaySelectedIPs)  #
            Collega il segnale alla funzione di visualizzazione
        self.ip_selector.exec_()

```

```
39      def displaySelectedIPs(self, ips):
40          # Salva gli IP selezionati nell'attributo della mainWindows
41          self.selected_ips = ips
42          self.ip_label.setText("Selected IPs: " + ", ".join(ips))
43
44      def create_ip_and_start_section(self):
45          # Sezione per selezionare l'IP e avviare il conteggio
46          top_layout = QHBoxLayout()
47
48          # Imposta i margini del layout (left, top, right, bottom)
49          top_layout.setContentsMargins(200, 5, 200, 5)  # Margini del
               layout,pixels
50          # Spazio fisso tra il pulsante e l'icona
51          spazio_fisso = QSpacerItem(5, 10, QSizePolicy.Fixed, QSizePolicy
               .Minimum)
52          top_layout.addSpacerItem(spazio_fisso)
53
54          # Pulsante info (cerchio)
55          self.info_button = QPushButton(self)
56          self.info_button.setIcon(QIcon(QPixmap(info_path)))
57          self.info_button.setIconSize(QSize(20, 20))  # Dimensione dell'
               icona
58          self.info_button.setFixedSize(30, 30)
59          self.info_button.setStyleSheet(
60              "background-color: white; "
61              "border-radius: 15px; "  # Crea l'effetto cerchio
62              "border: 3px solid rgba(96, 140, 165, 0.95); "
63              "padding: 5px;"
64          )
65
66          self.info_button.setToolTip("ehy")  # Aggiungi il tooltip
67          self.info_button.clicked.connect(lambda: self.show_info_message(
               "The IP Selector button: allows you to choose the detectors
               on which to begin measurements.\n Start: lets you initiate
               the measurements.\nWARNING!!!\nBefore starting the
               measurements, make sure to enter the measurement duration and
                the desired file name."))
68          top_layout.addWidget(self.info_button)
69
70          # Pulsante di selezione dell'IP
71          self.select_ip_button = QPushButton("Open IP Selector")
72          self.select_ip_button.setFont(QFont("Helvetica", 14))
73          self.select_ip_button.setStyleSheet("background-color: rgba(96,
               140, 165, 0.95); color:rgba(219,227,235,0.95); font-weight:
```

```
            bold;")
        self.select_ip_button.setFixedSize(300, 50)
        self.select_ip_button.clicked.connect(self.openIPSelector)
        top_layout.addWidget(self.select_ip_button)

        # Pulsante Start
        self.start_button = QPushButton('Start', self)
        self.start_button.setFont(QFont("Helvetica", 14))
        self.start_button.setStyleSheet("background-color: rgba(96, 140,
            165, 0.95); color:rgba(219,227,235,0.95); font-weight: bold;
            ")
        self.start_button.setFixedSize(300, 50)
        self.start_button.clicked.connect(self.send_data)
        #self.start_button.clicked.connect(self.open_new_window_counts)
        top_layout.addWidget(self.start_button)

        self.layout.addLayout(top_layout)

        # Sezione input per il tempo di misura e il filename
        input_layout = QVBoxLayout()
        self.label_max_count = QLabel('Enter measurement time:')
        self.line_edit_max_count = QLineEdit()

        self.label_filename = QLabel('Enter the file name:')
        self.line_edit_filename = QLineEdit()

        input_layout.addWidget(self.label_max_count)
        input_layout.addWidget(self.line_edit_max_count)
        input_layout.addWidget(self.label_filename)
        input_layout.addWidget(self.line_edit_filename)

        self.layout.addLayout(input_layout)

        # Aggiungi l'etichetta per visualizzare gli IP selezionati
        self.ip_label = QLabel("Please enter: the IPs to perform the
            measurement on, the measurement time, and the filename. ")
        self.ip_label.setFont(QFont("Helvetica", 12))
        self.layout.addWidget(self.ip_label)


    def create_raspberry_connection_section(self):
    # Layout orizzontale per il pulsante e l'icona di informazione
        top_layout = QHBoxLayout()
        # Imposta i margini del layout (left, top, right, bottom)
```

```
114         top_layout.setContentsMargins(200, 5, 200, 5)  # Margini del
                layout
115
116         # Spazio fisso tra il pulsante e l'icona (ad esempio, 20 pixel
                di larghezza)
117         spazio_fisso = QSpacerItem(5, 10, QSizePolicy.Fixed, QSizePolicy
                .Minimum)
118         top_layout.addSpacerItem(spazio_fisso)
119
120         # Pulsante info (cerchio)
121         self.info_button = QPushButton(self)
122         self.info_button.setIcon(QIcon(QPixmap(info_path)))
123         self.info_button.setIconSize(QSize(20, 20))  # Dimensione dell'
                icona
124         self.info_button.setFixedSize(30, 30)
125         self.info_button.setStyleSheet(
126             "background-color: white; "
127             "border-radius: 15px; "  # Crea l'effetto cerchio
128             "border: 3px solid rgba(96, 140, 165, 0.95); "
129             "padding: 5px;"
130         )
131         #self.info_button.setToolTip("we")
132         #self.info_button.clicked.connect(self.show_info_message)
133         # Usando la funzione show_info_message in altre funzioni
134         self.info_button.clicked.connect(lambda: self.show_info_message(
                "Connecting to Raspberry Pi: This allows you to establish a
                connection between your PC and the Raspberry Pi for file
                transfer.\nWARNING!!! \nAlways enable the connection"))
135
136         top_layout.addWidget(self.info_button)
137
138
139         # Pulsante di connessione a Raspberry
140         self.connessione_a_raspberry_button = QPushButton('Connecting to
                 Raspberry Pi', self)
141         self.connessione_a_raspberry_button.setFont(QFont("Helvetica",
                14))
142         self.connessione_a_raspberry_button.setStyleSheet("background-
                color: rgba(96, 140, 165, 0.95); color:rgba(219,227,235,0.95)
                ; font-weight: bold;")
143         self.connessione_a_raspberry_button.setFixedSize(395, 50)
144         self.connessione_a_raspberry_button.clicked.connect(self.
                connessione_a_raspberry_button_clicked)
145         top_layout.addWidget(self.connessione_a_raspberry_button)
```

```
146
147
148         # Rettangolo per la visualizzazione del messaggio di avvenuta
               connessione
149         self.connessione_a_raspberry_status_rect = QLabel("", self)
150         self.connessione_a_raspberry_status_rect.setStyleSheet("
               background-color: white; border: 3px solid rgba(96, 140, 165,
                0.95); font-weight: solid bold;")
151         self.connessione_a_raspberry_status_rect.setFixedSize(300, 50)
152         top_layout.addWidget(self.connessione_a_raspberry_status_rect)
153
154         # Aggiungi il layout al layout principale
155         self.layout.addLayout(top_layout)
156
157     def show_info_message(self,messaggio):
158         # Mostra un messaggio di informazione (self, titolo finestra,
               messaggio)
159         QMessageBox.information(self, "Information", messaggio)
160
161     def create_automatica_section(self):
162         top_layout = QHBoxLayout()
163         self.automatica_button = QPushButton('AUTOMATICO', self)
164         self.automatica_button.setFont(QFont("Helvetica", 14))
165         self.automatica_button.setStyleSheet("background-color: rgba(96,
                140, 165, 0.95); color:rgba(219,227,235,0.95); font-weight:
               bold;")
166         self.automatica_button.setFixedSize(300, 50)
167         self.automatica_button.clicked.connect(self.
               automatica_button_clicked)
168         top_layout.addWidget(self.automatica_button)
169
170         self.layout.addLayout(top_layout)
171
172     def create_ping_section(self):
173         top_layout = QHBoxLayout()
174
175         # Imposta i margini del layout (left, top, right, bottom)
176         top_layout.setContentsMargins(200, 5, 200, 5)  # Margini del
               layout
177         # Spazio fisso tra il pulsante e l'icona
178         spazio_fisso = QSpacerItem(5, 10, QSizePolicy.Fixed, QSizePolicy
               .Minimum)
179         top_layout.addSpacerItem(spazio_fisso)
180
```

```
        # Pulsante info (cerchio)
        self.info_button = QPushButton(self)
        self.info_button.setIcon(QIcon(QPixmap(info_path)))
        self.info_button.setIconSize(QSize(20, 20))  # Dimensione dell'icona
        self.info_button.setFixedSize(30, 30)
        self.info_button.setStyleSheet(
            "background-color: white; "
            "border-radius: 15px; "  # Crea l'effetto cerchio
            "border: 3px solid rgba(96, 140, 165, 0.95); "
            "padding: 5px;"
        )
        #self.info_button.setToolTip("ok")  # Aggiungi il tooltip
        self.info_button.clicked.connect(lambda: self.show_info_message("Ping: allows you to verify the connection between the PC and each connected device."))
        top_layout.addWidget(self.info_button)

        self.ping_button = QPushButton('Ping', self)
        self.ping_button.setFont(QFont("Helvetica", 14))
        self.ping_button.setStyleSheet("background-color: rgba(96, 140, 165, 0.95); color:rgba(219,227,235,0.95); font-weight: bold;")
        self.ping_button.setFixedSize(300, 50)
        self.ping_button.clicked.connect(self.ping_button_clicked)
        self.ping_button.clicked.connect(self.open_new_window)
        top_layout.addWidget(self.ping_button)


        self.ping_status_rect = QLabel("", self)
        self.ping_status_rect.setStyleSheet("background-color: white; border: 3px solid rgba(96, 140, 165, 0.95); font-weight: solid bold;")
        self.ping_status_rect.setFixedSize(300, 50)
        top_layout.addWidget(self.ping_status_rect)

        self.layout.addLayout(top_layout)

    def create_file_section(self):
        top_layout = QHBoxLayout()

         # Imposta i margini del layout (left, top, right, bottom)
        top_layout.setContentsMargins(200, 5, 200, 5) # Margini del layout
```

```
        # Spazio fisso tra il pulsante e l'icona
        spazio_fisso = QSpacerItem(5, 10, QSizePolicy.Fixed, QSizePolicy
            .Minimum)
        top_layout.addSpacerItem(spazio_fisso)

        # Pulsante info (cerchio)
        self.info_button = QPushButton(self)
        self.info_button.setIcon(QIcon(QPixmap(info_path)))
        self.info_button.setIconSize(QSize(20, 20))  # Dimensione dell'
            icona
        self.info_button.setFixedSize(30, 30)
        self.info_button.setStyleSheet(
            "background-color: white; "
            "border-radius: 15px; "  # Crea l'effetto cerchio
            "border: 3px solid rgba(96, 140, 165, 0.95); "
            "padding: 5px;"
        )

        self.info_button.setToolTip("ol")  # Aggiungi il tooltip
        self.info_button.clicked.connect(lambda: self.show_info_message(
            "External File: allows you to verify the presence on the PC
            of the file containing the IP addresses, location, and
            calibration factor of each individual detector."))
        top_layout.addWidget(self.info_button)


        self.check_file_in_directory_button = QPushButton('External File
            ', self)
        self.check_file_in_directory_button.setFont(QFont("Helvetica",
            14))
        self.check_file_in_directory_button.setStyleSheet("background-
            color: rgba(96, 140, 165, 0.95); color:rgba(219,227,235,0.95)
            ; font-weight: bold;")
        self.check_file_in_directory_button.setFixedSize(300, 50)
        self.check_file_in_directory_button.clicked.connect(self.
            check_file_in_directory_button_clicked)
        top_layout.addWidget(self.check_file_in_directory_button)

        self.check_file_in_directory_status_rect = QLabel("", self)
        self.check_file_in_directory_status_rect.setStyleSheet("
            background-color: white; border: 3px solid rgba(96, 140, 165,
             0.95); font-weight: solid bold;" )
        self.check_file_in_directory_status_rect.setFixedSize(300, 50)
        top_layout.addWidget(self.check_file_in_directory_status_rect)
```

```
249
250         self.layout.addLayout(top_layout)
251
252     def create_file_request_section(self):
253         input_layout = QVBoxLayout()
254         self.label_filename_RP = QLabel('Enter the name of the file you
               want to receive from the raspberry:')
255         self.line_edit_filename_RP = QLineEdit()
256
257         input_layout.addWidget(self.label_filename_RP)
258         input_layout.addWidget(self.line_edit_filename_RP)
259
260         self.layout.addLayout(input_layout)
261
262         top_layout = QHBoxLayout()
263
264         # Imposta i margini del layout (left, top, right, bottom)
265         top_layout.setContentsMargins(200, 5, 200, 5)  # Margini del
               layout
266         # Spazio fisso tra il pulsante e l'icona
267         spazio_fisso = QSpacerItem(5, 10, QSizePolicy.Fixed, QSizePolicy
               .Minimum)
268         top_layout.addSpacerItem(spazio_fisso)
269
270         # Pulsante info (cerchio)
271         self.info_button = QPushButton(self)
272         self.info_button.setIcon(QIcon(QPixmap(info_path)))
273         self.info_button.setIconSize(QSize(20, 20))  # Dimensione dell'
               icona
274         self.info_button.setFixedSize(30, 30)
275         self.info_button.setStyleSheet(
276             "background-color: white; "
277             "border-radius: 15px; "  # Crea l'effetto cerchio
278             "border: 3px solid rgba(96, 140, 165, 0.95); "
279             "padding: 5px;"
280         )
281         self.info_button.setToolTip("ehy")  # Aggiungi il tooltip
282         self.info_button.clicked.connect(lambda: self.show_info_message(
               "File Request: allows you to request any file from all
               connected devices. \nWARNING!!! \nRemember to enter the name
               of the file you wish to request."))
283         top_layout.addWidget(self.info_button)
284
285         self.file_request_raspberry_button = QPushButton('File Request',
```

```
            self)
286         self.file_request_raspberry_button.setFont(QFont("Helvetica",
                14))
287         self.file_request_raspberry_button.setStyleSheet("background-
                color: rgba(96, 140, 165, 0.95); color:rgba(219,227,235,0.95)
                ; font-weight: bold;")
288         self.file_request_raspberry_button.setFixedSize(300, 50)
289         self.file_request_raspberry_button.clicked.connect(self.
                file_request_raspberry_button_clicked)
290         top_layout.addWidget(self.file_request_raspberry_button)
291
292         self.file_request_raspberry_status_rect = QLabel("", self)
293         self.file_request_raspberry_status_rect.setStyleSheet("
                background-color: white; border: 3px solid rgba(96, 140, 165,
                 0.95); font-weight: solid bold;")
294         self.file_request_raspberry_status_rect.setFixedSize(300, 50)
295         top_layout.addWidget(self.file_request_raspberry_status_rect)
296
297         self.layout.addLayout(top_layout)
298
299     def create_db_update_section(self):
300         top_layout = QHBoxLayout()
301
302         # Imposta i margini del layout (left, top, right, bottom)
303         top_layout.setContentsMargins(200, 5, 200, 5)  # Margini del
                layout
304         # Spazio fisso tra il pulsante e l'icona
305         spazio_fisso = QSpacerItem(5, 10, QSizePolicy.Fixed, QSizePolicy
                .Minimum)
306         top_layout.addSpacerItem(spazio_fisso)
307
308         # Pulsante info (cerchio)
309         self.info_button = QPushButton(self)
310         self.info_button.setIcon(QIcon(QPixmap(info_path)))
311         self.info_button.setIconSize(QSize(20, 20))  # Dimensione dell'
                icona
312         self.info_button.setFixedSize(30, 30)
313         self.info_button.setStyleSheet(
314             "background-color: white; "
315             "border-radius: 15px; "  # Crea l'effetto cerchio
316             "border: 3px solid rgba(96, 140, 165, 0.95); "
317             "padding: 5px;"
318         )
319         self.info_button.setToolTip("ehy")  # Aggiungi il tooltip
```

```
320         self.info_button.clicked.connect(lambda: self.show_info_message(
                "Update DB: allows you to update the database"))
321         top_layout.addWidget(self.info_button)
322 
323         self.db_update_button = QPushButton('Update DB', self)
324         self.db_update_button.setFont(QFont("Helvetica", 14))
325         self.db_update_button.setStyleSheet("background-color: rgba(96,
                140, 165, 0.95); color:rgba(219,227,235,0.95); font-weight:
                bold;")
326         self.db_update_button.setFixedSize(300, 50)
327         self.db_update_button.clicked.connect(self.
                db_update_button_clicked)
328         top_layout.addWidget(self.db_update_button)
329 
330         self.db_update_status_rect = QLabel("", self)
331         self.db_update_status_rect.setStyleSheet("background-color:
                white; border: 3px solid rgba(96, 140, 165, 0.95); font-
                weight: solid bold;")
332         self.db_update_status_rect.setFixedSize(300, 50)
333         top_layout.addWidget(self.db_update_status_rect)
334 
335         self.layout.addLayout(top_layout)
336 
337     def create_show_data_section(self):
338         top_layout = QHBoxLayout()
339 
340         # Imposta i margini del layout (left, top, right, bottom)
341         top_layout.setContentsMargins(200, 5, 200, 5)  # Margini del
                layout
342         # Spazio fisso tra il pulsante e l'icona
343         spazio_fisso = QSpacerItem(5, 10, QSizePolicy.Fixed, QSizePolicy
                .Minimum)
344         top_layout.addSpacerItem(spazio_fisso)
345 
346         # Pulsante info (cerchio)
347         self.info_button = QPushButton(self)
348         self.info_button.setIcon(QIcon(QPixmap(info_path)))
349         self.info_button.setIconSize(QSize(20, 20))  # Dimensione dell'
                icona
350         self.info_button.setFixedSize(30, 30)
351         self.info_button.setStyleSheet(
352             "background-color: white; "
353             "border-radius: 15px; "  # Crea l'effetto cerchio
354             "border: 3px solid rgba(96, 140, 165, 0.95); "
```

```
355             "padding: 5px;"
356         )
357
358         self.info_button.setToolTip("ehy")  # Aggiungi il tooltip
359         self.info_button.clicked.connect(lambda: self.show_info_message(
                "Show data: allows you to see the database in an Excel format
                ."))
360         top_layout.addWidget(self.info_button)
361
362         self.show_data_button = QPushButton('Show data', self)
363         self.show_data_button.setFont(QFont("Helvetica", 14))
364         self.show_data_button.setStyleSheet("background-color: rgba(96,
                140, 165, 0.95); color:rgba(219,227,235,0.95); font-weight:
                bold;")
365         self.show_data_button.setFixedSize(300, 50)
366         self.show_data_button.clicked.connect(self.
                show_data_button_clicked)
367         top_layout.addWidget(self.show_data_button)
368
369         self.layout.addLayout(top_layout)
370
371     def create_stop_section(self):
372         top_layout = QHBoxLayout()
373
374         # Imposta i margini del layout (left, top, right, bottom)
375         top_layout.setContentsMargins(200, 5, 200, 5)  # Margini del
                layout
376         # Spazio fisso tra il pulsante e l'icona
377         spazio_fisso = QSpacerItem(5, 10, QSizePolicy.Fixed, QSizePolicy
                .Minimum)
378         top_layout.addSpacerItem(spazio_fisso)
379
380         # Pulsante info (cerchio)
381         self.info_button = QPushButton(self)
382         self.info_button.setIcon(QIcon(QPixmap(info_path)))
383         self.info_button.setIconSize(QSize(20, 20))  # Dimensione dell'
                icona
384         self.info_button.setFixedSize(30, 30)
385         self.info_button.setStyleSheet(
386             "background-color: white; "
387             "border-radius: 15px; "  # Crea l'effetto cerchio
388             "border: 3px solid rgba(96, 140, 165, 0.95); "
389             "padding: 5px;"
390         )
```

```
        self.info_button.setToolTip("ehy")  # Aggiungi il tooltip
        self.info_button.clicked.connect(lambda: self.show_info_message(
            "Stop: allows you to stop measurements at any time."))
        top_layout.addWidget(self.info_button)

        self.stop_button = QPushButton('Stop', self)
        self.stop_button.setFont(QFont("Helvetica", 14))
        self.stop_button.setStyleSheet("background-color: rgba(96, 140,
            165, 0.95); color:rgba(219,227,235,0.95); font-weight: bold;"
            )
        self.stop_button.setFixedSize(300, 50)
        self.stop_button.clicked.connect(self.stop)
        top_layout.addWidget(self.stop_button)
        self.layout.addLayout(top_layout)

    def create_show_counts_secion(self):
        top_layout = QHBoxLayout()

        # Imposta i margini del layout (left, top, right, bottom)
        top_layout.setContentsMargins(750, 5, 750, 5)  # Margini del
            layout
        # Spazio fisso tra il pulsante e l'icona
        spazio_fisso = QSpacerItem(5, 10, QSizePolicy.Fixed, QSizePolicy
            .Minimum)
        top_layout.addSpacerItem(spazio_fisso)

        # Pulsante info (cerchio)
        self.info_button = QPushButton(self)
        self.info_button.setIcon(QIcon(QPixmap(info_path)))
        self.info_button.setIconSize(QSize(20, 20))  # Dimensione dell'
            icona
        self.info_button.setFixedSize(30, 30)
        self.info_button.setStyleSheet(
            "background-color: white; "
            "border-radius: 15px; "  # Crea l'effetto cerchio
            "border: 3px solid rgba(96, 140, 165, 0.95); "
            "padding: 5px;"
        )

        self.info_button.setToolTip("ehy")  # Aggiungi il tooltip
        self.info_button.clicked.connect(lambda: self.show_info_message(
            "Show counts: allows you to see counts in real-time with an
            interval of 10 seconds."))
        top_layout.addWidget(self.info_button)
```

```
427
428         self.show_counts_button = QPushButton('Show counts', self)
429         self.show_counts_button.setFont(QFont("Helvetica", 14))
430         self.show_counts_button.setStyleSheet("background-color: rgba
            (96, 140, 165, 0.95); color:rgba(219,227,235,0.95); font-
            weight: bold;")
431         self.show_counts_button.setFixedSize(300, 50)
432         self.show_counts_button.clicked.connect(self.
            automatica_show_counts_button_clicked)
433         self.show_counts_button.clicked.connect(self.
            open_new_window_counts)
434         top_layout.addWidget(self.show_counts_button)
435         self.layout.addLayout(top_layout)
```

A.1.15. Server Raspberry Function Code

```
1  #!/usr/bin/env python3
2  import socket
3  import os
4
5  # Parametri del server
6  HOST = '0.0.0.0'
7  PORT = 12346      # Porta di ascolto
8  BASE_PATH = '/home/pi/Desktop/File_da_inviare_pc/'  # Percorso base dei
       file
9
10 # Creazione del socket TCP
11 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
12     s.bind((HOST, PORT))
13     s.listen()
14     print(f"Server in ascolto su {HOST}:{PORT}")
15     while True:
16         conn, addr = s.accept()
17         with conn:
18             print('Connessione stabilita con', addr)
19             # Ricevi il nome del file dal client
20             filename = conn.recv(1024).decode()
21             if not filename:
22                 print("Nome del file non ricevuto")
23                 conn.sendall(b"Errore: nome del file non ricevuto.")
24                 continue
25
26             FILE_PATH = os.path.join(BASE_PATH, filename)
27             print(f"Nome del file richiesto: {filename}")
```

```
            print(f"Percorso completo del file: {FILE_PATH}")

            if not os.path.isfile(FILE_PATH):
                print(f"Errore: il file {FILE_PATH} non esiste.")
                conn.sendall(b"Errore: file non trovato.")
                continue

            conn.sendall(b"OK")  # Notifica il client che il file
               pronto per essere inviato

            try:
                with open(FILE_PATH, 'rb') as f:
                    while True:
                        data = f.read(1024)  # Server legge il file in
                           blocchi di 1024 byte e li invia al client
                        if not data:
                            break
                        conn.sendall(data)
                print('File inviato con successo')
            except Exception as e:
                print(f"Errore durante l'invio del file: {e}")
```

A.2. C Code

A.2.1. Counting Program

```
 #include <stdio.h>
#include <stdlib.h>
#include <pigpio.h>
#include <time.h>
#include <unistd.h>
#include <string.h>
#include <math.h>
#include <signal.h>

int counter;
int timer;
int state;
int state_old;
int seconds;
char *max_count_char;
int max_count;
int old_counts, new_counts;
```

```
18 char *filename;
19 int i;
20 int step;
21 volatile int keep_running = 1;
22
23 int print_seconds[7] = {0};
24
25 //--------------------FUNCTION THAT PRINTS DATE, DAY, COUNTER
      -------------------//
26 void timer_print_save(void)
27 {
28     step = i + 1;
29
30     time_t t;
31     t = time(NULL);
32     struct tm tm;
33     tm = *localtime(&t);
34
35     FILE *fp;
36     fp = fopen(filename, "a");
37     fprintf(fp, "%d-%d-%d, %d:%d:%d ,%d\n", tm.tm_mday, tm.tm_mon + 1,
          tm.tm_year + 1900, tm.tm_hour, tm.tm_min, tm.tm_sec, counter);
38     fclose(fp);
39
40     // Aggiorno il timer
41     timer = timer + 1;
42 }
43
44 int main(int argc, char *argv[])
45 {
46     //-----------------INPUT PARAMETERS, FROM GUI-------------------//
47     max_count_char = argv[1];
48     max_count = atoi(argv[1]);
49     filename = argv[2];
50     printf("\n max_count = %d counter = %d \n", max_count, counter);
51
52     //----------------PARAMETERS---------------------//
53     counter = 0;
54     timer = 0;
55     seconds = 1;
56     old_counts = 0;
57     new_counts = 0;
58     state_old = 0;
59     i = 0;
```

```
    printf("%d", counter);

    //---------------- GPIO LIBRARY------------------//
    if (gpioInitialise() < 0)
    {
        return 1;
    }
    else
    {
        gpioSetMode(26, PI_INPUT);
    }


    //---------------COUNTING--------------//
    counter = 0;
    timer = 1;
    while (timer <= max_count && keep_running)
    {
        state = gpioRead(26);
        if (state_old == 0)
        {
            if (state == 1)
            {
                counter = counter + 1;
                state_old = 1;
            }
            else
            {
                state_old = 0;
            }
        }
        else
        {
            if (state == 0)
            {
                state_old = 0;
            }
        }

        gpioSetTimerFunc(0, 1000, timer_print_save); // ms
    }

    gpioTerminate();

```

```
    return 0;
}
```

A.2.2. Converting Program

```
#include <stdio.h>
#include <stdlib.h>
#define NUMBERS_PER_ROW 7

int main() {
    // Apri il file binario in modalit  lettura
    FILE *input_file = fopen("POS_IT15", "rb");
    if (input_file == NULL) {
        perror("Errore nell'apertura del file binario");
        return 1;
    }

    // Apri il file di testo in modalit  scrittura
    FILE *output_file = fopen("File_name.txt", "w");
    if (output_file == NULL) {
        perror("Errore nell'apertura del file di testo");
        fclose(input_file);
        return 1;
    }

    int print_seconds[NUMBERS_PER_ROW];

    // Leggi i numeri dal file binario e scrivi i decimali nel file di
       testo
    while (!feof(input_file) && fread(print_seconds, sizeof(int),
       NUMBERS_PER_ROW, input_file) == NUMBERS_PER_ROW) {
        fprintf(output_file, "%d-%d-%d %d:%d:%d %d\n",
                print_seconds[0], print_seconds[1], print_seconds[2],
                print_seconds[3], print_seconds[4], print_seconds[5],
                print_seconds[6]);
    }

    // Chiudi i file
    fclose(input_file);
    fclose(output_file);

    printf("Conversione completata con successo.\n");
    return 0;
}
```

B | Appendix B

B.1. Collected data during measurements in treatment room three

Device	Diode Number	Angle	Diode Diff. Counts	Cal. Coefficient $[nSv/counts]$	Dose [μSv]
W-1		-90°			0.097
W-1		-90°			0.061
Diodo	415	-70°	73	1.27	0.092
Diodo	415	-70°	64	1.27	0.081
Diodo	415	-70°	74	1.27	0.093
Diodo	415	-70°	71	1.27	0.090
Diodo	415	-70°	65	1.27	0.082
Diodo	415	-70°	65	1.27	0.082
Diodo	415	-70°	72	1.27	0.091
Diodo	1231	-60°	85	1.19	0.101
Diodo	1231	-60°	90	1.19	0.107
Diodo	1231	-60°	90	1.19	0.107
Diodo	1231	-60°	79	1.19	0.094
Diodo	1231	-60°	63	1.19	0.074
Diodo	1231	-60°	89	1.19	0.105
Diodo	1231	-60°	88	1.19	0.104
Diodo	305	-30°	171	1.18	0.201
Diodo	305	-30°	167	1.18	0.197
Diodo	305	-30°	183	1.18	0.215
Diodo	305	-30°	158	1.18	0.186
Diodo	305	-30°	151	1.18	0.178
Diodo	305	-30°	181	1.18	0.213
W-1		-30°			0.234
W-2		-30°			0.206
Diodo	1231	-15°	422	1.19	0.502
Diodo	1231	-15°	485	1.19	0.577

Table B.1: Results of the measurements on the head phantom from -90° to -15°.

Device	Diode Number	Angle	Diode Diff. Counts	Cal. Coefficient $[nSv/counts]$	Dose [μSv]
W-1		0°			3.22
W-1		0°			3.23
W-1		0°			3.19
Diodo	305	0°	1969	1.18	2.323
Diodo	305	0°	2009	1.18	2.370
W-2		0°			2.64
Diodo	305	15°	539	1.18	0.636
Diodo	305	15°	499	1.18	0.588
W-2		30°			0.218
W-2		30°			0.239
W-2		30°			0.217
Diodo	305	30°	180	1.18	0.212
W-1		30°			0.234
W-1		45°			0.144
W-2		60°			0.087
Diodo	1231	60°	80	1.19	0.095
Diodo	1231	60°	70	1.19	0.083
W-1		60°			0.085
W-1		70°			0.075
W-1		80°			0.060
W-2		90°			0.052
Diodo	415	90°	47	1.27	0.059
Diodo	415	90°	46	1.27	0.058
W-2		90°			0.134
W-2		90°			0.050

Table B.2: Results of the measurements on the head phantom from 0° to 90°.

Device	Diode Number	Angle	Diode Diff. Counts	Cal. Coefficient $[nSv/counts]$	Dose [μSv]
W-2		110°			0.049
Diodo	1231	110°	42	1.19	0.0499
Diodo	1231	110°	60	1.19	0.0714
W-2		120°			0.0456
Diodo	1231	120°	40	1.19	0.0476
Diodo	1231	120°	36	1.19	0.0428
Diodo	1231	120°	40	1.19	0.0476
Diodo	1231	120°	51	1.19	0.0606
Diodo	1231	120°	28	1.19	0.0333
Diodo	1231	120°	41	1.19	0.0487
W-2		130°			0.0497
Diodo	415	130°	38	1.27	0.0482
Diodo	415	130°	36	1.27	0.0457
W-2		140°			0.0481
Diodo	415	140°	36	1.27	0.0457
Diodo	415	140°	37	1.27	0.0469
Diodo	415	140°	33	1.27	0.0419
Diodo	415	140°	34	1.27	0.0431
Diodo	415	140°	31	1.27	0.0393
Diodo	415	140°	68	1.27	0.0863

Table B.3: Results of the measurements on the head phantom from 110° to 140°.

Device	Diode Number	Angle	Diode Diff. Counts	Cal. Coefficient [$nSv/counts$]	Dose [μSv]
W-1		-90°			0.482
Diodo	305	-60°	487	1.18	0.574
Diodo	305	-60°	560	1.18	0.660
Diodo	305	-60°	535	1.18	0.631
Diodo	305	-60°	550	1.18	0.649
Diodo	305	-60°	549	1.18	0.647
Diodo	305	-60°	510	1.18	0.601
Diodo	305	-60°	537	1.18	0.633
Diodo	305	-45°	880	1.18	1.038
Diodo	305	-45°	824	1.18	0.972
Diodo	305	-45°	829	1.18	0.978
Diodo	305	-45°	832	1.18	0.981
Diodo	305	-45°	814	1.18	0.960
W-1		-30°			1.97
Diodo	415	-15°	3189	1.18	3.763
Diodo	415	-15°	3096	1.18	3.653
Diodo	415	-15°	3159	1.18	3.727

Table B.4: Results of the measurements on the torso phantom from -90° to -15°.

Device	Diode Number	Angle	Diode Diff. Counts	Cal. Coefficient $[nSv/counts]$	Dose [μSv]
W-1		0°			46.8
W-1		0°			45.7
Diodo	1231	0°	25447	1.19	30.281
Diodo	1231	0°	25591	1.19	30.453
Diodo	1231	0°	25142	1.19	29.918
W-2		30°			1.73
W-2		30°			1.74
Diodo	1231	30°	1406	1.19	1.673
Diodo	1231	30°	1439	1.19	1.712
Diodo	1231	30°	1437	1.19	1.710
Diodo	1231	30°	1439	1.19	1.712
W-1		45°			1.19
W-2		60°			0.742
W-2		70°	0		0.557

Table B.5: Results of the measurements on the torso phantom from 0° to 70°.

Device	Diode Number	Angle	Diode Diff. Counts	Cal. Coefficient $[nSv/counts]$	Dose [μSv]
W-2		90°			0.413
Diodo	415	90°	276	1.27	0.350
Diodo	415	90°	332	1.27	0.421
Diodo	415	90°	281	1.27	0.356
Diodo	415	90°	292	1.27	0.370
W-2		100°			0.374
W-2		110°			0.336
Diodo	1231	120°	316	1.19	0.376
Diodo	1231	120°	323	1.19	0.384
Diodo	1231	120°	328	1.19	0.390
Diodo	1231	120°	314	1.19	0.373
Diodo	1231	120°	312	1.19	0.371
W-2		120°			0.335
W-2		130°			0.337
Diodo	415	140°	280	1.27	0.355
Diodo	415	140°	272	1.27	0.345
Diodo	415	140°	268	1.27	0.340
Diodo	415	140°	288	1.27	0.365
Diodo	415	140°	260	1.27	0.330
W-2		140°			0.339

Table B.6: Results of the measurements on the torso phantom from 90° to 140°.

C | Appendix C

C.1. The Monte Carlo code

The Monte Carlo method, which FLUKA relies upon, is a statistical technique used for simulating random processes. In particle physics and radiation transport, it involves simulating the interactions and paths of individual particles as they traverse through matter. The method repeatedly calculates random events—such as scattering, absorption, or decay—using probability distributions that represent the underlying physics. Over millions or billions of particle histories, these random events build up to provide a statistically accurate picture of the system being modeled.

C.1.1. The Monte Carlo code FLUKA

The FLUKA code is a general purpose Monte Carlo code for the interaction and transport of hadrons, heavy ions, and electromagnetic particles from few keV (or thermal energies for neutrons) to Cosmic Ray energies in arbitrary materials. It is built and maintained with the aim of including the best possible physical models in terms of completeness and precision,[40]. FLUKA operates by simulating the individual interactions of particles as they pass through matter. The code employs a comprehensive set of physical models to describe various processes, ensuring that the simulation is as realistic as possible. The following key components outline how FLUKA functions:

- **Geometry Definition:** Users can define complex geometries using a range of shapes and materials. This allows for the modeling of intricate experimental setups or environments, such as medical devices, particle detectors, or shielding configurations.
- **Particle Tracking:** Once the geometry is defined, FLUKA tracks particles as they propagate through the defined space. For each particle, the code calculates the probability of interactions based on the material properties and energy of the particle, employing Monte Carlo methods to determine the path and interactions.

- **Interaction Models:** FLUKA includes a variety of models to describe interactions, including:
  - *Hadronic Processes:* It simulates scattering and absorption processes for protons, neutrons, and heavy ions. Models account for both elastic and inelastic interactions, as well as nuclear reactions.
  - *Electromagnetic Processes:* The code accurately models ionization, Bremsstrahlung, and photon interactions, using well-established cross-section data to calculate the effects of these processes on the particles.
  - *Neutron Interactions:* Special models for neutron transport are implemented, allowing for detailed simulations of thermal and fast neutron behavior in various materials.
- **Scoring and Results:** After simulating numerous particle histories, FLUKA accumulates results for various quantities of interest, such as dose distributions, energy deposition, and particle flux. The scoring can be customized to provide specific outputs relevant to the user's research, whether in radiation therapy, shielding design, or radiation detection.
- **Post-Processing Tools:** FLUKA offers various tools for visualizing results, analyzing data, and generating reports. These tools facilitate the interpretation of simulation outputs and enable users to present their findings effectively.

FLUKA provides a user-friendly interface known as Flair, which utilizes a card-based system to streamline the setup and execution of simulations. Each card represents different functions, such as defining geometries, setting particle types, and specifying materials. This card-based approach simplifies the user experience, allowing users to easily configure simulations without extensive knowledge of the underlying code structure. All details regarding the cards are provided in the manual, [41]. By combining these components, FLUKA provides a powerful framework for researchers to simulate and analyze the behavior of particles in different materials.

List of Figures

1.1 Dose-effect curves for tumour local control and complication insurgence. . 4
1.2 Dose as function of the depth in water. 5
1.3 The layout of the underground floor. 7
1.4 The synchrotron. 8
1.5 Model of the compact layout of the CNAO synchrotron. 9
1.6 Model of the new buildings (in colours) in construction at CNAO. 9
1.7 Model of the proton therapy technology proposed by Hitachi. 10
1.8 Cross section for ^{7}Li(p,n) ^{10}Be reaction. 11
1.9 Beam Shaping Assembly design. 12
1.10 Mechanism of spallation reaction. 14
1.11 Geometry of CNAO implemented in FLUKA. 15
1.12 Beam loss map of the center for points from 1 to 29. 16
1.13 Neutron spectra generated inside the synchrotron room with and without magnets. 17
1.14 Photon spectra generated inside the synchrotron room with and without magnets. 18
1.15 Cross section versus neutron energy for some reactions of interest in neutron detection. [19] 19
1.16 Conversion coefficient from fluence to ambient dose equivalent values for neutrons. [20] 21
1.17 Absolute response functions (counts per unit fluence) for the following rem meters: WENDI-38, Hankins-NRD, Andersson Braun, Wendi-II , and cylindrical LINUS. [21] 22
1.18 Design of Andersson-Braun rem counter. 23
1.19 Design of the Wendi-II rem counter. 24
1.20 Longitudinal cross section of the cylindrical version of the rem counter LINUS. 24
1.21 Design of the LUPIN REM counter. 25
1.22 Internal structure of a DOMINO detector. 26

1.23 Deep microstructures backfilled with neutron-reactive material enhance both neutron absorption efficiency and reaction product detection efficiency in MSND®s. 27
1.24 Design of the moderator. 28
1.25 Plot of the linearity test of the rem counter. 29
1.26 Response function of the detector, expressed in $Sv{\cdot}cm^2$, as a function of the incident neutron energy. The black line indicates the fluence to ambient dose equivalent coefficients calculated by ICRP and Pelliccioni [32], the blue line represents the MSND rem counter results, and the green and red lines respectively indicate the LINUS and WENDI dose responses. 30

2.1 View of the old counting interface written with the python interface Tkinter. 34
2.2 Scheduled locations for the REM counters and switches on the basement level. 35
2.3 View of the new interface written with python. 38
2.4 Example of the information button. 38
2.5 Example of the ping window when no devices are properly connected. . . . 40
2.6 Example of the "File Esterno": from left: IP address, the Raspberry Pi's position and the calibration coefficient. 41
2.7 Ip Selector list. 43
2.8 Help message displayed to the user in the IP Selector window. 43
2.9 Example of the show counts function. 45
2.10 Example of the naming convention used for the files. 49
2.11 Example of a file generated by the counting program and received by the master PC. 50
2.12 Example of a Database created with SQL. 52
2.13 Example of a Database converted in Excel. 53
2.14 Example of the GUI folder structure. 54
2.15 Layout of source and detector positions for measurement setup. 55
2.16 Network Architecture for detector monitoring and control system. 56
2.17 Switch connected to the master PC and detectors via ethernet cables. . . . 56
2.18 FLUKA Simulation of the laboratory with the Am-Be source. 58
2.19 Ambient dose equivalent with radius. 59

3.1 Calibration room at CMI. 62
3.2 Plot of the linearity test of the REM counter; the shaded gray area represents the ideal response with a propagated uncertainty of 7%. 64

3.3 Cross sectional view of the irradiation area with the copper target under the concrete roof. 65
3.4 Cross sectional view of the irradiation area with the copper target under the iron roof. 66
3.5 Neutron fluence per lethargy unit in the reference positions CT8 (purple), CS2 (green) and IT4 (light blue). The values are normalized per incident beam particle on the copper target. 67
3.6 Positions available on the concrete top (CT). 68

4.1 Treatment Room three with the head phantom and the positioned detectors. 72
4.2 Angular dose distribution of MSND and WENDI detectors for torso phantom at 1 Gy proton irradiation. 75
4.3 Angular dose distribution of MSND and WENDI detectors for head phantom at 1 Gy proton irradiation. 75
4.4 Irradiation Setup: on the left the irradiated silicon-based detector. 77
4.5 Complete retup in the XPR room: an overview of equipment and errangement. 78

List of Tables

1.1 Reference neutron beam quality factors. [11] 10
1.2 List of ions with corresponding energies used in the simulations. 15
1.3 DOMINO V5.4 features. [28] 27

2.1 Results of the measurements conducted in the laboratory. 57

3.1 Results of the measurements and the evaluated calibration coefficients. . . 63
3.2 Results of the measurements. 64
3.3 Results of the measurements performed on the concrete top. 69
3.4 Results of the measurements performed on the iron top. 70

4.1 Results of the measurements on the head phantom. 74
4.2 Results of the measurements on the torso phantom. 74
4.3 Results of the first measurement with protons,with a total measurement duration of 1 hour. The angle increases from top to bottom. 79
4.4 Results of the second measurement with protons and carbon ions, with a total measurement duration of 1 hour and 40 minutes. The angle increases from top to bottom. 80
4.5 Results of the third measurement with carbon ions, with a total measurement duration of 1 hour and 45 minutes. The angle increases from top to bottom. 80
4.6 Results of the measurements conducted for four days. 81

B.1 Results of the measurements on the head phantom from -90° to -15°. . . . 126
B.2 Results of the measurements on the head phantom from 0° to 90°. 127
B.3 Results of the measurements on the head phantom from 110° to 140°. . . . 128
B.4 Results of the measurements on the torso phantom from -90° to -15°. . . . 129
B.5 Results of the measurements on the torso phantom from 0° to 70°. 130
B.6 Results of the measurements on the torso phantom from 90° to 140°. . . . 131