UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TRABAJO DE FIN DE GRAD0

Grado en Ingeniería Biomédica

DESARROLLO DEL SISTEMA DE CONTROL DEL MICROTRÓN DE LA UPC

Memoria y Anexos

Autor: Álvaro Igartua Aizpiri
Director: Youri A. Koubychine Merkulov
Co-Director: Raúl Benítez Iglesias

Convocatoria: Mayo 2019

Resum

Al llarg d'aquesta Memòria s'exposa el treball realitzat en la confecció del mòdul de control de dosi del microtró, un accelerador d'electrons, de la Universitat Politècnica de Catalunya (UPC) que, tal como està previst, s'utilitzarà per a experiments d'irradiació de diversos materials, en particular materials biològics, amb feix d'electrons. La funció d'aquest mòdul es calcular, controlar i monitoritzar de un mode segur la dosi o el temps, que es requereix per a la irradiació d'una mostra.

El desenvolupament d'aquest de este sistema està dividit en dues estructures, software i hardware. En les ambdues s'estudia la comunicació per diversos protocols utilitzant prototipus basats en Arduino i en llenguatge de programació C/C++, tant amb distints mòduls que composen el microtró, com amb una cambra d'ionització per a mesurar la dosi absorbida per una mostra en sessions de irradiació.

En la part software, s'ha desenvolupat un codi per a microcontroladors Arduino que permet la comunicació de l'operador del microtró amb el mòdul de control de dosi per al seguiment de sessions d'irradiació de mostres i s'han dut a termes proves del seu funcionament. A més, també s'ha programat la lectura mitjançant el protocol RS232 de la dosi mesurada prr la cambra d'ionització.

En la part hardware, s'ha desenvolupat una protoshield per a Arduino que proporciona la comunicació amb l'electròmetre de la cambra d'ionització i que permet manipular un relé del sistema de bloqueig (interlock) del microtró, implementant d'aquesta manera la seguretat eficient en la irradiació de materials.

Una vegada acabat el projecte, s'ha obtingut un mòdul capaç de comunicar-se per Ethernet i a través del port sèrie i de configurar i monitoritzar una sessió d'irradiació a partir de paràmetres introduïts per l'operador en funció temps, o de dosi, específics.

Resumen

A lo largo de esta Memoria se expone el trabajo realizado en la confección del módulo de control de dosis del microtrón, un acelerador de electrones, de la Universitat Politècnica de Catalunya (UPC) que, tal como está previsto, será utilizado para experimentos de irradiación de diversos materiales, en particular materiales biológicos, con el haz electrones. La función de este módulo es calcular, controlar y monitorizar de un modo seguro la dosis, o el tiempo, que se requiera para la irradiación de una muestra.

El desarrollo de este sistema está dividido en dos estructuras, software y hardware. En ambas se estudia la comunicación por diversos protocolos utilizando prototipos basados en Arduino y en lenguaje de programación C/C++, tanto con distintos módulos que componen el microtrón, como con una cámara de ionización para medir la dosis absorbida por una muestra en sesiones de irradiación.

En la parte software, se ha desarrollado un código para microcontroladores Arduino que permite la comunicación del operador del microtrón con el módulo de control de dosis para el seguimiento de sesiones de irradiación de muestras y se han realizado pruebas de su funcionamiento. Además, también se ha programado la lectura mediante el protocolo RS232 de la dosis medida por la cámara de ionización.

En la parte hardware, se ha desarrollado una protoshield para Arduino que proporciona la comunicación con el electrómetro de la cámara de ionización y que permite manipular un relé del sistema de bloqueo (interlock) del microtrón, implementando de este modo la seguridad eficiente en la irradiación de materiales.

Una vez acabado el proyecto, se ha obtenido un módulo del sistema de control del microtrón capaz de comunicarse por Ethernet y a través del puerto serie y de configurar y monitorizar una sesión de irradiación en base a parámetros introducidos por el operador en función de tiempo, o de dosis, específicos.

Abstract

In this Diploma thesis the work of construction of a dose control module of the race-track microtron, an electron accelerator developed at the Universitat Politècnica de Catalunya (UPC) is presented. As it is envisaged, it will be used for experiments of irradiation of different materials, in particular biomaterials, with an electron beam. The function of this module is to calculate, control and monitor in a safe way the dose of the time of sample irradiation.

The development this system divides in two sections, software and hardware. In both of them communication with various microtron systems and with an ionization chamber for measurements of the dose absorbed by a sample in irradiation sessions is studied. Different communication protocols are considered using prototypes based on Arduino y C/C++ programming language.

In the part of software, a code for Arduino microcontrollers that allows the operator to communicate with the dose control module to monitor irradiation sessions has been developed and tests of its operation have been carried out. Also, data reading of the dose measured by the ionization chamber via RS232 protocol has been programmed.

In the hardware part an Arduino protoshield has been developed. It provides communication with an electrometer of the ionization chamber and allows to manipulate an interlock relay, thus implementing efficient safety in the material irradiation.

As a result of the project, a microtron control system module able to communicate via Ethernet and via serial port and to configure and monitor an irradiation session based on parameters introduced by the operator as a function of specified time or dose.

Agradecimientos

Por un lado, de manera profesional, me gustaría agradecer la paciencia dedicada a este proyecto tanto del doctor Youri Kubyshin como del doctor Juan Antonio Romero; así como, la oportunidad de afrontar un reto diferente con el cual aprender y ganar experiencia en un sector que a día de hoy me entusiasma.

Por lo que respecta al terreno personal, me gustaría realmente hacerles sentir desde la distancia que nos separa el apoyo y cariño recibidos por mi madre y hermanos; ya que, desde Barcelona a Bilbao los kilómetros muchas veces se hacen muy largos.

Glosario

Baud Rate: Número de unidades de una señal por segundo.

Bootloader: Programa que permite programar el microcontrolador de la placa Arduino sin uso de hardware.

C: Velocidad de la luz en el vacío, se suele aproximar a 10^8 m/s.

CRT: Utiliza sus siglas en inglés *Cathodic Ray tube*. Se refiere a tubo de rayos catódicos.

ETSEIB: Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

EEBE: Escola d'Enginyeria Barcelona Est

INTE: Instituto de técnicas energéticas

IORT: *Intraoperative Radiotherapy*. Siglas en inglés de Radioterapia Intraoperatoria.

Linac: *Linear Accelerator*. Abreviatura en inglés de acelerador lineal.

MCD: Módulo de Control de Dosis.

Microcontrolador: Circuito integrado que contiene un microprocesador y lo mínimo para su funcionamiento.

Microprocesador: Circuito integrado especializado en la ejecución de operaciones matemáticas y la transferencia y, manipulación de información.

MSU: *Moscow State University***.** Siglas en inglés de la Universidad Estatal de Moscú.

Paridad: Códigos, o dígitos, añadidos en la transmisión de datos para detectar errores.

PING: Es una utilidad habitual en el diagnóstico de redes que comprueba el estado de uno o varios equipos.

Protoshield: Placa de circuitos modular.

PTW-UNIDOS: Cámara de ionización utilizada en el laboratorio de dosimetría de la ETSEIB.

PWM: Utiliza sus siglas en inglés *Pulse-width Modulation*. Señales que se pueden modular en función a su ciclo.

REPM: Utiliza sus siglas en inglés *Rare Earth Permanent Magnets*. Se refiere a un tipo de imán permanente.

RF: Radiofrecuencia.

RTM: Utiliza sus siglas en inglés *Race-Track Microtrón*. Se refiere al nombre completo del microtrón.

SINP: *Skobeltsyn Institute of Nuclear Physics*. Siglas en inglés del Instituto de Física Nuclear Skobeltsyn.

Solución: Conjunto de sketches que componen un programa de Arduino.

TCP/IP: Utiliza sus siglas en inglés *Transmission Control Protocol/Internet Protocol.* Se refiere a protocolos utilizados para el manejo de internet.

Telnet: Abreviatura en inglés de *Telecommunication Network.* Es el nombre un protocolo que permita acceder a otra máquina remotamente.

UPC: Universitat Politècnica de Catalunya.

Lista de tablas

Lista de imágenes

Índice

1. Prefacio

1.2. Origen del trabajo

El presente proyecto surge de la necesidad de incorporar un sistema de control sobre la dosis irradiada en el microtrón de la Universitat Politècnica de Catalunya (UPC). El microtrón tendrá como finalidad en un futuro irradiar con haces de electrones, o de rayos X, tanto materiales biológicos como no biológicos, para modificar su morfología y sus propiedades de cara al estudio de materiales; o por otro lado, para tratar tumores malignos, como es el caso de empleo en la radioterapia.

Es por ello, que un control de dosis seguro y eficiente garantizaría el correcto funcionamiento y una adquisición precisa de los datos en los distintos conceptos de sesión que ofrecería el microtrón de la UPC.

1.3. Motivación

A la hora de comenzar este trabajo, hubo diversas fuentes de motivación. Para empezar, ampliar los conocimientos sobre electrónica y programación en el ámbito de la medicina nuclear era un gran reto personal; puesto que, mis esperanzas de algún día poder trabajar en ese sector han ido incrementándose desede que empecé los estudios en Barcelona. Por otro lado, trabajar al lado de un equipo multidisciplinar me generaba grandes espectativas y la posibilidad de ganar experiencia.

Además, de esta manera, podía completar mis estudios de grado universitario en un proyecto con intenciones de salir al mercado empresarial en un horizonte relativamente cercano.

2. Introducción

2.1. Objetivos del proyecto

El objetivo de este trabajo de fin de grado (TFG) es desarrollar un módulo del sistema de control para monitorizar la dosis de irradiación del microtrón, un acelerador en construcción por el Instituto de Técnicas Energéticas de la UPC.

Los objetivos específicos del proyecto son los siguientes:

- ✓ Programar en lenguaje C/C++ un sistema de comunicación entre operador y un microcontrolador de tipo Arduino para la ejecución de comandos de control.
- ✓ Programar en lenguaje C/C++ un sistema de comunicación por medio de RS232 entre la cámara de ionización y Arduino.
- ✓ Desarrollar el hardware específico para llevar a cabo la comunicación entre Arduino y la cámara de ionización por medio de RS232.
- ✓ Desarrollar el hardware específico para el control de seguridad del microtrón, por medio del accionamiento de un relé electromecánico.
- ✓ Programar el entorno de comunicación entre el operador y Arduino por medio de puerto serie.
- ✓ Programar el entorno de comunicación entre el operador y Arduino por medio del protocolo Ethernet.
- ✓ Comprender y validar fórmulas para el cálculo de la dosis y el tiempo, en tiempo de ejecución.
- ✓ Comprender y validar la comunicación RS232 con la cámara de ionización para el control de la dosis y el tiempo, en tiempo de ejecución.

2.2. Alcance del proyecto

Este proyecto agrega la capacidad de controlar y monitorizar la emisión de radiaciones ionizantes en el microtrón, permitiendo la adquisición de datos detallados y de una trazabilidad en ejecución de las pruebas realizadas. Pese a que el proyecto no pretende cumplir con la exigente normativa de aplicación del microtrón con fines médicos, cubre de manera económica y fiable la monitorización y fijación de las dosis en distintos ámbitos. En el marco de este proyecto no está previsto realizar pruebas con la radiación o un haz de electrones generado por el microtrón ni pruebas de funcionamiento del módulo de control de dosis en conjunto con otros microcontroladores del sistema de control del acelerador. El proyecto se dedica al diseño e implementación de los protocolos de comunicación y la programación de secuencias de control de seguridad del equipo de dosimetría de radiaciones utilizando un dispositivo Arduino.

3. Aceleradores y sus aplicaciones

3.1. Aceleradores de electrones de medianas energías

Un acelerador es un equipo que incrementa la energía de las partículas subatómicas cargadas al pasar por una zona de aceleración, incrementando su velocidad, hasta llegar a velocidades relativistas; es decir, próximas a la velocidad de la luz. Hoy en día, su uso se ha vuelto habitual; puesto que, hay hospitales, clínicas privadas y empresas que cuentan con alguna gama de acelerador; además, en el último decalustro, se ha convertido en el referente dentro de la radioterapia médica con el objetivo del tratamiento de tumores malignos. En la industria, por lo general, se utilizan para el analizar mercancías en puertos y aeropuertos; también, para determinar la composición de los alimentos o analizar proteínas para fabricar fármacos [1]. Por otro lado, los aceleradores también se utilizan en investigación y, en el desarrollo de tecnologías; por ejemplo, en un análisis de alimentos para obtener información relevante de su composición, y en distintos procesos dentro del ámbito de los materiales, y biomateriales. Dentro de los hospitales y clínicas, su uso está dirigido generalmente a la medicina nuclear y el radiodiagnóstico; además, de la radioterapia.

En el campo de los materiales tiene diversas aplicaciones al tener la capacidad de modificar la morfología de los materiales y, de esta manera, alterar las propiedades de esos materiales. Por ejemplo, al irradiar ciertos tipos de polímeros, dependiendo de la dosis empleada, se pueden obtener escisiones, injertos o reticulaciones en la estructura del polímero irradiado; e incluso aplicando dosis relativamente bajas, modificar el polímero para conseguir una mejor adherencia a geles o recubrimientos superficiales [2]. Esos procesos empleados sobre placas de estudios con proteínas, ensayos inmunológicos y cultivos celulares permiten expandir las posibilidades en distintos campos de investigación al poseer una mayor adherencia sobre la placa.

Por otro lado, los aceleradores se utilizan en la radioquímica también. Su utilización en el radiodiagnóstico se basa en la aplicación de los distintos tipos de radiación en dosis controladas; de esta manera, mediante el análisis de sangre y procesamiento de imagen [3], se logran detectar enfermedades incluso antes de que se expongan por sintomatología en el paciente y; por lo tanto, pudiendo tratar al individuo de forma preventiva.

En el dominio público, pese a que la aparición los aceleradores es relativamente cercana, prácticamente toda la población europea habrá contado con un modelo conceptual simplificado en casa. Esto se debe a que, sin ir más lejos, el primer televisor sacado al mercado sin contar con ningún elemento mecánico para la generación de imagen, de la marca alemana "Telefunken" en 1934, se basaba en un tubo de rayos catódicos (CRT); el cual, puede servir de ejemplo para hacer un acercamiento al funcionamiento de un acelerador y entender los procesos que lo comprenden. La explicación básica de un CTR plantea que, desde un cátodo calentado se genera un flujo de partículas cargadas que se aceleran y llegan a la pantalla; la cual, es un ánodo (en el caso de un televisor, sería de fósforo) con una protección para la emisión de rayos hacia el espectador. El fósforo permitiría la visualización del haz de rayos catódicos que se haya generado, mostrándole al espectador imágenes en blanco y negro [4]. El esquema de un CRT se muestra en la siguiente imagen (Fig. 3.1).

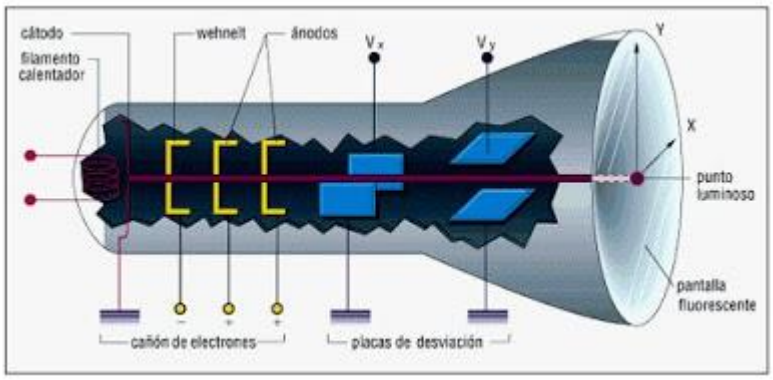


Fig. 3.1: *Tubo de rayos catódicos*

La evolución de los aceleradores ha sido constante desde sus inicios; en los cuales, a principios del siglo XX, los aceleradores de partículas se basaban en la aplicación de voltaje sobre partículas cargadas; por lo tanto, la energía alcanzable en ese momento era únicamente el producto de la carga de la partícula por el voltaje aplicado. Esto acarreaba un problema grave; debido a que, tras la aplicación de voltaje de alta magnitud el sistema generaba una ruptura dieléctrica, descargándose en el medio e impidiendo alcanzar altos voltajes y energías en su ejecución. Por ello, hubo otras investigaciones para obtener una alternativa válida entre las cuales destacó la idea del uso de un voltaje alternante de Gustav Ising.

Más tarde, Rolf Wideroe en 1928 logró llevar a cabo la idea de Gustav Ising, haciendo realidad la posibilidad de tener un voltaje alternante como pilar en la evolución energética de los aceleradores. Este modelo de acelerador se compone de un número establecido de tubos cilíndricos. Los tubos alternos se conectan entre si formando una diferencia de potencial oscilante entre los dos conjuntos y generando este efecto sobre el sistema. De esta forma, esta diferencia de potencial generada entre tubos es la que da la aceleración a las partículas cargadas haciendo que vayan de un tubo a otro; además, los tubos actúan de caja de Faraday, aislando a las partículas del campo eléctrico y, de este

modo, no están sometidas a aceleración hasta emerger al otro lado de los tubos. Cada vez que hay un empuje (una salida del tubo), los tubos han de alargarse; puesto que, la velocidad de las partículas se incrementará y es necesaria una sincronía entre el tiempo constante que deben tardar en recorrer cada uno de los tubos y la oscilación de los pulsos eléctricos. No obstante, en 1931 Ernest O. Lawrence, con la ayuda de David H. Sloan, obtuvo una energía de más de 1.2MeV en la Universidad de California; para ello, empleó campos de alta frecuencia para acelerar iones de mercurio, logrando así desbancar la anterior marca obtenida por Wideroe, al modificar su modelo confeccionando el primer ciclotrón [5].

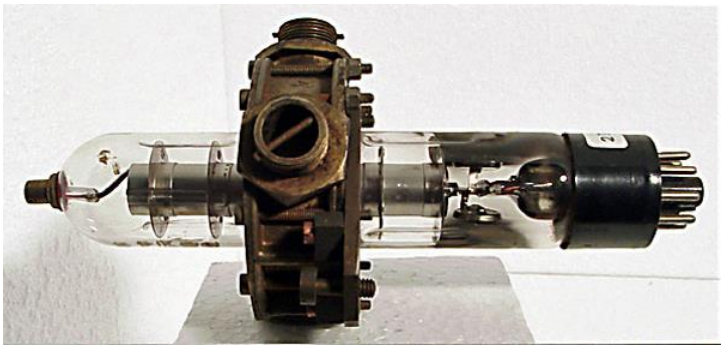


Fig. 3.2: *Klistrón*

En 1937 los hermanos Rusell y Sigurd Varian revolucionaron los aceleradores con su invento: el klistrón (Fig. 3.2) [6]. Este aparato es una válvula de vacío de electrones; la cual, se puede utilizar como un oscilador y amplificador. En la etapa final de este aparato, puede generar ondas electromagnéticas de una frecuencia comprendida entre 300MHz y 3GHz (microondas), estas ondas se incluyen en las bandas de la radiofrecuencia (RF) [7]; además, dependiendo del número de cavidades que tenga, puede utilizarse para amplificación de alta potencia, con frecuencias de hasta 200GHz.

Más adelante, entre las décadas de los 60 y 80 hubo avances; aunque, ciertamente, el concepto y los usos solamente han sido optimizados, llegando a día de hoy a niveles energéticos bastante más altos, sobre todo en el campo de la investigación.

En la actualidad, hay dos clases de aceleradores de electrones de utilización habitual: lineales y circulares. Los aceleradores circulares pueden ser de diferentes diseños: ciclotrones, sincrotrones, microtrones y betatrones; sin embargo, los principales son los ciclotrones y los sincrotrones. Los ciclotrones (Fig. 3.3) son los primeros modelos construidos, en ellos se aplica un campo eléctrico para acelerar a las partículas y un campo magnético constante que hace que las partículas giren.

El ciclotrón consiste en dos *Des*, llamadas así por su forma, que son las cámaras semicirculares huecas situadas en un campo magnético uniforme, orientado perpendicularmente a las placas, creado por un imán. Tras hacer el vacío en el interior de esta estructura, se aplica un voltaje alterno entre las cámaras. Debido a esta combinación, durante un semiciclo, el campo eléctrico acelera los iones, que recorren una trayectoria circular debido al campo magnético [8].

Mientras que, los sincrotrones, estructura de la cual se habla en el apartado 3 de este capítulo, son más modernos y ambos campos se hacen variar para llegar a energías más elevadas (hasta el orden de TeV).

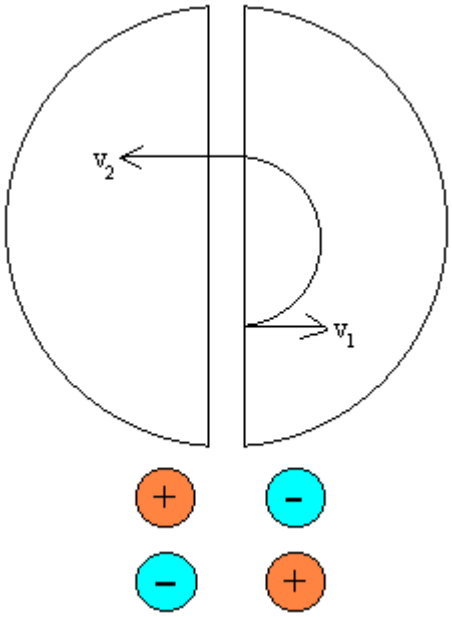


Fig. 3.3: *Ciclotrón*

El microtrón de pista (race-track microtron, RTM) está mejor adaptado para interactuar con partículas ligeras como los electrones. En el caso de este concepto de acelerador, se utilizan cavidades de aceleración, semejantes a la de un acelerador lineal. Incluye dos imanes semicirculares de retorno que hacen que al haz de electrones dar vueltas y pasar varias veces por la estructura de aceleración. El microtrón está ideado para funcionar a una frecuencia de campo constante y a un campo magnético en el límite ultra relativista. Una descripción más detallada de su funcionamiento se presenta en el capítulo 4.

3.2. Aplicaciones médicas de los aceleradores

Entre sus aplicaciones, la medicina es una de las más extendidas; en su momento, los primeros aceleradores utilizados para la radioterapia fueron los linacs electrostáticos con fuente de alta tensión como el generador de Van de Graaff (Fig. 3.4). Una esfera metálica hueca que llegaba a crear grandes diferenciales de potencial gracias a la electrostática, que tuvo su aplicación médica llegando a alcanzar los 5 MeV, permitiendo generar rayos X (7).

Conforme los aceleradores avanzaban, la medicina fue adaptando sus ventajas a tratamientos y metodologías con las que enfrentarse a distintas enfermedades; de este modo, tomando forma su función en la radioterapia. No obstante, hasta día de hoy, no ha habido ningún otro equipo médico que supere en utilización y efectividad a los aceleradores, en el campo de la radioterapia. De este modo, han ido monopolizando una sección crucial en la lucha oncológica y en los tratamientos paliativos habituales.

Fig. 3.4: *Generador de Van Graff*

A lo largo del crecimiento y emprendimiento dentro del ámbito de la ingeniería biomédica, se desarrollaron dos empresas que, en la actualidad, constituyen casi en su totalidad el negocio de la fabricación de aceleradores utilizados en la medicina para tratamientos con electrones y rayos X. Las dos empresas de referencia son Varian y Elekta. Un nuevo concepto de tratamiento comenzó a utilizarse en 1954, cuando se emplearon por primera vez partículas pesadas para tratar tumores. Es cierto, que hay distinciones entre la eficacia y necesidad de la diferencia entre aceleradores de protones, y aceleradores de electrones; sin embargo, el modo de utilización es muy similar. El uso actual más común es la radioterapia externa o de haz externo, donde el paciente recibe una dosis de radiación en puntos específicos y exhaustivamente calibrada para la eliminación, o reducción, de células tumorales [9]. En la actualidad, para la radioterapia externa se utilizan haces de radiación X de entre 1,5 y 25MV, o también, haces de electrones [10]. Se trata de un método indoloro en el momento

de la práctica, que deja bastantes secuelas sobre el paciente en los días posteriores al tratamiento; aunque, está entre las soluciones más eficaces; puesto que, en pacientes con tumores muy localizados, la tasa de curación alcanza entre un 80-90% y, en el caso de pacientes con una enfermedad más avanzada ayuda a controlar el avance en el crecimiento del tumor; por lo tanto, siempre se valora su aplicación en pacientes oncológicos [11]. Por otro lado, en la medicina nuclear también se emplean los aceleradores para producción de radioisótopos de tipo radiofármacos que se utilizan para diagnósticos por imagen al utilizar las diversas técnicas de detección de partículas. De esta manera, se desarrollaron técnicas como la imagen PET entre otros.

La imagen PET, o tomografía por emisión de positrones, se basa en la detección de dos fotones producidos en la aniquilación de un positrón (e^+) emitido por los radiofármacos al colisionar con un electrón cortical (e^-) del cuerpo del paciente; de esta manera, se obtienen imágenes de la actividad metabólica de los distintos órganos del cuerpo humano [12]. Para poder emplear esta técnica de imagen, es necesario generar radioisótopos y, para este cometido se suele emplear un ciclotrón. El PET, es un método no invasivo y denominado "in vivo"; esto se debe, a la introducción del radiofármaco de vida media ultracorta para el estudio de su distribución tridimensional.

De cara al tratamiento mediante el uso de la radioterapia es donde más destaca la utilización de los aceleradores, generalmente los lineales. La física empleada en un acelerador lineal para la generación de rayos X es la siguiente; para empezar, el cátodo se calienta mediante una corriente elevada hasta que los electrones de los orbitales más externos se ven ionizados. Después, mediante un campo eléctrico se aceleran hasta colisionar contra el ánodo, pasando primero por una guía de ondas; la cual, posee un campo magnético de alta frecuencia y alta potencia. La colisión de los electrones contra el metal genera fotones que se encuentran en el ancho de banda de los rayos X. Este fenómeno para la generación de rayos X, se denomina radiación de frenado, o Bremsstrahlung y, depende del voltaje aplicado para la aceleración y del material que constituya al ánodo; además, teniendo en cuenta que, la eficiencia de adquisición de rayos es solo del 1%, el ánodo tiene que ser refrigerado constantemente. Debido al tamaño que debería tener la guía para obtener energías elevadas, se emplea un potente electroimán deflector para orientar el haz hacía el paciente. A la salida de los electrones del electroimán, se encuentra el blanco empleado, que será el que haga de objetivo para la generación de rayos X; este, debe ser de un alto número atómico para aumentar la radiación de frenado, y debe aguantar las altas temperaturas [13].

Para el tratamiento, el paciente se encuentra sobre una camilla móvil (generalmente constituida por fibra de carbono), y se utilizan rayos láser para obtener la información espacial del paciente. De este modo, se asegura que cada vez que el sujeto se vea expuesto a la radiación, se encuentre en la misma posición desde que se inició el tratamiento. Además de los láseres, muchos modelos, emiten rayos X para hacer una verificación más precisa. La estructura de soporte que orienta el cabezal del acelerador

se denomina gantry, este elemento es móvil también; por lo tanto, la posibilidad de mover la camilla y el gantry, hacen muy cómodo el posicionamiento para el tratamiento; puesto que, el objetivo es

Fig. 3.5: *Radioterapia de haz externo*

irradiar únicamente los tejidos dañados, y no los periféricos. En la figura 5 (Fig. 3.5), se puede observar cómo se lleva a cabo el estudio de posicionamiento antes de realizar un tratamiento de radioterapia a un paciente oncológico, cuyo tumor se haya ubicado en la parte central del abdomen.

3.3. Aplicaciones en la industria y bioingeniería.

Algunos de los aceleradores más importantes, están dirigidos al campo de la investigación; este es el caso, del ALBA Synchrotron. Este inmenso acelerador de partículas, generalmente utilizado para la irradiación de rayos X en distintas pruebas, sobre todo en materiales, se encuentra en la provincia catalana de Barcelona, cerca de la universidad autónoma de Barcelona (UAB). Siendo el más importante del área del Mediterráneo, el sincrotrón ALBA (Fig. 3.6) se trata de un complejo de aceleradores de electrones diseñados para producir luz de sincrotrón; mediante ellos, se puede visualizar la estructura atómica y molecular de los materiales expuestos a estudio. La energía emitida por el haz de electrones generado por el ALBA es de 3GeV [14], para alcanzar esos altos rangos energéticos se combinan un LINAC y un propulsor de baja emitancia ($W \cdot m^{-2}$); es decir, la potencia emitida por unidad de superficie de la fuente radiante, y máxima potencia colocado en túnel que alberga el anillo de almacenamiento.

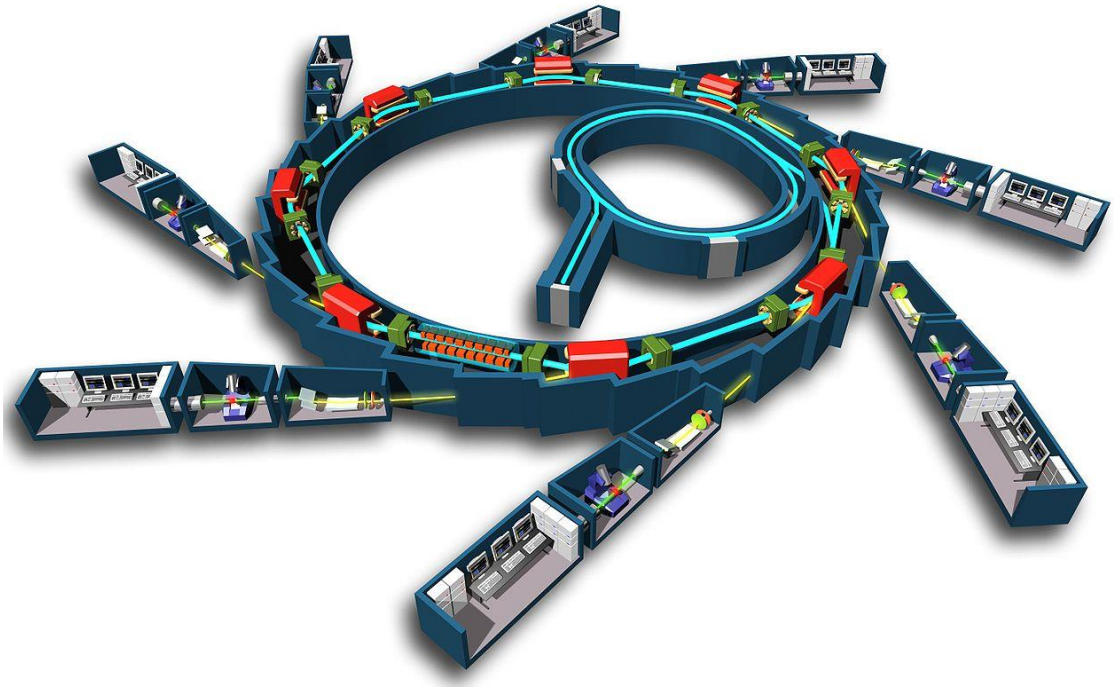


Fig. 3.6: *Esquema del sincrotrón ALBA.*

Las dimensiones del sincrotrón abarcan un perímetro 270 metros y 17 tramos rectos donde instalar dispositivos de inserción. Dentro de las diferentes doctrinas para las cuales se puede emplear el ALBA como un motor en el campo de la investigación, están las biociencias, la materia condensada y la ciencia de los materiales.

Como ejemplo, la física de la materia condensada es el campo de la física que se ocupa de las características físicas macroscópicas de la materia. En concreto, está enfocada a las fases condensadas; es decir, la fase en la cual el número de elementos de un sistema sea extremadamente grande y las interacciones entre los componentes sean fuertes. Un ejemplo clásico es el de los sólidos y los líquidos, en el caso de la investigación del sincrotrón ALBA está dirigido a la nanociencia y las propiedades magnéticas y electrónicas.

Por otro lado, dentro del ámbito industrial de los aceleradores, como por ejemplo la esterilización de alimentos o instrumentos médicos, la radiografía industria, irradiación de materiales para la mejora de sus características o implantación de iones, la arqueología, paleoantropología, etc. Esto se debe a la técnica de la espectrometría de masas con aceleradores (AMS); en la cual, se utiliza un acelerador y su sistema de transporte de haces como instrumento ultrasensible que puede detectar las abundancias isotópicas de radioisótopos de periodo largo en muestras pequeñas (Fig. 3.7). En la mayoría de las mediciones en las que se pretende datar por radiocarbono, se emplea la AMS; puesto que, provee una sensibilidad más alta que la prueba por recuento de desintegraciones del carbono 14 [15].

Fig. 3.7: Taller de espectrometría en masas con aceleradores

3.4. Radiaciones ionizantes y dosimetría

La radiación no es otra cosa que la emisión, propagación y transferencia de energía en forma de ondas electromagnéticas o partículas subatómicas a través del vacío o un medio material [16]. En el caso de los aceleradores la naturaleza de la radiación es electromagnética; es decir, no precisa de un medio conductor; por lo tanto, se requiere la utilización de materiales especiales para evitar las exposiciones indeseadas a esta fuente nociva para el ser humano (Fig. 3.8).

La radiación utilizada en el tratamiento oncológico es de carácter ionizante. Esto es que, pese a que el origen de las radiaciones ionizantes de naturaleza electromagnética es fundamentalmente el mismo que el de las no ionizantes, se dan a energías bastante más elevadas. Para aclarar esto se ha de expresar que, los iones se producen cuando la radiación tiene energía suficiente para arrancar los electrones de los orbitales de los átomos que se encuentran en el material radiactivo; de esta manera, los átomos quedan con un exceso de carga eléctrica. Por lo tanto, cuando se da este fenómeno, se clasifica la radiación como ionizante, si la radiación no es lo suficientemente radioactiva, se denomina no ionizante.

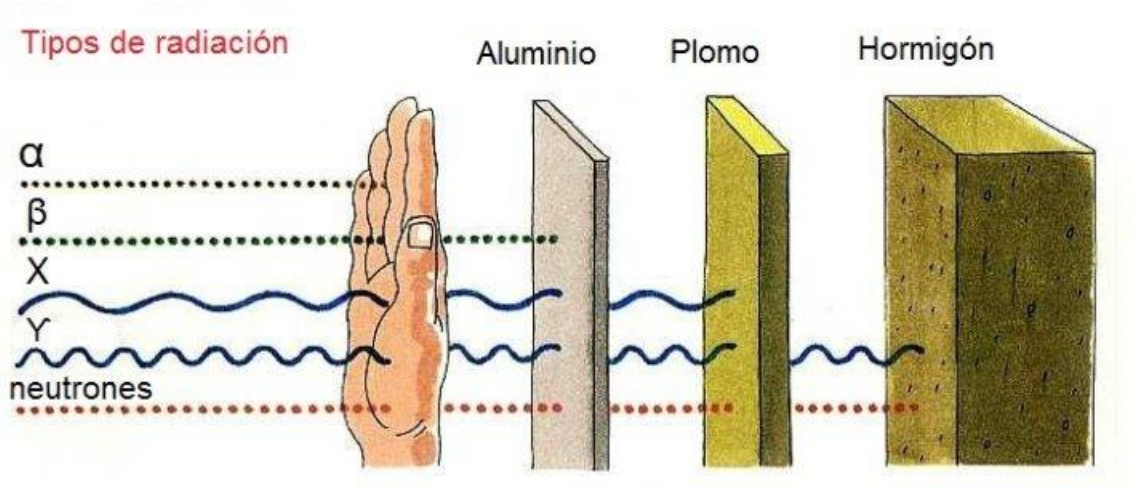


Fig. 3.8: *Interacción de las radiaciones ionizantes con la materia.*

Dentro de las emisiones de origen atómico más frecuentes están las siguientes:

- Radiación o desintegración alfa (α):

Son núcleos de He (helio) con carga positiva que poseen energías muy altas que; por otro lado, se contrapone con su baja capacidad de penetración. A continuación, se expone el esquema de la desintegración alfa.

$$ {}^{A}_{Z}X \rightarrow {}^{A-4}_{Z-2}Y + {}^{4}_{2}He^{2+} = {}^{A-4}_{Z-2}Y + \alpha $$

- Radiación beta (β-):

Se generan debido a la transformación de un neutrón en un protón y un electrón, en concreto son los electrones emitidos desde el núcleo del átomo a consecuencia de ese suceso. Además, tienen una capacidad de penetración más elevada que la radiación alfa.

$$ n \rightarrow p^{+} + e^{-} + \bar{\nu}e $$

- Radiación (β+):

En este caso es la emisión de un positrón, generado en la conversión de un protón en un neutrón y un positrón. Tienen un poder de penetración mayor que las partículas alfa; no obstante, cuentan con un nivel energético menor.

$$ p^{+} \rightarrow n + e^{+} + \nu e $$

- Radiación gamma (γ):

Tiene una capacidad de penetración mayor que las radiaciones beta y alfa; por lo tanto, son más difíciles de controlar. En sí, son radiaciones electromagnéticas procedentes del núcleo atómico.

$$Ni^* \rightarrow Ni + \gamma$$

- Rayos X:

Pese a que también son de naturaleza electromagnética, se originan al colisionar un haz de electrones acelerado contra los átomos de un material que haga de blanco; como puede ser, el tungsteno. En este choque, los electrones se frenan, al perder parte de su energía; entonces, una parte de esa energía pérdida se transforma en calor y la otra se emite en forma de rayos X [17]. Este tipo de radiación también es conocida como radiación de frenado o "Bremmsstrahlung". Para absorber los rayos X, son necesarios apantallamientos especiales de grosor elevado.

- Radiación de neutrones:

Se debe a partículas sin carga y una gran capacidad de penetración; generalmente, este tipo de radiación se da en los reactores nucleares y aceleradores de energías de haz superiores a 10 MeV.

Los efectos que la radiación ionizante puede causar son diversos; por supuesto, al margen de la capacidad para tratar tumores, también tiene efectos nocivos sobre la salud humana; los cuales, se exponen a continuación. A pesar de que las lesiones que se generan por radiaciones de naturalezas no corpusculares; es decir, rayos X y gamma, son menos lesivas y tienen mayor facilidad de reparación por parte del ser humano; tanto las radiaciones de naturaleza no corpuscular como las corpusculares, pueden dañar el ADN humano [18]. Debido a la naturaleza inestable de los iones generados por la radiación, tienden a producir nuevos enlaces con las moléculas cercanas; de esta manera, las reacciones químicas ocasionadas pueden alterar la

Fig. 3.9: *Relación entre la dosis y la respuesta de los efectos de las radiaciones ionizantes.*

estructura del ADN de las células. En las ocasiones en las que los cambios generados no son reparables, el ciclo reproductivo de las células lo detecta y programa la muerte celular de las células alteradas, evitando así su multiplicación. Aprovechando que las moléculas de ADN son más sensibles en ciertas fases y los tejidos tumorales tienen una multiplicación incontrolada, se utiliza la radiación para generar la muerte celular de ellos; puesto que, son más radiosensibles. Cómo efecto adverso, los daños que puede generar la radiación al ser humano se dividen principalmente en dos; en el caso de que el daño se le haga al propio individuo se considera un efecto somático y, si el daño se genera en la herencia del ser humano afectado, se denomina efecto genético. Por otro lado, dependiendo que la dosis absorbida por el individuo los efectos pueden ser inmediatos o darse tras un periodo de latencia y; además, es necesario mencionar la diferencia entre el efecto estocástico y no estocástico (Fig. 3.9). Es decir, la relación entre la respuesta y la dosis se puede dar de forma probabilística, o en el momento en el que sobrepasa una cantidad de dosis, denominada "dosis umbral". Las etapas de los efectos de la radiación ionizante son las siguientes:

- Etapa física: La respuesta de esta etapa es prácticamente inmediata y, en ella se produce la ionización de los átomos, provocando un efecto cascada.
- Etapa química: En esta etapa; debido a los efectos de la etapa física, comienzan las reacciones moleculares en cadena.
- Etapa biológica: Debido a los cambios moleculares de las dos etapas anteriores, se pueden generar lesiones o daños. Puesto que, los átomos al interaccionar con la radiación quedan en un estado inestable, se recuperan tanto recobrando electrones, como iniciando reacciones en cadena que desencadenan en daños biológicos, cuya gravedad es variable.

Los diferentes efectos biológicos [19] que se pueden dar y, que son debidos a las alteraciones biomoleculares dados en las etapas mencionadas, se pueden estimar dependiendo de la dosis recibida y el tiempo de exposición; además, de los órganos afectados.

Para detectar la radiación ionizante, es necesario utilizar algún sistema de medida; puesto que, no es observable por el ser humano. Es necesario cuantificar la radiación y establecer unos parámetros que sirvan de referencia para la correcta observación de la radiación. La dosimetría se define como la "medida física que se correlaciona con los efectos de las radiaciones" y, dentro de esta definición se contemplan varios tipos de dosis con sus respectivas magnitudes de lectura; no obstante, estas son las que se deben tener en cuenta habitualmente [20]:

- Dosis absorbida: La dosis absorbida, es el cociente entre la energía media ΔE impartida por la radiación ionizante a un volumen y la masa m de ese volumen; es decir:

  $$D = \frac{\Delta E}{m}$$

  En esta ecuación, D es la dosis absorbida y en el Sistema Internacional (SI) las unidades están definidas como $J*kg^{-1}$, comúnmente denominado como Gray (Gy).

- Dosis absorbida en un órgano o tejido: En este caso hay que considerar que dependiendo del órgano o tejido sobre el cual se focalice la radiación, el umbral de dosis absorbida será mayor o menor. Por lo cual, hay una variación elevada entre la dosis absorbida y este formato de dosis; el cual, se calcula teniendo en cuenta la dosis absorbida en un punto, el volumen del tejido y, la densidad másica en ese punto. Se representa como D_T y su magnitud es equivalente a la de la dosis absorbida (D), siendo esta en Gy.

- Dosis equivalente en un órgano o tejido: Para poder calcular la dosis equivalente, es necesario calcular la dosis absorbida en un órgano o tejido primero; para posteriormente, obtener el valor equivalente en función del punto afectado. Se pueden observar las diferencias en la tabla que se muestra en la figura 10 (Fig. 3.10), teniendo en cuenta que las unidades de la dosis equivalente son los Sievert (Sv).

$$H_T = \sum_R w_R D_{TR}$$

Table B.3. Radiation weighting factors[1] (ICRP 1991b).

Type and energy range[2]	Radiation weighting factors, w_R
Photons, all energies	1
Electrons and muons, all energies[3]	1
Neutrons, energy < 10 keV	5
10 keV to 100 keV	10
> 100 keV to 2 MeV	20
> 2 MeV to 20 MeV	10
> 20 MeV	5
Protons, other than recoil protons, energy>2 MeV	5
Alpha particles, fission fragments, heavy nuclei	20

[1] All values relate to the radiation incident on the body or, for internal sources, emitted from the source.
[2] The choice of values for other radiations is discussed in paragraph A14 in ICRP (1991b).
[3] Excluding Auger electrons emitted from nuclei bound to DNA (see paragraph A13 in ICRP 1991b).

Fig. 3.10: *Tabla de ponderación en función de la energía de irradiado*

- Dosis efectiva: Una vez calculada la dosis equivalente en un órgano o tejido, se puede calcular la dosis efectiva. Esta dosis es el parámetro que especifican el efecto que ha desencadenado sobre el cuerpo humano una dosis de radiación recibida. En la figura 11 (Fig. 3.11), se puede observar una ponderación orientativa sobre los distintos órganos del cuerpo humano. Siendo la dosis medida en la unidad Sv.

$$E = \sum_T w_T H_T \, ; \sum_T w_T = 1$$

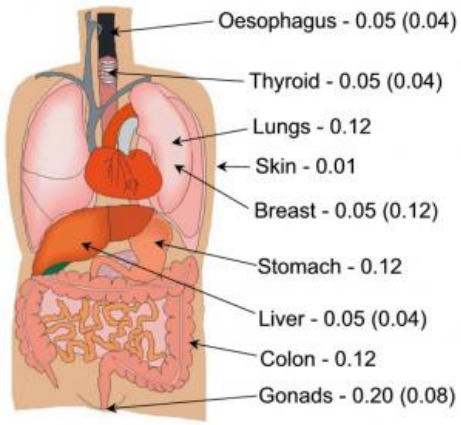


Fig. 3.11: *Factor de ponderación por tejidos (Wt)*

4. El Microtrón de 12MeV de la UPC

4.1. Diseño y las características del microtrón

El *race-track microtron*, o RTM, es un modelo de acelerador circular que se encuentra bajo desarrollo en las instalaciones de la ETSEIB (UPC) por el INTE (Instituto de Técnicas Energéticas) y que se elabora bajo la colaboración del *Skobeltsyn Institute of Nuclear Physics* (SINP) de la Universidad Estatal de Moscú (MSU).

Este modelo, alcanza un máximo haz de 12 MeV de energía [21]; además, desde un principio, se ideó para ser utilizado en el desempeño de la terapia de radiación intraoperatoria, o Intraoperational Radiation Therapy (IORT), y también con objeto de emplearse en la radiografía de materiales y en inspecciones de carga. Para esta aplicación, el RTM posee rasgos que le otorgan una significante ventaja sobre los modelos clásicos de utilización en esta rama médica, que son los linac. Más concretamente, debido a la recirculación del haz dentro del acelerador, produce el haz de energías hasta 12MeV con una estructura de aceleración muy corta en comparación con un linac de la misma energía. También permite obtener el haz de pequeña dispersión de energía; es decir, el haz es prácticamente monoenergético, y alterar la energía del haz de salida de un modo simple, tan solo extrayendo el haz desde la órbita correspondiente. Por otro lado, el costo de potencia para una pequeña ganancia en la subida de una RF a otra en el RTM, lo hace más eficiente que los linac convencionales.

En el diseño del microtrón como fuente del campo magnético de los imanes se utiliza material magnético permanente de tierras raras (REPM). Los imanes de retorno (end magnets) alcanzan un valor de campo de 0.8T; al margen de ello, para explicar los distintos detalles se expondrá la imagen siguiente del cabezal del microtrón:

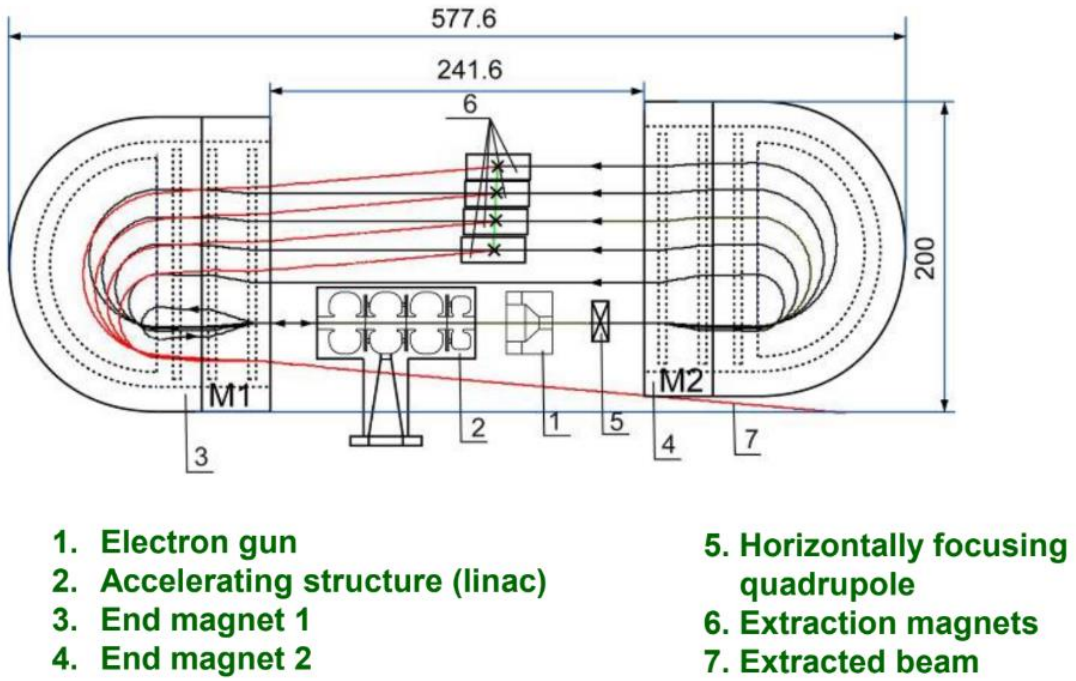


Fig. 4.1: *RTM (Race-Track Microtron).*

Como se puede apreciar en el esquema (Fig. 4.1), el RTM está constituido por varios elementos comunes en el resto de los aceleradores circulares; no obstante, el desarrollo de su funcionamiento, le proporciona las características distintivas que se exponían anteriormente en este mismo apartado.

El cañón de electrones 1, emite un haz de electrones que es acelerado en la estructura de aceleración 2 que proporciona una ganancia de energía de 2MeV por vuelta. Después de la primera aceleración el haz tiene energía de 2 MeV que no sería suficiente para poder rebasar la estructura de aceleración después de pasar por el imán de retorno M1. Entonces los imanes M1 y M2 tienen una distribución del campo magnético muy especial que refleja el haz de 2 MeV a la entrada de la estructura de aceleración. Al estar siendo utilizada una estructura de aceleración de onda estacionaria, la estructura permite la entrada del haz por ambos extremos. En la segunda aceleración, el haz ya estará suficientemente energizado como para después de hacer la vuelta de 180º empezar la recirculación dentro de la cámara de vacío del microtrón. Para focalizar el haz en el plano horizontal se coloca el imán 5, una lente cuadripolar que enfoca en ese plano al haz; mientras que, la focalización en el plano vertical se consigue mediante un perfil especial del campo magnético en los imanes M1 y M2. El haz de electrón irá ganando energía a cada vuelta y podrá ser extraído por uno de los cuatro imanes de extracción de la estructura 6. Estos imanes, desvían el haz directamente hacía la salida del RTM, obteniendo; de esta manera, un haz de electrones con una cantidad energética dependiente de la posición del imán empleado para su extracción. El elemento 7, muestra la trayectoria del haz de electrones saliendo del

RTM; no obstante, se debe tener en cuenta que todos los elementos que se ven en la figura 4.1, están cuidadosamente fijados en una plataforma en el interior de una cámara de vacío, para poder concederle al acelerador las condiciones necesarias y óptimas para su funcionamiento [22].

El RTM permite obtener un haz dependiendo de la órbita de extracción a 6, 8, 10, y 12 MeV, operando con una corriente de haz media hasta 1 µA y; de este modo, alcanzando una cantidad de dosis emitida entre 10 y 30 Gy por minuto. A su vez, las dimensiones del cabezal son bastante reducidas al ocupar 57,8 x 20 x 12,3 cm y pesando menos de 100 Kg. En la imagen siguiente (Fig. 4.2), se puede ver la estructura del RTM vista desde el exterior para poder observar el cabezal que alberga los elementos de la figura anterior (Fig. 4.1):

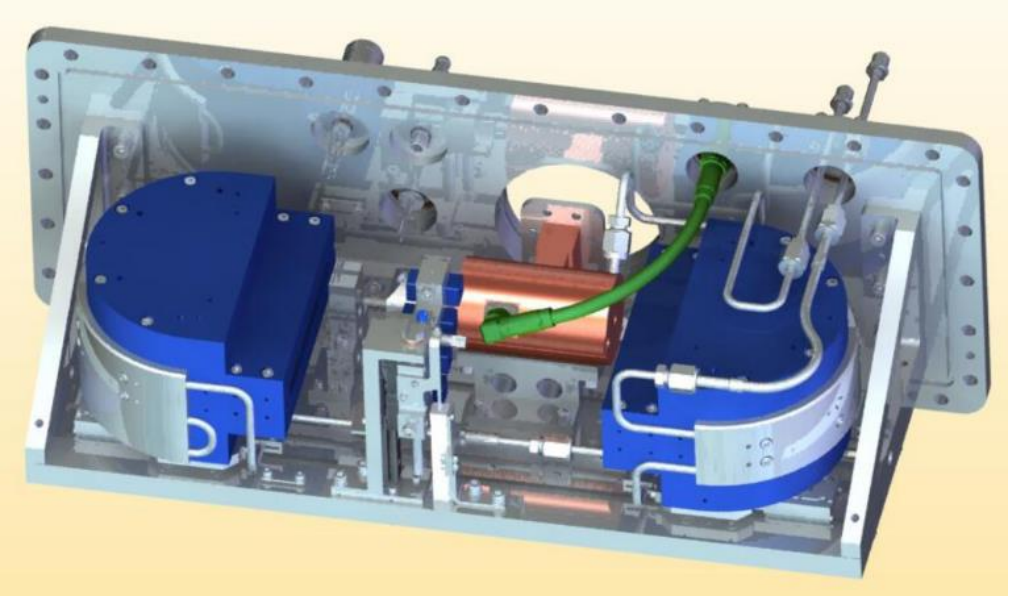


Fig. 4.2: *Vacuum Chamber (RTM).*

4.2. Sistema de control

La estructura del sistema de control del RTM [23], se puede describir de manera breve como una arquitectura modular, dividida en dos tipos de programación. Por un lado, la programación basada en el programa LabView y, por otro lado, la programación funcional de los microcontroladores, la mayoría de la familia ATmega.

De acuerdo con el diseño conceptual que se ideó en un principio, el sistema de control del RTM, con sus consiguientes versiones, debe proveer un control total del sistema del acelerador y ofrecer un entorno seguro tanto al operador como a los demás subsistemas del RTM. Además, también debe

poder ser controlado mediante una conveniente interfaz, o GUI, para el control simplificado por el operador. Por lo tanto, teniendo en cuenta la estructura, a continuación, se enumeran los distintos componentes del sistema de control del RTM:

- Un PC haciendo de elemento principal, con una GUI.
- Sistema Interlock
- Un sistema de microprocesadores comunicándose con los instrumentos y el PC principal

En la figura 3 (Fig. 4.3), se muestra el esquema de la arquitectura del sistema de control; en el cual, no se encuentra, a día de hoy, introducido el módulo de control de dosis (MCD).

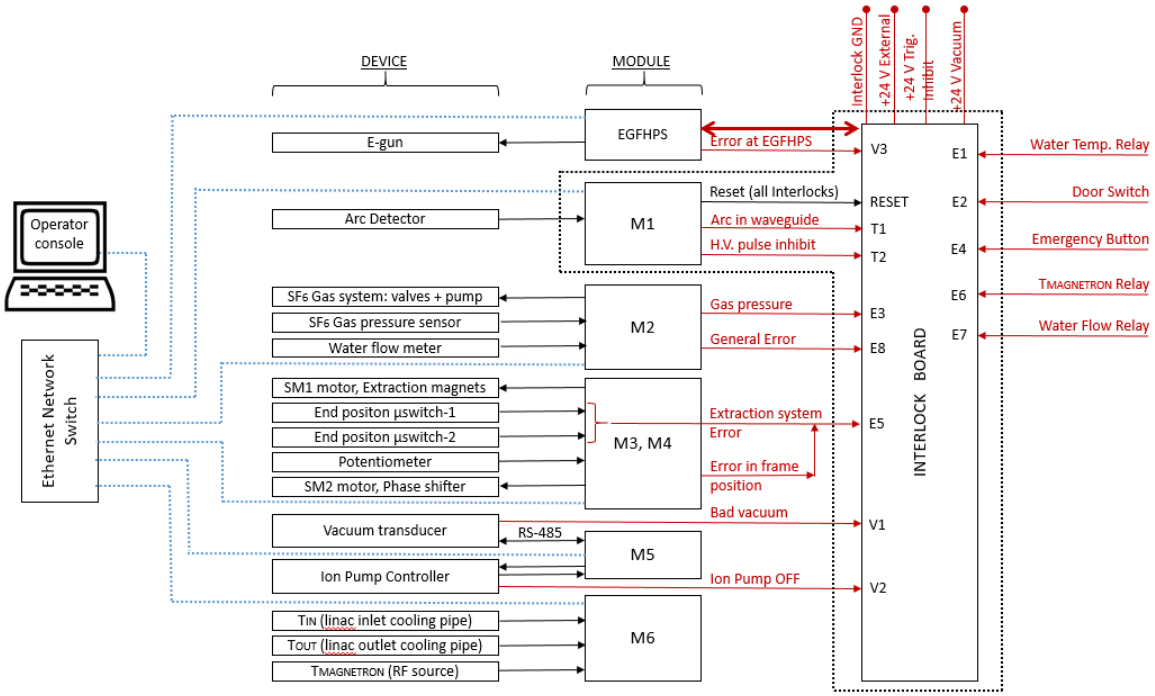


Fig. 4.3: *12MeV RTM control system architecture.*

Dentro de la arquitectura del microtrón, el sistema de control cumple la ardua tarea de interactuar, supervisando y controlando, los relés del interlock y diferentes sistemas del acelerador. Tanto el modulador (generador de pulsos de alto voltaje), como la unidad de calentamiento del cañón de electrones, como las bombas y galgas de vacío, se comunican por su cuenta con el PC principal, o con su correspondiente microprocesador del sistema de control. Por defecto, todos los elementos integrados en el sistema de control del RTM, establecen la comunicación entre ellos mediante Ethernet usando el protocolo TCP/IP; dejando de lado la programación del PC principal, que está realizada en

LabView; mientras que, el módulo de control de dosis tiene una programación individual añadida utilizando el protocolo de comunicación mediante el uso de puerto serie. Además, al margen del microprocesador que se encarga de las lecturas del sensor de temperatura del magnetrón; el cual es, el ADAM-6015 ADC, el resto de los microprocesadores se basan en el ATmega328. Este microprocesador, se integra en la estructura del Arduino Leonardo Ethernet.

Cada controlador, tiene definidos dos estados, o modos, que se cambian de forma automatizada, o manual, el modo "Safe State", o modo seguro y, el modo "Ready", o modo normal de funcionamiento. El modo seguro se activa en diversas situaciones en las cuales, se considera que la seguridad del operador, o del equipo, pueden verse expuestas a riesgos; por ejemplo, una posible reacción del sistema cambiando a modo seguro, es la pérdida de comunicación al no haber una respuesta de algún elemento dentro del tiempo de respuesta establecido por el sistema de control. Para poder volver al modo de funcionamiento normal, el operador, después de resolver la incidencia tiene que cambiar el modo desde el PC principal de forma manual. Para que esta arquitectura sea eficiente, el sistema de control del acelerador supervisa el estado del sistema interlock y testea tanto su comunicación, como la del PC principal, enviando mensajes de respuesta "ping" en un periodo de seguridad establecido.

Fig. 4.4: *RTM Interlock System.*

El sistema interlock (Fig. 4.4), se encarga de asegurar que ciertos regímenes de funcionamiento del RTM, como son la emisión del haz o las actividades a alto voltaje, solo sean posibles si el personal de seguridad y el equipo de protección los aprueban. En algunos casos, una simple variación en los valores nominales de alguno de los sensores es suficiente para que se detenga el haz, o se abra el circuito de

alimentación. Como el haz se emite solamente durante el pulso de 3 µs generado por el modulador, el sistema de control utiliza circuitos interlock del mismo modulador alimentados a 24VDC. En la versión actual del microtrón se utiliza la línea de interlock externa (*external interlock*) y de vacío (*vacuum interlock*). Para asegurar la máxima seguridad posible, prácticamente todas las conexiones con el equipo; sobre todo, las que puedan generar fallos de señal, fallos aislados o riesgos, en general, están conectadas por medio de cableado adaptado a cada necesidad. Por otro lado, en el caso de valores variables, que puedan llevar al RTM a modo seguro, es el propio microprocesador el que se encarga de mandar la orden de apertura de su respectivo relé en el circuito de interlock. En resumen, el sistema de control del acelerador se encarga de supervisar el estado de los relés del sistema interlock y de procesar las órdenes de rearme (reset) enviadas por el operador. Además, debido a la lógica empleada en el sistema de control, el haz solo puede ser emitido cuando todos los elementos de control se encuentren en el modo "Ready"; de tal manera que, en caso de que algún elemento hubiera fallado, habría sido obligatoriamente revisado y puesto a punto por el operador. A continuación, se muestra el menú principal de la aplicación diseñada (Fig. 4.5) y programada en LabView que hace de GUI; con la cual, se puede supervisar y tratar el sistema de control del RTM:

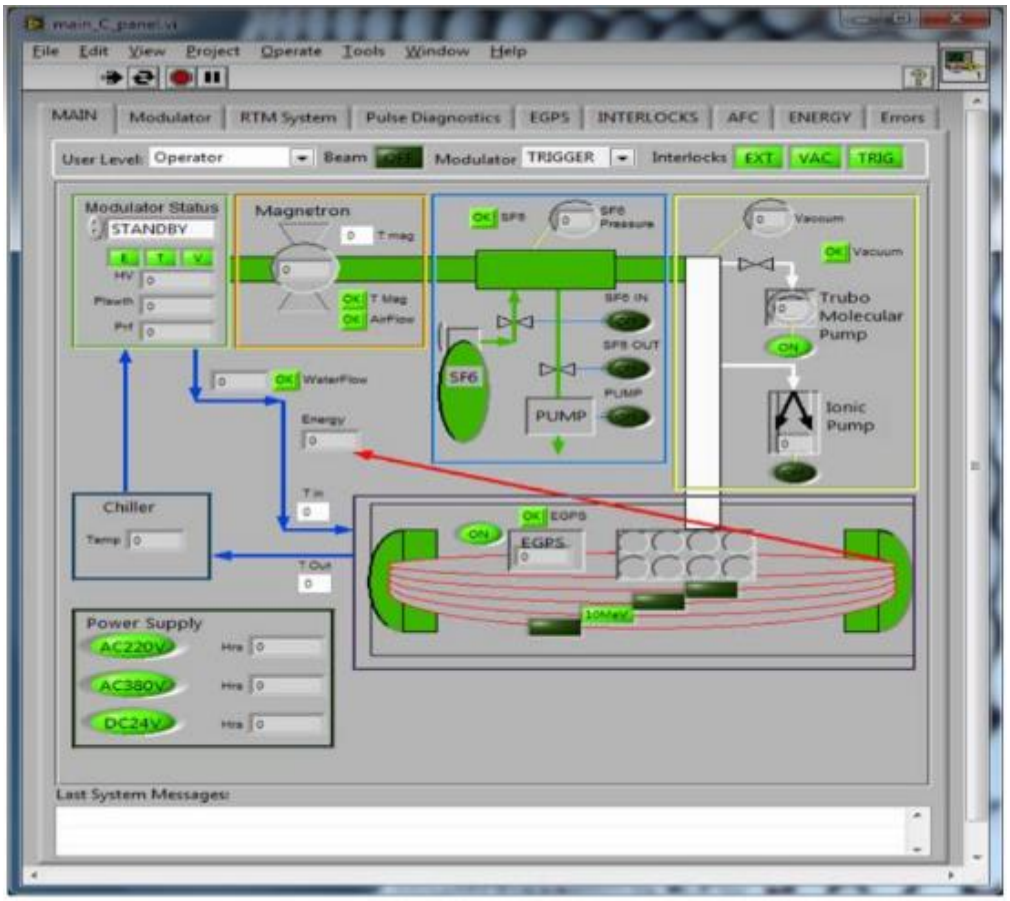


Fig. 4.5: *Menú principal de la GUI del sistema de control.*

4.3. Microcontroladores Arduino

Un Arduino es una plataforma hardware *open source*, de venta a nivel comercial, que incluye un microprocesador como elemento principal. Está placa se utiliza de forma habitual en el diseño de prototipos, tanto a nivel amateur como a nivel profesional. En el ámbito profesional, se han generado proyectos importantes, como impresoras 3D, basados en un prototipo inicial de Arduino en una de sus variantes. Hay bastantes tipos de placas Arduino en el mercado, aunque en este proyecto se utilizaron fundamentalmente dos que se explican a continuación, Arduino UNO y Arduino Leonardo Eth.

- Arduino UNO

Está placa Arduino (Fig. 4.6), además de ser la más común para el prototipado por ser relativamente sencilla, es la placa con la que se ha llevado a cabo la mayor parte del proyecto. Está basada en el microprocesador ATmega328 y para comunicarse con el ordenador, tiene un chip ATmega16U2. Dentro de sus especificaciones nos interesan las que se enumeran a continuación:

1. Funciona a 5V y a 16MHz.
2. La alimentación recomendada es de 7-12V
3. Tiene 32KB de memoria flash
4. Tiene conector Jack (para la alimentación)
5. Dispone de un conector USB (para la transmisión de datos)
6. Dispone de 14 pines I/O digitales
7. Tiene 6 pines analógicos

De esos 14 pines de entrada y salida digitales 6 pines nos permiten generar PWM (Pulse Width Modulation). Además de los pines estándar tanto analógicos cómo digitales, tiene una sección de headers especiales para la alimentación.

- GND: también llamado ground, es el polo negativo de la alimentación; el cual, hace referencia a los 0V de un circuito electrónico. Tiene 3 pines dirigidos a este cometido.
- 5V: Este pin proporciona una tensión de 5V, regulada y estabilizada por la placa. El estabilizador interno es capaz de dar hasta 1A de corriente.
- 3,3V: Al pasar por un regulador de tensión, hay un pin regulado a 3,3V de tensión.
- VIN: Este pin está enlazado con la entrada de alimentación y proporcionará la tensión que se le introduzca a la placa por medio de la entrada tipo Jack.

- AREF: A este pin es al cual la placa permite agregarle una tensión de referencia en el caso de que se necesite más precisión en la lectura de datos analógicos.

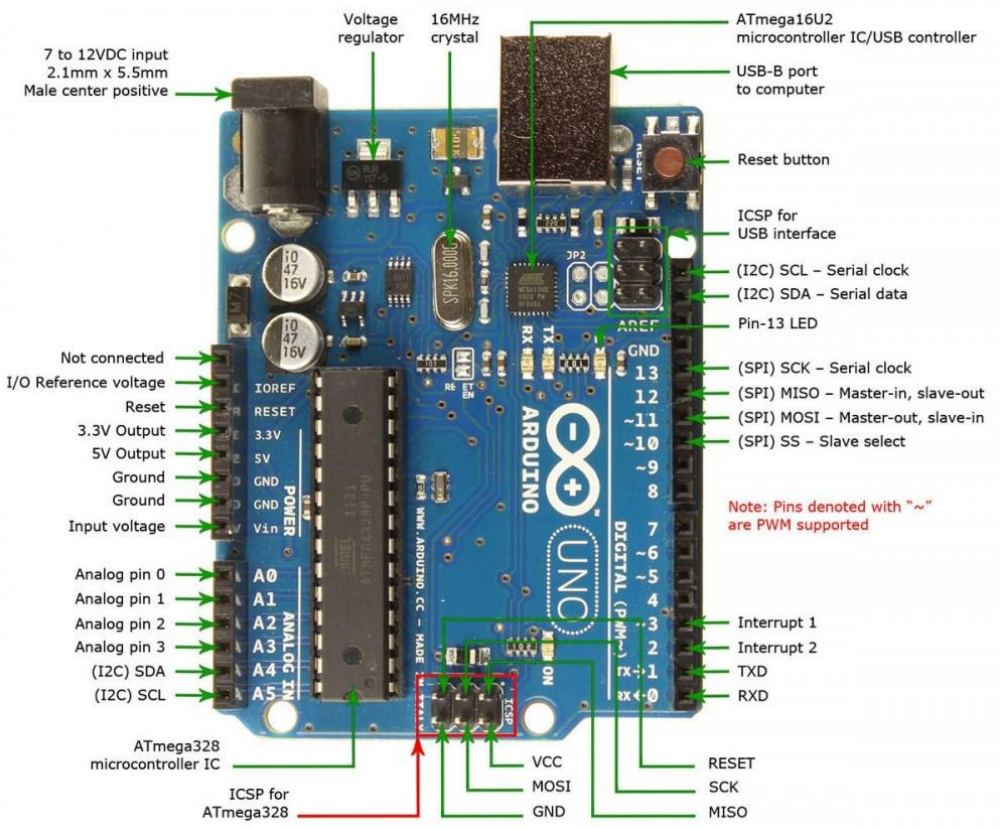


Fig. 4.6: *Arduino UNO.*

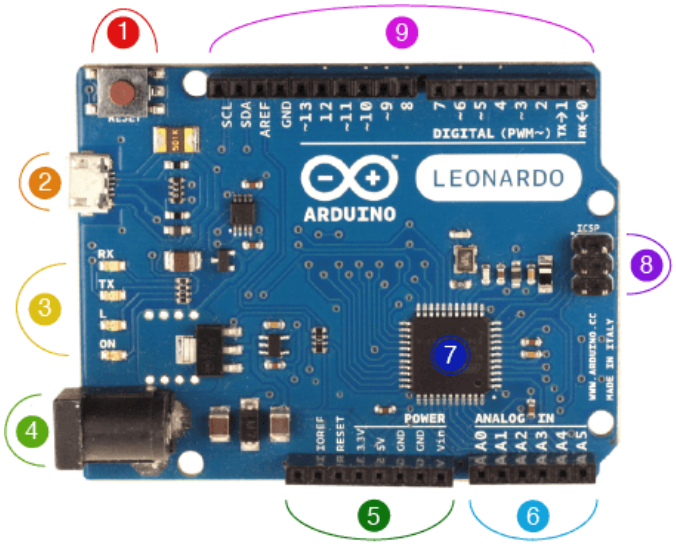
- Arduino Leonardo Eth

Esta plataforma hardware vendría a ser la sucesora del Arduino UNO en cuanto a cronología dentro de esa marca. A diferencia del Arduino UNO, la placa contiene un microprocesador ATmega32U4, lo cual, evita tener un segundo chip para el control de la transmisión de datos con el ordenador; sin embargo, esta modificación genera que el código a ejecutar ocupe más espacio de memoria al tener que soportar también la comunicación con el ordenador. Además de permitir las mismas operaciones que el Arduino UNO, este hardware incorpora una entrada Ethernet; por lo tanto, nos permite comunicarnos vía Ethernet con el ordenador y otros objetos; por otro lado, el modo de programarlo es diferente. Mientras que en el Arduino UNO, el puerto de comunicación para la programación del microprocesador es USB; en el caso del Arduino Leonardo, es micro-USB. Las prestaciones del Arduino Leonardo Eth. (Fig. 4.7) se enumeran a continuación:

1. Funciona a 5V y a 16MHz.
2. La alimentación recomendada es de 7-12V.
3. Tiene 32KB de memoria flash.

4. Tiene un conector Jack (para la alimentación).
5. Dispone de un conector micro-USB (para la transmisión de datos)
6. Dispone de 20 pines I/O digitales.
7. Tiene 12 pines de entrada analógica.
8. Solo funciona para las versiones 1.0.1 del IDE de Arduino.

De los 20 pines de entrada analógica 7 son capaces de proporcionar señales PWM, siendo bastante más completo que el Arduino UNO y estando más preparado para realizar labores en sistemas de control.



Color	Tech Name	What it does:
1	Reset Button	Restarts the Arduino
2	USB	Program your Arduino from your Computer
3	Indicator LEDs	Shows the user what the Arduino is doing
4	DC Power	Power your project - without USB
5	Power Pins	Connect power to your circuit
6	Analog Pins	Connect sensors to your Arduino
7	ATmega32u4	Brain of your Computer
8	ISP Connector	Update bootloader (software) on the Arduino
9	Digital I/O	Connect sensors or actuators to your Arduino

Fig. 4.7: *Arduino Leonardo Ethernet.*

- Interrupciones

Las interrupciones son un mecanismo que permiten controlar eventos que tengan una urgencia alta, o una necesidad sobre el código. Mediante ellas, se puede hacer que el microprocesador salte directamente desde un punto del código a otro para controlar un proceso principal. Las interrupciones se suelen detectar mediante la alteración de algunos pines específicos dentro del microprocesador. En

el caso de Arduino UNO, los pines habilitados para este tipo de operaciones son exclusivamente los pines digitales 2 y 3; mientras que, en el caso del Arduino Leonardo, son los pines 0, 1, 2, 3 y 7.

- Tipos de Memoria

Un microcontrolador tiene tres tipos de memoria, flash, RAM y EEPROM.

1. Memoria flash: Esta memoria es permanente y también reescribible unas 10.000 veces, en esta memoria es donde están escritos tanto el sketch, como el bootloader.
2. Memoria RAM: La memoria RAM (Random Access Memory) es la memoria de solo lectura, rápida y perecedera; puesto que, se borra al cortar la alimentación. Generalmente, se utiliza para mantener las variables del programa.
3. EEPROM: Está es una memoria permanente y se emplea para guardar parámetros y ajustes que no deberían desaparecer al cortar la alimentación; aunque, sus dimensiones rondan 1kB.

- Timers

Los timers, son contadores que se pueden utilizar para desarrollar operaciones temporalizadas y para ajustarlos hay que deshabilitar las interrupciones. Tras haberlos configurado, cuando se ejecuten, invocarán una interrupción que llamará a la función de código que se deba ejecutar. En el caso del microprocesador ATmega328, hay tres contadores:

1. Contador0: Es de 8 bits, llega a un máximo de 16,4 microsegundos y la placa los utiliza para el funcionamiento de las funciones millis() y delay().
2. Contador1: Este es de 16 bits; por lo que, llega a un máximo de 4,1 milisegundos.
3. Contador2: También de 8 bits, este lo utiliza la función tone()

A lo largo del proyecto se han utilizado ambos Arduino para la elaboración de distintas fases de la programación; puesto que, Arduino Uno está limitado a la utilización del puerto serie para la comunicación. Además de Arduino, también se ha contado con un shield (elemento que se explica a continuación) con el cual se ha podido llevar a cabo la comunicación por RS232 y el control de seguridad del acelerador por medio de un relé.

Una shield (Fig. 4.8) es una placa de circuitos modular que se monta encima de Arduino para darle una utilidad extra. Son apilables; por lo tanto, realmente es posible apilarla sobre el mismo Arduino que se vaya a utilizar, o directamente, sobre otra shield, ampliando de manera gradual el hardware empleado en un prototipo o diseño. En el proyecto se usó un modelo especifico de shield, llamado protoshield.

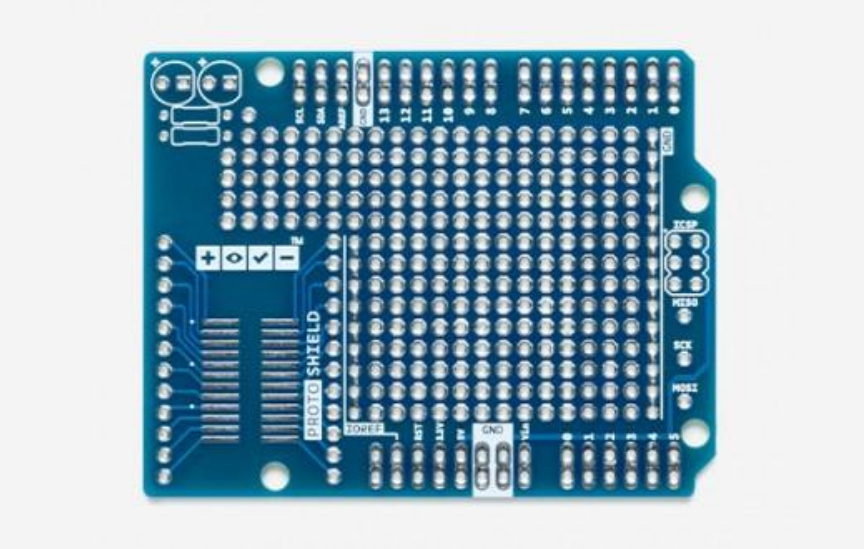


Fig. 4.8: *Arduino ProtoShield Rev 3.*

- Hardware empleado

Para que Arduino pueda comunicarse con la cámara de ionización, es necesario el empleo de la comunicación RS232; para ello, se requiere emplear un hardware específico que permita a Arduino convertir TTL en RS232 para la emisión de información a la cámara y facilitar la conversión inversa para la recepción de los datos que el código procese para monitorizar y controlar la dosis de radiación. Por otro lado, para controlar la emisión del haz de electrones se utiliza un relé del circuito de interlock externo que debe ser controlado por el Arduino; por lo tanto, en el mismo protoshield, se incluyen ambos conjuntos de elementos hardware; es decir, tanto el sistema de comunicación RS232, como el relé de seguridad del acelerador.

En primer lugar, el sistema de comunicación RS232 (Fig. 4.10) está constituido por los siguientes elementos:

- MAX3232: Es un circuito integrado (Fig. 4.9) o transceptor, diseñado para realizar la conversión entre TTL y RS232, para ello necesita un esquema de montaje proporcionado por el fabricante, que constituye el resto del diseño del sistema de comunicación RS232.

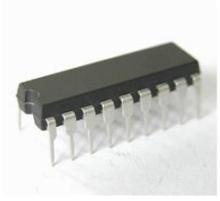


Fig. 4.9: *MAX3232 Transceiver.*

- Condensadores: Básicamente, se encargan de elevar y estabilizar la tensión del sistema.

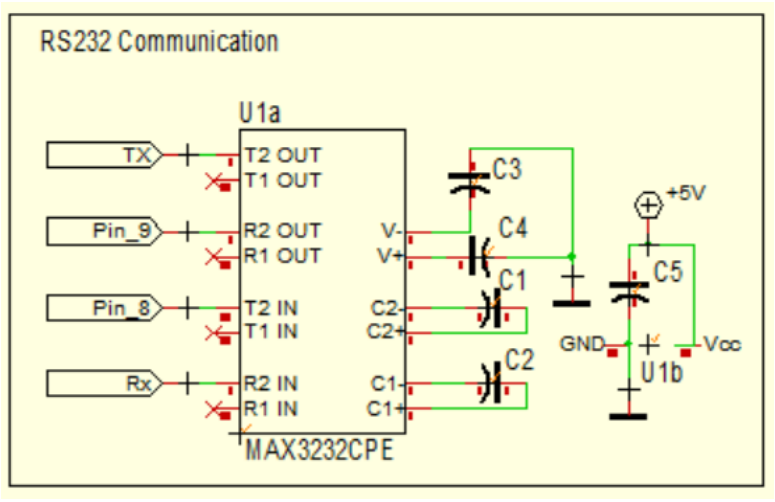


Fig. 4.10: *Esquemático comunicación RS232.*

En segundo lugar, el sistema de control de seguridad (Fig. 4.11) manejado por relé está formado por los siguientes elementos:

- Relé: Es un relé de la empresa Finder alimentado a una tensión de 5V y accionado al emitir tensión desde un pin de Arduino; el cual, será el que cierre y abre la línea de 15VDC que alimenta uno de los relés del circuito de interlock externo situado en la caja de interlock mostrada en la figura 4.4.
- Diodo: En el caso del protoshield empleado, es un diodo colocado a los dos extremos de la bobina del relé. Este se encarga de evitar las corrientes de Focault.
- Transistor: En el cuerpo del protoshield se sitúa un transistor PNP, modelo BC 557 B, que se encargará de estabilizar y controlar la entrada de tensión que activará el relé. Está conectado a GND y a un pin de Arduino que emitirá la tensión de activación.
- Resistencia: Entre el pin de Arduino que emitirá la tensión de activación del relé y la base del transistor, se encuentra una resistencia para evitar los picos de corriente en el circuito.

Para diseñar y optimizar ambos sistemas, se emplea una única protoshield, y para la elaboración de los esquemáticos y la construcción por ordenador de la PCB, se utilizó el programa de diseño Target 3001, al ser gratis hasta un número elevado de conexiones.

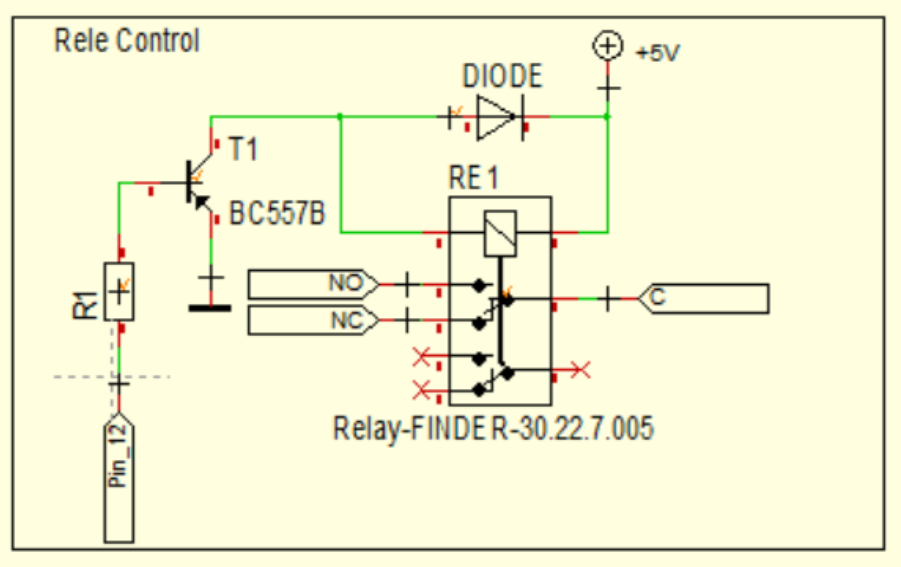


Fig. 4.11: *Esquemático control de relé.*

En la siguiente figura (Fig. 4.12), se muestra el posible diseño de ambos sistemas en una sola placa, o PCB, para la utilización del protocolo de comunicación RS232 y el control de seguridad manejado por relé:

Fig. 4.12: *Diseño de Sistema de comunicación RS232/Control de Relé.*

4.4. Módulo de control de dosis

Debido a que una de las posibles aplicaciones del RTM, es la irradiación de muestras de diferentes materiales utilizando el haz de electrones para la investigación de los mismos, se planeó la inserción de un módulo dedicado al control de la dosis de radiación absorbida. Se contemplan dos modos de irradiación.

(1) "Fixed time": En este caso, la irradiación se lleva a cabo durante un tiempo que fija el operador; mientras que, la dosis emitida es determinada por el propio microprocesador del módulo de control.
(2) "Fixed dose": Al contrario que en el "Fixed time", en este caso el operador determina una dosis y el microprocesador determina el tiempo de irradiación.

Para la determinación de ambos modos de irradiación, se emplean dos métodos para la determinación de los valores a utilizar.

(a) "Analytic": Se emplea una fórmula analítica semiempírica que relaciona el tiempo de emisión del haz con la dosis depositada. Para ello, se toma como base el modo de irradiación seleccionado y distintos parámetros que toman parte en el cálculo de la variable resultante.
(b) "Measurement": En este caso, se hace valer el dosímetro PTW-UNIDOS, con el cual se toman datos en tiempo real y; de este modo, se comprueba el valor, hasta que iguale el valor predeterminado de dosis, o de tiempo, fijados.

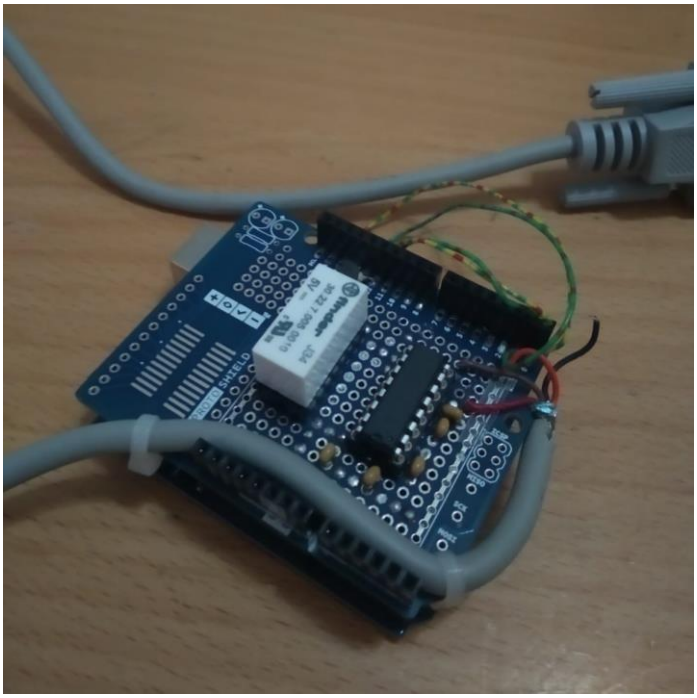


Fig. 4.13: *Módulo de control de dosis del microtrón.*

El módulo de control de dosis (Fig. 4.13), se comunica con la consola del operador por medio de comunicación puerto serie, o utilizando Ethernet; por otro lado, recibe datos de la cámara de ionización empleando el protocolo de comunicación RS232 y controla un relé normalmente abierto (N.O.), que controla la línea de interlock externo del modulador. De esta manera, el MDC mantiene el relé cerrado cuando el RTM, este convenientemente programado para emitir el haz de electrones, y no haya ninguna pérdida de comunicación intermedia. El análisis del módulo se trata en profundidad a lo largo del capítulo 5; no obstante, en la siguiente figura (Fig. 4.14), se puede analizar el esquema de conexiones del MCD.

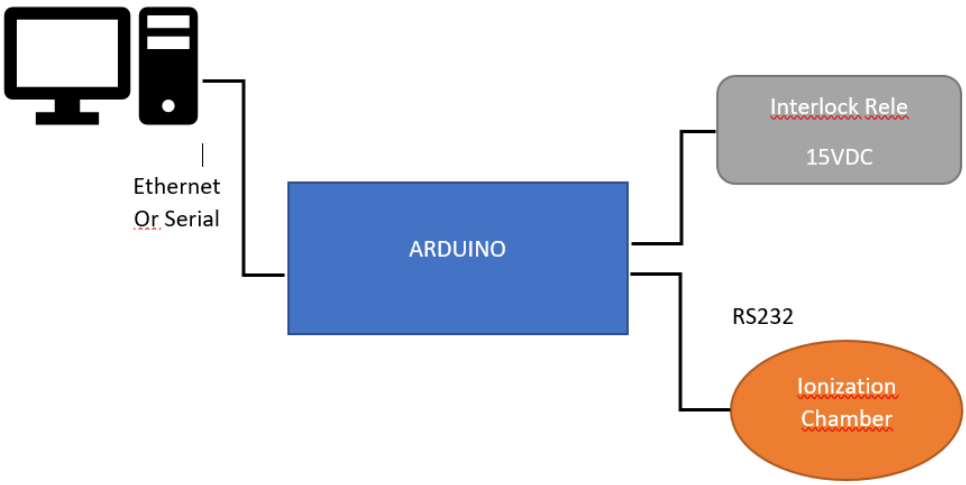


Fig. 4.14: *Conexiones del módulo de control de dosis con otros sistemas del RTM de 12MeV.*

5. Módulo de control de dosis del Microtrón

5.1. Arquitectura y funciones del módulo

El programa escrito en lenguaje C/C++, tiene diferentes funciones y métodos de funcionamiento que le permiten trabajar utilizando la cámara de ionización, o dosímetro, PTW-UNIDOS, o en dos formatos distintos de control autónomo. De esta forma, el módulo de control de dosis del microtrón, actúa a modo de entorno con la cámara de ionización. El equipo se encarga de medir la radiación emitida por el acelerador y poder hacer un control preciso de la radiación absorbida por el material a irradiar. A lo largo de este apartado, se definirán las distintas variables y modos que determinan el comportamiento del RTM y la emisión del haz, y se procederá a la descripción de la arquitectura del módulo.

El MCD de Arduino funciona utilizando un patrón denominado request/response mediante el que el microprocesador responde a instrucciones que son enviadas por el operador. El módulo tiene dos tipos de instrucciones que serán válidas dependiendo del contexto que se le haya designado, mediante los modos de detección o de irradiación. Estas dos opciones son 'W' y 'R', W es el inicio de una instrucción que planteé introducir un cambio en alguna de las variables de control que determinan el funcionamiento de programa; además, también se utilizará para el accionamiento y detención de la emisión del haz. Por otro lado, R es el inicio de una instrucción que devolverá el estado de una variable de forma aislada; esto se debe, a que el módulo de control ya cuenta con una instrucción al margen de estas dos, para la comprobación del estado global del acelerador y de las condiciones de lectura en cada momento. Esa entrada de instrucción está definida como 'P', es la que envía un PING al sistema.

A continuación, se explican distintos componentes de la arquitectura del MCD y las diferentes funciones que controlan el patrón de request/response.

- PING: El programa del MCD cuenta con una instrucción de comunicación con el operador basada en la estructura PING; es decir, debe emitir un mensaje compuesto por los valores de las variables FMST/DCON/SCDE/TRAD/DRAD/SSTA/IM/MD/TSES/DSES/K1/K2/K3/K/F/ DBUG/VCC cada vez que el operador lo vea conveniente.
- Safe State: El MCD cuenta con un sistema de seguridad; el cual, podrá ser verificado por el operador en todo momento; además, de poder ser alterado por el mismo. El sistema de seguridad será activado automáticamente en el caso en el que una instrucción no sea registrada en un tiempo máximo definido por el software del MCD. De este modo, el tiempo máximo desde que se envía una última instrucción está definido por la variable cmdT, la cual

se determina al final de la emisión de cada instrucción. Por otro lado, en el setup() del código, se determina la constante para controlar la comunicación, CTmax (Communication Timeout); la cual, tiene un valor establecido por defecto del MCD fijado en 10 ms. En el momento en el que, por medio de interrupciones, el tiempo desde la última instrucción sobrepasa esta variable, se activa el "Safe State". En este modo de funcionamiento del módulo de control de dosis del microtrón, el relé mantiene la alimentación abierta, bloqueando la posibilidad de que el acelerador emita el haz de electrones; además, se modifica el valor de la variable SSTA (SSTA = 4); puesto que, la sesión se encontrará interrumpida por el MCD; no obstante, la comunicación entre el operador y el módulo de control seguirá abierta. Para poder volver al modo de funcionamiento normal, el operador deberá mandar una instrucción específica indicando que desea volver al modo de funcionamiento normal. Esta acción hará que el estado de la variable SSTA se restablezca (SSTA = 0).

- New Session: Para comenzar una nueva sesión, el operador deberá enviar una petición específica para este cometido. De esta manera, en cualquier momento a lo largo de la realización de la sesión podrá corregir y reiniciar la sesión. Cada vez que el operador haga una petición de nueva sesión, todos los valores introducidos se almacenarán y guardarán, reinstaurando la sesión a sus valores iniciales.
- Session Code setting: Al especificar que se desea generar una nueva sesión, se podrá escoger entre asignarle un valor seleccionado por el operador a la variable SECD (Session Code); en el caso de que, en la escritura de la instrucción, se añada un valor. O, en el caso de que no se le añada un valor específico, el módulo de control le asigne un valor, basándose en la última sesión ejecutada. Puesto que un microprocesador ATmega2560 no guarda, en principio, las sesiones anteriores; para ello, se ha reservado un espacio para variables persistentes en la memoria EEPROM; de este modo, cada vez que se crea una sesión se recupera la SECD guardada en el byte 0 de la memoria EEPROM en la sesión anterior. Del mismo modo, al finalizar una sesión se guarda la SECD en ese mismo byte.
- Irradiation Mode setting: Para definir el modo de operación (o de irradiación), el operador determina cuál de los dos modos desea utilizar, las opciones se describen a continuación (tabla 5.1):

Tabla 5.1: *Modos de irradiación del MCD.*

IMOD (Irradiation Mode)	Descripción
FT → IMOD = '1'	"Fixed Time". En este modo de irradiación, el código utilizará un tiempo proporcionado por el técnico y, a partir de él, calculará la dosis requerida para dar por concluida la irradiación.
FD → IMOD = '2'	"Fixed Dose". En este modo de irradiación, el código utilizará una dosis proporcionada por el técnico y a partir de ella, calculará el tiempo requerido para irradiar esa dosis en el material.

- Delivery setting: Dependiendo del modo de irradiación escogido en el apartado anterior, el operador deberá introducir por medio de comandos un valor que; en el caso de FD, será la dosis, y en el caso de FT, será el tiempo. En el caso de la dosis, el parámetro que contendrá el valor de la cantidad proporcionada por el operador será DRAD; para el cual, habrá que introducir la dosis en Grays [Gy]. Por otro lado, en el caso de que el tiempo sea fijo, la variable que deberá inicializarse será TRAD, la unidad de medida que precisa el módulo de control de dosis para el cálculo mediante el uso de un tiempo fijado por el operador es la de los segundos [s].
- Method of dose control setting: Una vez que se haya definido la dosis o el tiempo necesario, el programa del MCD, calculará los respectivos valores que falten. Para ello, el programa tiene dos métodos para determinar cuáles serán los valores necesarios, y uno para obtener la información de la cámara de ionización. Los métodos se encuentran descritos en la tabla 5.2:

Tabla 5.2: *Métodos de cálculo del MCD.*

METD (Method of Dose Control)	Descripción
METD = 11	"Analytic". Para este método, el programa requerirá los distintos factores K1, K2 y la frecuencia de repetición, para poder calcular la dosis, o el tiempo, a irradiar durante la sesión.
METD = 12	"Conversion Factor". En este caso, el programa empleará únicamente la variable KSET (Convertion factor) para la determinación de la dosis.
METD = 20	"PTW-UNIDOS". Al seleccionar este método, el programa habilita la comunicación RS232 con la cámara de comunicación. De este modo, los datos de la irradiación serán introducidos directamente desde esta vía. En este momento del desarrollo, este método no se encuentra plenamente operativo; más adelante, se especifican sus funcionalidades actuales.

Caso METD = 11

- K1 factor: En este caso, el operador deberá introducir un valor obtenido por medio de la relación entre el material a irradiar y la energía que el acelerador aplicará a lo largo de la sesión. El operador podrá introducir ese valor en la variable global K1. Las unidades; con las cuales, el MCD podrá operar, serán de [nGy * cm^2] y este valor se aplicará para el cálculo tanto de la dosis, como del tiempo.
- K2 factor: El operador podrá introducir por medio de instrucciones el valor de un cálculo; en el cual, se relacionan las variables descritas, posteriormente, en el apartado 2 de este mismo capítulo; donde, se definen los parámetros que operan en este factor. Este factor, es necesario para el cálculo de la dosis o el tiempo: por otro lado, en caso de que K2 tenga un valor equivalente a cero, se estará indicando que no se cuenta con un colimador y, por lo tanto, el haz se encontraría irradiando directamente el material. Las unidades con las que deberá introducir este valor para que el MCD pueda trabajar con ellas serán [cm^{-2}].
- Frequency control setting: En este apartado, como último dato necesario el operador deberá introducir la frecuencia de repetición de pulsos con la que trabajará el acelerador; de este modo, el programa obtiene la última variable que necesita para calcular la dosis o el tiempo cuando METD = 11. Esta información quedará almacenada en la variable K3. El valor introducido deberá estar cuantificado en [Hz]

Caso METD = 12

- Conversion Method control setting: En este caso, el operador únicamente tendrá que introducir el factor de conversión almacenándose en la variable K, que se empleará para la obtención de la dosis o el tiempo. Este método, permite al operador introducir un valor directo que sirva para el cálculo momentáneo de la dosis, o el tiempo, a lo largo de toda la ejecución. Para que el valor pueda ser utilizado por el MCD, deberá estar cuantificado con las unidades correspondientes [Gy/s].

Caso METD = 20

Para este caso no hay un método específico, simplemente son necesarias las entradas de datos de las variables SECD, IMOD y METD; además de, DRAD o TRAD, para especificar la dosis, o el tiempo, a irradiar en la sesión. Esto se debe a que serán los parámetros por los cuales se estime la medida que se captará desde la cámara de ionización. Para la calibración precisa de la medida que se realizará, el operador deberá introducir el valor que se almacenará en la variable global F. Esta variable sirve para guardar el factor de calibración. No obstante, esta funcionalidad se encuentra en desarrollo y, debido a diferentes sucesos a lo largo del proyecto, se verá incompleto de cara a la entrega, pese a que, se haya definido el estándar por el cual se continuará este proyecto en un futuro.

5.1.1. Emisión del haz

Además de los métodos para el control de las variables que participan en el cálculo de la dosis, o el tiempo, el MCD cuenta con distintas acciones que están consideradas como instrucciones de inicialización; es decir, estarán marcadas como instrucciones de escritura. Estas instrucciones, son las que permiten al operador iniciar la emisión del haz, pararla de manera temporal, o detenerla definitivamente. Todas ellas alteran una variable denominada SSTA, que reporta el estado en el cual se encuentra la emisión en cada momento. Las instrucciones están definidas como "Beam ON" y "Beam OFF", de cara a la utilización del RTM.

5.1.2. Método de depurado

Para poder controlar el modo de funcionamiento y transmitir cierta cantidad de información normalmente no reportada al operador, el programa del módulo de control de dosis del microtrón, permite accionar una funcionalidad denominada "debug", o depurado. Al activar esta función, serán impresas en pantalla ciertas variables que permitirán aclarar dudas, o trazar la entrada de los datos al operador, profundizando en el sistema. Es un sistema de validación que se empleó a lo largo del desarrollo y que permite al operador verificar datos en tiempo de ejecución, en caso de que fuese necesario.

5.1.3. Safe State

Por otro lado, también se puede controlar de modo manual el estado del programa, esto es práctico para activar o desactivar el Safe State En el modo de funcionamiento normal, la sesión te permite cerrar el circuito para activar el haz una vez que se hayan introducido los parámetros necesarios para su utilización de manera segura. No obstante, en caso de que se haya activado el modo seguro, o Safe State, el MCD permanecerá con las siguientes especificaciones, hasta que el operador recuperé la comunicación y cambie el modo de control de la sesión.

1. En caso de que la comunicación con el módulo no se vea interrumpida, podrá responder a instrucciones de lectura; como, por ejemplo, "PING".
2. El relé que cierra la alimentación del acelerador permanecerá abierto.
3. No se ejecutará ninguna medida, o cálculo, de dosis, o tiempo.
4. El MCD modificará el modo de funcionamiento de la sesión de "Safe State" a "Normal Mode" a petición del operador.

5.2. Método de control de dosis

Como se específica en el apartado 5.1, el módulo de control de dosis de microtrón, tiene tres métodos diferentes para el cálculo de la dosis emitida por el acelerador. Estos modos, o métodos, son seleccionados por el operador al dar un valor a la variable METD. Para inicializar la emisión de radiación con cualquiera de los tres métodos, es necesario que estén seleccionados previamente el número de sesión, el modo de irradiación y la dosis, o el tiempo, a emitir; de otro modo, el módulo no permitirá el cierre del relé, evitando la posibilidad de irradiar ningún material. Las distintas posibilidades serán explicadas a continuación:

- "Analytic" (METD = 11)

En este método; además de las características habituales a definir, será necesario introducir los datos de todas las variables restantes. Esto se debe a que, la fórmula mediante la que se calcula tanto el tiempo, como la dosis, depende de la energía emitida, el material empleado, el colimador utilizado y la frecuencia con la que se repiten los pulsos. La relación entre la dosis depositada y el tiempo de irradiación se muestra en la siguiente fórmula:

$$D[Gy] = K * T[s]$$

El coeficiente K será determinado por las diferentes variables introducidas; de este modo, K se obtendrá del producto de K_1, K_2 y K_3.

(1) $K_1 = \frac{1}{\rho}\left(\frac{dE}{dx}\right) = K_1$ (E, material)

Para el cálculo de K1 por parte del operador, se deberá contar con la densidad del material a irradiar, el grosor, y la energía; con la cual, funcionará el haz del RTM.

(2) $K_2 = \frac{K}{A} = K_2$ (colimador)

Para el cálculo de K2 por parte del operador, se tendrán que tener en cuenta las variables que definen el área del campo de irradiación (A) y, la fracción del flujo inicial generado por el haz de electrones que, finalmente, llega al material objetivo (K).

(3) $K_3 = \frac{l}{|e|} = \frac{Qpulse}{|e|} frep = Npulsefrep = K_3$ (frep)

Por último, para que el operador determine K3 pueden realizarse procedimientos diferentes cuyo resultado es equivalente entre sí, dependiendo de los datos que se tengan de inicio. Por lo tanto, se puede relacionar la corriente media del haz (l), con la carga del electrón, también, la carga total del pulso de electrones ($Qpulse$), con la carga de electrones y la frecuencia de repetición del haz ($frep$); así como, el número de electrones contenido en cada pulso ($Npulse$) con la frecuencia de repetición del haz.

Una vez introducidas los tres parámetros: K1, K2 y K3, el programa calculará K, aplicando la siguiente fórmula, en la cual, el producto resultante se expresará en [Gy/s]:

$$K = K1 * K2 * K3$$

Cuando el operador introduzca los datos deberá tener en cuenta las unidades definidas en el apartado 5.1.

- "Conversion factor" (METD = 12)

Al emplear este método solo serán necesarios, el código de sesión, el modo de irradiación, la dosis, o tiempo, a emitir y el factor de conversión; evidentemente, al margen del propio método de determinación de la dosis. Para determinar la dosis, o el tiempo, en este método se establece una relación lineal utilizando el factor de conversión proporciono por el operador. En la siguiente fórmula, se expone la relación descrita anteriormente:

$$D[Gy] = F * T[s]$$

Para determinar F, se emplea el dato almacenado en la variable global KSET. El cálculo de F estará sujeto a diversos condicionante; aunque, se encuentra estandarizada [24].

Ambos métodos se emplean de forma autónoma; puesto que, no se emplea ningún elemento al margen del módulo de control de dosis del microtrón. Una vez facilitados los datos requeridos desde el operador, será posible la activación de la emisión de radiación. No obstante, el método de "PTW-UNIDOS", sí que necesita un elemento externo que es la cámara de ionización.

- "PTW-UNIDOS" (MDDC = 20):

Este método no requiere más que la introducción de los datos comunes en los tres métodos; de todos modos, para el funcionamiento óptimo de esta opción, son necesarias las comprobaciones que se verán explicadas en el apartado 6.3. Por otro lado, requiere de la introducción por parte del operador del valor que se almacenará en la variable FSET; el cual, servirá para calibrar la dosis, o el tiempo, medido desde la cámara de ionización. El cálculo que se debe realizar para que el ajuste sobre la medida adquirida se vea reflejado; de tal manera, que los datos recibidos por parte de la cámara de ionización PTW-UNIDOS sean válidos, se ve descrito en la fórmula siguiente:

$$D[Gy] = F * D_{meas}$$

En esta fórmula, la dosis medida está representada como D_{meas}, y el factor de calibración como F.

5.3. Dosímetro PTW-UNIDOS

Para poder captar de forma activa la dosis de radiación emitida por el acelerador de electrones, el módulo de control guiado por Arduino se conecta por medio de RS232 con la cámara de ionización PTW-UNIDOS. Esta cámara pertenece al laboratorio de dosimetría del INTE en la ETSEIB y, se utiliza de forma habitual para distintas pruebas; dato por el cual, se volvió complicado a lo largo de la confección del trabajo llevar a cabo de manera efectiva los distintos experimentos para corroborar la eficacia y el correcto funcionamiento del módulo de control de dosis del microtrón.

La cámara de ionización PTW-UNIDOS, posee una interfaz con la cual, el operador, debe hacer una puesta a punto, antes de utilizar el sistema de control. Esto se debe a que, pese a la posible manipulación permitida por el código del MCD al propio sistema PTW-UNIDOS, es necesario que la máquina se encuentre en unas condiciones previas específicas.

Para la comunicación de Arduino por medio de RS232 con la cámara de ionización, es necesario que PTW-UNIDOS se encuentre en la pantalla de medición de radiación; a la cual, se accede pulsando el botón físico MEAS (Fig. 5.1). Además de esta necesidad, deben estar predefinidos los criterios de funcionamiento del protocolo de comunicación. A continuación, se listan los puntos específicos a preseleccionar para la comunicación entre el dosímetro y el módulo de control:

- Baud Rate: es la velocidad de transmisión (bits por segundo) y el dosímetro admite desde 1200 hasta 115000.
- Paridad: en el caso del dosímetro es 8N1.
- RS232: Es necesario asegurarse que la comunicación por medio de RS232 este activada; puesto que PTW-UNIDOS, no lo tiene activado por defecto.

Todos los datos acerca del funcionamiento y configuración de la cámara de ionización PTW-UNIDOS se obtuvieron desde el manual de usuario elaborado por el proveedor de dicho instrumento. Para la configuración de la utilización del dosímetro por medio del protocolo RS232, se utilizó el documento "User Manual UNIDOS webline Interfaces" [25].

Una vez asignados los valores requeridos por el código del módulo de control de dosis de microtrón, ya se podrán llevar a cabo medidas de manera activa utilizando el dosímetro, o cámara de ionización.

Fig. 5.1: *Cámara de ionización PTW-UNIDOS.*

Cabe resaltar que el sistema de instrucciones request/response, que emplea el sistema de dosimetría, debe estar acompañado siempre de dos símbolos de la tabla ASCII tanto para la emisión de una instrucción desde el MCD guiado por Arduino, como para la recepción de las repuestas del sistema PTW-UNIDOS. Estos dos iconos son CR (carriage return) y LF (line feed), los elementos 13 y 10 de la tabla ASCII respectivamente [26]; además, siempre han de ir en ese orden específico; por lo tanto, es necesario su tratamiento tanto para la emisión de datos como para la recepción. De esta manera

cualquier mensaje emitido o recibido, llevaría agregado al final una extensión de dos iconos declarados como CR+LF.

6. Programación del microprocesador

El módulo de control de dosis del microtrón de la UPC, está basado en la placa Arduino Leonardo Ethernet; el cual, contiene un microprocesador ATmega32U4. Mediante la programación de este circuito integrado, es como se calculará y controlará la dosis, o el tiempo, irradiados; además, de monitorizar los distintos procesos que podrían inducir a algún error a lo largo de la sesión. Para ello, se ha llevado a cabo un planteamiento de programación estructurada [27], dividida en distintas carpetas destinadas a las distintas funciones que deberá llevar a cabo la placa Arduino Leonardo Ethernet. Por ejemplo, la capeta denominada "XXXXX_Service" está dirigida al almacenado de variables; las cuales, contendrán los datos para el cálculo de la dosis, o el tiempo, en base a una dosis dada, o un tiempo establecido por el operador. Además del concepto estructural del microprocesador, también ha sido establecido un sistema de control de procesos a lo largo de la sesión. Estos sistemas, se describen a lo largo de este apartado 6.

6.1. Control de la sesión de irradiación

El control de la sesión dentro del programa es un sistema combinado de varios de los módulos, o carpetas; el cual, se basa en alteraciones de distintas variables, con el objetivo de indicar al operador el estado de la sesión en cada momento del proceso; además, de guardarlo para su posterior utilización, o comprobación. Fue introducido con la idea de mantener una trazabilidad dentro de un entorno controlado; no obstante, en el caso de los microprocesadores, generalmente efímero; puesto que, normalmente no se guardan datos complejos de sesiones anteriores. Además, también se encarga del control a lo largo del proceso de ejecución; de tal manera que, en caso de alguna anomalía o error, el microprocesador detuviese o finalizase el proceso de un modo óptimo y seguro. Para ello, se hizo uso de las distintas variaciones de memoria con las que cuenta este microprocesador ATmega32U4, explicadas en el apartado 4.3: la memoria flash, EEPROM y RAM.

La única de esas tres que nos permite una reescritura sencilla y sin problemas de recuperación es la EEPROM; por lo tanto, se optó por esta memoria para hacer la escritura de los datos de la última sesión realizada. De este modo, se utilizan las funciones habilitadas por la librería EEPROM.h; ya incluida en la IDE de Arduino, concretamente, EEPROM.get(dato, pos.memoría) y EEPROM.put(dato,

pos.memoria). Este proceso se lleva a cabo de forma automatizada en el caso de la escritura; no obstante, requiere de una petición por parte del operador para la recuperación de los datos de sesión anterior. De esta manera, la orden PRSS permite al operador la visualización de los datos adquiridos y tratados en la sesión inmediatamente anterior.

Por otro lado, el control de la sesión se hace de forma dinámica a lo largo de todo el tiempo de ejecución; para ello, el programa mantiene una estructura guiada por el uso de "flags". Una flag, o bandera, "es una variable que se utiliza para conservar el estado, verdadero o falso, de una condición" [28]. Dependiendo de las flags validadas, el módulo de control de dosis del microtrón habilita, o no, ciertas opciones al operador. Como ya se ha descrito en el apartado 5, el MCD maneja cierto rango de valores admisibles en las variables, en función de que opción de funcionamiento se escoja para detectar o calcular la dosis, o el tiempo; además, cada vez que un parámetro es introducido y, se encuentra dentro del rango, o del formato correcto, una flag pasa del estado "false", al estado "true". De esta manera, el MCD conoce el estado de la variable modificada dentro del proceso y; debido a ello, habilita o deshabilita el acceso a ciertas configuraciones posibles dentro del programa. En el caso del módulo de control de dosis del microtrón, se distinguen dos tipos de flag, las de verificación de variables y las de acceso a procesos.

- Verificación de variables: Los booleanos dedicados a la verificación de las variables, únicamente se activan en el momento en el que es configurado el valor de una variable dentro del rango adecuado de los valores admisibles por esa variable. Al validarse estos booleanos, simplemente se confirman las variables que el operador ha ido completando a lo largo de la sesión (Fig. 6.1); no obstante, algunas de estas flag se pueden considerar, también, del formato de acceso a procesos.

```
if (strncmp(reg, "TRAD", 4) == 0) {
  if (act[1] == 'R') {
    resp += TString;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    TimeDeliveryCatcher();;
    resp += TString;
  }
  else {
    badcmd;
  }
  goto endsub;
}
void TimeDeliveryCatcher() {
  T = flParam;
  strcpy(TString, stParam);
  DeliveryRequestValidator = true;
}
```

Fig. 6.1: *Ejemplo de flag para verificar la introducción de una variable.*

- Acceso a procesos: Al margen de las anteriores, hay un formato de flag que al validarse dan pie al inicio de ciertos procesos dentro del programa en ejecución. Este es el caso, por ejemplo, del flag que se activa cuando el operador introduce un valor dentro del rango de verificación

de la variable METD. Dependiendo del valor que se introduzca, se validará una de las diferentes flag asociadas a esta variable; haciendo así, que el MCD redirija el proceso hacía uno de los tres formatos de cálculo y medida de dosis, descritos en el apartado 1 del capítulo 5 con los que cuenta el código.

Además de las flag para el control de los accesos a ciertos procesos, el MCD cuenta con una variable que indica el estado en el cual se encuentra la sesión en todo momento. Está variable denominada como SSTA, tiene hasta seis estados posibles dependiendo del momento en el que se encuentre el proceso y, de la interacción del operador con el módulo de control de dosis del microtrón. De esta manera, cada vez que se comienza una nueva sesión, el valor de esta variable se reinicia y, cada vez que ocurre un error que obliga al MCD a interrumpir algunos de sus procesos a lo largo de la sesión que se esté llevando a cabo, la variable SSTA cambia de valor, indicando el tipo de excepción que se ha detectado durante el proceso. Los valores que puede adoptar SSTA y sus definiciones, están descritos en la tabla 6.1.

Tabla 6.1: *Definición del estado de la variable SSTA.*

Valores SSTA	Definición
0	La sesión no ha sido realizada.
1	La sesión está en marcha.
2	La sesión ha finalizado correctamente.
3	La sesión ha sido interrumpida por el operador.
4	La sesión ha sido interrumpida por el MCD.
5	Pérdida de comunicación durante la sesión.

6.2. Comunicación con la consola del operador

Como ya ha sido explicado en el apartado 4, la comunicación entre el microprocesador y la consola del operador se da en un formato de pregunta y respuesta, conocido como request/response en el mundo de la programación. No obstante, para llevar a cabo esta modalidad de comunicación, se debió generar un entorno que hiciese posible una interacción segura y precisa. En el programa, dependiendo del formato de comunicación que se desee utilizar; puesto que, en el caso del módulo de control de dosis del microtrón, hay una versión en la que comunicación se lleva cabo a través del puerto serie (COM), y otra versión se comunica mediante Ethernet, utilizando una de las dos funciones que se describen a continuación.

- Comunicación puerto serie: En el caso de la comunicación puerto serie [29], el operador envía comandas al microprocesador utilizando el terminal del ordenador a 115200 bits por segundo. Para recibir los mensajes, el microprocesador utiliza la función comandoSerial(); la cual, devuelve un byte. El byte será 1 en el caso de que el mensaje haya sido procesado con éxito y, en la misma función, se fracciona, mediante el uso de punteros, para su posterior procesado en la función SerialTranslator(). En caso de que el byte devuelto sea 0, simplemente el programa continúa devolviendo una de las posibles respuestas de error la pantalla de monitorización serie.
  Para poder interactuar entre el ordenador y el operador simplemente se necesita el cable de programación de puerto serie/USB; además, el código es compatible tanto para Arduino Uno, como para Arduino Leonardo Ethernet.
- Comunicación Ethernet: Por otro lado, cuando se está utilizando el modelo de comunicación Ethernet [30], la interacción entre el MCD y el operador; pese a ser similar, requiere la utilización de más material, al utilizar un cable Cat5, o Cat6, con un conector RJ45, y deja de ser compatible con el Arduino Uno. No obstante, la interacción necesaria una vez elaborado el montaje es similar, el operador envía instrucciones mediante la escritura en el terminal y, el código evalúa la información recibida en la función comandoEthernet(). Esta función responde con un byte, en el caso de que sea un 1, significará que la instrucción pasa a ser procesable por la función EthernetTranslator() y, dependiendo de la instrucción, el MCD reaccionará de un modo u otro. En el caso de que la respuesta sea 0, el código devolverá una de las posibles respuestas de error y seguirá con su rutina habitual.

Cuando el operador envía una instrucción errónea, el programa devuelve una respuesta dependiendo del motivo del error. Está respuesta ante el error, está diseñada para que el operador sea consciente de que la orden no ha sido procesada, o es inválida y, generar de este modo un "feedback" más completo entre el proceso y el operador. Las respuestas posibles son las siguientes y son devueltas por la función de procesado de las instrucciones, no por ninguna de las funciones

de recepción denominadas comandoXXXXX, donde "XXXXXX" es el nombre del tipo de comunicación:

- Comando inexistente → (#/comandoerróneo): En este caso, la función comandoXXXXX, ha captado un mensaje; no obstante, cuando ha sido procesado, no se ha encontrado coherencia entre el comando escrito y ninguna de las opciones planteadas en el MCD. Por lo tanto, el sistema muestra al operador el mensaje dónde "comandoerróneo", representa el comando escrito.
- Comando incongruente → (!/comandoerróneo): Por otro lado, en esta situación sí que se ha encontrado una opción que sea coherente con la instrucción enviada; no obstante, el valor aportado no está dentro de los que esa opción admite en su rango de valores específicos. Dentro del código, esto se representa con "comandoerróneo" como la instrucción enviada; puesto que, ha fallado y el código continúa con su rutina habitual, a la espera de instrucciones.

De la forma en la que está estructurado el código, solo se admite una instrucción por petición; esto es, en el caso de que se enviasen varias órdenes simultaneas, el MCD solo leería la primera y desecharía las posteriores. Esto se debe a que, el modelo de comunicación requiere del empleo de una estructura en la cual, el símbolo '(' representa el inicio del mensaje y, ')' el final de este; los elementos 40 y 41 [26], respectivamente, en la tabla ASCII expandida. A continuación, se muestra la consola del operador de Arduino, con la cual se emiten las instrucciones y también se reciben cuando se utiliza la versión con comunicación puerto serie (Fig. 6.2):

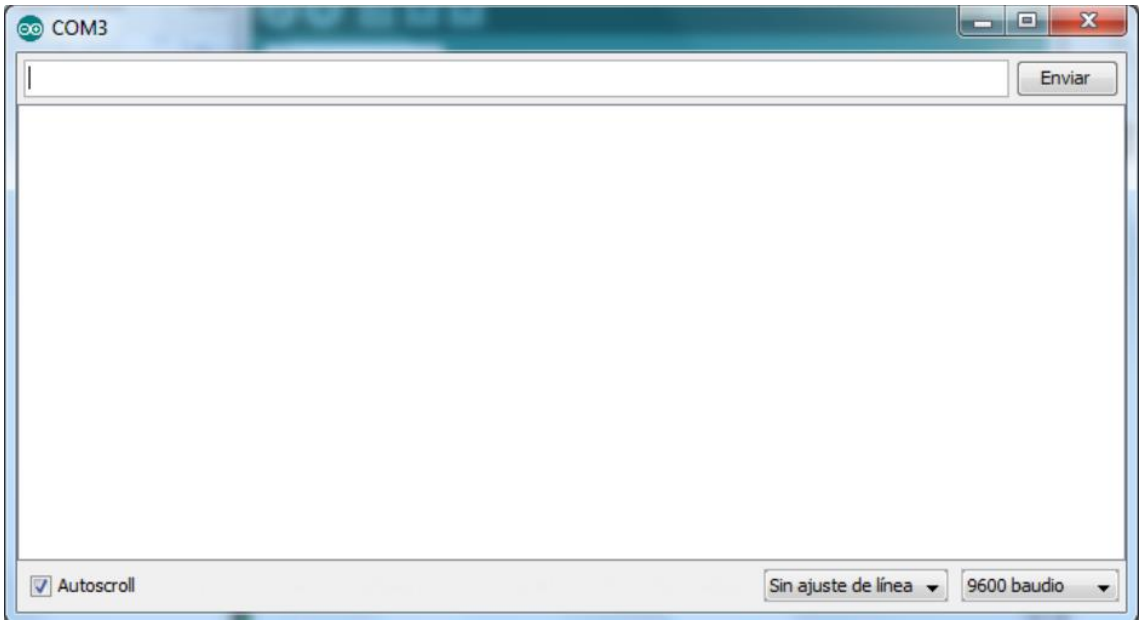


Fig. 6.2: *Monitor serie del entorno Arduino*

6.3. Comunicación RS232 con el dosímetro

A lo largo del capítulo 4 (apartado 3), se especifican los elementos que se emplean para llevar a cabo la comunicación RS232; para ello, se emplea un circuito integrado modelo MAX3232, que hace de transceptor y, el esquema de montaje del mismo. El protoshield en el cual va soldado ese sistema se acopla a la placa Arduino; de esta manera, completando el módulo de control de dosis del microtrón y, habilitando la opción de control por medio del dosímetro PTW-UNIDOS cuando METD adquiere el valor 20, debido a una asignación por parte del operador.

Para que el MCD este operativo y, este habilitada la opción de control de dosis por medio de la cámara de ionización, el operador deberá introducir ciertos datos requeridos por el programa para permitir la emisión. De este manera, el MCD debe haber obtenido un código de sesión (SECD), el modo de irradiación (IMOD), la dosis, o el tiempo, a irradiar (TRAD/DRAD) y asignar a la variable METD el valor 20. Una vez hecho esto, el sistema comprobará si la cámara de ionización está conectada; y también, si es viable comenzar a leer datos desde ella.

El manejo del dosímetro PTW-UNIDOS es completamente autónomo una vez que el operador introduce los valores para configurar la sesión y activa el modo “Beam On”; de este modo, el operador únicamente debe manejar las respuestas de la instrucción PING, o en el caso de que lo vea oportuno, parar la emisión del haz.

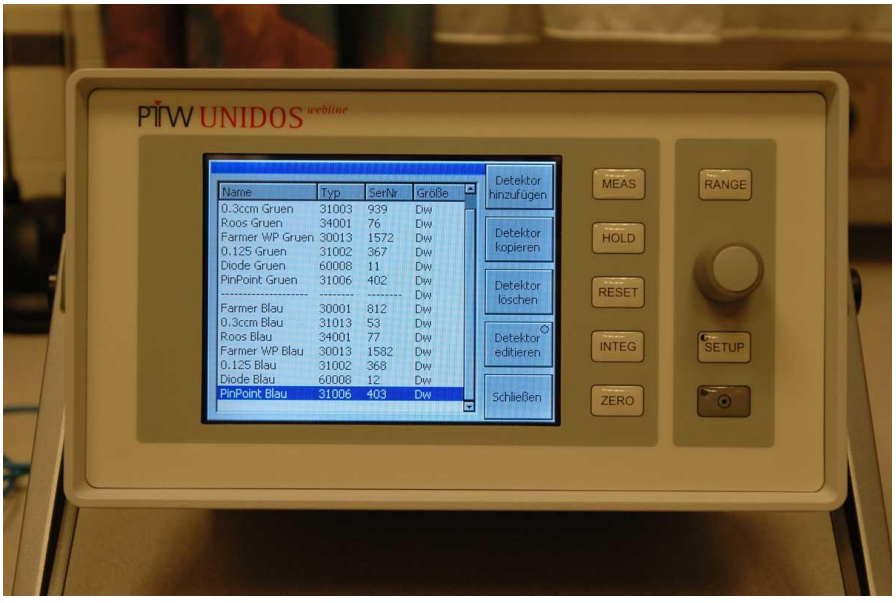


Fig. 6.3: *Ionization chamber.*

Para iniciar la comunicación RS232 con la cámara de ionización (Fig. 6.3) deben hacerse dos configuraciones iniciales:

- Configuración física:

  Primero, la cámara de ionización debe estar preparada para comunicar por medio de RS232; para ello, hay que configurarla de una manera específica. Dentro de la UI (User Interface) PTW-UNIDOS, es necesario ajustar el protocolo de comunicación RS232 de "OFF" a "ON"; además, es necesario hacer ajustes para preparar la comunicación. El baud rate (bits/s) y la paridad deben estar acorde con los datos que se empleen en el módulo de control de dosis del microtrón; de otro modo, la comunicación será imprecisa o nula. En el caso del dosímetro, se encuentran por defecto en un baud rate de 57600 bits por segundo y una paridad de 8N1. Además, todas las instrucciones que se utilizan a lo largo del programa para comunicarse con la cámara de ionización están estimadas en un tiempo de respuesta menor o igual a 0.5s, salvo "STA" que se estima en menos de 2s.

- Configuración software:

  La configuración software viene implementada en el programa del MCD; esto es, una vez preparadas las condiciones para la utilización de la cámara de ionización, el programa entrará en la función RS232_CheckCommunication(). Mediante esta función, se envía, utilizando el protocolo de comunicación RS232 la instrucción "PTW" para verificar la conexión estable con la cámara de ionización. Este proceso se repite un máximo de tres veces, a la espera de recibir una respuesta con el número de identificación del dosímetro. En caso de que la respuesta no sea la esperada, el programa de Arduino generará un mensaje de error "E"; no obstante, sea o no afirmativa la respuesta de la cámara de ionización, se envía una segunda instrucción definida como "SE". La respuesta de este mensaje debería ser 0; puesto que, aunque la conexión este establecida de forma correcta, la respuesta por parte del dosímetro a esta instrucción indicará si hay algún error en el dosímetro o en la interfaz gráfica del mismo. Por último, en caso de que se haya conseguido una conexión entre cámara de ionización y el módulo de control de dosis del microtrón; además de que, no haya ningún error en el proceso, se enviará una última instrucción de configuración definida como "STA". Este mensaje, inicia la actividad de medida del dosímetro; por lo tanto, a partir de esta instrucción, el sistema estará operativo y remitirá datos al MCD.

Una vez que la configuración haya sido completada, se iniciará la emisión del haz de electrones y el código empezará a leer datos de la cámara de ionización mediante el uso del mismo modelo de pregunta y respuesta que utiliza para la interacción habitual. Esto es, la comprobación por parte del operador del proceso en tiempo de ejecución del programa, únicamente, se deberá al empleo de la instrucción PING, como en el funcionamiento del MCD en el modo normal; sin embargo, el programa empleará la instrucción "MV"; cuya respuesta, será procesada en la función ControlValue(); la cual, tiene como entrada una cadena de texto, proveniente de la respuesta de la cámara de ionización. A continuación, se muestra la tabla 6.2, con el formato de empleo de la instrucción "MV" y respuesta de la cámara de ionización PTW-UNIDOS:

Tabla 6.2: *Instrucciones admitidas por la cámara de ionización.*

Función	Request / Response	Parámetros
Lectura del ID y la versión del firmware	PTW PTW; UNIDOS; x.xx; y	UNIDOS2 ID x.xx firmware y número de compilación (opcional)
Lectura del estado de error	SE SE; x; y	x error en la unidad de medida 0: no error 1: error y error en la alimentación 0: no error 1: error
Comenzar la lectura de la dosis	STA STA	Errores posibles: E; 02 E; 03
Lectura del valor medido	MV MV; a; bb; cc.c; ±d.dddE±ee; f ; g; ±h.hhhE±ii; k; ±llllE±mm; nnnnn	±d.dddE±ee valor de dosis medido ±h.hhhE±ii valor medido de la dosis o corriente.

6.4. Comunicación Ethernet

Para comunicar el MCD por medio de comunicación Ethernet, tan solo era necesario adaptar la comunicación entre el Arduino y el monitor. Para ello es necesaria la utilización de las librerías Ethernet.h y SPI.h; la cual, permite acceder a funciones específicas de la IDE de Arduino que están pensadas para habilitar al microprocesador el manejo de este sistema de comunicación. El programa tiene que utilizar una MAC y una IP para poder funcionar adecuadamente. La dirección MAC (Media Access Control) es un identificador único del dispositivo; por lo que, puede ser aleatorio mientras no se repita para otro módulo; por otro lado, la IP (Internet Protocol) depende de la configuración de la red [31]. En el caso del MCD, también se emplea una máscara de subred; no obstante, de cara al manejo de un servidor y un cliente mediante el uso de la librería Ethernet.h, es opcional.

Una vez asignados, los valores de las direcciones MAC, IP y la máscara de subred, es necesario que el programa genere un servidor en el puerto que vaya a utilizar para comunicarse; en el caso del MCD es el 23 (por defecto es el telnet), inicializarlo y comprobar si algún cliente se comunica por ese puerto fijado. Telnet es un protocolo TCP/IP, que nos permite conectarnos por el terminal con un equipo remoto; es por ello, que se escoge este puerto, para poder conectarse al conjunto de módulos que constituyen el sistema de control del RTM De esta manera, utilizando un cable Cat5 o Cat6, con un conector RJ45, se pueden conectar al mismo puerto todos los módulos del microtrón. En la siguiente imagen (Fig. 6.4), se puede observar la parte del código que conecta con el servidor.

```
/// MAC address
//
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xE1, 0x22 };

/// IP address
//
byte ip[] = { 10, 0, 0, 14 };

/// Mask
//
byte subnet[] = { 255, 0, 0, 0 };

/// Port to listen (default 23, Telnet service)
//
EthernetServer server = EthernetServer(23);
```

Fig. 6.4: *Configuración versión Ethernet del MCD.*

6.5. Programación del microcontrolador Arduino

El código que controla el microprocesador del MCD está dividido en distintos sketches dentro de la misma solución, con el objetivo de ser más comprensible y mantenible en el tiempo. Además, tanto la nomenclatura de las distintas variables, como de las funciones y los sketches, busca ser clara y precisa; de tal manera, que pese a encontrarse por primera vez con el programa, resulté relativamente sencillo ubicar cada uno de las funciones y las variables a las cuales se hagan referencia. Tanto el programa que se comunica mediante Ethernet, como el que hace lo mismo mediante comunicación por medio del puerto serie, tienen el mismo número de sketches que se describen en la figura siguiente (Fig. 6.5):

Fig. 6.5: *Sketches dentro de la solución para Arduino (protocolo de comunicación por puerto serie).*

6.5.1. Programa Puerto Serie

El programa que se comunica por el protocolo de comunicación mediante el uso puerto serie para la interacción con el operador y por medio del protocolo RS232 para el control de la medida por medio del dosímetro mantiene la siguiente estructura:

- ProgramaDetector_Service: Este sketch se encuentra destinado al manejo de los servicios. En su interior, no alberga lógica de programación concerniente al cálculo o el control de la dosis; puesto que, su objetivo es el de dar soporte al programa asignando los valores introducidos por el operador a las distintas variables. Además de eso, también se encarga de validar la mayoría de las flag que verifican variables; permitiendo así, que el programa dirija el proceso durante la sesión a las funciones que se encuentran dentro del sketch ProgramaDetector_Library. La tabla 6.3 especifica el contenido de este sketch:

Tabla 6.3: *Funciones del sketch "Services".*

Función	Variables tratadas	Flag
NewSessionMaker(int), void	SECD	NewSessionMakerValidator, NewSessionNeeded
ModeOperationSelector(),void	IMOD	ModeOperationSelectorValidator
TimeDeliveryCatcher(),void	TRAD	DeliveryRequestValidator
DoseDeliveryCatcher(),void	DRAD	DeliveryRequestValidator
MeasureOperationSelector(),void	METD	MeasureOperationSelectorValidator, RS232_Flag, NewSessionNeeded, NewSessionMakerValidator, ModeOperationSelectorValidator
ConversionMethodCatcher(),void	F	ConversionMethodSelectorValidator
KConstantsCatcher(byte), void	K, K, K2, Frep	KSetupValidator, K1SetupValidator, K2SetupValidator, K3SetupValidator
CTMaxCatcher(), void	CTMax	N/A
SessionEnd(),void	----------	NewSessionMakerValidator, ModeOperationSelectorValidator, DeliveryRequestValidator, MeasureOperationSelectorValidator, ConversionMethodSelectorValidator, NewSessionNeeded, Beam_OnFlag, Beam_OffFlag, RS232_Flag

- ProgramaDetector_Library: Este sketch contiene la lógica que se aplica dentro de los procesos concernientes únicamente al cálculo y control de la dosis; en los cuales, no intervienen ningún elemento externo; es decir, no se hace uso de la cámara de ionización para la medición de la dosis, o el tiempo. La lógica de esta carpeta facilita operaciones que van desde la creación de la sesión, hasta el cálculo de la dosis. Además, también se encarga de controlar el estado de los procesos a lo largo de la sesión y durante la aplicación de la dosis de irradiación. Algunas de las funciones implementados en el sketch (tabla 6.4), son continuaciones de otros implementados en el ProgramaDetector_Services; de tal manera que, cuando ese sketch descrito en el punto anterior necesita aplicar algo de lógica de programación, se redirige a este

sketch para hacerlo. Las funciones implementadas en ese sketch están definidas en la siguiente tabla:

Tabla 6.4: *Funciones del sketch "Library".*

Función	Definición
NewSessionCreation(int), void	Se encarga de asignar un nuevo código de sesión a la variable SECD, e interactúa con la memoria EEPROM actualizando la posición de memoria en la que se define esta variable. Se le llama desde la función del sketch de ProgramaDetector_Service, NewSessionMaker() y tiene una variable de entrada que contiene el código de sesión que será asignado.
EndSessionSafe(), void	Cuando se finaliza una sesión de irradiación, el código llama a esta función. En ella, se lanza un guardado dentro de la memoria EEPROM, manteniendo así la trazabilidad de la sesión inmediatamente anterior. Por otro lado, también hace una llamada a la función de NewSessionCleanUp(), dentro del mismo sketch.
DoseCalculator(int,float,int,unsigned int,float,float), float	Esta función resuelve mediante la entrada de DRAD la dosis, o por medio de la entrada de TRAD el tiempo, a irradiar. Tiene distintas variaciones de cálculo dependientes del IMOD y el METD.
DoseRevelator(), void	Mediante esta función el programa muestra el resultado que se cálcula periódicamente en el DoseCalculator(). Durante el proceso de irradiación el programa llama reiteradamente a esta función para verificar la dosis, o el tiempo que queda por irradiar.
SftEvent(), void	Se utiliza un TIMER en el ProgramaDetector_vX.X; por el cual, se llama a esta función periódicamente. Si el tiempo que

	pasa desde la última instrucción es mayor que el CTMax; entonces, el MCD pasará a estar en "Safe State".
BeamCanStart(), byte	Esta función es un control de seguridad que permite, en caso de que devuelva un 1, activar el haz de electrones. Es una verificación de que todos las flag estén correctamente validadas de cara al inicio de una sesión de irradiación; de esta manera, la seguridad de la sesión no se ve comprometida. Además, también controla el estado de la variable CDON; la cual, permite al operador conocer la disponibilidad del inicio de emisión del haz, a petición.
BeamControlExecuter(), void	Durante el proceso de aplicación de la dosis a irradiar, esta función es la que controla la aplicación; es decir, es la parte del programa que se recorre durante la irradiación por parte del haz de electrones. Hace llamadas a la función DoseCalculator() para calcular la dosis; además, comprueba el estado de la emisión y; en caso de que se quiera anular la emisión, se encarga llamar a la función EndSafe(), para guardar los datos de la sesión.
NewSessionCleanUp(), void	Esta función, se encarga de reiniciar todas las variables que toman parte en cualquiera de los procesos a lo largo de la sesión de irradiación. Se le llama cada vez que se guarda una sesión terminada, o que se reclama una nueva sesión.
PreviewLastSession(), void	Mediante esta función, el programa permite al usuario recuperar los datos de la sesión finalizada inmediatamente anterior a la que se esté llevando a cabo.
KIsSet(), void	Esta función verifica si los factores K han sido asignados; para ello, manipula el valor de la flag ConversionMethodSelectorValidator.
CoherenceValidator(), bool	Es una función, que valida que haya coherencia entre el modo de irradiación, el método de

	cálculo, los factores introducidos para el cálculo, y la dosis, o el tiempo, escogidos por el operador. Esto se debe a bloquear posibles emisiones mal configuradas; de este modo, el MCD no permite la iniciación de la sesión de irradiación en casos en los que no estén bien seleccionados los parámetros de configuración de sesión.
readVcc(), void	Esta función permite leer el voltaje suministrado por Arduino cada vez que el operador lo requiera; por medio de, la instrucción ping.

- ProgramaDetector_LeerComandoRS232: Este sketch se encarga de inicializar y habilitar la comunicación de RS232; no obstante, solo se utiliza en el caso de que la flag RS232_Flag se encuentre validada. Envía y recibe los datos, manteniendo la comunicación abierta con el dosímetro; además, también ejecuta la configuración software requerida y explicada en el apartado 6.3. La tabla 6.5, muestra el contenido de este sketch, que en este momento se encuentra en fase de desarrollo.

Tabla 6.5: *Funciones del sketch para la lectura de instrucciones por RS232.*

Función	Definición
RS232_CheckCommunication(), void	Se encarga de enviar la instrucción “PTW” a la cámara de ionización; dependiendo de la respuesta, o la ausencia de ella, lo llega a hacer un máximo de 3 veces. En caso de que haya una, validará la flag CheckCommunication_Flag y enviará la instrucción “STA”, para que se inicie la toma de datos.
RS232_Communication(), void	Esta función, en el caso de que la flag CheckCommunication_Flag se encuentre validada, enviará una instrucción “MV” a la cámara de ionización para recibir la medida momentánea de la cámara de ionización. En el caso contrario, llamará a la función RS232_CheckCommunication().

RS232cmd(String), void	Mediante esta función, se envían las instrucciones por medio del protocolo de comunicación RS232 a la cámara de ionización.
RS232read(), String	Por otro lado, de manera inversa a la función RS232cmd(String), esta función recibe la información remitida por parte de la cámara de ionización.

- ProgramaDetector_InterpretarRS232: Este sketch está encargado de procesar la información que se recibe durante el proceso de irradiación desde la cámara de ionización. Para ello, previamente esos datos son registrados por el sketch ProgramaDetector_LeerComandoRS232 y, posteriormente son tratados en este sketch (tabla 6.6). Además, también se encarga de controlar la dosis cuando está siendo controlada por el dosímetro. No obstante, este sketch se encuentra en fase de desarrollo y no está completado; de este modo, quedará abierto a modificaciones en el futuro.

Tabla 6.6: *Funciones del sketch para la interpretación de instrucciones por RS232.*

Función	Definición
ControlValues(String), void	Procesa el mensaje recibido desde el dosímetro para poder emplear la información a lo largo de los distintos procesos de control de dosis que se dan a lo largo de la sesión de irradiación.
st2ft(String), float	Debido a la notación científica; que es un formato empleado por la cámara de ionización para enviar cierta información, esta función convierte ese tipo de variable a uno útil durante el proceso.
RS232Control(), void	De la misma modo que actúa el BeamControlExecuter(), dentro del sketch ProgramaDetector_Library, esta función controla todo el proceso de irradiación. También, es capaz de detener, o finalizar el proceso, en caso de que sea necesario.

- LeerComandoSerial: En este sketch se procesa y redirecciona el programa hacía las diferentes funciones (Tabla 6.7) que gestionan las variables que se utilizan a lo largo de la sesión de irradiación dependiendo de la instrucción introducida por el operador. Todas las instrucciones posibles dentro del programa están tratadas en el Anexo A y; al margen de ello, las llamadas a las funciones que complementan a esas instrucciones se hacen desde este sketch.

Tabla 6.7: *Funciones de lectura de instrucciones por medio del puerto serie.*

Funciones	Definición
SerialTranslator(), void	Esta función es el que recibe todas las instrucciones y, dependiendo de la coincidencia de estas con las distintas posibilidades descritas en el Anexo A, se redirigirá el programa hacía una de las diferentes funciones que gestionan las variables de entrada, introducidas por el operador. Además, también se encarga de devolver una respuesta al operador; de tal manera, que se pueda mantener una comunicación entre el operador y el programa. Es necesario añadir, que esa respuesta es enviada siempre y cuando el operador envíe una instrucción al MCD.
badcmd(), void	Remite un mensaje de error en caso de que la instrucción enviada por el operador sea adecuada; no obstante, el valor adjunto no se encuentre en el rango de valores adecuado, o tenga una errata.
badsyn(), void	Remite un mensaje de error en caso de que el valor conjunto a la instrucción enviada por el operador no esté dentro del rango admisible de valores.

- ProgramaDetector_vX.X: Este sketch se encarga del proceso principal del programa; puesto que, en él se generan e inicializan todas las variables globales del programa de funcionamiento del MCD. Además, este sketch alberga la estructura clásica de la IDE de Arduino; en la cual, están definidas las variables globales, junto a dos funciones; los cuales se llaman, setup() y loop(). Todo lo que pasa en la aplicación viene enlazado con la secuencia de procesos que se da en este sketch; debido a que, sin este esquema de ejecución, lo demás quedaría en librerías de funciones sin inicializar. Al margen de las variables, también están definidos e inicializados los TIMER y la función SerialTranslator(), que se encarga de captar los mensajes enviados por puerto serie. Por otro lado, en este sketch también se incluyen las librerías EEPROM.h y SoftwareSerial.h. La tabla 6.8, muestra las distintas funciones que se dan en este sketch.

Tabla 6.8: *Funciones y rutinas principales.*

Función	Definición
setup(), void	En el setup(), se inicializan los TIMER, primero deshabilitando y, después volviendo a habilitar las interrupciones para ese cometido; también, se establecen los parámetros con los cuales se configurarán los distintos canales de comunicación. Esta función por defecto, solo se recorre una única vez en todo el proceso; puesto que, es una de las características de la IDE de Arduino.
loop(), void	El loop es una función que se recorre repetidamente en el programa del MCD; por lo tanto, debe ser el fragmento de código que dependiendo de las flag validadas redirija el programa hacía los distintos métodos. Por otro lado, también se encarga de comprobar si hay mensajes por parte del operador encolados en el buffer de entrada utilizado para la comunicación por medio del puerto serie y de reiniciar las variables de comunicación al final de cada recorrido.
comandoSerial(), byte	Cuando esta función envía una respuesta igual a 1, quiere decir que se ha recibido un mensaje por el canal de comunicación por medio del puerto serie.

6.5.2. Programa Ethernet

Una vez explicada la versión del programa con el protocolo de comunicación por medio del puerto serie; debido a que, entre esa versión y la versión que se basa en un sistema de comunicación mediante el uso de Ethernet, descrita en este apartado, se diferencian únicamente en dos de los sketches, ProgramaDetector_Services, ProgramaDetector_Library, ProgramaDetector_LeerComandoRS232 y ProgramaDetector_InterpretarRS232 no será necesario explicarlos (están definidos en el apartado 6.4.1); sin embargo, tanto ProgramaTFGEthernet como el sketch InterpretarComandoEthernet, serán descritos a lo largo del siguiente apartado. De esta manera, la solución del programa que maneja el MCD por medio de comunicación Ethernet estaría definida tal y como aparece en la siguiente figura (Fig. 6.6):

Fig. 6.6: *Programa del módulo de control de dosis con protocolo de comunicación Ethernet.*

- InterpretarComandoEthernet: De la misma manera que LeerComandoSerial lo hacía en el caso de la versión con comunicación mediante el uso de puerto serie, el proyecto InterpretarComandoEthernet interpreta las instrucciones enviadas por el operador, para posteriormente, dirigir el proceso hacía las funciones (Tabla 6.9) que sean necesarios para hacer funcionar la instrucción pedida. Para ello, se mantiene el modelo request/response, en el cual, a petición del operador, el programa devuelve una respuesta.

Tabla 6.9: *Funciones de lectura de instrucciones por medio de Ethernet.*

Función	Definición
EthernetTranslator(), void	Esta función es el que recibe todas las instrucciones y, dependiendo de la coincidencia de estas con las distintas posibilidades descritas en el Anexo A, se redirigirá el programa hacía una de las diferentes funciones que gestionan las variables de entrada, introducidas por el

	operador. Además, también se encarga de devolver una respuesta al operador; de tal manera, que se pueda mantener una comunicación entre el operador y el programa. Es necesario añadir, que esa respuesta es enviada siempre y cuando el operador envíe una instrucción al MCD.
badcmd(), void	Remite un mensaje de error en caso de que la instrucción enviada por el operador sea adecuada; no obstante, el valor adjunto no se encuentre en el rango de valores adecuado, o tenga una errata.
badsyn(), void	Remite un mensaje de error en caso de que el valor conjunto a la instrucción enviada por el operador no esté dentro del rango admisible de valores.

- ProgramaTFGEthernet: Es este sketch es el que contiene la estructura principal dentro de la versión con comunicación Ethernet del programa; aunque esta versión sea semejante a la que cuenta con un modelo de comunicación basado en el puerto serie, en esta versión se utiliza una librería más incluida en la IDE de Arduino, Ethernet.h. También está compuesto por las dos funciones setup() y loop(), que se muestran en la tabla 6.10.

Tabla 6.10: *Funciones y rutinas principales del programa manejado por Ethernet.*

Función	Definición
setup(), void	En el setup(), se inicializan los TIMER, deshabilitando y volviendo a habilitar las interrupciones; también, se establecen los parámetros con los cuales se configurarán los distintos canales de comunicación. Esta función, por defecto, solo se recorre una única vez en todo el proceso; puesto que, es una de las características de la IDE de Arduino.

loop(), void	El loop es una función que se recorre repetidamente en el MCD; por lo tanto, debe ser el fragmento de código que dependiendo de las flag validadas guíe el proceso a las distintas funciones posibles. Por otro lado, también se encarga de comprobar si hay mensajes por parte del operador por medio de Ethernet y, de reinicializar las variables de comunicación al final de cada recorrido.
comandoEthernet(), byte	Dentro de esta función se genera un cliente para poder añadirse a la cola de mensajería generada por el sistema Ethernet. Cuando esta función envía una respuesta igual a 1, quiere decir que se ha recibido un mensaje por el canal de comunicación mediante el uso de Ethernet. Además, dentro de esta función se procesa la instrucción para poder interpretarla en el sketch EthernetTranslator y, se redirige el código para este cometido.

7. Pruebas de funcionamiento del código

Para verificar que el programa que controla el módulo de control de dosis del microtrón funcionaba de manera óptima y eficiente, se llevaron a cabo distintos tipos de prueba. No obstante, la mayoría de las pruebas se dieron de forma experimental; puesto que, la IDE de Arduino, no cuenta con ningún sistema de depurado ("debug") que permita tratar las funciones de manera aislada, con tal de poder observar su comportamiento por separado y de forma unitaria; evitando así, las pruebas en tiempo de ejecución. Este matiz, ralentiza la resolución de problemas, u obstáculos; debido a que, cada vez que se da un error, es necesario reproducir el proceso entero en busca de un cambio de comportamiento en el programa. En el caso de poder hacer pruebas unitarias [32], el proceso se agiliza, permitiendo al desarrollador buscar errores en líneas aisladas del código; haciendo de este modo, un código más seguro y mantenible en menos tiempo. Al margen de las pruebas del programa utilizando Arduino como plataforma para poder validar el código, se hicieron comprobaciones mediante el uso de software para validar que los cálculos empleados eran correctos; los cuales, están explicados en el apartado siguiente.

7.1. Pruebas de cálculo

La verificación de que los distintos cálculos que permiten la obtención de la dosis, o el tiempo, a irradiar durante la sesión llevada a cabo por el MCD, se ha realizado por medio del entorno de desarrollo software Matlab. De esta manera, se obtuvieron gráficos que permitieron observar la evolución de las distintas variables conforme el transcurso de la sesión, teniendo en cuenta, tanto el modo de irradiación seleccionado por el operador, como el método de cálculo; también, elegido por el mismo. Debido a esas métricas, se puede apreciar el comportamiento lineal de las funciones; de tal manera que, el control de la dosis irradiada se ve simplificado en el código. A continuación, se explicarán las gráficas logradas en cada una de las posibles sesiones a realizar.

Cálculo de la dosis:

- **METD = 11:** Para visualizarlo se puede observar la figura (Fig. 7.1); en la cual, el eje de las coordenadas representa el tiempo en segundos y, el de las ordenadas la dosis en Grays.

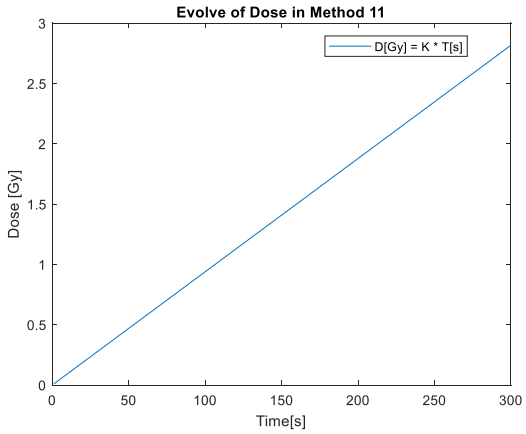


Fig. 7.1: Evolución de dosis fija mediante el método de cálculo 11.

- **METD = 12 (Cuando el factor K es igual a 9400Gy/s):** En este caso, se emplea un factor calculado previamente por el operador, reflejándose en los gráficos de la misma forma que se puede apreciar en la siguiente figura (Fig. 7.2):

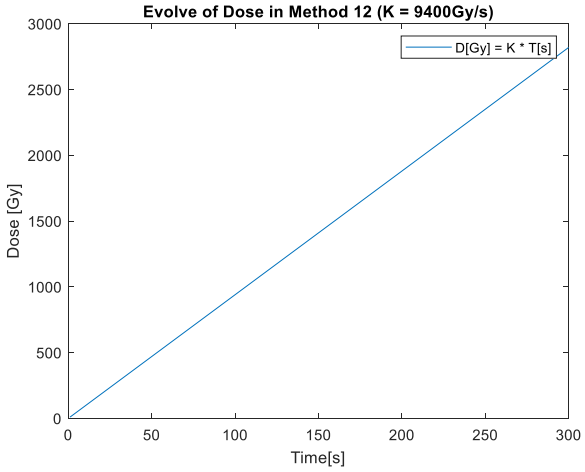


Fig. 7.2: Evolución de una dosis fija mediante el método de cálculo 12, cuando K es equivalente a 9400Gy/s.

- **METD = 12 (Cuando el factor K es igual a 24Gy/s):** Del mismo modo (Fig. 7.3), se calcula la dosis con un factor K diferente, para poder observar un comportamiento diferente de la dosis irradiada que mantenga la linealidad.

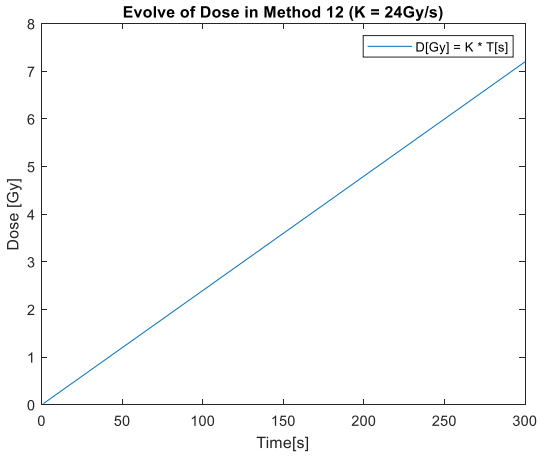


Fig. 7.3: Evolución de una dosis fija mediante el método de cálculo 12, cuando K es equivalente a 24Gy/s.

Cálculo del tiempo:

- **METD = 11:** Para el cálculo del tiempo, es necesario modificar la ecuación; no obstante, los ejes se mantienen en la siguiente figura (Fig. 7.4).

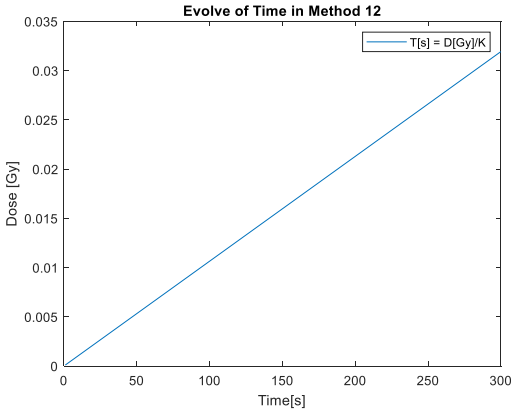


Fig. 7.4: Evolución de un tiempo fijo mediante el método de cálculo 11.

- **METD = 12 (Cuando el factor K es equivalente a 9400Gy/s):** Este caso se muestra en la siguiente figura (Fig. 7.5), para la cual, el operador deberá introducir el factor K.

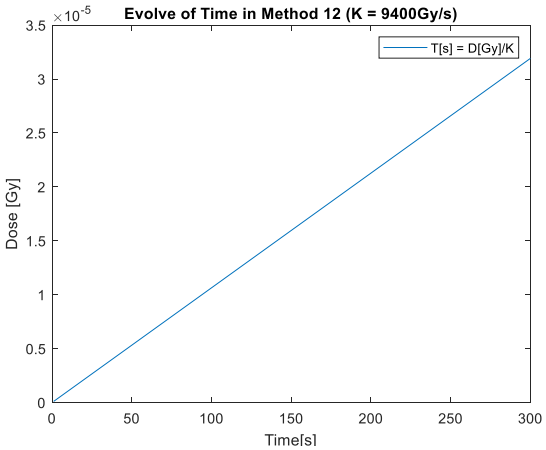


Fig. 7.5: Evolución de un tiempo fijo mediante el método de cálculo 12, cuando K es equivalente a 9400Gy/s.

- **METD = 12 (Cuando el factor K es equivalente a 24Gy/s):** Este último caso, se muestra en la figura (Fig. 7.6) siguiente.

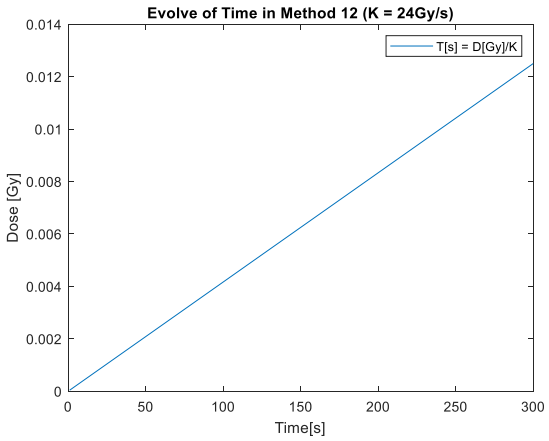


Fig. 7.6: *Evolución de un tiempo fijo mediante el método de cálculo 12, cuando K es equivalente a 24Gy/s.*

El código empleado para el cálculo y obtención de las distintas métricas mostradas se encuentra adjunto a estos documentos en el Anexo B3.

7.2. Pruebas del programa

En el caso de las pruebas con el programa que manipula el módulo de control de dosis del microtrón, se ha empleado un sistema empírico de estudio; por el cual, mediante la experimentación, se han ido completando las distintas instrucciones que puede llevar a cabo el módulo. Las pruebas se basan en la utilización de la IDE de Arduino para la verificación de que las distintas instrucciones son recibidas por el microcontrolador y, a su vez, las respuestas esperadas en el modelo request/response, cumplen con el estándar a seguir para poder adaptarse a una utilización habitual por un operador.

7.2.1. Pruebas en modo normal

En la figura 7.7 y en la figura 7.8, se puede observar el comportamiento mínimo que debe tener el programa del MCD, en ellas se muestran las instrucciones ping y su respuesta, respectivamente. Los datos que deben aparecer, como se especifica en el apartado 1 del capítulo 5, son FMST/DCON/SCDE/TRAD/DRAD/SSTA/IM/MD/TSES/DSES/K1/K2/K3/K/F/DBUG/VCC; de tal manera que, la instrucción ping, responda con la información suficiente como para que el operador tenga constancia de los cambios que se efectúan en el transcurso de la sesión de irradiación.

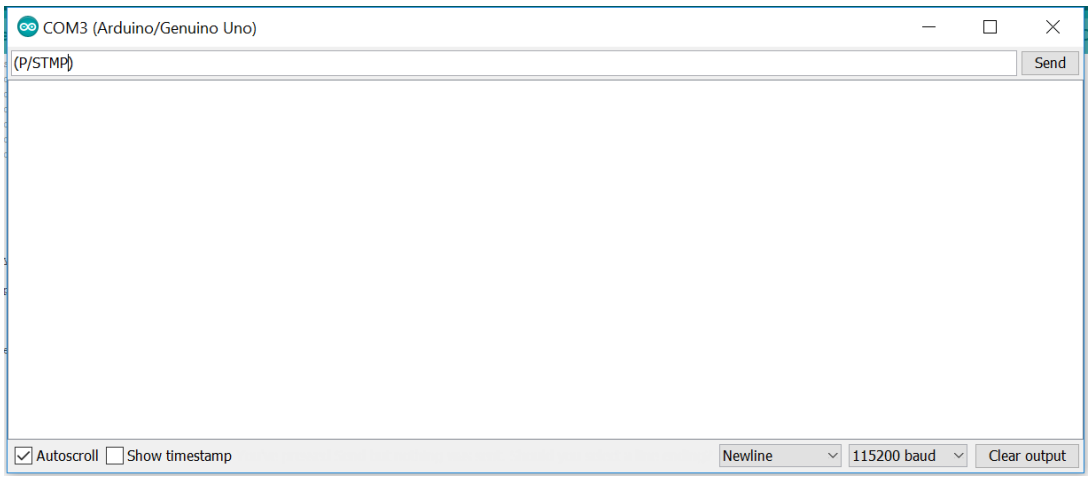


Fig. 7.7: *Instrucción "ping".*

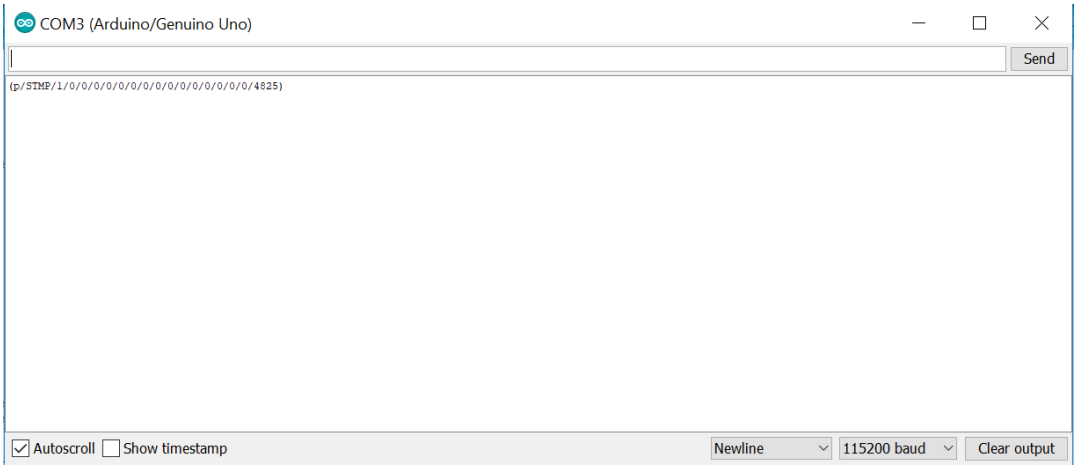


Fig. 7.8: *Respuesta instrucción "ping".*

El último dato respondido (Fig. 7.8), correspondiente a Vcc, nos responde con el voltaje saliente del Arduino.

El transcurso normal de la sesión sería la siguiente, el operador iniciaría una nueva sesión (Fig. 7.9); a la cual, posteriormente le asignaría un código de sesión (Fig. 7.10) para reconocerla y hacer un seguimiento de ella.

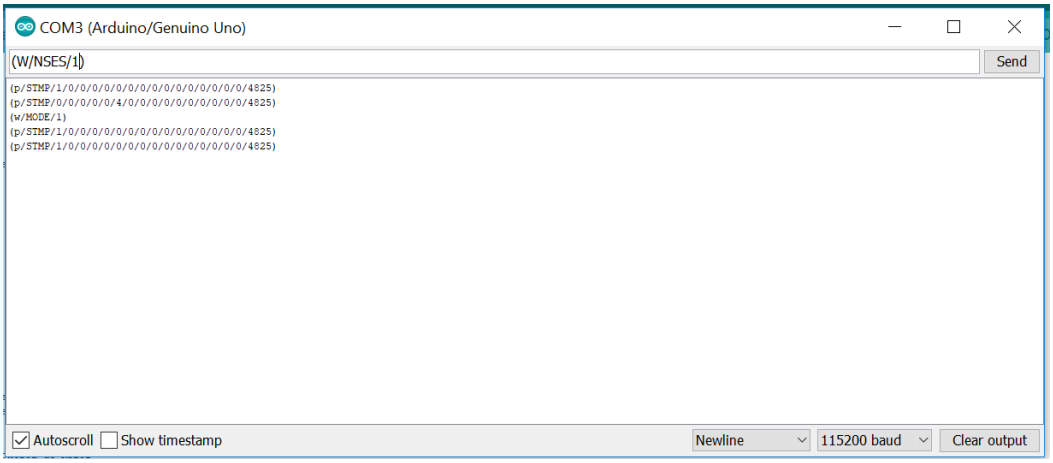


Fig. 7.9: *Instrucción NSES, para generar una nueva sesión.*

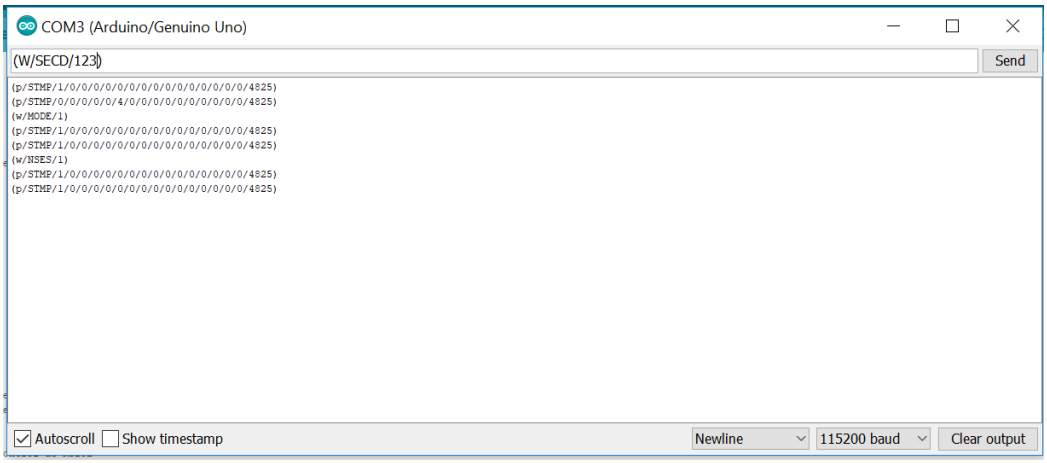


Fig. 7.10: *Instrucción SECD, para asignar un código de sesión.*

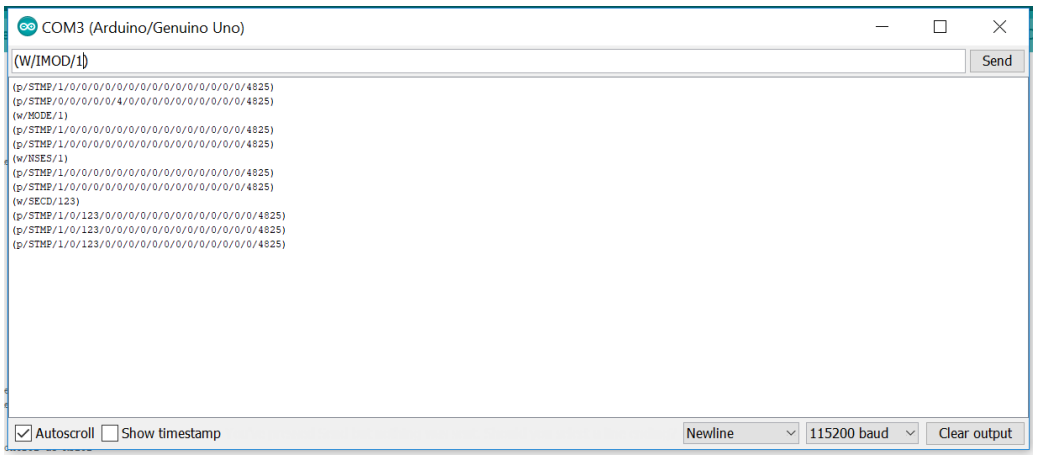


Fig. 7.11: *Instrucción IMOD, para asignar el modo de irradiación.*

Una vez inicializada la sesión, el operador debería introducir los datos necesarios en función de la sesión que se vaya a llevar a cabo; es decir, dependiendo de qué modo de irradiación vaya a emplear y de que método de cálculo requiera para hacer el seguimiento de la sesión. En la figura 7.11 se muestra la instrucción; con la cual, el operador introduciría el modo de irradiación (en este caso 1) y, en la siguiente figura (Fig. 7.12) se muestra la misma operación de selección con el método de cálculo seleccionado (en este caso 12).

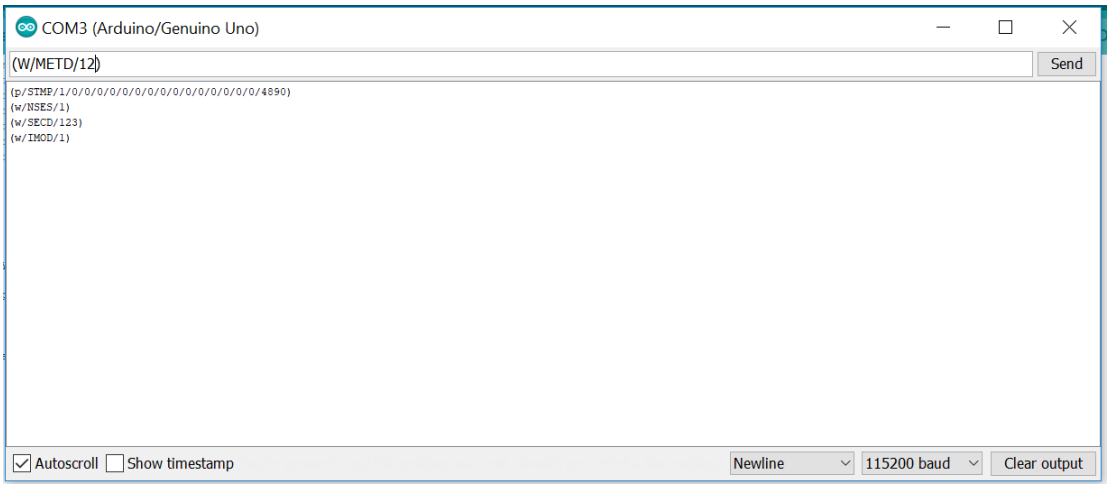


Fig. 7.12: *Instrucción METD, para seleccionar el método de cálculo.*

Una vez escogidos esos dos últimos parámetros, el operador necesitaría introducir tanto el tiempo a irradiar (Fig. 7.13) (debido a que se está probando el caso de tiempo fijo), como el factor de conversión mediante la instrucción KSET (Fig. 7.14) (teniendo en cuenta que se está probando el método de cálculo 12)

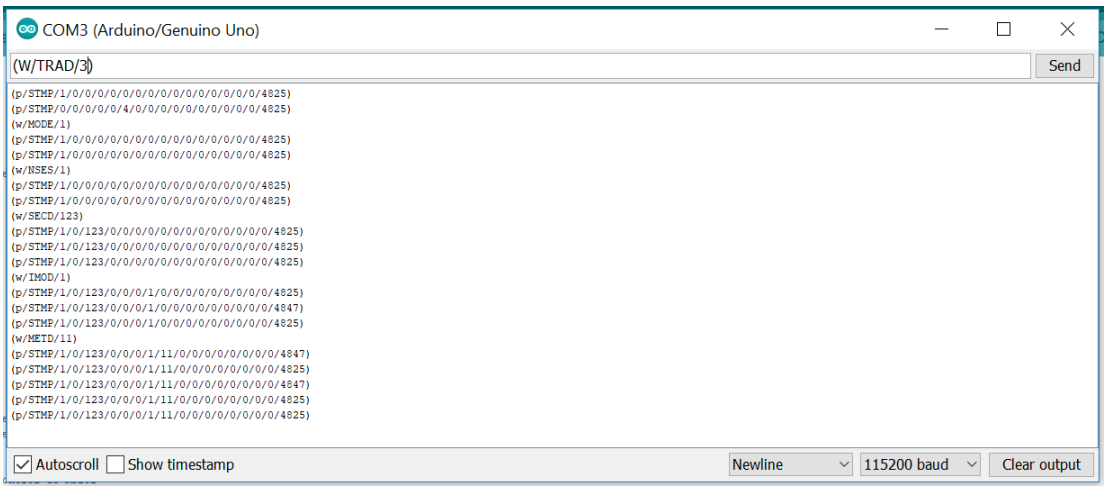


Fig. 7.13: *Instrucción TRAD, para asignar un tiempo fijo.*

Al haber introducido todos los valores necesarios para completar la inicialización de los parámetros necesarios en la sesión de irradiación, la variable CDON, cambiará su valor de 0 a 1; de esta manera, en cuarta posición de la respuesta del ping (Fig. 7.15), se podrá observar dicho cambio. Esto significará que la sesión está lista para realizarse y el operador sabrá que la instrucción de emisión del haz estará lista para utilizarse.

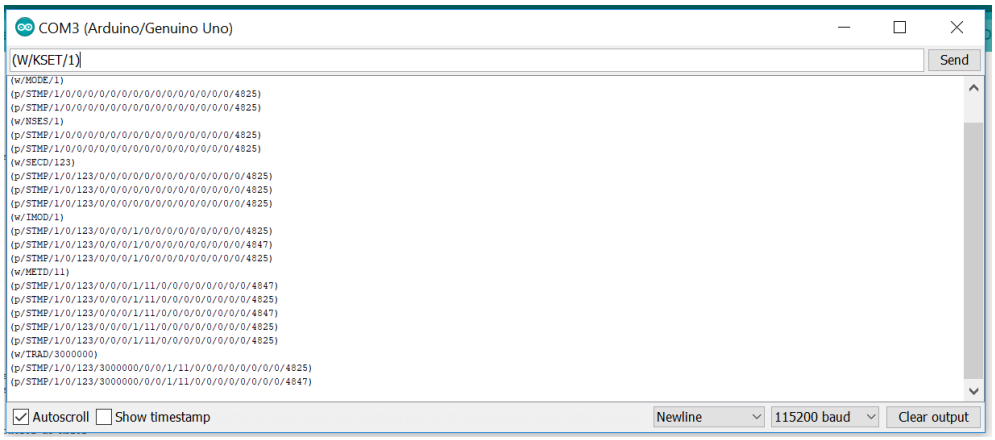


Fig. 7.14: *Instrucción KSET, para asignar el valor al factor de conversión en METD (12)*

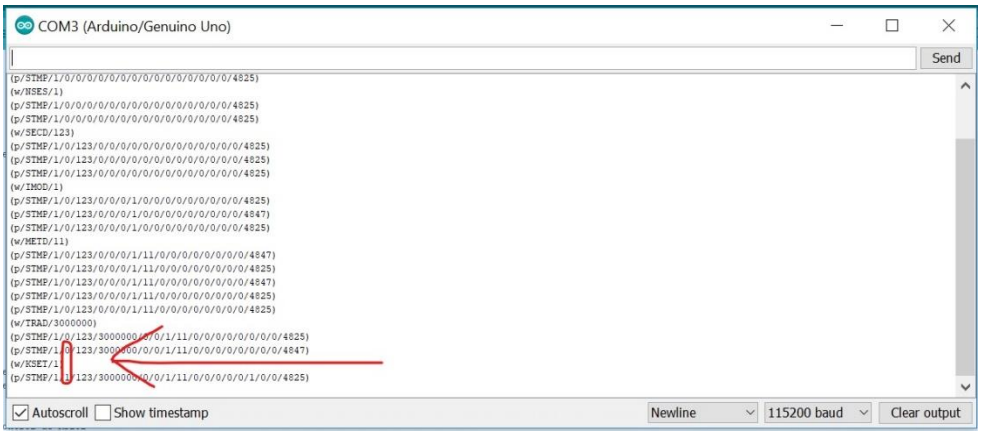


Fig. 7.15: *Cambio en la variable CDON, conforme a que el haz está listo para emitirse.*

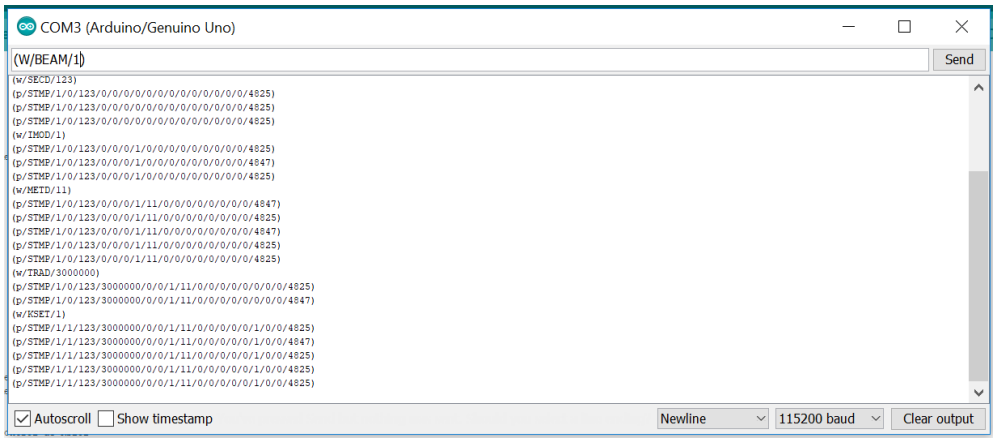


Fig. 7.16: *Instrucción BEAM, para la activación de la emisión del haz.*

Por último, se inicializará la emisión del haz; con lo cual, se cerrará el relé, con la instrucción (W/BEAM/1) (Fig. 7.16) y se observará el comportamiento habitual del programa del MCD, cuando la sesión no se ve interrumpida hasta terminar la sesión configurada por el operador. Los mensajes que aparecen en la imagen (Fig. 7.17), únicamente son de referencia para el operador, en una hipotética adaptación a LabView no aparecerían.

Por otro lado, si el operador quisiera parar la sesión antes de tiempo tendría que introducir la instrucción (W/BEAM/0); de este modo, cambiando el estado de SSTA y dejando la sesión sin terminar. En este caso, para que la sesión se pudiese proseguir, el operador debería introducir los datos de modo que se iniciase una nueva sesión. A continuación, en la figura 7.18, se pueden ver los datos de una sesión iniciada; en la cual, ya se ha introducido un código de sesión, el modo de irradiación es mediante la introducción de una dosis fija, el modo de cálculo es equivalente a 11; por lo cual, se introducen tanto K1, como K2 y la frecuencia de repetición de los pulsos y, ya se ha introducido una dosis fijada para, posteriormente, detener la sesión.

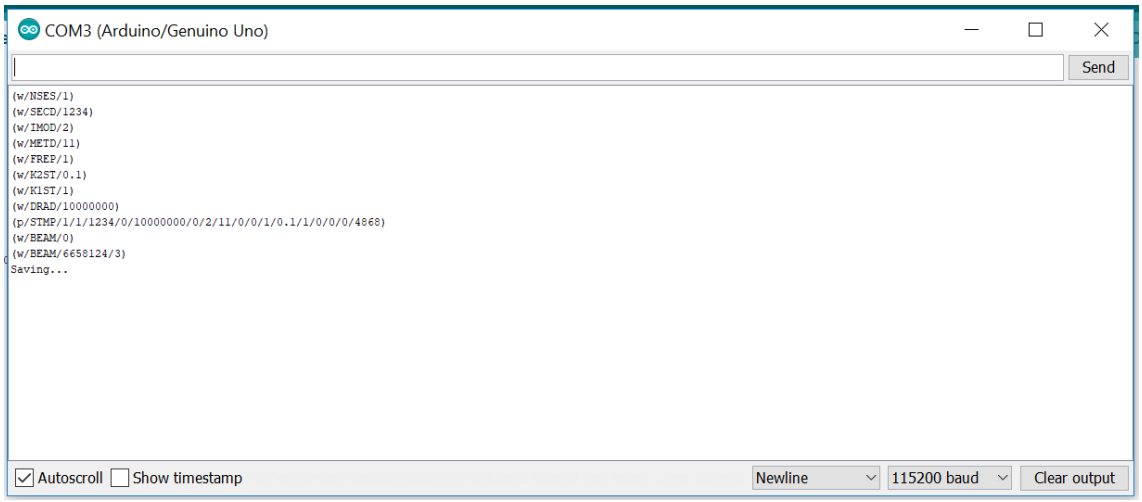


Fig. 7.17: *Sesión finalizada por el operador.*

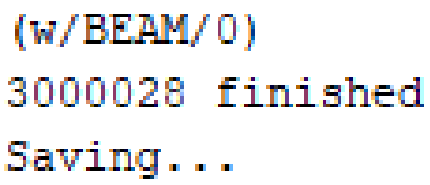


Fig. 7.18: *Final de la sesión de irradiación.*

Además, a lo largo de la sesión de irradiación el MCD devuelve al operador información de los procesos que se estén llevando a cabo, por medio de la instrucción ping (Fig. 7.19); de esta manera, en caso de que el operador quisiese detener el haz, tendría en su disposición los datos de sesión hasta ese momento. En la figura 7.19, se puede observar el cambio de valor del parámetro SSTA (que índica el estado de la sesión) y la evolución de la dosis irradiada.

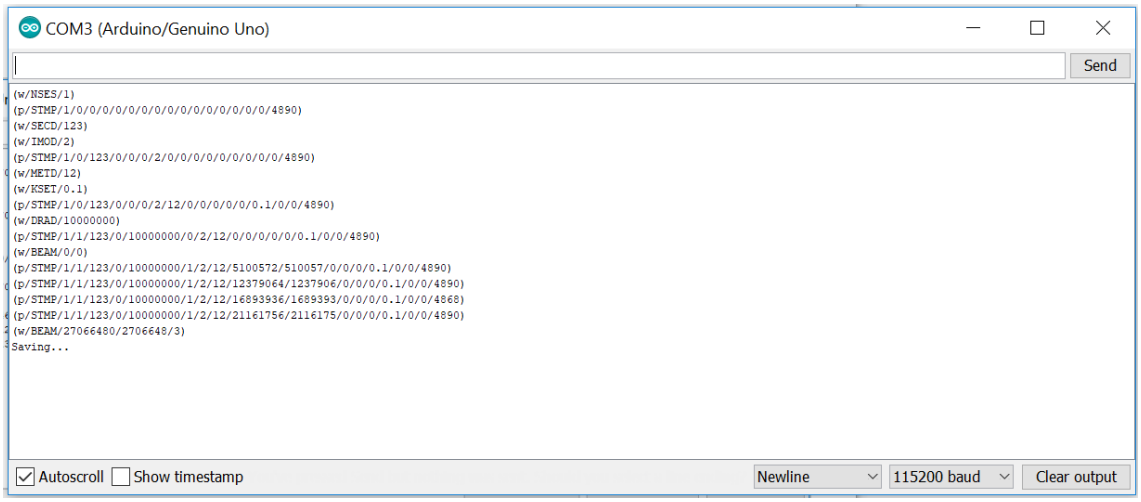


Fig. 7.19: *Sesión finalizada por el operador con múltiples instrucciones ping.*

7.2.2. Pruebas con el depurado activado

Para poder analizar el comportamiento del programa con más detalle; es decir, en busca de errores que son difíciles de hallar mediante la lectura del código, se utiliza un “debug”, o depurado. Al mandar

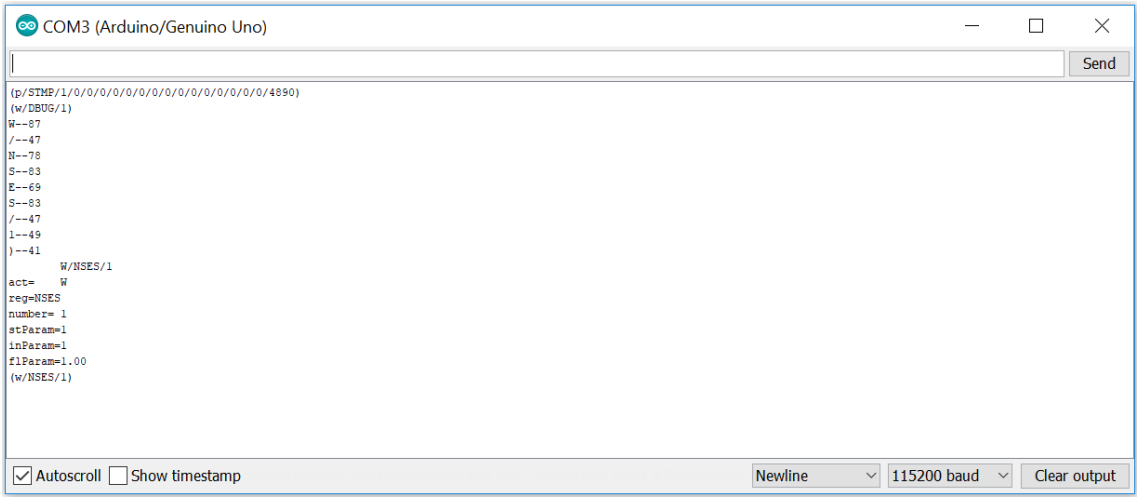


Fig. 7.20: *Instrucción NSES con "debug" activado.*

el operador la instrucción "(W/DBUG/1)", el programa devolverá otro tipo de datos que expliquen los sucesos que se estén dando en los distintos procesos a lo largo del transcurso de la sesión (Fig. 7.20).

En la imagen 7.20, se puede observar como el sistema de depurado, devuelve los caracteres introducidos por el operador en función de la tabla ASCII; además de, mostrar los diferentes valores que se asignarán a los valores de los parámetros del programa del MCD.

7.2.3. Pruebas con el Safe State

El sistema de seguridad es una de las funciones principales del MCD; es por ello, que la sesión de irradiación debe pasar al Safe State en caso de que algún proceso se vea comprometido. En la figura que se muestra a continuación (Fig. 7.21), se puede observar como el programa cambia del modo normal al modo seguro cuando el MCD no detecta una instrucción en un tiempo superior al asignado como valor del parámetro CTMax. De este modo, el programa modifica el valor de SSTA (estado de la sesión) y mantiene abierto el relé que controla, hasta que el operador modifique manualmente el modo de funcionamiento del módulo de control; por lo tanto, no permite que el haz pueda ser generado. El valor del parámetro SSTA, se modifica debido a que es el propio módulo de control de dosis del microtrón el que altera el modo de funcionamiento automáticamente; así que, el valor de SSTA pasará de ser 0 a ser equivalente a 4. En la figura 7.22, se muestra como el programa vuelve al modo de funcionamiento normal, una vez que, el operador de la orden "(W/MODE/1)"; de esta manera, revertiendo la situación del módulo de control.

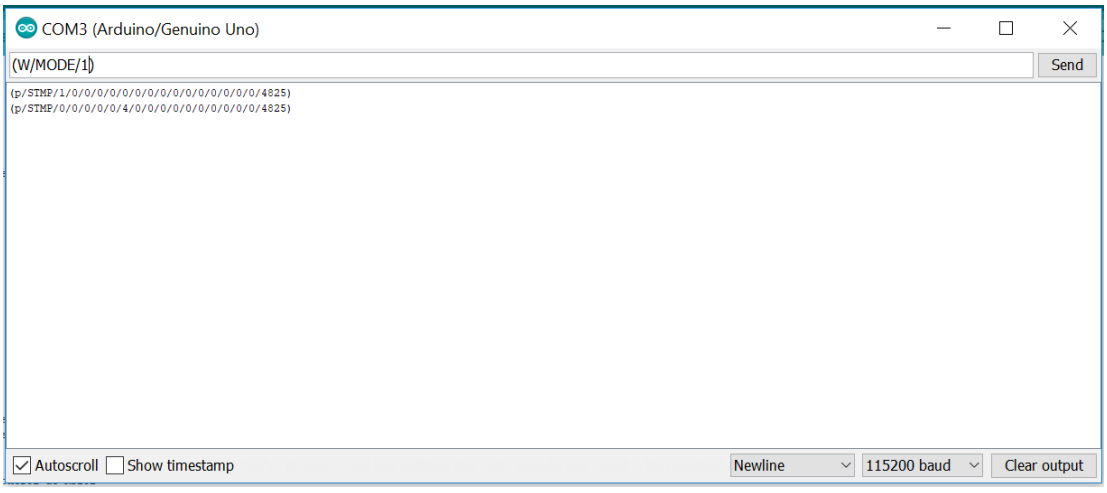


Fig. 7.21: *Cambio desde el modo normal al modo seguro.*

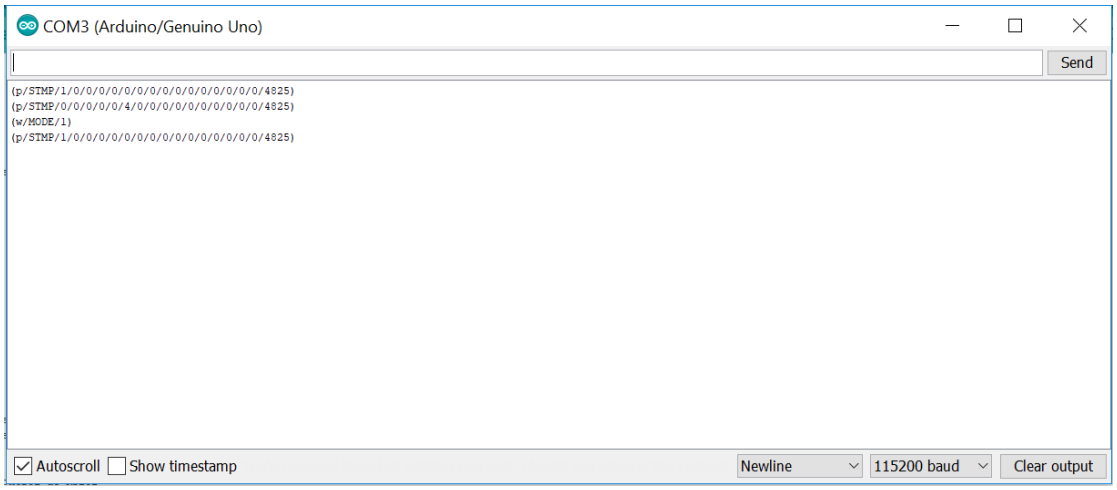


Fig. 7.22: *Cambio desde el modo seguro al modo normal por parte del operador.*

7.2.4. Fallos de la entrada de las instrucciones

Cuando el programa del MCD detecta que una instrucción enviada por el operador es errónea tiene dos respuestas posibles, dependiendo del fallo que se dé. En las siguientes dos imágenes (Fig. 7.23 y Fig.7.24), se observa como en la primera, el MCD responde ante un fallo debido a que el valor del parámetro introducido por el operador es erróneo; y en la segunda, la respuesta ante un fallo a la hora de insertar la instrucción de lectura o escritura.

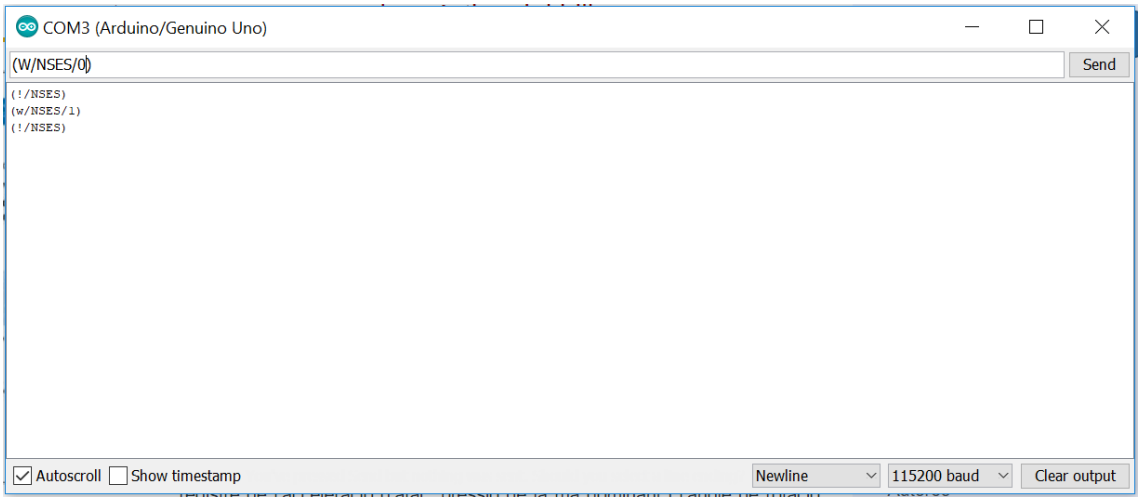


Fig.7. 23: *Respuesta ante fallo por introducción de valor erróneo.*

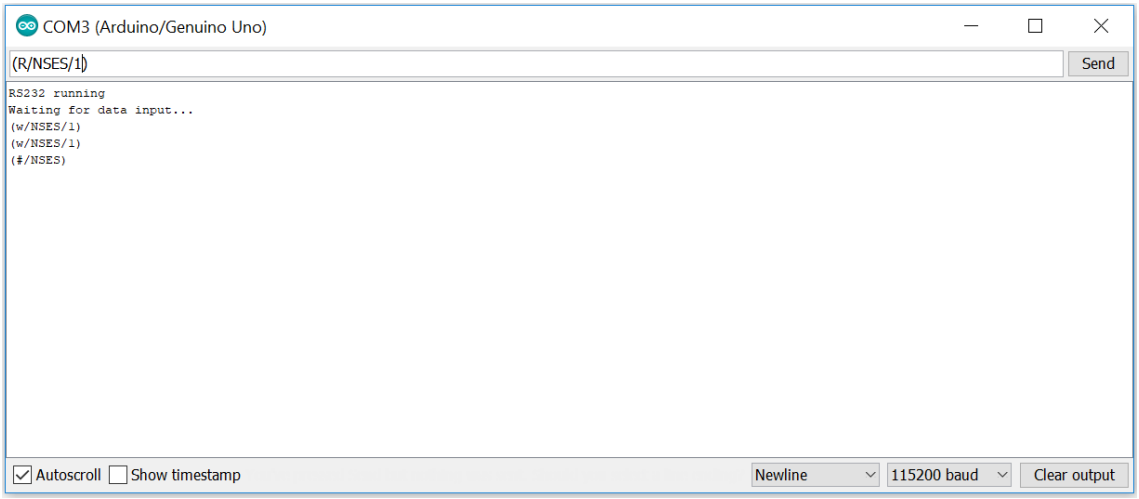


Fig. 7.24: *Repuesta ante fallo por la introducción de una instrucción errónea.*

7.2.5. Pruebas con la recuperación de sesión

Con tal de mantener la trazabilidad con la última sesión y adquirir los valores necesarios para comenzar otra sesión con los mismos parámetros utilizados en esa sesión anterior, el MCD utiliza la instrucción (R/PRSS). Cuando el operador introduce esta instrucción, el programa de módulo de control, lee la memoria EEPROM del microprocesador y devuelve los valores de los parámetros guardados en ella (Fig. 7.25). Esos datos los mantiene en la memoria una vez apagado el dispositivo; puesto que, esa memoria es permanente.

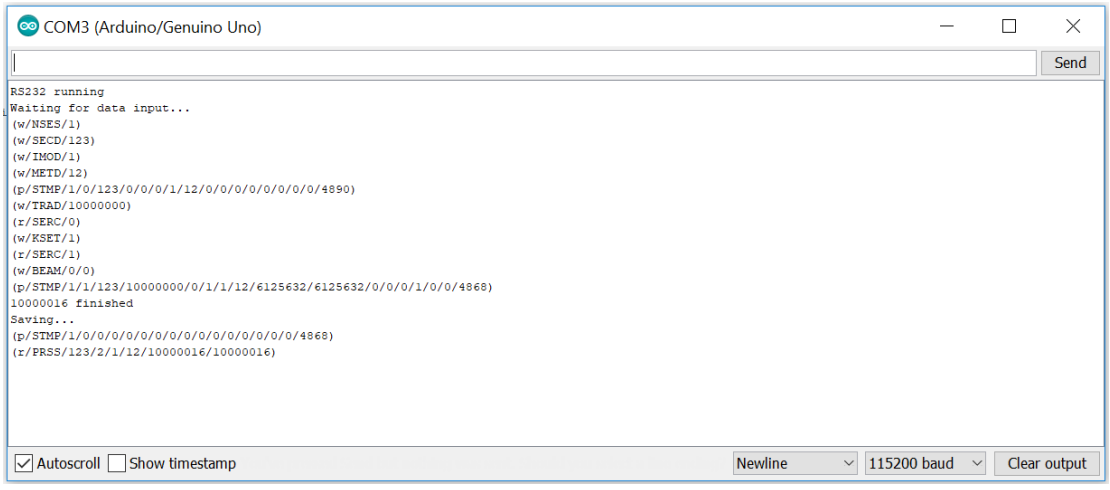


Fig. 7.25: *Instrucción PRSS devolviendo los datos de la sesión anterior.*

7.3. Pruebas con PTW-UNIDOS

Para verificar el funcionamiento de la cámara de ionización en conjunto con el MCD, se hicieron pruebas en el laboratorio de dosimetría de la ETSEIB junto al equipo PTW-UNIDOS. Se consiguió enviar instrucciones desde el programa del módulo de control; no obstante, el dosímetro no llegó a devolver las respuestas esperadas una vez concluido el periodo de tiempo destinado al presente proyecto. De manera que se pudiese verificar que el formato de instrucciones enviadas desde el MCD al dosímetro era el adecuado, fue utilizada la herramienta software "Terminal" (Fig. 7.26); mediante la cual, se pudo observar el mensaje real que se transmitía a por la vía de comunicación RS232. El dosímetro reaccionaba de forma correcta al recibir las instrucciones desde el MCD; de este modo, confirmando que el sistema de emisión de mensajes mediante el protocolo RS232 del módulo funciona correctamente. Por otro lado, la respuesta remitida por el dosímetro era incongruente; por lo que, el proyecto se vio bloqueado en este punto.

Fig. 7.26: *Programa "Terminal" recibiendo instrucciones por RS232 (ASCII y Hexadecimal).*

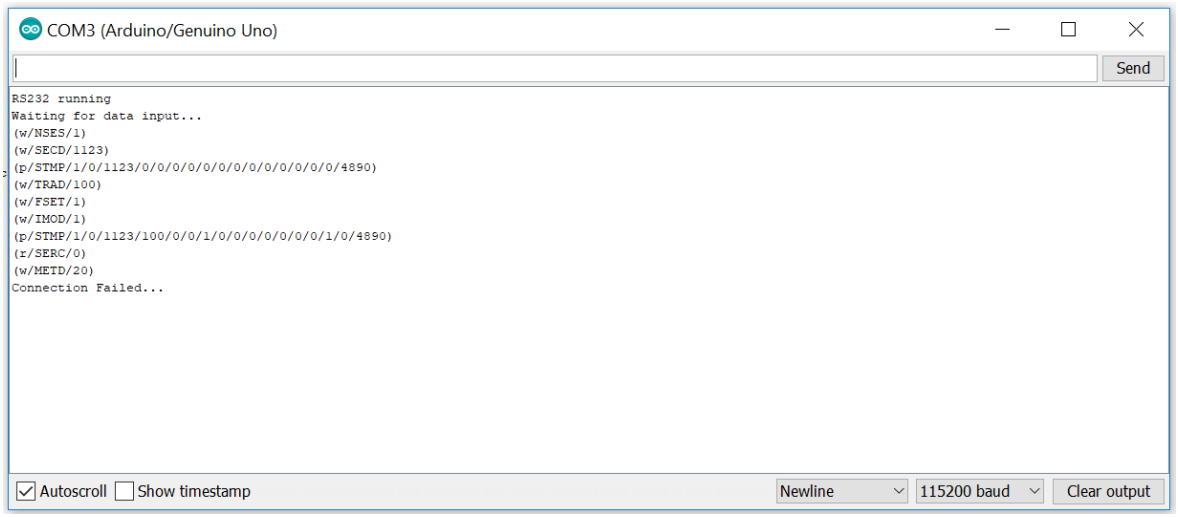


Fig. 7.27: *Fallo de conexión utilizando tratando de habilitar el dosímetro desde el MCD.*

En el momento en el que se entregó el presente proyecto, la opción de conectar el dosímetro al MCD, devolvía un mensaje de fallo de conexión (Fig. 27); puesto que, una vez enviada la instrucción "PTW" tres veces y no habiendo hallado respuesta por parte de ningún equipo que no indujese a un mensaje de error, el programa que controla el MCD dejaría de intentarlo reinicializando la sesión y restaurando los valores de los parámetros empleados a su valor original.

8. Presupuesto

A lo largo del desarrollo del proyecto se emplearon distintos materiales que implicaron un coste; el cual, se ve desglosado a continuación en los siguientes presupuestos.

8.1. Presupuestos materiales

En la siguiente tabla se muestran los distintos materiales utilizados, su cantidad y el coste de cada uno de ellos, tras haber tenido en cuenta la cantidad. El proveedor ha sido seleccionado dependiendo del precio y de descuentos; por ejemplo, "Diotronic" al estudiantado universitario le reduce el I.V.A.

Tabla 8.1: *Tabla con el presupuesto de los materiales.*

MATERIAL - empresa	CANTIDAD	PRECIO (€)
Arduino UNO (Diotronic)	1	17.50 (sin I.V.A)
Arduino Leonardo Ethernet (RobotDyn)	1	31.12
Protoshield (Diotronic)	1	6.90 (sin I.V.A)
MAX3232 (Diotronic)	1	1.65 (sin I.V.A)
Resistencia 1kΩ (Diotronic)	1	0.10 (sin I.V.A)
Cable Ethernet (FS)	1	1.04
Cable RS232 (Amazon)	2	2.10
Contacto macho DB9 (Farnell)	2	0.38
Contacto hembra DB9 (Farnell)	2	0.38
Carcasa para contactores (Farnell)	4	3.28

Diodo (Diotronic)	1	0.06 (sin I.V.A)
Estaño (Diotronic)	1 rollo	3.56 (sin I.V.A)
Cable micro-USB (Diotronic)	1	2.55 (sin I.V.A)
Cable USB (Diotronic)	1	1.47 (sin I.V.A)
Condensadores (Diotronic)	1	0.70 (sin I.V.A)
Relé (Finder)	1	2.56
Transistor	1	0.12 (sin I.V.A)
Zócalos	2	2.08 (sin I.V.A)
	TOTAL (sin I.V.A)	77.55
	TOTAL	85.26

8.2. Presupuesto diseño, ingeniería y documentación

En la tabla 8.2, se muestran los diferentes puestos referidos a la mano de obra en función a la media de coste:

Tabla 8.2: *Tabla con el presupuesto de la mano de obra.*

DEFINICIÓN	HORAS	PRECIO (HORA)	PRECIO (TOTAL)
Administración	40	16€	640€
Investigación	75	25€	1875€
Ingeniero	300	32.5€	9750€
Redacción	115	12€	1380€
		TOTAL	13645€

8.3. Presupuesto total del proyecto

Por último, en la tabla 8.3, se expone el coste total del proyecto teniendo en cuenta la mano de obra y los materiales empleados:

Tabla 8.3: *Presupuesto total.*

DEFINICIÓN	PRECIO (€)
Materiales	77.55
Mano de obra	13645
TOTAL (sin I.V.A)	13722.55
TOTAL	13727.26

9. Impacto medioambiental

Según la definición del art. 5 del Real Decreto 1131/88, de 30 de septiembre [33], la Evaluación del Impacto Ambiental (EIA) corresponde al "… conjunto de estudios y sistemas técnicos que permiten estimar los efectos que la ejecución de un determinado proyecto, obra o actividad causan sobre el medio ambiente"; además, en el artículo 6 del mismo Real Decreto especifica que "La evaluación de impacto ambiental debe comprender, al menos, la estimación de los efectos sobre la población humana, la fauna, la flora, la vegetación, la gea, el suelo, el agua, el aire, el clima, el paisaje y la estructura y función de los ecosistemas presentes en el área previsiblemente afectada".

El presente proyecto se ha desarrollado mediante sistemas informáticos y ha basado un gran porcentaje del tiempo dedicado en programación; no obstante, precisa de un análisis de impacto medioambiental; debido a que, aunque se haga especialmente difícil concretar tiempos de producción y gastos energéticos, además del propio impacto humano sobre el entorno para poder llevar acabo la realización del proyecto, se han desarrollado actividades de mayor impacto sobre el entorno. Por lo tanto, pese a que el desarrollo del proyecto se basa en la programación; lo cual, genera entre 52 y 234 gramos equivalentes de CO_2 por hora [34]; también se ha de tener en cuenta el material electrónico empleado para la confección de la estructura hardware utilizada.

Para tratar con los componentes electrónicos se ha de tener precaución; puesto que, tanto los componentes pasivos como los activos, están constituidos por materias primas obtenidas a raíz de métodos como la extracción minera; para la cual, se emplean componentes tóxicos entre los que se engloban: el cianuro, plomo, mercurio, y demás. Estas sustancias contaminan el agua, el suelo y el aire; de esta manera, convirtiéndose en un riesgo para la salud del ser humano y del planeta [35].

Además de la obtención de los materiales electrónicos; también, se ha de tener en cuenta el impacto de su reciclado una vez acabada su vida útil. Esto se debe a que, según el Programa de las Naciones Unidas para el Medio Ambiente (PNUMA) el aumento de chatarra electrónica es exponencial y tan solo el 20% de esos desechos se recicla de forma adecuada [36]. Se estima que, en 2025, el ser humano generará un total de casi 54 millones de toneladas de residuos electrónicos [37] si la curva de crecimiento sigue al mismo ritmo. Para reducir este impacto, PNUMA propone la reutilización de todo el material posible dentro de los marcos establecidos; además, de una serie de normativas regulatorias para la distribución, uso y producción de artefactos tecnológicos.

Sin embargo, al tratarse este proyecto de un prototipo aislado y no de un producto de producción masiva; además, de la utilización de zócalos para la posible extracción y reutilización de los

componentes empleados, el impacto medioambiental queda bastante mitigado. En caso contrario, los daños al medioambiente requerirían de un análisis en mayor profundidad que permitiese decidir la viabilidad del producto; teniendo en cuenta, las líneas de producción, los materiales, las emisiones, el espacio utilizado, y otras variables nocivas

10. Conclusiones

Una vez acabado el plazo de entrega y habiendo visto los resultados obtenidos, nos encontramos ante una labor de investigación en distintos ámbitos convirtiendo el presente proyecto en un trabajo multidisciplinar. Esto sucede debido a la interacción que se ha dado a lo largo del último siglo entre los aportes que la física ha otorgado a la humanidad, gracias a la investigación, y a la adaptación que se ha hecho en el ámbito médico de los nuevos recursos que aparecían.

En el proyecto se presenta un módulo de control capaz de llevar a cabo una sesión obteniendo valores de ciertos parámetros introducidos por el operador; de este modo, mediante cálculos embebidos en el programa que manipula el módulo de control, se determina la dosis, o el tiempo, necesarios para alcanzar las medidas requeridas en una sesión particular. Además, el módulo controla la seguridad de la sesión mediante distintos procesos de lectura de variables, permitiendo al operador tener un control del estado de la sesión en todo momento, e incluso, frenarlo en caso de que fuera necesario. También, el módulo capacita al operador a emplear un programa adaptable a distintas formas de comunicación con un hardware extraíble y, actualmente, permitiendo el envío de instrucciones al dosímetro PTW-UNIDOS.

Para completar el módulo de control y habilitarlo de modo que sea totalmente acoplable con el resto de los módulos que monitorizan y controlan el RTM, serían necesarios los siguientes pasos:

- Completar la comunicación con el dosímetro PTW-UNIDOS.
- Testear en más profundidad esa conexión.
- Configurar la GUI en LabView para acoplar el módulo

Una vez concluidas esas pautas el MCD estaría acoplado con el resto de los módulos, siendo plenamente operativo.

Bibliografia

[1] CPAN, “Medicina - La fructífera relación entre Física y Medicina,” 2010. [Online]. Available: https://www.i-cpan.es/es/content/medicina. [Accessed: 04-Jan-2019].

[2] Interempresas, “Aplicaciones de la radiación ionizante en polímeros - Química,” 2018. [Online]. Available: https://www.interempresas.net/Quimica/Articulos/213975-Aplicaciones-de-la-radiacion-ionizante-en-polimeros.html. [Accessed: 28-Apr-2019].

[3] Y. Fandiño, “Radioquímica - EcuRed,” 2017. [Online]. Available: https://www.ecured.cu/Radioquímica#Aplicaciones_en_las_ciencias. [Accessed: 28-Apr-2019].

[4] J. Fernández Ferrer, M. Pujal Carrera, and J. F. Ferrer, *Iniciación a la física*. Reverté, 1981.

[5] C. Sutton, “Particle accelerator | instrument,” *Britannica.com*, 2006. [Online]. Available: https://www.britannica.com/technology/particle-accelerator#ref364998. [Accessed: 30-Mar-2019].

[6] R. Valencia, “INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA "ESTUDIO DEL KLYSTRON REFLEX",” Mexico. DF, 1981.

[7] D. M. Pozar, *Microwave engineering*. Addison-Wesley, 1990.

[8] Á. Franco, “Acelerador de partículas cargadas. El ciclotrón,” 2010. [Online]. Available: http://www.sc.ehu.es/sbweb/fisica/elecmagnet/ciclotron/ciclo.html#El ciclotrón. [Accessed: 30-Mar-2019].

[9] National Cancer Institute, “Radioterapia de haz externo para el cáncer,” 2018. [Online]. Available: https://www.cancer.gov/espanol/cancer/tratamiento/tipos/radioterapia/haz-externo. [Accessed: 30-Mar-2019].

[10] Asociación Argentina para el Progreso de las Ciencias., *Ciencia e investigación.* Emecé Editores, 1945.

[11] La Razón, “«La radioterapia cura hasta el 90 por ciento de los tumores localizados»,” *La Razón*, 2011. [Online]. Available: https://www.larazon.es/historico/10003-la-radioterapia-cura-hasta-el-90-por-ciento-de-los-tumores-localizados-MLLA_RAZON_384968. [Accessed: 31-Mar-2019].

[12] Radiology.org, “PET/TC (tomografía por emisión de positrones – tomografía computada),” *Radiology.org*, 2017. [Online]. Available: https://www.radiologyinfo.org/sp/info.cfm?pg=pet. [Accessed: 31-Mar-2019].

[13] J. M. Escobar, “Acelerador lineal - Apuntes de Electromedicina Xavier Pardell,” 2019. [Online]. Available: https://www.pardell.es/acelerador-lineal.html. [Accessed: 31-Mar-2019].

[14] CELLS, “ALBA Synchrotron,” 2014. [Online]. Available: https://www.cells.es/es/. [Accessed: 31-Mar-2019].

[15] V. Valkovic and W. Zyszkowski, “Los aceleradores en la ciencia y la industria: Énfasis en el Oriente Medio y Europa,” 1994.

[16] Foro Nuclear, “¿Qué sabes de la radiación?,” *Foro Nuclear*, 2019. [Online]. Available: https://www.foronuclear.org/es/el-experto-te-cuenta/119909-que-sabes-de-la-radiacion. [Accessed: 02-Apr-2019].

[17] P. Turmero, “Generación de RX. Características físicas de los equipos de radiodiagnóstico - Monografias.com,” 2015. [Online]. Available: https://www.monografias.com/trabajos105/generacion-rx-caracteristicas-equipos-rd/generacion-rx-caracteristicas-equipos-rd.shtml. [Accessed: 02-Apr-2019].

[18] Foro Nuclear, “5.Efectos biológicos de la radiación ionizante,” *foronuclear.org*, 2019. [Online]. Available: http://rinconeducativo.org/contenidoextra/radiacio/5efectos_biolgicos_de_la_radiacin_ionizante.html. [Accessed: 02-Apr-2019].

[19] A. Real, G. Dra, and C. Biológicas, “Efectos Biológicos de las Radiaciones Ionizantes.”

[20] IAEA - Protección Radiológica para pacientes, “Magnitudes y unidades de radiación,” *iaea.org*, 2013. [Online]. Available: https://rpop.iaea.org/RPOP/RPoP/Content-es/InformationFor/HealthProfessionals/1_Radiology/QuantitiesUnits.htm. [Accessed: 02-Apr-2019].

[21] V. Blasco, Y. Kubyshin, J. A. Romero, A. Sanchéz, and V. Shedunov, “MITA Control System : Architecture , Functions and Commands MITA Control System :,” pp. 1–43, 2017.

[22] Y. Kubyshin, “Construction of a Compact 12 MeV Race-track Microtron at the UPC Yuri Kubyshin,” 2017.

[23] Y. Kubyshin and J. A. Romero, “RTM control system specification,” no. October, 2008.

[24] P. Bergstrom and S. Seltzer, “Radiation Dosimetry Data | NIST,” *National Institute of Standards and Technology*, 2019. [Online]. Available: https://www.nist.gov/pml/radiation-dosimetry-data. [Accessed: 22-Apr-2019].

[25] UNIDOS, “User Manual UNIDOS webline Interfaces,” 2009.

[26] ASCII, “Códigos ASCII - Tabla de caracteres y simbolos ascii,” *ascii.cl.es*, 2015. [Online]. Available: https://ascii.cl/es/. [Accessed: 08-Apr-2019].

[27] R. Remón, “Programacion estructurada,” *EcuRed*, 2014. [Online]. Available: https://www.ecured.cu/Programacion_estructurada. [Accessed: 07-Apr-2019].

[28] UTN-FRM, “PROGRAMACION ESTRUCTURADA,” 2014.

[29] J. Pérez and M. Merino, “Definición de puerto serial - Qué es, Significado y Concepto,” 2013. [Online]. Available: https://definicion.de/puerto-serial/. [Accessed: 07-Apr-2019].

[30] Linksys, “¿Qué es Ethernet?,” *Linksys*, 2018. [Online]. Available:

https://www.linksys.com/es/r/resource-center/que-es-ethernet/. [Accessed: 07-Apr-2019].

[31] J. M. Ruiz Gutiérrez, "Arduino + Ethernet Shield," 2013.

[32] Geeky Theory, "La importancia de UI Testing y Unit Testing," *Geeky Theory*, 2019. [Online]. Available: https://geekytheory.com/la-importancia-de-ui-testing-y-unit-testing. [Accessed: 16-Apr-2019].

[33] Ministerio de Obras Públicas y Urbanismo, "Real Decreto 1131/1988, de 30 de septiembre," *BOE*, 1988. [Online]. Available: http://noticias.juridicas.com/base_datos/Admin/rd1131-1988.html. [Accessed: 19-Apr-2019].

[34] Planeta Recicla, "Los ordenadores también emiten CO2," *EcoEmbes*, 2016. [Online]. Available: https://www.ecoembes.com/es/planeta-recicla/blog/los-ordenadores-tambien-emiten-co2. [Accessed: 19-Apr-2019].

[35] Redacción El Tiempo, "MINERÍA AFECTA AL MEDIO AMBIENTE," *eltiempo.com*, 1995. [Online]. Available: https://www.eltiempo.com/archivo/documento/MAM-497060. [Accessed: 19-Apr-2019].

[36] J. Elcacho, "La ONU reclama acciones contra la plaga mundial de la basura electrónica," *lavanguardia.com*, 2019. [Online]. Available: https://www.lavanguardia.com/natural/20190125/454292252661/la-onu-reclama-acciones-contra-la-plaga-mundial-de-la-basura-electronica.html. [Accessed: 19-Apr-2019].

[37] EFE, "Basura electrónica aumentó 30% en menos de una década | ELESPECTADOR.COM," *elespectador.com*, 2018. [Online]. Available: https://www.elespectador.com/noticias/medio-ambiente/el-mundo-generara-54-millones-de-toneladas-de-chatarra-electronica-en-2025-articulo-791189. [Accessed: 19-Apr-2019].

Anexo A - Instrucciones del Módulo de Control de Dosis del microtrón

Instrucción	Tipo	Req/ Resp	Formato de escritura	Descripción y comentarios
STMP	Ping	P	(P/STMP)	Expone el estado del microprocesador y los parámetros de la sesión.
		p	(p/STMP/FS/DCON/SC/TRAD/DRAD/SSTA/IM/MD/TSES/DSES/K1/K2/K3/K/F/DBUG/VCC)	
NSES	Write	W	(W/NSES/1)	Inicia una nueva sesión.
			(w/NSES/1)	
SECD	Write/ Read	W / R	(W/SECD/nnn), (R/SECD)	Configura el código de la sesión y lo confirma. El código se trata como un entero introducido por el operador.
		w / r	(w/SECD/nnn), (r/SECD/nnn)	
IMOD	Write/ Read	W / R	(W/IMOD/IM), (R/ IMOD)	Configura y confirma el modo de irradiación IM=1, tiempo fijo IM=2, dosis fija.
		w / r	(w/IMOD/IM), (r/IMOD/IM)	
METD	Write/ Read	W / R	(W/METD/MD), (R/METD)	Configura y confirma el método de cálculo

		w / r	(w/METD/MD), (r/METD/MD)	para el control de la dosis: MD = 11, 12 o 20.
TRAD	Write	W	(W/TRAD/TRAD)	Configura el tiempo de irradiación de la sesión cuando IM = 1, TRAD [µs].
		w	(w/TRAD/TRAD)	
DRAD	Write	W	(W/DRAD/DRAD)	Configura la dosis que será emitida cuando IM = 2, DRAD [Gy].
		w	(w/DRAD/DRAD)	
K1ST	Write	W	(W/K1ST/K1)	Configura el valor de K1 cuando MD = 11, K1 [nGy · cm^2].
		w	(w/K1ST/K1)	
K2ST	Write	W	(W/K2ST/K2)	Configura el valor de K2 cuando MD = 11, K2 [cm^{-2}].
		w	(w/K2ST/K2)	
FREP	Write	W	(W/FREP/FREP)	Configura el valor de la f_{rep} cuando MD = 11, FREP [Hz].
		w	(w/FREP/FREP)	
KSET	Write	W	(W/KSET/K)	Configura el valor de K cuando MD = 12, K [Gy/sec].
		w	(w/KSET/K)	
FSET	Write	W	(W/FSET/F)	Configura el valor del factor de conversión F cuando MD = 20.
		w	(w/FSET/F)	
SERC	Read	R	(R/SERC)	Confirma que se puede iniciar una sesión DCON = 0, faltan datos DCON = 1, listo
		r	(r/SERC/DCON)	

BEAM	Write	W	(W/BEAM/1)	Instrucción que inicia la sesión: "Beam on"
		w	(w/BEAM/1)	
BEAM	Write	W	(W/BEAM/0)	Instrucción que termina la sesión: "Beam off", se envía en caso de que el operador desee finalizar la sesión, o que uno de los interlocks del RTM lo requiera.
		w	(w/BEAM/0)	
SSTA	Read	R	(R/SSTA)	Lee el estado de la sesión: SSTA=0, "La sesión no ha sido realizada", SSTA = 1, "La sesión está en marcha", SSTA = 2, "La sesión ha finalizado correctamente", SSTA = 3, "La sesión ha sido interrumpida por el operador", SSTA = 4, "La sesión ha sido interrumpida por el módulo de control de dosis" SSTA = 5, "Pérdida de comunicación durante la sesión".
		r	(r/SSTA/SSTA)	

SRES	Read	R	(R/SRES)	Lee los resultados de la sesión de irradiación.
		r	(r/SRES/SSTA/TSES/DSES)	
PRSS	Read	R	(R/PRSS)	Lee los resultados de la última sesión de irradiación llevada a cabo desde la memoria EEPROM, y sustituye en la sesión actual el modo de irradiación y el método de cálculo.
		R	(r/PRSS/SC/SSTA/IMOD/METD/ TSES/DSES)	
MODE	Write	W	(W/MODE/0)	Pasa a Safe State, FS=0
		w	(w/MODE/0)	
MODE	Write	W	(W/MODE/1)	Pasa a Normal Mode, FS=1
		w	(w/MODE/1)	
COMT	Write / Read	W / R	(W/COMT/T1max), (R/COMT)	Configura o confirma el tiempo máximo sin comunicación, T1max [µs].
		w / r	(w/COMT/T1max), (r/COMT/T1max)	
DBUG	Write/ Read	W/R	(W/DBUG/dbug), (R/DBUG)	Activa y desactiva el modo de depurado, dbug = 0, debug mode off dbug = 1, debug mode on
		w/r	(w/DBUG/dbug), (r/DBUG/dbug)	

N/A	Read	!	(!/Instrucción/dato)	Devuelve el valor de una instrucción con valores erróneos.
N/A	Read	#	(!/Instrucción/dato)	Devuelve una instrucción errónea o inexistente introducida por el operador

Datos que se tratan en las instrucciones de la tabla:

- **FS:** Estado del modo seguro (0: normal, 1: "Safe State").
- **DCON:** Estado de la sesión de irradiación (0: no lista, 1: lista).
- **SC:** Código de sesión (entero).
- **TRAD:** Tiempo a irradiar en microsegundos.
- **DRAD:** Dosis a irradiar en Grey.
- **SSTA:** Estado de la sesión.
- **IM:** Modo de irradiación, (1: tiempo fijo, 2: dosis fija).
- **MD:** Método de cálculo de dosis, (11: factores K1, K2 y Frep, 12: factor K, 20: cámara de ionización).
- **TSES:** Tiempo irradiado en microsegundos.
- **DSES:** Dosis irradiada en Greys.
- **K1, K2, K3:** Parámetros para el cálculo de la dosis cuando MD = 11.
- **K:** Factor para el cálculo de la dosis cuando MD = 12.
- **F:** Factor de conversión cuando MD = 20.
- **DBUG:** Estado del modo de depurado, (0: régimen normal, 1: depurado activo).
- **VCC:** Nivel de voltaje suministrado por el microprocesador (regulador interno).

Anexo B - Código empleado

B1. Programa completo con comunicación por puerto serie

```
//PROGRAMA <PROGRAMADETECTOR>

 /*The states of the control session:

                0: Session Not Realized

                1: Session Ongoing

                2: Session Performed Correctly

                3: Session Interrupted by the operator

                4: Session Interrupted by the dose control module

                5: Loss of communication during the irradiation

*/

//Libraries

#include <EEPROM.h>

#include <SoftwareSerial.h> //It would be using to communicate with the Ionization Chamber via RS232


#define delaycom 50 //this is the delay after sending a command via RS232 port


SoftwareSerial RS232(9, 8); //Rx & Tx pins


///RS232 control variables

//
```

```
const char LF = 10;

const char CR = 13;

String reply232; //String to receive the response from the Ionization Chamber

bool CheckCommunication_Flag = false;

bool RS232_Flag = false;

float receiverDose = 0.00;

float receiverTime = 0.00;

unsigned long delayTime = 0;


///Control Variables

//

boolean debug = false;

boolean mode = true;


///Led Status

//

#define Rele 12

#define Light 13


///System Booleans

//

bool NewSessionMakerValidator = false; //New Session Validator Boolean

bool ModeOperationSelectorValidator = false; //Operation Mode Validator Boolean

bool DeliveryRequestValidator =  false; //Delivery Dose Validator Boolean
```

```
bool MeasureOperationSelectorValidator = false; //Measurement Mode Validator Boolean

bool ConversionMethodSelectorValidator = false; //Conversion Method Validator Boolean

bool NewSessionNeeded =  true; //New Session Creation Validator Boolean


///K Selections Booleans

//

bool KSetupValidator = false;

bool K1SetupValidator = false;

bool K2SetupValidator = false;

bool K3SetupValidator = false;


///Coherence validator

//

bool DRADFlag = false;

bool TRADFlag = false;

///Session Settings

//


#define direcEEPROM 0 //EEPROM byte where is keeping the value of the session code


unsigned int SC = 0; //This is the session code


byte NSES = 0; //This is the new session request variable

byte DCON = 0; //This is the variable which allows the beam
```

```
///Acquisition Dates To PING
//
unsigned int IM; //Irradiation Mode
float T = 0.00; //Delivered Time
float D = 0.00; //Delivered Dose
unsigned int MD; //Dose Determination Mode
float F = 0.00; //Eventual Conversion Factor

///Ping Variables
//
char TString[10] = "0";
char DString[10] = "0";
char k1String[10] = "0";
char k2String[10] = "0";
char k3String[10] = "0";
char KString[10] = "0";
char FString[10] = "0";-
char CTmaxString[10] = "50000000";

// Variables declared for SerialTranslator function
char act[5];// define ACTION to do with register
char reg[5]; //define which REGISTER is in use to make the defined action
byte nParam;// contain de numbers of parameters introduced by the operator
```

```
char stParam[10];// contain the string passed to parameter where the maximum are 10 chars

float flParam;// contain the param converted to float

int inParam;// contain the param converted to int



char buffer[20] = "";


String response = "";

String resp = "";


///Dose/Time Operation needed variables

//

float k1 = 0.00;

float k2 = 0.00;

float k3 = 0.00;

float K = 0.00;

float KGet = 0.00;


///Last Command Timer

//

unsigned long cmdT = 0; // contain time spam of last command process


///Dosimeters in Dose Delivery

//
```

```
float Dtoget = 0.00;

float Dmoment = 0.00;


///Timers in Dose Delivery

//

unsigned long Ttoget;

unsigned long TIter;


///Run Control Timer

unsigned long MomentumTimer;


///Maximum Time to do a PING

unsigned long CTmax = 50000000; //50 segundos

unsigned long CT = 0; //Connection Timeout


///Task Control Timers

#define ControlTime 500000 //periodo control de RS232

unsigned long int ts1;

unsigned long int t01;



///Execution Flags

//

bool Beam_OnFlag = false;
```

```
bool Beam_OffFlag = false;

bool Beam_StpFlag = false;


///Execution Indicator

//

byte SSTA = 0;

unsigned  long TSES = 0.00;

unsigned  long DSES = 0.00;

unsigned long ATimeEmitted = 0.00;

const long scaleConst = 1100.300 * 1000; //internal ref * 1023 * 1000


void setup() {

 // OUTPUT Pin definitions

 //

 pinMode(Rele, OUTPUT);

 pinMode(Light, OUTPUT);

 digitalWrite(Rele, LOW);

 digitalWrite(Light, LOW);


 // Here starts the Serial Communication

 //

  Serial.begin(115200);

 Serial.println("RS232 running");

 //The communication speed is set
```

```
 RS232.begin(57600);

 Serial.println("Waiting for data input...");

 // Timers Initialization

 //

 ts1 = 0;

 t01 = 0;

 ///Here start the timers

 //

 cli();


 TCCR1A = 0x00;

 //preescaler

 TCCR1B = 0x00;

 TCNT1 = 0;

 OCR1A = 15640; //(16*10^6)/(5*1024)-1

 TCCR1B |= (1 << WGM12);

 // Set CS12 and CS10 bits for 1024 preescaler

 TCCR1B |= (1 << CS12) | (1 << CS10);

 // enable timer compare interrupt

 TIMSK1 |= (1 << OCIE1A);


 sei();

}

ISR(TIMER1_COMPA_vect) {
```

```
 SftEvent();
}

void loop() {
 if (BeamCanStart() == 1) {
  if (!RS232_Flag) {
   BeamControlExecuter();
  }
  else {
   RS232_Communication();
  }

 }
 if (comandoSerial() == 1) {
  ///Those commands that the Detector reads
  if ((act[1] == 'W') || (act[1] == 'R') || (act[1] == 'P')) {
   SerialTranslator();
  }
 }
 // Get rid of the values:  act, reg, stParam
 memset(act, '\0', sizeof(act));
 memset(reg, '\0', sizeof(reg));
 memset(stParam, '\0', sizeof(stParam));
 nParam = 0; inParam = 0; flParam = 0;
```

```
}


byte comandoSerial() {
 byte cmdok = 0;
 while (Serial.available()) {
  char inChar = (char)Serial.read();
  if (inChar == '(') {
   double t = millis();
   char c = 0;
   byte j = 0;
   boolean salir = true;
   while ( salir ) {
    c = (char)Serial.read();
    if ( c > 0 ) {
     buffer[j + 1] = c;
     j++;
     if ( debug ) {
      Serial.print(c); Serial.print("--"); Serial.println(c, DEC);
     }
     if ( c == ')' ) {
      j--;
      cmdok = 1;
      break;
     }
```

```
  }
 }
 if ( cmdok == 1 ) {
  buffer[0] = j + 1; buffer[j + 1] = '\0'; //finalizar buffer quitar el carracter ')' final
  char * pch; //puntero array de chars
  if ( debug ) {
   Serial.println(buffer);
  }
  pch = strtok(buffer, "/"); //Look for  / and find the action
  strcpy(act, pch); act[5] = '\0'; //set the value
  int count = 1;
  while ( pch != NULL ) {
   if ( count == 2 ) {
    strcpy(reg, pch);  //the second one is reg variable value
    reg[5] = '\0';
   }
   if ( count == 3 ) {
    strcpy(stParam, pch);  //the parameter is at the third input
    inParam = atoi(pch);
    flParam = atof(pch);
   }
   pch = strtok(NULL, "/");
   count++;
  }
```

```
      nParam = count - 3;


      if ( debug ) {
       Serial.print("act="); Serial.println(act);
       Serial.print("reg="); Serial.println(reg);
       Serial.print("number= "); Serial.println(nParam);
       Serial.print("stParam="); Serial.println(stParam);
       Serial.print("inParam="); Serial.println(inParam);
       Serial.print("flParam="); Serial.println(flParam);
      }
     }
    }
   return cmdok;
  }
}


//PROGRAMA <LEERCOMANDOSERIAL>
void SerialTranslator() {
  String cadena = "";
  if (strncmp(reg, "STMP", 4) == 0) {
    if (act[1] == 'P') {
      resp += mode; resp.concat('/');
      resp += DCON; resp.concat('/');
      resp += SC; resp.concat('/');
```

```
    resp += TString; resp.concat('/');
    resp += DString; resp.concat('/');
    resp += SSTA; resp.concat('/');
    resp += IM; resp.concat('/');
    resp += MD; resp.concat('/');
    resp += TSES; resp.concat('/');
    resp += DSES; resp.concat('/');
    resp += k1String; resp.concat('/');
    resp += k2String; resp.concat('/');
    resp += k3String; resp.concat('/');
    resp += KString; resp.concat('/');
    resp += FString; resp.concat('/');
    resp += debug; resp.concat('/');
    resp += readVcc();


   }
   else {
    badcmd();
   }
   goto endsub;
  }
 if (strncmp(reg, "NSES", 4) == 0) {
   if ((act[1] == 'W') && (nParam != 0)) {
    if (inParam == 1) {
```

```
      NSES = 1;
      NewSessionCleanUp();
      resp += NSES;
    }
    else {
      badsyn();
    }
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "SECD", 4) == 0) {
  if (act[1] == 'R') {
    resp += SC;
  }
  else if (act[1] == 'W' && (nParam != 0)) {
    if (!NewSessionNeeded) {
      badsyn();
    }
    else {
      NewSessionMaker(inParam);
    }
```

```
    resp += SC;
  }
  else {
    badcmd();
  }
  goto endsub;
}

if (strncmp(reg, "IMOD", 4) == 0) {
  if (act[1] == 'R') {
    resp += IM;
  }
  else if ((act[1] == 'W') && (nParam != 0) ) {
    if ((inParam == 1) || (inParam == 2)) {
      ModeOperationSelector();
      resp += IM;
    }
    else {
      badsyn();
    }
  }
  else {
    badcmd();
  }
```

```
  goto endsub;
 }
if (strncmp(reg, "METD", 4) == 0) {
  if (act[1] == 'R') {
    resp += MD;
  }
  else if ((act[1] == 'W') && (nParam != 0) ) {
    if ((inParam == 11) || (inParam == 12) || (inParam == 20)) {
      MeasureOperationSelector();
      resp += MD;
    }
    else {
      badsyn();
    }
  }
  else {
    badcmd();
  }
  goto endsub;
 }
if (strncmp(reg, "TRAD", 4) == 0) {
  if (act[1] == 'R') {
    resp += TString;
  }
```

```
  else if ((act[1] == 'W') && (nParam != 0)) {
    TimeDeliveryCatcher();
    TRADFlag = true;
    resp += TString;
  }
  else {
    badcmd();
  }
  goto endsub;
 }
 if (strncmp(reg, "DRAD", 4) == 0) {
  if (act[1] == 'R') {
    resp += DString;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    DoseDeliveryCatcher();
    DRADFlag = true;
    resp += DString;
  }
  else {
    badcmd();
  }
  goto endsub;
 }
```

```
if (strncmp(reg, "K1ST", 4) == 0) {
  if (act[1] == 'R') {
    resp += k1String;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    byte KSelector = 1;
    KConstantsCatcher(KSelector);
    KIsSet();
    resp += k1String;
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "K2ST", 4) == 0) {
  if (act[1] == 'R') {
    resp += k2String;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    byte KSelector = 2;
    KConstantsCatcher(KSelector);
    KIsSet();
    resp += k2String;
```

```
  }
  else {
    badcmd();
  }
  goto endsub;
 }
 if (strncmp(reg, "FREP", 4) == 0) {
  if (act[1] == 'R') {
    resp += k3String;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    byte KSelector = 3;
    KConstantsCatcher(KSelector);
    KIsSet();
    resp += k3String;
  }
  else {
    badcmd();
  }
  goto endsub;
 }
 if (strncmp(reg, "KSET", 4) == 0) {
  if (act[1] == 'R') {
    resp += KString;
```

```
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    if (!ConversionMethodSelectorValidator) {
      byte KSelector = 0;
      KConstantsCatcher(KSelector);
      KIsSet();
      resp += KString;
    }
    else {
      resp += KString;
    }
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "FSET", 4) == 0) {
  if (act[1] == 'R') {
    resp += FString;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    ConversionMethodCatcher();
    resp += FString;
```

```
  }
  else {
    badcmd();
  }
  goto endsub;
 }
 if (strncmp(reg, "SERC", 4) == 0) {
  if (act[1] == 'R') {
    resp += DCON;
  }
  else {
    badcmd();
  }
  goto endsub;
 }
 if (strncmp(reg, "BEAM", 4) == 0) {
  if ((act[1] == 'W') && (nParam != 0)) {
    if (inParam == 1) {
      if (BeamCanStart() == 1) {
        DoseRevelator();
        Beam_StpFlag = false;
        Beam_OnFlag = true;
        MomentumTimer = micros();
        SSTA = 1;
```

```
      resp += TSES; resp.concat('/');
      resp += DSES;
    }
    else {
      badsyn();
    }
  }
  if (inParam == 0) {
    if (BeamCanStart() == 1) {
      Beam_OnFlag = false;
      Beam_OffFlag = true;
      SSTA = 3;
      resp += TSES; resp.concat('/');
      resp += DSES; resp.concat('/');
      resp += SSTA;
    }
    else {
      badsyn();
    }
  }

}
else {
  badcmd();
```

```
  }
  goto endsub;
 }
 if (strncmp(reg, "SSTA", 4) == 0) {
  if (act[1] == 'R') {
   resp = SSTA;
  }
  else {
   badcmd();
  }
  goto endsub;
 }
 if (strncmp(reg, "SRES", 4) == 0) {
  if (act[1] == 'R') {
   resp += SSTA; resp.concat('/');
   resp += TSES; resp.concat('/');
   resp += DSES;
  }
  else{
   badcmd();
  }
  goto endsub;
 }
 //  if (strncmp(reg, "BSTP", 4) == 0) {
```

```
//   if (act[1] == 'W') {
//     if (BeamCanStart() == 1) {
//       Beam_StpFlag = true;
//       Beam_OnFlag = false;
//       ATimeEmitted = TSES;
//       SSTA = 3;
//       resp += TSES;
//     }
//     else {
//       badsyn();
//     }
//   }
//   else {
//     badcmd;
//   }
//   goto endsub;
// }
//MODE DEBUG
if ( strncmp(reg, "DBUG", 4) == 0 ) {
  if ( act[1] == 'R' ) {
    resp += debug;
  }
  else if ( act[1] == 'W' ) {
    if ( (inParam == 0 || inParam == 1) ) {
```

```
      debug = inParam;
      resp += debug;
    }
    else {
      badsyn();
    }
  }
  else {
    badcmd();
  }
  goto endsub;
}
//MODE STATUS
if ( strncmp(reg, "MODE", 4) == 0 ) {
  if ( act[1] == 'R' ) {
    resp += mode;
  }
  else if ( act[1] == 'W') {
    if ( (inParam == 0 || inParam == 1) ) {
      mode = inParam;
      SSTA = 0;
      resp += mode;
    }
    else {
```

```
      badsyn();
    }
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "COMT", 4) == 0) {
  if (act[1] == 'R') {
    resp += CTmax;
  }
  else if ((act[1] == 'W') && (nParam != 0) ) {
    CTMaxCatcher();
    resp += CTmax;
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "PRSS", 4) == 0) {
  if (act[1] == 'R') {
    PreviewLastSession();
```

```
    resp += SC; resp.concat('/');
    resp += SSTA; resp.concat('/');
    resp += IM; resp.concat('/');
    resp += MD; resp.concat('/');
    resp += TSES; resp.concat('/');
    resp += DSES;
   }
   else {
    badcmd();
   }
   goto endsub;
 }

endsub:
 {
   cmdT = micros(); //actualize timestamp of last command

   if (resp != "") {
    response = (act[1] == 'W' ? "(w/" : (act[1] == 'R' ? "(r/" : "(p/"));
    response.concat(reg); response.concat('/'); response.concat(resp); response.concat(')');
    Serial.println(response);

    //clear string to be sent to the operator
    response = "";
```

```
    resp = "";
   }


 }
}


void badcmd() {
 Serial.print("(#/"); Serial.print(reg); Serial.println(")"); //error message
}
void badsyn() {
 Serial.print("(!/"); Serial.print(reg); Serial.println(")"); //error message
}


//PROGRAMA <PROGRAMADETECTOR_LIBRARY>


void NewSessionCreation(int inParam) {
 SSTA = 0;
 SC = inParam;
 EEPROM.write(direcEEPROM, SC);
}
void NewSessionCleanUp() { //Reset all of the variables that could be initialized
 strcpy(TString, "0");
 strcpy(DString, "0");
 strcpy(k1String, "0");
```

```
strcpy(k2String, "0");

strcpy(k3String, "0");

strcpy(KString, "0");

strcpy(FString, "0");

strcpy(CTmaxString, "50000000");


NewSessionMakerValidator = false;

ModeOperationSelectorValidator = false;

DeliveryRequestValidator =  false;

MeasureOperationSelectorValidator = false;

ConversionMethodSelectorValidator = false;

NewSessionNeeded =  true;


KSetupValidator = false;

K1SetupValidator = false;

K2SetupValidator = false;

K3SetupValidator = false;


SC = 0;

DCON = 0;

IM = 0;

T = 0.00;

D = 0.00;

MD = 0;
```

```
  F = 0.00;
  k1 = 0.00;
  k2 = 0.00;
  k3 = 0.00;
  K = 0.00;
  SSTA = 0;
  TSES = 0.00;
  DSES = 0.00;

  Beam_OnFlag = false;
  Beam_OffFlag = false;
  Beam_StpFlag = false;

}
void EndSessionSafe() {
  EEPROM.put(6, SSTA);
  EEPROM.write(8, IM);
  EEPROM.write(10, MD);
  EEPROM.put(46, TSES);
  EEPROM.put(14, DSES);
  //SafeData();
  NewSessionCleanUp();
}
void PreviewLastSession() {
```

```
  SC = EEPROM.read(direcEEPROM);
  EEPROM.get(6, SSTA);
  IM = EEPROM.read(8);
  MD = EEPROM.read(10);
  EEPROM.get(46, TSES);
  EEPROM.get(14, DSES);
  NewSessionNeeded = false;
  MeasureOperationSelectorValidator = true;
  NewSessionMakerValidator = true;
  ModeOperationSelectorValidator = true;
}
float DoseCalculator(int IM, float D, float T, int MD, float F) {
  if ((IM == 2) && (MD == 11)) {
    if (!KSetupValidator) {
      KGet = (k1 * k2 * k3);
    }
    return D / KGet;
  }
  else if ((IM == 2) && (MD == 12)) {
    return D / K;
  }
  else if ((IM == 1) && (MD == 11) ) {
    if (!KSetupValidator) {
      KGet = (k1 * k2 * k3);
```

```
    }
    return T * KGet;
  }
  else if ((IM == 1) && (MD == 12)) {
    return T * K;
  }
  else {
    return 0;
  }
}
float DoseCatcher(float TREC) {
  if (MD == 11) {
    if (!KSetupValidator) {
      KGet = (k1 * k2 * k3);
    }
    return TREC / KGet;
  }
  else if (MD == 12) {
    return TREC * K;
  }
  else {
    return 0;
  }
}
```

```
void DoseRevelator() {
  if (IM == 1) {
    Dtoget = DoseCalculator(IM, D, T, MD, F);
  }
  else if (IM == 2) {
    Ttoget = DoseCalculator(IM, D, T, MD, F);
  }
}
void SftEvent() {
  CT = micros() - cmdT;
  if (CT > CTmax) {
    mode = false;
    SSTA = 4;
    digitalWrite(Rele, LOW);
    digitalWrite(Light, LOW);
  }
}
byte BeamCanStart() {
  byte beamok = 0;
  if (NewSessionMakerValidator && CoherenceValidator() && ModeOperationSelectorValidator && DeliveryRequestValidator && MeasureOperationSelectorValidator && ConversionMethodSelectorValidator && !NewSessionNeeded && mode) {
    beamok = 1;
    DCON = 1;
```

```
    return beamok;
  }
  else {
    DCON = 0;
    return beamok;
  }
}
bool CoherenceValidator(){
  if((IM == 1) && (DRADFlag)){
    DRADFlag = false;
    DeliveryRequestValidator = false;
    ModeOperationSelectorValidator = false;
    return false;
  }
  if ((IM == 2) && (TRADFlag)){
    TRADFlag = false;
    DeliveryRequestValidator = false;
    ModeOperationSelectorValidator = false;
    return false;
  }
  if((MD == 11) && (KSetupValidator)){
    KSetupValidator = false;
    MeasureOperationSelectorValidator = false;
    return false;
```

```
  }
  if ((MD == 12) && ((K1SetupValidator)||(K2SetupValidator)||(K3SetupValidator))){
    K1SetupValidator = false;
    K2SetupValidator = false;
    K3SetupValidator = false;
    MeasureOperationSelectorValidator = false;
    return false;
  }
  return true;
}
void KIsSet() {
  if (KSetupValidator) {
    ConversionMethodSelectorValidator = true;
  }
  else if (K1SetupValidator && K2SetupValidator && K3SetupValidator) {
    ConversionMethodSelectorValidator = true;
  }
}
void BeamControlExecuter() {
  if (Beam_OnFlag) {
    if (IM == 1) {
      //     if (ATimeEmitted > 0) {
      //       MomentumTimer = micros() - ATimeEmitted;
      //       ATimeEmitted = 0;
```

```
  //    }
  TIter = micros() - MomentumTimer;
  TSES = TIter;
  Dmoment = DoseCalculator(IM, DSES, TSES, MD, F);
  DSES = Dmoment;
  digitalWrite(Rele, HIGH);
  digitalWrite(Light, HIGH);
  if (Dmoment >= Dtoget) {
   Serial.print(TSES);
   SSTA = 2;
   Serial.println(" finished");
   SessionEnd();
  }
 }
 else if (IM == 2) {
  //    if (ATimeEmitted > 0) {
  //      MomentumTimer = micros() - ATimeEmitted;
  //      ATimeEmitted = 0;
  //    }
  TIter = micros() - MomentumTimer;
  TSES = TIter;
  DSES = DoseCatcher(TSES);
  digitalWrite(Rele, HIGH);
  digitalWrite(Light, HIGH);
```

```
    if (TSES >= Ttoget) {
      Serial.print(TSES);
      SSTA = 2;
      Serial.println(" finished");
      SessionEnd();
    }
  }
 }
 // else if (Beam_StpFlag) {
 //   SSTA = 2;
 //   digitalWrite(Rele, LOW);
 //   digitalWrite(Light, LOW);
 // }
 else if (Beam_OffFlag) {
   SessionEnd();
 }
}
// Read 1.1V reference against AVcc
int readVcc()
{

#if defined(__AVR_ATmega32U4__) || defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
 ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
```

```
#elif defined (__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) || defined(__AVR_ATtiny84__)
  ADMUX = _BV(MUX5) | _BV(MUX0);
#elif defined (__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
  ADMUX = _BV(MUX3) | _BV(MUX2);
#else
  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
#endif
  // set the reference to Vcc and the measurement to the internal 1.1V reference


  delay(2); // Wait for Vref to settle
  ADCSRA |= _BV(ADSC); // Start conversion
  while (bit_is_set(ADCSRA, ADSC)); // measuring


  uint8_t low  = ADCL; // must read ADCL first - it then locks ADCH
  uint8_t high = ADCH; // unlocks both


  long result = (high << 8) | low;


  //result = 1125300L / result; // Calculate Vcc (in mV); 1125300 = 1.1*1023*1000
  result = scaleConst / result; // Calculate Vcc (in mV); 1125300 = 1.1*1023*1000
  return result; // Vcc in millivolts


}
```

```
//PROGRAMA <PROGRAMADETECTOR_SERVICE>
void NewSessionMaker(int inParam) {
  NewSessionCreation(inParam);
  NewSessionNeeded = false;
  NewSessionMakerValidator = true;
}
void ModeOperationSelector() {
  IM = inParam;
  ModeOperationSelectorValidator = true;
}
void TimeDeliveryCatcher() {
  T = flParam;
  strcpy(TString, stParam);
  DeliveryRequestValidator = true;
}
void DoseDeliveryCatcher() {
  D = flParam;
  strcpy(DString, stParam);
  DeliveryRequestValidator = true;
}
void MeasureOperationSelector() {
  MD = inParam;
  if (MD == 11) {
    MeasureOperationSelectorValidator = true;
```

```
  }
  else if (MD == 12) {
    MeasureOperationSelectorValidator = true;
  }
  else if (MD == 20) {
    ModeOperationSelectorValidator = true;
    RS232_Flag = true;
    NewSessionNeeded = false;
    NewSessionMakerValidator = true;
    MeasureOperationSelectorValidator = true;
    ConversionMethodSelectorValidator = true;
  }
}
void KConstantsCatcher(byte KSelector) {
  if (KSelector == 0) {
    K = flParam;
    strcpy(KString, stParam);
    KSetupValidator = true;
  }
  else if (KSelector == 1) {
    k1 = flParam;
    strcpy(k1String, stParam);
    K1SetupValidator = true;
  }
```

```
  else if (KSelector == 2) {
    k2 = flParam;
    strcpy(k2String, stParam);
    K2SetupValidator = true;
  }
  else if (KSelector == 3) {
    k3 = flParam;
    strcpy(k3String, stParam);
    K3SetupValidator = true;
  }
}
void ConversionMethodCatcher() {
  F = flParam;
  strcpy(FString, stParam);
  ConversionMethodSelectorValidator = true;
}
void CTMaxCatcher() {
  CTmax = inParam;
  strcpy(CTmaxString, stParam);
}
void SessionEnd() {
  digitalWrite(Rele, LOW);
  digitalWrite(Light, LOW);
  EndSessionSafe();
```

```
  Serial.println(F("Saving..."));
  NewSessionCleanUp();
  RS232_Flag = false;
}

//PROGRAMA <PROGRAMADETECTOR_INTERPRETARRS232>

void ControlValues(String response232) {
  char checkpoint = '.'; //Point to check were the decimal time start
  char separator = ';'; //response separator by default
  String analyzer = ""; //String to analyze dose and time
  String receivertimeString = "";
  String receiverdoseString = "";
  analyzer = response232.indexOf(checkpoint);

  receivertimeString = response232.substring(analyzer.length() + 2,
response232.lastIndexOf(separator));
  receiverTime = st2fl(receivertimeString);

  receiverdoseString = response232.substring(analyzer.length() + 3,
response232.indexOf(separator));
  receiverDose = st2fl(receiverdoseString);
}
```

```
float st2fl(String st)

{

 float fl = 0.0;

 byte e;

 //convert reply string to numeric value

 for (byte i = 0; i < st.indexOf('E'); i++) { //to parse the value without taking into account the '.'

  if ( (st[i] > 47) && (st[i] < 58) ) {

   fl *= 10; fl += (st[i] - 48);

  }

 }

 fl = fl / pow(10, (st.indexOf('E') - st.indexOf('.') - 1));

 //to parse the multiplier

 e = st[st.indexOf('E') + 2] - 48;

 if ( st[st.indexOf('E') + 1] == '-' ) {

  fl = fl / pow(10, e);

 }

 else if ( st[st.indexOf('E') + 1] == '+' ) {

  fl = fl * pow(10, e);

 }

 return fl;

}

void RS232Control() {

 if (Beam_OnFlag) {

  if (IM == 1) {
```

```
  if (ATimeEmitted > 0) {
    delayTime += receiverTime - ATimeEmitted;
    ATimeEmitted = 0;
  } else {
    delayTime = 0;
  }
  TSES = receiverTime - delayTime;
  digitalWrite(Rele, HIGH);
  if (TSES >= Ttoget) {
    digitalWrite(Rele, LOW);
    Serial.println(TSES);
    Serial.println(" finished");
    SessionEnd();
  }
}
else if (IM == 2) {
  if (ATimeEmitted > 0) {
    delayTime += receiverTime - ATimeEmitted;
    ATimeEmitted = 0;
  }
  TSES = receiverTime;
  digitalWrite(Rele, HIGH);
  if (receiverDose >= Dtoget) {
    digitalWrite(Rele, LOW);
```

```
      Serial.println(TSES);
      Serial.println(" finished");
      SessionEnd();
    }
   }
  }
  else if (Beam_StpFlag) {
   ATimeEmitted = TSES;
   SSTA = 2;
   digitalWrite(Rele, LOW);

  }
  else if (Beam_OffFlag) {
   SessionEnd();
  }
}

//PROGRAMA <PROGRAMADETECTOR_LEERCOMANDORS232>

void RS232_Communication() {
  if (CheckCommunication_Flag) {
   RS232cmd("MV");
   ControlValues(RS232read());
   RS232Control();
```

```
  }
    else {
      RS232_CheckCommunication();
    }
}



///Function to send String from RS232
void RS232cmd (String command232) {
  reply232 = "";
  RS232.listen(); //sets listen of Softwareserial on RS232 port.
  RS232.print(command232 + CR +LF);
  delay (delaycom);

  if (debug) {
    Serial.print(F("cmd232 : "));
    Serial.println(command232 + CR +LF);
  }
}

///Function to read RS232 response
String RS232read() {
  String cadena = "";
  while (RS232.available()) {
```

```
    cadena.concat(char(RS232.read()));
  }
  if (debug) {
    Serial.print(F("RS232 read: "));
    Serial.println(cadena);
  }
  return (cadena[0] != 'M' ? (cadena[0] != 'P' ? "E": cadena.substring(0)) : cadena.substring(3));
}

void RS232_CheckCommunication() {
  int i;
  String caller = "";
  for (i = 0; i <= 2; i++) {
    RS232cmd("PTW");
    caller = RS232read();
    if (caller != "E") {
      CheckCommunication_Flag = true;
      Serial.print("Connection is Ready");
      break;
    }
  }
  if (!CheckCommunication_Flag) {
    Serial.println(F("Connection Failed..."));
    MD = 0;
```

```
   MeasureOperationSelectorValidator = false;
  } else {
   RS232cmd("STA");
  }
}
```

B2. Programa principal y de interpretación de instrucciones por protocolo TCP/IP

```
//PROGRAMA <PROGRAMADETECTORETHERNET>
#include <SPI.h>
#include <EEPROM.h>
#include <Ethernet.h>
#include <SoftwareSerial.h>

#define delaycom 50 //this is the delay after sending a command via RS232 port

SoftwareSerial RS232(2, 3);//Rx & Tx pins

/// MAC address
//
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0E, 0xE1, 0x22 };
```

```
/// IP address
//
byte ip[] = { 10, 0, 0, 14 };


/// Mask
//
byte subnet[] = { 255, 0, 0, 0 };


/// Port to listen (default 23, Telnet service)
//
EthernetServer server = EthernetServer(23);


///Initial Status
//
boolean debug = false;
boolean mode = true;
boolean online = true;


///RS232 control variables
//
const char LF = 10;
const char CR = 13;
String reply232; //String to receive the response from the Ionization Chamber
bool CheckCommunication_Flag = false;
```

```
bool RS232_Flag = false;

float receiverDose = 0.00;

float receiverTime = 0.00;

unsigned long delayTime = 0;


///Led Status

//

#define Rele 12

#define Light 13


///System Booleans

//

bool NewSessionMakerValidator = false; //New Session Validator Boolean

bool ModeOperationSelectorValidator = false; //Operation Mode Validator Boolean

bool DeliveryRequestValidator =  false; //Delivery Dose Validator Boolean

bool MeasureOperationSelectorValidator = false; //Measurement Mode Validator Boolean

bool ConversionMethodSelectorValidator = false; //Conversion Method Validator Boolean

bool NewSessionNeeded =  true; //New Session Creation Validator Boolean


///K Selections Booleans

//

bool KSetupValidator = false;

bool K1SetupValidator = false;

bool K2SetupValidator = false;
```

```
bool K3SetupValidator = false;


///Coherence validator
//
bool DRADFlag = false;
bool TRADFlag = false;
///Session Settings
//

#define direcEEPROM 0 // EEPROM byte where is keeping the value of the session code

unsigned int SC = 0; //This is the session code

byte NSES = 0; //This is the session begin request
byte DCON = 0; //This is the Beam ableness variable

///Adquisition Dates To PING
//
unsigned int IM; //Irradiation Mode
float T = 0.00; //Delivered Time
float D = 0.00; //Delivered Dose
unsigned int MD; //Dose Determination Mode
float F = 0.00; //Eventual Conversion Factor
```

```
///Ping Variables

//

char TString[10] = "0";

char DString[10] = "0";

char k1String[10] = "0";

char k2String[10] = "0";

char k3String[10] = "0";

char KString[10] = "0";

char FString[10] = "0";

char CTmaxString[10] = "50000000";


// Variables declared for SerialTranslator function

char act[5];// define ACTION to do with register

char reg[5]; //define which REGISTER is in use to make the defined action

byte nParam;// contain de numbers of parameters

char stParam[10];// contain the string passed to parameter where the maximum are 10 chars

float flParam;// contain the parameter converted to float

int inParam;// contain the parameter converted to int



char buffer[20] = "";


String response = "";

String resp = "";
```

```
///Dose/Time Operation needed variables

//

float k1 = 0.00;

float k2 = 0.00;

float k3 = 0.00;

float K = 0.00;

float KGet = 0.00;


///Last Command Timer

//

unsigned long cmdT = 0; // contain time stamp of last command process


///Dosimeters in Dose Delivery

//

float Dtoget = 0.00;

float Dmoment = 0.00;


///Timers in Dose Delivery

//

unsigned long Ttoget;

unsigned long TIter;


///Run Control Timer
```

```
unsigned long MomentumTimer;


///Maximum Time to do a PING
unsigned long CTmax = 50000000; //50 seconds
unsigned long CT = 0; //Connection Timeout


///Task Control Timers
#define ControlTime 500000 //control period from RS232
unsigned long int ts1;
unsigned long int t01;



///Execution Flags
//
bool Beam_OnFlag = false;
bool Beam_OffFlag = false;
bool Beam_StpFlag = false;


///Execution Indicator
//
byte SSTA = 0;
unsigned  long TSES = 0.00;
unsigned  long DSES = 0.00;
unsigned long ATimeEmitted = 0.00;
```

```
const long scaleConst = 1100.300 * 1000; //internal ref * 1023 * 1000



void setup() {
 // OUTPUT Pin definitions
 //
 pinMode(Rele, OUTPUT);
 pinMode(Light, OUTPUT);
 digitalWrite(Rele, LOW);
 digitalWrite(Light, LOW);


 // Here starts the Serial Communication
 //
 //  Keyboard.begin();
 Serial.begin(115200);
 Serial.println("RS232 running");
 //Set the speed of the communication
 RS232.begin(57600);
 Serial.println("Waiting for data input...");
 // Timers Initialization
 //
 ts1 = 0;
 t01 = 0;
 ///Here start the timers
```

```
  //
  cli();

  TCCR1A = 0x00;
  //prescaler
  TCCR1B = 0x00;
  TCNT1 = 0;
  OCR1A = 15640; //(16*10^6)/(5*1024)-1
  TCCR1B |= (1 << WGM12);
  // Set CS12 and CS10 bits for 1024 prescaler
  TCCR1B |= (1 << CS12) | (1 << CS10);
  // enable timer compare interrupt
  TIMSK1 |= (1 << OCIE1A);

  sei();
}
ISR(TIMER1_COMPA_vect) {
  SftEvent();
}
void loop() {
  if (BeamCanStart() == 1) {
    if (!RS232_Flag) {
      BeamControlExecuter();
    }
```

```
   else {
     RS232_Communication();
   }

 }
 if (comandoEthernet() == 1) {
   ///Those commands that the Detector can reads
   if ((act[1] == 'W') || (act[1] == 'R') || (act[1] == 'P')) {
     EthernetTranslator();
   }
 }
 // Get rid of the values:  act, reg, stParam
 memset(act, '\0', sizeof(act));
 memset(reg, '\0', sizeof(reg));
 memset(stParam, '\0', sizeof(stParam));
 nParam = 0; inParam = 0; flParam = 0;
}

byte comandoEthernet() {
 EthernetClient client = server.available();
 byte cmdok = 0;
 if (client.read() == '(') {
   double t = millis();
   char c = 0;
```

```
byte j = 0;
boolean salir = true;
while ( salir ) {
 c = client.read();
 if ( c > 0 ) {
  buffer[j + 1] = c;
  j++;
  if ( debug ) {
   server.print(c); server.print("--"); server.println(c, DEC);
  }
  if ( c == ')' ) {
   j--;
   cmdok = 1;
   break;
  }
 }
 if ( cmdok == 1 ) {
  buffer[0] = j + 1; buffer[j + 1] = '\0'; //finalizar buffer quitar el carracter ')' final
  char * pch; //puntero array de chars
  if ( debug ) {
   server.println(buffer);
  }
  pch = strtok(buffer, "/"); //Look for / to find the action
  strcpy(act, pch); act[5] = '\0'; //Set the act variable value
```

```
int count = 1;
while ( pch != NULL ) {
 if ( count == 2 ) {
  strcpy(reg, pch);  //The second one is the reg parameter
  reg[5] = '\0';
 }
 if ( count == 3 ) {
  strcpy(stParam, pch);  //The param is at third input
  inParam = atoi(pch);
  flParam = atof(pch);
 }
 pch = strtok(NULL, "/");
 count++;
}
nParam = count - 3;

if ( debug ) {
 server.print("act="); server.println(act);
 server.print("reg="); server.println(reg);
 server.print("number= "); server.println(nParam);
 server.print("stParam="); server.println(stParam);
 server.print("inParam="); server.println(inParam);
 server.print("flParam="); server.println(flParam);
}
```

```
    }
   }
   return cmdok;
  }
}

//PROGRAMA <INTERPRETARCOMANDOETHERNET>

void EthernetTranslator() {
 String cadena = "";
 if (strncmp(reg, "STMP", 4) == 0) {
  if (act[1] == 'P') {
   resp += mode; resp.concat('/');
   resp += DCON; resp.concat('/');
   resp += SC; resp.concat('/');
   resp += TString; resp.concat('/');
   resp += DString; resp.concat('/');
   resp += SSTA; resp.concat('/');
   resp += IM; resp.concat('/');
   resp += MD; resp.concat('/');
   resp += TSES; resp.concat('/');
   resp += DSES; resp.concat('/');
   resp += k1String; resp.concat('/');
   resp += k2String; resp.concat('/');
```

```
    resp += k3String; resp.concat('/');
    resp += KString; resp.concat('/');
    resp += FString; resp.concat('/');
    resp += debug; resp.concat('/');
    resp += readVcc();


  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "NSES", 4) == 0) {
  if ((act[1] == 'W') && (nParam != 0)) {
    if (inParam == 1) {
      NSES = 1;
      NewSessionCleanUp();
      resp += NSES;
    }
    else {
      badsyn();
    }
  }
  else {
```

```
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "SECD", 4) == 0) {
  if (act[1] == 'R') {
    resp += SC;
  }
  else if (act[1] == 'W' && (nParam != 0)) {
    if (!NewSessionNeeded) {
      badsyn();
    }
    else {
      NewSessionMaker(inParam);
    }
    resp += SC;
  }
  else {
    badcmd();
  }
  goto endsub;
}

if (strncmp(reg, "IMOD", 4) == 0) {
```

```
  if (act[1] == 'R') {
    resp += IM;
  }
  else if ((act[1] == 'W') && (nParam != 0) ) {
    if ((inParam == 1) || (inParam == 2)) {
      ModeOperationSelector();
      resp += IM;
    }
    else {
      badsyn();
    }
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "METD", 4) == 0) {
  if (act[1] == 'R') {
    resp += MD;
  }
  else if ((act[1] == 'W') && (nParam != 0) ) {
    if ((inParam == 11) || (inParam == 12) || (inParam == 20)) {
      MeasureOperationSelector();
```

```
      resp += MD;
    }
    else {
      badsyn();
    }
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "TRAD", 4) == 0) {
  if (act[1] == 'R') {
    resp += TString;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    TimeDeliveryCatcher();
    TRADFlag = true;
    resp += TString;
  }
  else {
    badcmd();
  }
  goto endsub;
```

```
 }
 if (strncmp(reg, "DRAD", 4) == 0) {
  if (act[1] == 'R') {
    resp += DString;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    DoseDeliveryCatcher();
    DRADFlag = true;
    resp += DString;
  }
  else {
    badcmd();
  }
  goto endsub;
 }
 if (strncmp(reg, "K1ST", 4) == 0) {
  if (act[1] == 'R') {
    resp += k1String;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    byte KSelector = 1;
    KConstantsCatcher(KSelector);
    KIsSet();
    resp += k1String;
```

```
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "K2ST", 4) == 0) {
  if (act[1] == 'R') {
    resp += k2String;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    byte KSelector = 2;
    KConstantsCatcher(KSelector);
    KIsSet();
    resp += k2String;
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "FREP", 4) == 0) {
  if (act[1] == 'R') {
    resp += k3String;
```

```
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    byte KSelector = 3;
    KConstantsCatcher(KSelector);
    KIsSet();
    resp += k3String;
  }
  else {
    badcmd();
  }
  goto endsub;
 }
if (strncmp(reg, "KSET", 4) == 0) {
  if (act[1] == 'R') {
    resp += KString;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    if (!ConversionMethodSelectorValidator) {
      byte KSelector = 0;
      KConstantsCatcher(KSelector);
      KIsSet();
      resp += KString;
    }
    else {
```

```
      resp += KString;
    }
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "FSET", 4) == 0) {
  if (act[1] == 'R') {
    resp += FString;
  }
  else if ((act[1] == 'W') && (nParam != 0)) {
    ConversionMethodCatcher();
    resp += FString;
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "SERC", 4) == 0) {
  if (act[1] == 'R') {
    resp += DCON;
```

```
  }
  else {
    badcmd();
  }
  goto endsub;
}
if (strncmp(reg, "BEAM", 4) == 0) {
  if ((act[1] == 'W') && (nParam != 0)) {
    if (inParam == 1) {
      if (BeamCanStart() == 1) {
        DoseRevelator();
        Beam_StpFlag = false;
        Beam_OnFlag = true;
        MomentumTimer = micros();
        SSTA = 1;
        resp += TSES; resp.concat('/');
        resp += DSES;
      }
      else {
        badsyn();
      }
    }
    if (inParam == 0) {
      if (BeamCanStart() == 1) {
```

```
      Beam_OnFlag = false;
      Beam_OffFlag = true;
      SSTA = 3;
      resp += TSES; resp.concat('/');
      resp += DSES; resp.concat('/');
      resp += SSTA;
    }
    else {
     badsyn();
    }
  }

 }
 else {
  badcmd();
 }
 goto endsub;
}
if (strncmp(reg, "SSTA", 4) == 0) {
 if (act[1] == 'R') {
  resp = SSTA;
 }
 else {
  badcmd();
```

```
  }
  goto endsub;
 }
 if (strncmp(reg, "SRES", 4) == 0) {
  if (act[1] == 'R') {
   resp += SSTA; resp.concat('/');
   resp += TSES; resp.concat('/');
   resp += DSES;
  }
  else {
   badcmd();
  }
  goto endsub;
 }
 if (strncmp(reg, "COMT", 4) == 0) {
  if (act[1] == 'R') {
   resp += CTmax;
  }
  else if ((act[1] == 'W') && (nParam != 0) ) {
   CTMaxCatcher();
   resp += CTmax;
  }
  else {
   badcmd();
```

```
  }
  goto endsub;
}
if (strncmp(reg, "PRSS", 4) == 0) {
  if (act[1] == 'R') {
    PreviewLastSession();
    resp += SC; resp.concat('/');
    resp += SSTA; resp.concat('/');
    resp += IM; resp.concat('/');
    resp += MD; resp.concat('/');
    resp += TSES; resp.concat('/');
    resp += DSES;
  }
  else {
    badcmd();
  }
  goto endsub;
}
//  if (strncmp(reg, "BSTP", 4) == 0) {
//    if (act[1] == 'W') {
//      if (BeamCanStart() == 1) {
//        Beam_StpFlag = true;
//        Beam_OnFlag = false;
//        ATimeEmitted = TSES;
```

```
//       SSTA = 3;
//       resp += TSES;
//     }
//     else {
//       badsyn();
//     }
//   }
//   else {
//     badcmd;
//   }
//   goto endsub;
// }
//MODE DEBUG
if ( strncmp(reg, "DBUG", 4) == 0 ) {
  if ( act[1] == 'R' ) {
    resp += debug;
  }
  else if ( act[1] == 'W' ) {
    if ( (inParam == 0 || inParam == 1) ) {
      debug = inParam;
      resp += debug;
    }
    else {
      badsyn();
```

```
    }
  }
  else {
    badcmd();
  }
  goto endsub;
}
//MODE STATUS
if ( strncmp(reg, "MODE", 4) == 0 ) {
  if ( act[1] == 'R' ) {
    resp += mode;
  }
  else if ( act[1] == 'W') {
    if ( (inParam == 0 || inParam == 1) ) {
      mode = inParam;
      resp += mode;
    }
    else {
      badsyn();
    }
  }
  else {
    badcmd();
  }
```

```
  goto endsub;
 }
 //MODE ONLINE
 if ( strncmp(reg, "ONLN", 4) == 0 ) {
  if ( act[1] == 'R' ) {
   resp += online;
  }
  else if ( act[1] == 'W' ) {
   if ( (inParam == 0 || inParam == 1) ) {
    online = inParam;
    resp += online;
   }
   else {
    badsyn();
   }
  }
  else {
   badcmd();
  }
  goto endsub;
 }
endsub:
 {
  cmdT = millis(); //actualize timestamp of last CMD
```

```
   if (resp != "") {
     response = (act[1] == 'W' ? "(w/" : (act[1] == 'R' ? "(r/" : "(p/"));
     response.concat(reg); response.concat('/'); response.concat(resp); response.concat(')');
     server.println(response);

     //clear string to be sent to the operator
     response = "";
     resp = "";
   }

  }
}

void badcmd() {
  server.print("(#/"); server.print(reg); server.println(")"); //error message
}
void badsyn() {
  server.print("(!/"); server.print(reg); server.println(")"); //error message
}
```

B3. Programa para el estudio del cálculo de la dosis y el tiempo (MatLab)

```
%% This is a script was made for the visualization of the calibration curve that

%% there will be in the accelerator doses delivery.


clc;

clear all;


%% Variables definitions

K1 = 0.253;

K2 = 7.9;

K3 = 4.7;

T = 1:1:300;


%% Assigned Time in Method 11

KDef1 = K1 * K2 * K3;

for i = 1:1:300

   X = i/1000;

  D1(i, :) = [(KDef1*X) i];

end


figure(1);
```

```
plot(T, D1(:,1)); ylabel('Dose [Gy]'); xlabel('Time[s]');

legend('D[Gy] = K * T[s]'); title('Evolve of Dose in Method 11');


%% Assigned Dose in Method 12

%Example 1: K = 9400;


KDef2 = 9400;

for i = 1:1:300

   X = i/1000;

  D2(i, :) = [(KDef2*X) i];

end


figure(2);

plot(T, D2(:,1)); ylabel('Dose [Gy]'); xlabel('Time[s]');

legend('D[Gy] = K * T[s]'); title('Evolve of Dose in Method 12 (K = 9400Gy/s)');


%% Assigned Time in Method 12

%Example 2: K = 24Gy/s


KDef3 = 24;

for i = 1:1:300

   X = i/1000;

  D3(i, :) = [(KDef3*X) i];

end
```

```
figure(3);
plot(T, D3(:,1)); ylabel('Dose [Gy]'); xlabel('Time[s]');
legend('D[Gy] = K * T[s]'); title('Evolve of Dose in Method 12 (K = 24Gy/s)');

%% Assigned Time in Method 11

KDef4 = K1*K2*K3;
for i = 1:1:300
   X = i/1000;
  T1(i, :) = [(X/KDef4) i];
end

figure(4);
plot(T, T1(:,1)); ylabel('Dose [Gy]'); xlabel('Time[s]');
legend('T[s] = D[Gy]/K'); title('Evolve of Time in Method 12');

%% Assigned Time in Method 12
%Example 1 : K = 9400Gy/s

KDef5 = 9400;
for i = 1:1:300
   X = i/1000;
  T2(i, :) = [(X/KDef5) i];
```

```
end


figure(5);
plot(T, T2(:,1)); ylabel('Dose [Gy]'); xlabel('Time[s]');
legend('T[s] = D[Gy]/K'); title('Evolve of Time in Method 12 (K = 9400Gy/s)');


%% Assigned Time in Method 12
%Example 1 : K = 24Gy/s


KDef6 = 24;
for i = 1:1:300
   X = i/1000;
  T3(i, :) = [(X/KDef6) i];
end
```