Ferran Peralta Queral

Activació i Monitoratge d'un Sistema d'Anodització d'Alúmina Nanoporosa mitjançant Raspberry Pi

Treball Fi de Grau

dirigit per Josep Ferré Borrull

Grau d' Enginyeria en Electrònica Industrial i Automàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2023

Índex

1. Introducció

1.1 Objectiu del treball

Aquest treball sorgeix com una necessitat de millora del sistema de control i monitoratge del procés d'anodització d'alúmina nanoporosa que s'utilitza als laboratoris del grup de recerca NEPHOS (NanoElectronic and PHOtonic Systems) de l'ETSE, liderat pel professor Lluis Marsal.

1.2 Procés d'anodització

Per tal de posar en context el procés a controlar, es fa una breu explicació sobre en què consisteix.

L'anodització consisteix en fer que una làmina d'alumini sigui l'ànode en una cel·la electroquímica que conté un electròlit, el qual normalment és una dissolució d'un àcid. Per tal d'aconseguir que l'alumini sigui l'ànode, aquest es posa en contacte amb una base de coure connectada al terminal positiu d'una font de corrent o tensió. L'altra cara de la làmina d'alumini és la que es troba en contacte amb l'electròlit. El càtode es connecta a un elèctrode de platí, que és un metall inert que es troba submergit a l'electròlit. El corrent que passa a través de l'alumini provoca la formació d'una capa d'òxid damunt seu. Al mateix temps, l'àcid ataca la capa d'òxid, de manera que es formen petits orificis al llarg de tota la làmina.

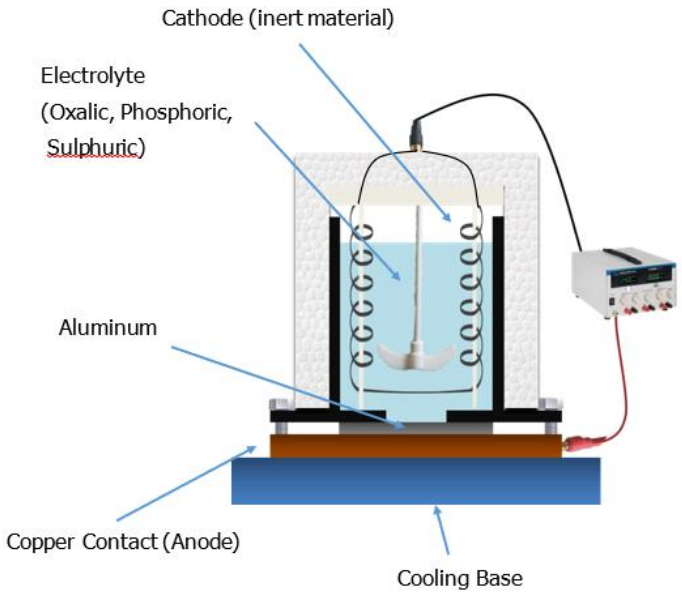

Figura 1. Representació de la cel·la electroquímica emprada al laboratori.

A continuació es poden veure diferents fases del procés en dos tipus de funcionament diferents, corrent o tensió constants.

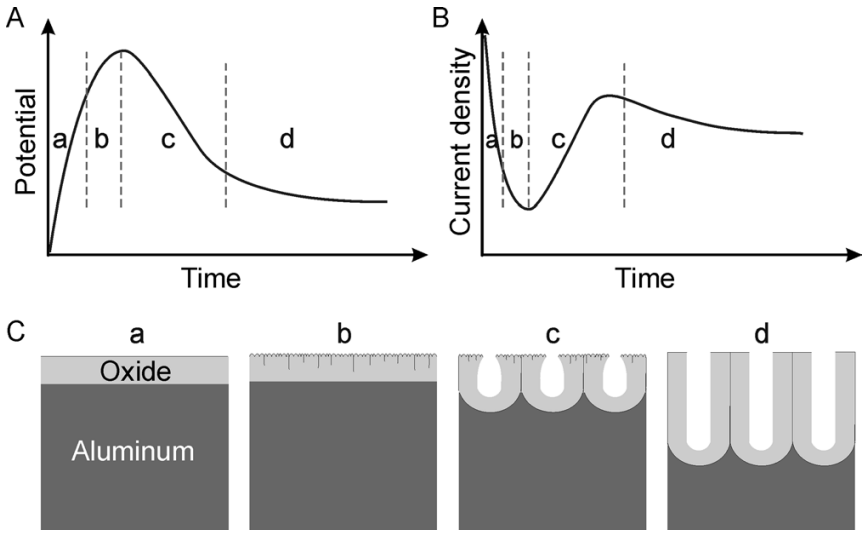

Figura 2. Diferents etapes del procés.

A l'inici del procés, es forma una capa d'òxid que modifica la resistivitat de l'alumini, provocant que el corrent o tensió no es mantingui constant. Aquesta capa d'òxid s'anomena capa barrera. En un cert instant, aquesta capa barrera comença a dissoldre's a l'interfície òxid-electròlit. En aquests punts el gruix de la capa barrera disminueix donant lloc a una disminució de la resistència del circuit i per tant a una disminució de la tensió aplicada o un augment del corrent aplicat. En els punts on s'ha començat la dissolució de la capa barrera, la reacció electroquímica fa que es creï una nova capa a més profunditat, fent que els porus comencin a créixer. Després d'un cert temps, el creixement dels porus s'estabilitza donant lloc a una estabilització del corrent o la tensió. El sistema de control i monitoratge s'ha d'encarregar de mantenir constant la tensió o el corrent. En el cas de tenir una tensió constant, el procés s'anomena potenciostàtic (gràfica de la dreta), mentre que en el cas de tenir un corrent constant el procés s'anomena galvanostàtic (gràfica de l'esquerra).

En aquest projecte es realitza un sistema de control per un procés potenciostàtic i es donen indicacions de com fer un procés galvanostàtic. Per tant, es manté constant la tensió aplicada a la cel·la i mesura periòdicament la intensitat del corrent subministrat.

1.3 Sistema actual i problemes que planteja

El sistema implementat actualment als laboratoris fa ús d'uns "sourcemeter" que permeten aplicar les tensions i corrents corresponents a la cel·la electroquímica utilitzada en el procés d'anodització i alhora fer les mesures pertinents per controlar el procés que es porta a terme. Aquests aparells es comuniquen amb un ordinador via port sèrie RS-232 o a través d'una interfície GPIB, on l'usuari pot ajustar els paràmetres de l'aparell en funció del procés a dur a terme. El model utilitzat al laboratori és el Keithley 2400 Series SourceMeter. Aquests dispositius ofereixen una connexió molt fiable, una mesura precisa i compleixen a la perfecció la funció a realitzar durant el procés d'anodització. Per altra banda, el llenguatge de comunicació a través del port sèrie és estàndard i és vàlid per altres instruments de mesura i fonts d'altres fabricants com Agilent o Hewlett-Packard.

La comunicació es realitza mitjançant el port GPIB de la font, on cal fer ús d'un adaptador a USB per realitzar la connexió amb l'ordinador. El port GPIB (General Purpose Interface Bus) és un estàndard de comunicació d'un ordinador amb dispositius electrònics de mesura, com oscil·loscopis, generadors de funcions o multímetres que va ser desenvolupat per part de la empresa tecnològica Hewlett-Packard sobre l'any 1970.

Tot i ser un port de comunicació altament fiable, està sobredimensionat per a les necessitats d'instrumentació del laboratori de recerca. El fet d'haver d'adquirir els adaptadors corresponents per realitzar la connexió amb l'ordinador també és un punt negatiu ja que implica un cost addicional.

Figura 3. Adaptador GPIB a USB

Finalment, el software que permet monitoritzar i actuar sobre els aparells és un programari basat en LabView, el qual és un llenguatge de programació de molt alt nivell utilitzat per l'anàlisi de mesures i adquisició de dades. Malauradament, el programari utilitzat és d'ús comercial i requereix actualitzar la llicència periòdicament, la qual és d'ús educacional però no utilitzable per a recerca.

2. Definició d'especificacions

A continuació s'exposen les especificacions que ha de complir el nou sistema per tal de millorar l'actual.

- **Keithley 2400 Series SourceMeter**

Per realitzar les mesures corresponents i aplicar les tensions i corrents per als processos es farà ús d'aquest sourcemeter, el qual ja s'utilitzava actualment. Tal i com s'ha mencionat anteriorment, aquests dispositius ofereixen una comunicació altament fiable, motiu pel qual es decideix no substituir-los per uns de nous.

- **Dispositiu on executar el software**

El software de control i monitoratge no requereix de grans prestacions. Solament és necessària una bona comunicació via port sèrie amb el "sourcemeter" per obtenir les mesures necessàries. Per aquest motiu, es planteja la possibilitat de deixar de banda l'utilització d'ordinadors, i estudiar l'ús d'un dispositiu molt més econòmic, amb dimensions més reduïdes per ser més portable, amb un sistema operatiu que pugui executar sense problemes el software i que disposi d'un port sèrie d'alta fiabilitat.

- **Protocol de comunicació**

Es requereix substituir l'actual port GPIB per un protocol de comunicació més simple i més adequat per al procés a realitzar. És fonamental que mantingui una alta fiabilitat en la comunicació.

- **Utilització de software lliure i de fàcil manteniment**

Per tal de solucionar les mancances que presenta el software, es demana fer el programari des de nou. Haurà de basar-se en un programari lliure que presenti les eines necessàries per fer una bona interfície d'usuari per realitzar una bona comunicació amb els aparells.

- **Sistema interactiu per a l'usuari/gràfic/monitoratge remot**

També es sol·licita un mètode per poder realitzar un monitoratge remot del procés que es porta a terme al laboratori.

- **Possibilitat d'ampliació per part del "client/usuari"**

El projecte ha de permetre que el client que en faci ús pugui realitzar les modificacions i ampliacions que trobi oportunes de la manera més senzilla possible.

3. Solució proposada

3.1 Eines utilitzades al sistema

Un cop clares les especificacions del projecte, es procedeix a explicar la solució proposada. Primerament, es defineixen les eines que s'han triat per complir les especificacions donades.

- **Keithley 2400 Series SourceMeter**

Tal i com s'ha introduït prèviament, els dispositius encarregats de fer les mesures de tensions, corrents i impedàncies, i aplicar els voltatges i les intensitats corresponents per a dur a terme el procés d'anodització són els Keithley 2400 Series SourceMeter.

Aquests aparells es caracteritzen per la seva alta precisió, proporcionant unes mesures amb cinc dígits. Un altre punt fort és la seva gran estabilitat, tant en el seu funcionament com a font com en la comunicació sèrie, permetent fer fins 2000 mesures per segon sense cap problema.

A continuació es mostra una imatge de l'aparell:

Figura 4. Aparell Keithley 2400 model que es disposa al laboratori.

A la part dreta, es poden observar les femelles on van connectades les sondes que realitzen les mesures o apliquen els corrents i tensions. El dispositiu també permet realitzar mesures a quatre fils, de manera que es pot veure la potència instantània consumida per la càrrega. A la part posterior també es tenen quatre femelles iguals. Amb el botó que s'observa a la part inferior dreta, es pot seleccionar quines femelles volem utilitzar.

Just a la seva esquerra, es troba el botó que permet activar o desactivar la sortida de corrent o voltatge, de manera que si configurem la font amb uns determinats valors, no activarà la sortida fins que es premi aquest botó. Al seu costat, es troben un parell de fletxes que permeten modificar el rang desitjat de la mesura.

Els botons de la part central es divideixen en dos grans grups: els que corresponen al funcionament com a font i els que pertanyen a la part de mesures. A la part de font es pot triar si es vol aplicar un voltatge o un corrent mitjançant dos botons amb la lletra "V" i "I" respectivament. Al seu costat apareixen dues fletxes, una apuntant amunt i una altra avall, que serveixen per incrementar o disminuir el valor de sortida aplicat. A la banda de mesures es repeteixen els dos botons amb les lletres "V" i "I", i s'afegeix un altre amb la lletra "Ω", Els tres botons permeten seleccionar quin paràmetre es vol mesurar: tensió, corrent o impedància. El botó "FCTN" permet fer operacions matemàtiques amb les mesures fetes. Amb la tecla "CONFIG" i posteriorment prement la tecla corresponent es pot accedir als diversos modes que té l'aparell. De la resta de botons es pot destacar el que posa "SPEED", el qual permet

configurar la velocitat de les mesures, el de "DIGITS" amb el qual es pot canviar el número de dígits que apareixen al display i el de "STORE" que activa l'emmagatzematge de mesures.

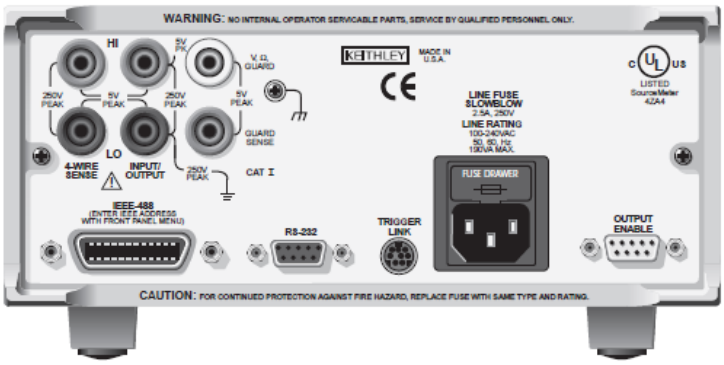

Figura 5. Part posterior del "sourcemeter"

Per la part posterior de l'instrument podem veure les femelles per connectar les sondes de les quals s'ha parlat abans. Just baix, estan els GPIB i RS-232 per realitzar la comunicació sèrie. També es troba el port per connectar el "trigger", el port d'alimentació i el connector per a sortides digitals.

El motiu més important pel que s'elegeix aquesta font és per la seva facilitat per realitzar la comunicació sèrie. Per tal de configurar els diferents modes i paràmetres, se li envien una sèrie de ordres en forma de comanda que ordenen la acció a realitzar. A continuació es mostren totes les comandes disponibles:

- :SOURce:FUNCtion[:MODE] <name>: posa l'aparell com a font de corrent o tensió, en funció del que es posi a la posició d'on ara apareix "name" a la comanda (VOLT o CURR).
- :SOURce:CURRent:MODE FIXed: estableix un corrent fixe durant tota la execució.
- :SOURce:VOLTage:MODE FIXed: estableix una tensió fixa durant tota la execució.
- :SOURce:CURRent:RANGe <n>: configura l'interval de corrent desitjat.
- :SOURce:VOLTage:RANGe <n>: configura l'interval de voltatge desitjat.
- :SOURce:CURRent:LEVel <n>: configura el valor de voltatge desitjat.
- :SOURce:VOLTage:LEVel <n>: configura el valor de corrent desitjat.
- :SENSe:FUNCtion <function>: estableix el mode de mesura.
- :SENSe:CURRent:PROTection <n>: ajusta un valor màxim de corrent de protecció.
- :SENSe:VOLTage:PROTection <n>: ajusta un valor màxim de tensió de protecció.
- :SENSe:CURRent:RANGe <n>: configura un rang fe mesura de corrent.
- :SENSe:VOLTage:RANGe <n>: configura un rang de mesura de tensió.
- :OUTPut <state>: selecciona l'estat de la sortida de l'aparell (ON o OFF).
- :READ?: adquireix una lectura.

En aquestes comandes, cal substituir el que es troba entre "<>" pel valor o paràmetre corresponent per a cada comanda.

- **Port RS-232**

Figura 6. Aspecte d'un connector RS-232.

Per tal d'adequar millor el sistema a les necessitats dels processos del laboratori, s'opta per fer ús del port RS-232.

RS-232 es un protocol creat als anys 60 que defineix com es transfereixen les dades. És l'estàndard comú que s'utilitza als ports sèrie, definint les propietats elèctriques, la sincronització i interpretació de senyals, la mida i la configuració dels pins del connector. Tot i que en les darreres generacions d'ordinadors s'ha anat substituint per l'USB, avui en dia encara són molt utilitzats en grans màquines industrials i equips científics on connexions de dades per cable de baixa velocitat (menys de 20 kbps) són suficients.

El connector a utilitzar disposa de nou pins, dels quals podem destacar els següents:

- RxD (pin de recepció): s'encarrega de rebre les dades.
- TxD (pin de transmissió) que s'encarrega de transmetre les dades.
- Protecció de terra: aquesta senyal està connectada al xassís del connector metàl·lic.
- Terra comú: estableix el nivell de voltatge zero de referència per a tots els senyals.
- DTR (terminal de dades preparades): indica si el DTE (Data Terminal Equipment) està llest per acceptar la sol·licitud de connexió.
- DCD (Detector de dades): accepta l'enviament d'un DTE.
- DSR (conjunt de dades preparat): indica si el DCE està preparat per enviar i rebre la informació.

Donat que la majoria de dispositius actuals ja no utilitzen aquest port, es requereix fer ús d'un adaptador RS-232 a USB.

Figura 7. Adaptador RS-232 a USB

- **Raspberry Pi**

Figura 8. Raspberry Pi 4 Model B

Per tal de substituir l'ús actual d'un ordinador convencional per realitzar el control del procés d'anodització, s'opta per fer ús d'una Raspberry Pi.

Aquest dispositiu és un ordinador monoplaca (computadora completa en un sol circuit) pensat per a que tot tipus de persones es puguin iniciar en el món de la informàtica. L'idea de la Raspberry Pi neix l'any 2006 de la mà de l'enginyer britànic Eben Upton, el qual es va proposar crear una placa que integrés tots els elements per funcionar per ella sola i buscant que tingués un baix cost per a que qualsevol usuari pugues disposar d'un petit ordinador. Per tal de fer-ho possible, es crea "The Raspberry Foundation", la qual tenia com a objectiu garantitzar que el cost d'aquest dispositiu fos d'uns 30 dòlars aproximadament. No va ser fins al 2012 quan es va aconseguir llançar la primera Raspberry Pi al mercat, amb els models Zero i Pico. Fins al dia d'avui, han sortit al mercat multitud de models, permetent que els usuaris puguin triar entre tots ells adaptant el model del dispositiu al tipus d'ús que se li vol donar.

Tots els models inclouen almenys un port USB, una connexió HDMI i un jack 3.5mm, de manera que permet la connexió de perifèrics com ratolins, teclats, pantalles o altaveus per tal de poder navegar per l'interfície d'usuari, cosa que per al ús que se li pretén donar és més que suficient. Apart, al mercat hi ha disponibles molts més perifèrics compatibles amb el dispositiu, com lectors d'empremta o càmeres.

Tot i que la Raspberry Pi té un software oficial, és possible instal·lar-li altres sistemes operatius com Windows o Android. En el cas d'aquest projecte s'opta per fer ús de Raspberry OS, que és la última versió del sistema oficial del dispositiu, el qual està basat en Linux. La principal raó és per fer ús d'un sistema operatiu que s'ha dissenyat específicament per aquest dispositiu, evitant problemes de rendiment i incompatibilitats. També és molt més còmode per executar el software que controlarà el "sourcemeter", ja que el programari lliure que s'utilitzarà ja està implementat de sèrie al sistema operatiu.

- **Python**

L'entorn de programació elegit per programar el software que controlarà el sistema és Python. Python és un llenguatge de programació d'alt nivell creat a finals dels anys 80 per Guido van Rossum, i és majoritàriament utilitzat en aplicacions web i desenvolupament de software.

Presenta una sintaxis molt clara i fàcil de entendre, i disposa de gran quantitat de biblioteques de funcions, les quals faciliten considerablement la programació. Algunes de les biblioteques utilitzades en aquest projecte són:

- sys: proporciona funcions i variables per a la interacció amb l'intèrpret de Python i el sistema.
- os: conjunt de funcions i mètodes que permeten interactuar amb el sistema operatiu. Es poden gestionar fitxers i directoris, controlar processos, manipular rutes de fitxers i obtenir informació sobre el sistema.
- PyQt6: és la biblioteca que permet crear la part gràfica del programa. Dins la biblioteca hi ha altres biblioteques, com la QtWidgets, la qual conté les classes que permeten afegir elements com botons, gràfics, caixes de text, etc. També es fa ús de la biblioteca QtCore, la qual és part del framework PyQt i proporciona les funcionalitats bàsiques per a l'aplicació d'interfícies gràfiques d'usuari amb PyQt. Dins la biblioteca anterior tenim la classe QTimer, que permet crear temporitzadors en una aplicació PyQt, permetent programar l'execució de tasques en moments específics o a intervals regulars, i que serà útil per realitzar les mesures de forma periòdica.
- matplotlib: és la biblioteca de visualització de dades en 2D i 3D que s'utilitza àmpliament en Python. Proporciona una ampla gamma de funcions per a la creació de gràfics, diagrames, histogrames i altres representacions visuals per a l'anàlisi de dades.
- serial: ofereix una interfície per a la comunicació sèrie amb dispositius externs mitjançant ports sèrie. També es fa ús de la biblioteca "serial.tools.list_ports", que realitza la detecció i llistat dels ports disponibles en el sistema per a la comunicació sèrie.
- telebot: és una biblioteca de Python que permet interactuar amb l'API de Telegram.
- datetime: proporciona la data i hora local

Un altre gran avantatge que ofereix Python és que fa ús de la programació orientada a objectes mitjançant la creació de classes. Les classes en Python permeten encapsular dades i lògica relacionada en objectes, fomentant la reutilització de codi, l'organització i la modularitat del programa, cosa que afavoreix notablement la comprensió del codi.

Finalment, el propi Python incorpora funcions molt interessants, com la funció "try", la qual gestiona les excepcions que es generen durant la execució del codi, permetent que l'usuari pugui indicar quines accions s'han d'executar si es dóna el cas, i evitant problemes d'execució del codi.

És per tots aquest motius, sumat a que Python es tracta d'un programari lliure, que es decideix desenvolupar el software amb aquest llenguatge de programació.

3.2 Implementació

Per tal de facilitar la comprensió de la implementació feta, es desglossa el programa en una sèrie de blocs, els quals engloben diverses funcions, que es representen en el següent diagrama de flux i que posteriorment s'expliquen:

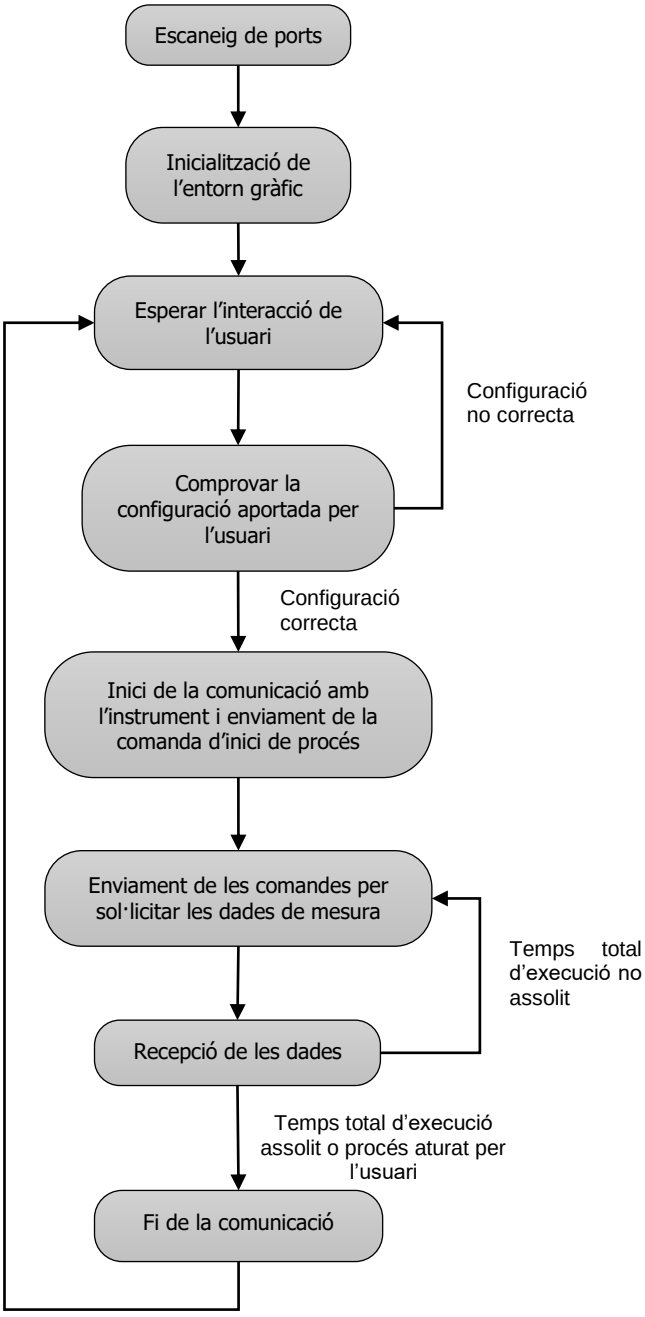

Figura 9. Diagrama de flux del software

- **Escaneig de ports:** realitza un primer escaneig dels ports sèrie per tal l'obtenir el nom identificador de cadascun dels dispositius connectats i mostrar-los a la llista de dispositius un cop s'ha iniciat el programa.

- **Inicialització de l'entorn gràfic:** s'obre la finestra del programa amb tots els elements gràfics necessaris.

- **Esperar l'interacció de l'usuari:** un cop s'ha iniciat el programa i s'ha obert la finestra, el software es manté a l'espera de que l'usuari introdueixi els valors dels paràmetres necessaris i doni la ordre per iniciar el procés d'anodització.

- **Comprovar la configuració aportada per l'usuari:** un cop s'han introduït els paràmetres, s'ha de comprovar que no s'hagi quedat algun valor per configurar i que els que ja estan introduïts són correctes. Aquesta comprovació es realitza quan es dóna l'ordre d'inici.

- **Inici de la comunicació amb l'instrument i enviament de la comanda d'inici de procés:** un cop la configuració és correcta i s'ha donat l'ordre d'inici, s'inicialitza la comunicació amb l'instrument desitjat i se li envien les comandes corresponents per configurar l'instrument amb els paràmetres introduïts.

- **Enviament de les comandes per sol·licitar les dades de mesura**: s'enviarà de manera periòdica la comanda corresponent per rebre les dades mesurades per l'instrument.

- **Recepció de les dades:** es representen les dades de tensió i corrent rebudes en dos gràfics que s'incorporen en l'entorn gràfic. Per tal de que l'usuari pugui fer un seguiment del procés de forma remota, també s'envien les dades rebudes via Telegram gràcies a un bot que s'ha vinculat a l'execució del programari. Finalment, també es desen les dades en un fitxer de text.

- **Fi de la comunicació:** es posa fi a la comunicació amb el dispositiu, s'atura la representació gràfica i es notifica via programa i via Telegram de que el procés ha finalitzat o s'ha aturat.

Un cop entès el funcionament a gran escala del programari, cal parlar sobre l'entorn gràfic que permet l'interacció amb l'usuari. Aquest s'ha realitzat amb la biblioteca PyQt6. A continuació es mostra l'aspecte de la finestra principal del programa:

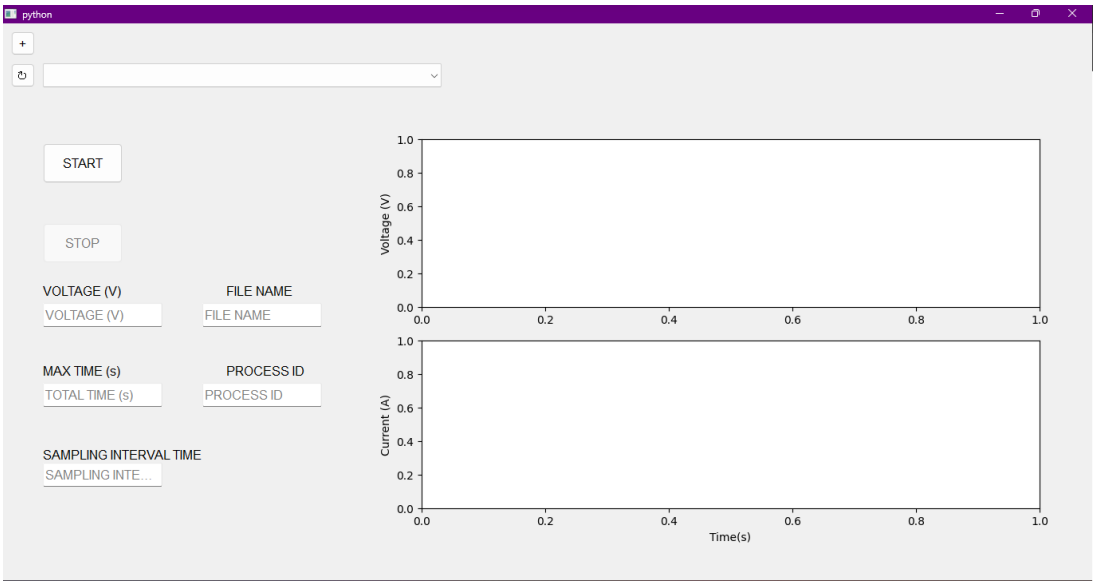

Figura 10. Aspecte de l'entorn gràfic

Els diferents elements que conté la finestra són els següents:

- **Botó "+":** aquest petit botó situat a la part superior esquerra permet obrir una nova finestra idèntica a la que s'obre al iniciar el programari. Es poden obrir tantes finestres com es desitgen, de manera que cada finestra podria controlar un instrument diferent.

- **Llista de dispositius:** llista desplegable que permet triar quin dispositiu dels que hi ha connectats als diferents ports sèrie volem utilitzar.

- **Botó "↻":** situat a l'esquerra de la llista de dispositius, permet actualitzar la llista de dispositius quan es desitgi. En el cas de tenir obertes múltiples finestres, la llista de dispositius s'actualitza a totes, independentment de en quina s'accioni el botó.

- **Botó "START":** dóna l'ordre d'inici d'un procés. Solament estarà habilitat quan no hi hagi cap procés en curs a la finestra on es pretén iniciar el nou procés.

- **Botó "STOP":** permet aturar un procés que s'estigui executant. Estarà habilitat solament quan hi hagi un procés en curs.

- **Quadres de text ("VOLTAGE", "MAX TIME", "SAMPLING INTERVAL TIME", "FILE NAME", "PROCESS ID"):** permeten parametritzar els valors de voltatge a aplicar, temps màxim del procés, temps de mostreig de dades, nom del fitxer on guardar les dades i nom d'identificador del procés respectivament.

- **Gràfiques de dades:** a la dreta de la finestra trobem les dos gràfiques on es representaran les dades de corrent i tensió mesurades pel "sourcemeter" al llarg de tota l'execució del procés.

3.3 Detalls de la programació

A continuació s'aprofundeix a explicar en detall la programació del software i les eines de Python que s'utilitzen per portar-la a terme.

3.3.1 Eines de Python

Una de les característiques més utilitzades que té Python és la programació orientada a objectes. Aquesta és una tècnica molt útil per estructurar i organitzar el codi d'una forma lògica i coherent, facilitant la comprensió del codi i afavorint la reutilització de codi.

La base de la programació orientada a objectes són les classes, les quals serveixen com a plantilla per posteriorment crear diversos objectes. Dins les classes es defineixen un conjunt d'atributs que defineixen l'objecte, normalment solen ser funcions i variables, que posteriorment se'ls hi podrà fer crida de forma externa, com per exemple, des de dins d'una altra classe. Aquí és on entra en joc el concepte de "self". Si es vol fer ús de una variable o una funció definida dins una classe fora de la mateixa classe, el nom de la variable o funció ha d'anar precedit de "self.". El mateix passa si es vol utilitzar una variable entre diferents funcions dins d'una mateixa classe.

Una altra de les eines utilitzades és la funció "try". Aquesta característica de Python serveix per gestionar possibles errors o excepcions que puguin ocórrer durant l'execució del programari, permetent saltar les línies de codi que generen el conflicte, inclús es poden definir certes línies a executar en funció de l'excepció o error que ha ocorregut. L'ús d'aquesta funció és fonamental per evitar errors durant el procés i mals comportaments del programa i inclús evitar que s'aturi l'execució de forma inesperada.

3.3.2 Funcions utilitzades

Per tal de poder interactuar amb els diferents elements del programa i amb l'objectiu de reutilitzar codi i simplificar-lo al màxim, cal fer l'ús de funcions. A continuació es fa un repàs de les diferents funcions que s'han creat per implementar el programa de control:

- **scan():** fa un escaneig dels dispositius connectats al port sèrie. S'obtenen els noms i els ports per tal de afegir-los a la llista de dispositius de la finestra.

- **new_window():** s'executa al prémer el botó '+'. Obre una nova finestra idèntica i totalment independent de la resta que hi puguin haver obertes.

- **new_scan():** s'executa al prémer el botó situat a l'esquerra de la llista de dispositius. Realitza un nou escaneig fent crida de la funció "scan()" vista anteriorment. Actualitza la llista de dispositius a totes les finestres que hi ha obertes, independentment de a quina finestra s'acciona.

- **take_port():** identifica a quin port està connectat el dispositiu seleccionat a la llista desplegable, ja que és necessari saber-ho per posteriorment iniciar la comunicació sèrie.

- **start_com():** inicia la comunicació amb el dispositiu seleccionat de la llista desplegable i envia les comandes corresponents per iniciar el procés.

- **click_bstart():** funció que s'executa al prémer el botó START. Inicia la comunicació sèrie fent crida de la funció "start_com". També s'inicialitzen els gràfics i el temporitzador per tal de començar la representació gràfica, es crea el fitxer on es guarden les dades si no existeix i es notifica via Telegram que s'ha iniciat el procés.

- **click_bstop():** de manera anàloga a la funció vista anteriorment, aquesta s'executa quan fem clic al botó STOP. Atura la representació gràfica i la sortida de la font, tanca el fitxer i notifica via Telegram que el procés ha finalitzat.

- **read():** funció que sol·licita la mesura al dispositiu amb l'enviament d'una comanda. Espera a rebre els paràmetres i guarda els valors de corrent i tensió per posteriorment ser representats. També s'envia la mesura realitzada mitjançant Telegram.

- **graf():** funció que s'executa periòdicament i que sol·licita la mesura del multímetre (fent crida de la funció "read()" vista anteriorment) i representa les dades als gràfics. També calcula el temps d'execució i controla que no s'hagi assolit el màxim. En aquest cas, s'atura el procés. També escriu al fitxer la última mesura.

3.3.3 Programació de l'entorn gràfic

Per la creació de la interfície del programa es fa ús de la biblioteca de Python PyQt6. Es comença declarant la classe MplCanvas, la qual defineix la configuració dels gràfics animats on es representen les mesures. Els paràmetres configurats són el color dels gràfics i la seva disposició, en aquest cas un a sobre de l'altre. A continuació es mostra el codi que ho implementa:

```
class MplCanvas(FigureCanvas):
    def __init__(self, parent=None, width=5, height=4, dpi=100):
        # Es crea una nova figura de matplotlib amb l'amplada, l'alçada i
DPI especificats
        fig = Figure(figsize=(width, height), dpi=dpi)
        # S'afegeixen dos subplots a la figura
        self.axes1 = fig.add_subplot(211)
        self.axes2 = fig.add_subplot(212)
        # Es posa el fons de la figura transparent
        fig.set_facecolor('none')
        # S'estableix la figura com a figura principal del canvas
        super(MplCanvas, self).__init__(fig)
        # Es posa el fons del canvas transparent
        self.setStyleSheet("background-color:transparent;")
```

Posteriorment, es defineix la classe MainWindow, la qual inclou la creació de la finestra i tots els elements gràfics que conté, ajustant la seva mida i posició dins la mateixa. Cada tipus d'element gràfic (botó, caixa de text, llista...) es crea i configura d'una manera diferent. Per més informació es recomana consultar una guia de PyQt6.

Dins la classe MainWindow també es defineixen totes les funcions i variables que interactuen amb algun dels elements creats a la finestra. És fonamental que es declarin dins la classe ja que sinó, no es pot fer ús d'elles. També es defineixen dos petites finestres que notifiquen a l'usuari quan s'inicia el programa i quan es realitza un nou escaneig de dispositius, ja que aquests dos fets requereixen d'un cert temps per executar-se.

Per incloure els gràfics dins la finestra, primerament es crea un "layout". Un "layout" permet organitzar els diferents "widgets" que conté una finestra. En aquest cas, només es fa ús per col·locar el "canvas" que conté els gràfics. Un "canvas" és una superfície de la finestra que permet dibuixar gràfics, formes i altres elements visuals. Donat que es fa ús de gràfics animats, s'han d'incloure els gràfics dins del "canvas".

Finalment, cal aclarir que la declaració d'aquestes classes no crea directament la finestra i els gràfics, sinó que defineix els seus paràmetres i els elements que contenen. crea En aquest cas, es crida al constructor MplCanvas dintre la classe MainWindow per tal d'incorporar els gràfics a l'interior de la finestra.

Per crear la finestra, primerament es crea una instància de la classe "QtWidgets.QApplication". Posteriorment, es crida el constructor per crear la finestra i s'ajusta la seva mida al màxim que permet la pantalla. Finalment, es fa crida a l'instància creada anteriorment per tal de que la execució de l'aplicació entri en un bucle d'events de l'interfície gràfica, permetent que la finestra reaccioni a interaccions de l'usuari.

```
app = QtWidgets.QApplication(sys.argv)
w = MainWindow()
w.showMaximized()
app.exec()
```

3.3.4 Implementació de les funcions

A continuació s'aprofundeix més sobre l'implementació de cada funció.

scan()

Primerament, es defineixen dos matrius on es guardaran els noms i número o nom de port sèrie dels dispositius connectats. Posteriorment, es realitza l'escaneig dels dispositius i es guarda dins d'una mateixa variable anomenada "ports_disp", la qual conté tota la informació sobre tots els dispositius connectats.

```
self.num=[]
self.name=''
ports_disp = serial.tools.list_ports.comports()
```

El següent pas tracta en separar les dades recollides per dispositius i classificar-les en les matrius creades inicialment, però hi ha un petit inconvenient. Com es fa ús d'un adaptador port sèrie a USB per a la connexió dels "sourcemeter", quan es fa la cerca dels noms dels aparells, apareix el nom de l'adaptador i no el del dispositiu. Per aquest motiu, s'intenta establir la connexió amb cadascun dels dispositius i s'envia la comanda "*IDN?\n", amb la qual la font respon amb el seu nom d'identificació. El motiu d'enviar la comanda a cadascun d'ells és perquè no es sap a quin dels ports està connectat el "sourcemeter", de manera que cal fer-ho a tots. Per tal d'evitar que es generin errors, ja sigui intentant iniciar la connexió a un dispositiu ja connectat o esperant la resposta de la comanda "*IDN?\n" a un dispositiu al "sourcemeter", es fa ús de la funció de Python "try" per tal de gestionar les possibles excepcions i evitar errors d'execució. Tots els noms i ports obtinguts es van desant dintre les dues matrius creades inicialment, i que posteriorment seran representades a la llista de dispositius.

```
try:
    ser = serial.Serial(port.device, 9600, timeout=0.1, write_timeout=0.1)
    a = '*IDN?\n'
    ser.write(a.encode('utf-8'))
    s0 = b'0'
    name = str()
    try:
        while s0 != b'\r':
            s0 = ser.read()
            name = name + s0.decode('utf-8')
            name = name + ' (' + port.device + ')'
            self.desc.append(name)
            self.num.append(port.device)
            ser.close()
        except:
            name = port.description
            self.desc.append(name)
            self.num.append(port.device)
    except (serial.SerialException, serial.SerialTimeoutException):
        name = port.description
        self.desc.append(name)
        self.num.append(port.device)
self.starting.close()
```

Un cop obtingudes totes les dades necessàries per a la llista de dispositius, es tanca una petita finestra anomenada "self.starting", la qual s'obre prèviament a l'execució d'aquesta funció, just al iniciar el programa, i notifica a l'usuari que el software ja s'està executant, ja que aquest procés requereix d'un cert temps, i si no es notifica pot donar la impressió que el programari no està funcionant.

new_window()

Aquesta funció simplement obre una nova finestra, i s'executa al polsar el botó "+". El primer que fa al executar-se és fer un nou escaneig dels dispositius. Posteriorment, es netegen els dispositius que apareixen abans de polsar el botó a la llista de dispositius i s'afegeixen els que s'han detectat en la nova recerca. A continuació es crea la nova finestra fent crida al constructor de la classe "MainWindow()" i es maximitza. Finalment, cal tornar a actualitzar la llista de dispositius per tal que aparegui a la nova finestra.

new_scan()

Aquesta funció s'executa al polsar el botó situat a l'esquerra de la llista de dispositius. Al polsar, s'obre una petita finestra que indica que s'està actualitzant la llista, al igual que passa amb la funció "scan()". Mentre es manté la finestra oberta, es realitza un nou escaneig i novament es neteja la llista de dispositius i s'afegeixen els resultats de la nova recerca. Es fa el mateix amb les finestres que s'han obert posteriorment, però s'incorpora dins d'un bloc "try", ja que en el cas que solament hi hagi una finestra oberta es podria generar una excepció. Un cop ja s'han actualitat les llistes de totes les finestres, es tanca la petita finestra que notifica que s'estava actualitzant.

take_port()

L'objectiu d'aquesta funció és obtenir el número/nom identificador del dispositiu al port sèrie que s'ha elegit a la llista de dispositius, i que és necessari per realitzar la connexió posteriorment. La pròpia llibreria de PyQt6 permet obtenir aquest paràmetre fàcilment, i posteriorment es desa a una variable local de la funció. Finalment, la funció retorna aquesta variable.

start_com()

Inicialment, s'obté l'identificador del port sèrie mencionat anteriorment i el guarda a la variable local "com". Posteriorment, ja s'estableix la connexió, passant com a paràmetre l'identificador, la velocitat en bauds (la qual sempre son 9600 bauds), i es posa el paràmetre "timeout" a 1. Seguidament, s'envien les comandes al dispositiu que estableixen el "sourcemeter" com a font de tensió i es configura per a mesurar tensió i corrent. També s'activa la sortida de la font, i s'envia la comanda que estableix el valor de tensió especificat a l'interfície d'usuari. Finalment, per tal d'evitar errors, es posa un "delay" d'un mil·lisegon. Aquesta petita pausa pot ajudar a que l'instrument tingui un temps per processar les dades enviades.

click_bstart()

Aquesta funció s'executa al polsar el botó START. Primerament, es posa a zero la variable "self.stop", que ens marca si el programari està aturat o no. Seguidament, es netegen els gràfics per si es donés el cas que continguessin dades d'un procés anterior. També s'inicialitza una nova variable que posteriorment servirà per calcular el temps d'execució i tres matrius ("self.x", "self.y" i "self.z"), les quals contindran les dades a representar (temps, corrent i tensió). L'eix d'abscisses ("self.x") correspon al temps d'execució, el qual que és comú per a les dos variables, motiu pel qual solament n'hi ha un per a les dos gràfiques.

El següent pas és comprovar que tots els paràmetres a configurar tenen algun valor i no estan buits. Si es dóna el cas, s'obre una petita finestra que notifica a l'usuari que falta algun paràmetre. Si els paràmetres són correctes, s'inicia la comunicació sèrie fent crida de la funció "start_com()" vista anteriorment.

Seguidament, s'extreu el nom de la carpeta on es troba l'arxiu Python que s'està executant i es crea un arxiu de text a la mateixa carpeta. El nom d'aquest arxiu serà l'especificat per l'usuari prèviament. Un cop creat, s'obre l'arxiu amb mode escriptura, i se li escriu una capçalera on apareix el nom d'identificació del procés i el nom dels tres paràmetres que es guardaran (temps, voltatge i corrent), ja que les dades s'escriuran al arxiu de text en forma de tres columnes, on cadascuna de elles correspondrà a un paràmetre diferent. També es notifica via Telegram a l'usuari que s'ha iniciat un nou procés.

A continuació es converteixen a enters les variables que contenen el temps màxim d'execució i el valor de tensió a aplicar, les quals són especificades per l'usuari a l'interfície, i per defecte es guarden com una cadena de caràcters a dintre dels "QLineEdit" anomenats "tmax" i "valorV". Donat que posteriorment s'han de realitzar operacions matemàtiques amb aquestes variables, cal fer aquesta conversió.

Seguidament, s'inicialitza el temporitzador que permetrà fer les mesures de dades de forma periòdica i se li assigna com a interval d'execució el temps de mostreig especificat per l'usuari. També cal assignar-li quina funció es vol que s'executi periòdicament la qual és l'encarregada de representar-les mesures (la funció "graf()"). Un cop ja configurat, s'inicia el temporitzador.

Per tal d'evitar modificacions dels paràmetres i errors, es desactiven tots els quadres de text on l'usuari pot modificar els paràmetres, la llista de dispositius i el botó START, i s'activa el botó STOP per poder aturar el procés quan es desitgi.

Finalment, es configuren dos petites finestres que notificaran quan el procés s'hagi aturat o acabat per assolir el temps màxim d'execució especificat.

click_bstop()

De forma anàloga a la funció anterior, aquesta s'executa al prémer el botó STOP. El primer que fa aquesta funció és reactivar tots els elements que es desactiven al polsar el botó START per poder modificar de nou els paràmetres del procés, es desactiva el botó STOP i es torna a habilitar el botó START.

Seguidament, es desactiva la sortida de la font i es posa a zero volts enviant la comanda corresponent, i un cop enviada es tanca la comunicació. També es posa a 1 la variable "self.stop", que indica quan està en parada, i es notifica vía Telegram que el procés s'ha aturat. També es netegen les matrius on es guarden les dades i s'atura i elimina el temporitzador.

Finalment, s'obre una petita finestra notificant a l'usuari que ha acabat el procés, i es tanca el fitxer on es guarden les dades.

read()

Aquesta funció envia una comanda a l'instrument per tal que prengui una mesura de corrent i tensió. Per fer-ho, envia la comanda corresponent de mesura, i desa les dades rebudes en una cadena de caràcters. El "sourcemeter" retorna els valors de tots els paràmetres en un mateix missatge, els quals són: voltatge mesurat, corrent mesurat, resistència mesurada, temps des de l'engegada de l'instrument i l'estat del dispositiu, el qual proporciona una cadena de 24 bits que informa sobre el mode de funcionament actual del dispositiu. Per aquest motiu, cal extreure solament les dades que interessen, en aquest cas les de tensió i corrent. Donat que el missatge ja separa els diferents valors amb una coma, amb la funció de Python "split" es separen tots els valors i es desen cadascun en una matriu, on el primer element de la matriu correspon al voltatge i el segon al corrent. Aquests dos valors es desen en dues variables locals, i se'ls hi fa la conversió a número real, ja que al missatge s'expressen en notació científica. Finalment, es retornen les dos variables que emmagatzemen les dades.

```
data = meas.split(",") # Separa els números en una llista
volt = float(data[0]) # Es desa el valor de tensió com a número real
curr = float(data[1]) # Es desa el valor de corrent com a número real
return volt, curr
```

graf()

És la funció que el temporitzador executa periòdicament, amb l'interval especificat per l'usuari.

Primerament, es comprova si s'ha assolit el temps màxim especificat o no. En el cas que encara no s'hagi assolit, es sol·licita una lectura i es desen els dos paràmetres en dos variables locals, "a" i "b". Posteriorment, s'afegeixen els valors llegits de tensió i corrent i el temps actual a les matrius "self.x", "self.y" i "self.z". Seguidament, es netegen els gràfics i s'assignen les dades corresponents a cadascun dels eixos dels gràfics. És important netejar els eixos abans ja que no es representen els punts individualment, sinó les matrius de punts al complet, i per tal d'actualitzar les matrius dels gràfics és clau fer-ho. A continuació, s'assignen els títols als eixos de cada gràfic i es dóna la ordre per tal que els punts siguin representats. En el cas de que no s'hagi polsat el botó STOP, s'escriu al fitxer i s'envia via Telegram les mesures fetes. Finalment, s'incrementa la variable "self.i" per indicar que s'ha realitzat una nova execució de la funció i es configuren dos petites finestres que notificaran quan el procés s'hagi aturat o acabat per assolir el temps màxim d'execució especificat.

Tot aquest codi es troba dins d'un bloc "try" per si es donés el cas de que hi hagués una excepció durant una lectura concreta no interrompi l'execució del programa. Si es dóna el cas, es notifica via Telegram i fitxer de que hi ha hagut un error.

En el cas de que s'hagués assolit el temps màxim, el comportament seria similar al de quan es polsa el botó STOP, amb la diferència que la finestra de notificació ens informaria que el procés no ha sigut aturat, sinó que ha finalitzat.

3.4 Prevenció d'errors

El control del procés d'anodització és un punt crític dels experiments, motiu pel qual cal mantenir el programari en execució en tots els possibles casos de contingència. Per aquest motiu, és necessari parar molta atenció a les possibles incidències que es puguin produir durant l'execució del programari, de forma que s'han de aplicar totes les mesures de prevenció possibles per evitar-les. En els processos experimentals, poden ocórrer diversos incidents els quals poden fer que el programa deixi de respondre, tot i que no té per què afectar a la execució del procés.

A continuació s'expliquen les mesures adoptades per minimitzar les incidències que puguin ocórrer:

- Habilitar/desactivar elements: PyQt6 ens permet poder activar o desactivar els diferents elements gràfics que s'utilitzen a la finestra. És molt important que l'usuari no pugui modificar paràmetres mentre un procés està en ús. Per tal de evitar-ho, quan es prem el botó START, queden bloquejades les caixes de text, la llista de dispositius i el botó de START. El botó de STOP queda habilitat per poder aturar el procés. Si es prem el botó d'aturada o finalitza el procés, es tornen a activar els elements anteriors i es desactiva el botó de STOP per tal de poder iniciar un nou procés diferent.

- Fer ús de la funció "try": Com s'ha fet menció en apartats anteriors, es fa ús de la funció pròpia de Python "try". Aquesta permet saltar una línia de codi si ha generat un error o una excepció. Un exemple d'aplicació d'aquesta funcionalitat la podem trobar quan intentem establir connexió amb un dispositiu del port sèrie. El sistema intentarà realitzar la connexió, però si es dona el cas que el dispositiu ja ha establert una connexió prèviament, no es podrà fer efectiva i generarà una excepció. Normalment, l'excepció bloquejaria el programa o fins i tot podria tancar-lo. Aquest

mètode evita que passi, de manera que simplement es salta aquesta línia de codi i la resta que estan dins del bloc "try".

- Evitar paràmetres buits o incorrectes: Si deixem alguna caixa de text buida pot donar problemes a l'execució del programa. Per exemple, si no s'introdueix un temps de mostreig, el timer tindrà un interval d'execució de zero segons, el qual no és possible i provocarà un error a l'execució del procés. Per tal d'evitar-ho, cada cop que anem a iniciar un procés prement el botó START, es comprova que no hi hagi cap caixa de text buida. Si es dona el cas que falta algun paràmetre, es mostra una petita finestra d'alerta informant a l'usuari sobre que ha de revisar els paràmetres. També cal que els valors estiguin dins d'un cert rang. Si no es compleix, també es notifica al usuari amb una finestra emergent.

Tot i així, poden ocórrer algunes incidències difícils de preveure que poden afectar a l'execució del programa. El programari implementat permet un monitoratge continu de l'execució del procés, tant de forma presencial com remota, ja sigui vigilant el dispositiu on s'executa el programari o controlant-ho via Telegram.

3. Joc de proves

3.1 Proves a realitzar

Per tal de realitzar el joc de proves i assegurar-nos que el funcionament del software és correcte, es deixa el software funcionant durant 65000 segons, unes 18 hores, amb un temps de mostreig de 10000 mil·lisegons, connectat mitjançant la Raspberry Pi a l'instrument, que subministrarà 10 volts a una resistència de 1000 Ω, de manera que el valor esperat de corrent a mesurar és d'uns 0,01 A. Es comprovarà que el procés arribi al final del seu temps màxim i que s'han realitzat totes les mesures esperades.

El muntatge realitzat és el següent:

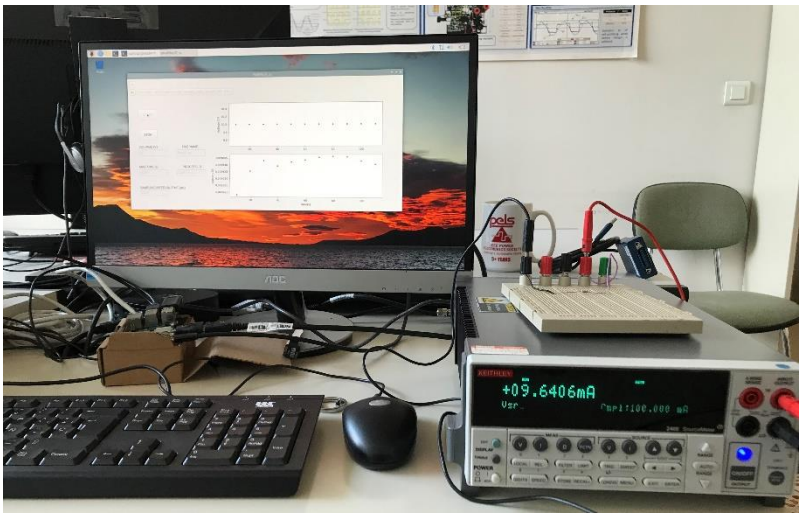

Figura 11. Muntatge realitzat per al joc de proves

Via Telegram, s'anirà comprovant que l'execució no s'aturi fins arribar al temps màxim, i que els valors de les mesures siguin els esperats. També es comprovarà que s'hagin desat totes les mesures al arxiu.

A continuació es mostra la finestra del programari on es configura el procés. Es pot apreciar com apareix el nom del dispositiu a la llista desplegable de la part superior.

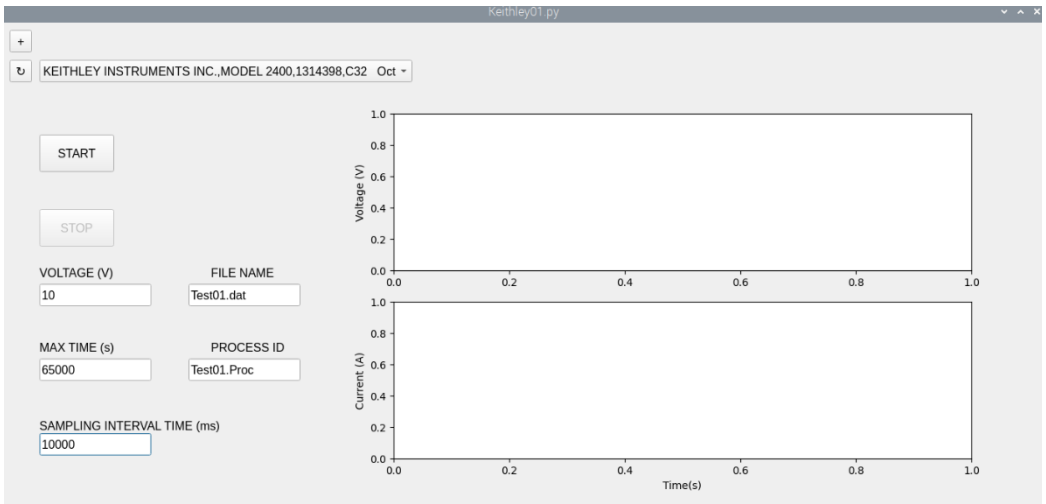

Figura 12. Configuració dels paràmetres del procés.

3.2 Resultats

El resultat de la prova és favorable. El programari i el procés han funcionat a la perfecció durant tota l'execució. A continuació s'adjunta una captura de pantalla al final del procés, on es notifica que el procés ja ha finalitzat i la seva durada total.

Figura 13. Final d'un procés amb el dispositiu monitoritzat amb el programari

S'observa que el procés ha durat 2,33 segons més dels 65000 segons esperats. Aquest fet pot ser degut a una petita desincronització entre el rellotge de Python que executa de forma periòdica la funció de mesura i representació de dades amb el rellotge de l'instrument, fent que es produeixi aquest petit desajust, que comparat amb la durada total del temps del procés és ínfim.

A continuació es mostren les mesures que s'han fet al llarg de tot el procés:

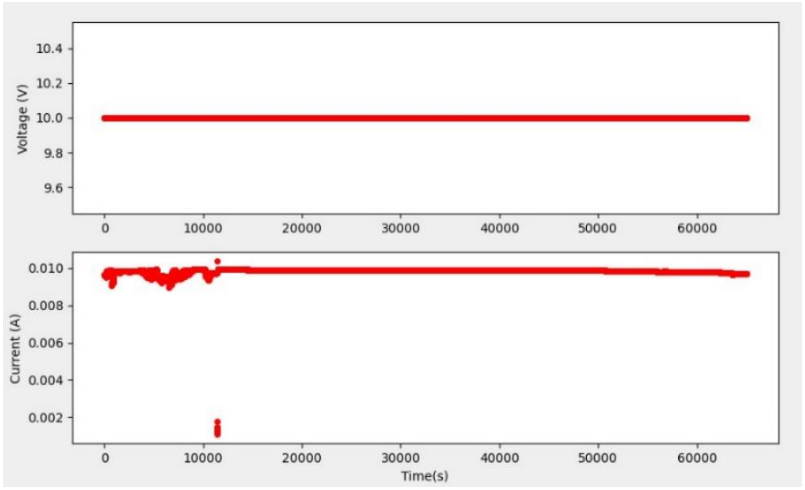

Figura 14. Representació al software de les dades obtingudes

En quan al voltatge, observem que es manté fixe als 10 volts que s'han configurat. En canvi, en el corrent sí s'observen petites variacions, sobretot a l'inici. Cal destacar que la mesura es dona amb vuit decimals (tot i que al gràfic apareixen tres), cosa que ens proporciona un alt grau de precisió a la mesura. Els quatre primers decimals no varien mai, donant una valor de corrent d'uns 0.009 A. El valor és lleugerament inferior al esperat, segurament degut a la tolerància de la resistència, fet que fa que el seu valor real sigui lleugerament superior.

Pel que fa als cinc decimals de menys pes, s'observen petites variacions, normalment davallades, que podem considerar poc significatives. Aquestes variacions poden ser degudes a petits canvis de temperatura o pertorbacions provocades per altres fonts de soroll. On sí es poden observar set valors fora de rang és entre els segons 11413 i 11473, on els valors mesurats són 10 cops més petits de l'esperat, segurament degut a un petit error de mesura de l'instrument. Per sort, aquesta és l'única incidència a destacar.

Pel que fa el monitoratge remot via Telegram, s'han rebut totes les mesures. A continuació es mostra la notificació del moment en el que s'inicia i acaba el procés.

Figura 15. Notificació via Telegram de l'inici i finalització del procés

Efectivament, el temps d'execució és correcte, i el nom del procés coincideix amb el que s'ha configurat a la finestra del programari.

Per últim, cal comprovar que s'hagin guardat correctament les dades al fitxer. Es comprova que estiguin totes les dades i que s'hagin desat en el format corresponent. A continuació es mostra l'inici del procés i el final tal com queda registrat a l'arxiu:

```
Process ID: Test01.Proc
TIME (s) VOLTAGE (V) CURRENT(A)
10.12891 10.0 0.00961802
20.05664 10.0 0.009635536
30.04492 10.0 0.009643191
64972.35 10.0 0.00969039
64982.35 10.0 0.009690247
64992.34 10.0 0.009690546
65002.33 10.0 0.009690511
Process finished
```

Figura 16. Inici i final de procés registrat a l'arxiu

Per tal de comprovar que els valors són correctes, es representen amb Excel per comprobar que les gràfiques coincideixen amb les representades al programari.

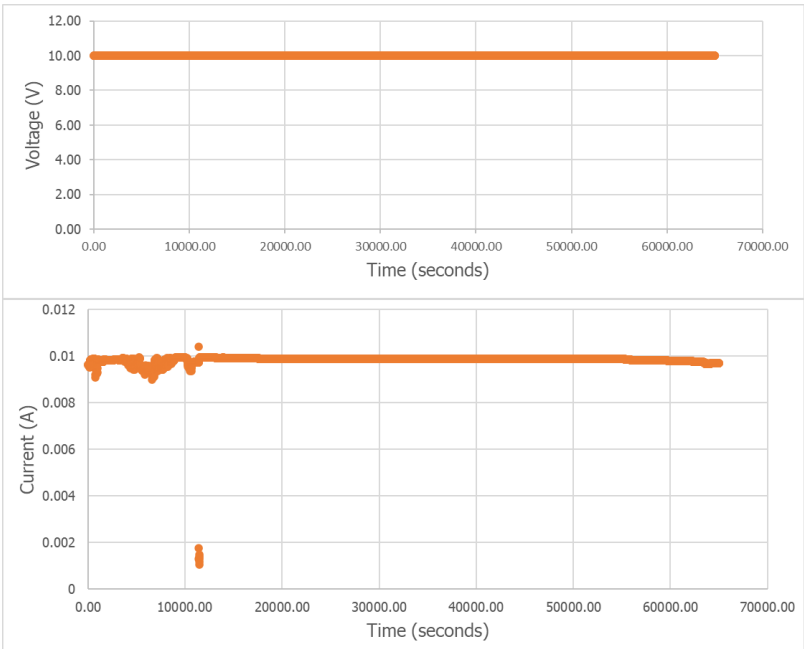

Figura 17. Representació de les dades desades al fitxer

Efectivament, els punts són els mateixos representats a la finestra del programari. Per tant, podem concloure que el joc de proves realitzat ha estat un èxit.

4. Manual d'ús

4.1 Instruccions del programa

A continuació s'explica als usuaris com fer un ús correcte del programari.

Els primer passos a seguir consisteixen en comprovar que la connexió amb el dispositiu sigui via port sèrie. Cal que la connexió s'hagi configurat a una velocitat de transmissió de 9600 bauds. També és necessari habilitar la connexió sèrie RS-232 al dispositiu a controlar i configurar-la com a font de tensió. Al manual del dispositiu es pot consultar sobre com fer-ho.

El següent pas és executar el software. A continuació s'explica com fer-ho per a Raspberry Pi OS, Linux i Windows:

- Raspberry Pi OS i Linux
  - Amb la terminal de comandes: Primerament cal obrir la terminal de comandes i indicar en quina carpeta es troba l'arxiu Python que es vol executar amb la comanda "cd". Un cop la carpeta seleccionada, executem l'arxiu amb la comanda Python. A continuació és mostra un exemple:

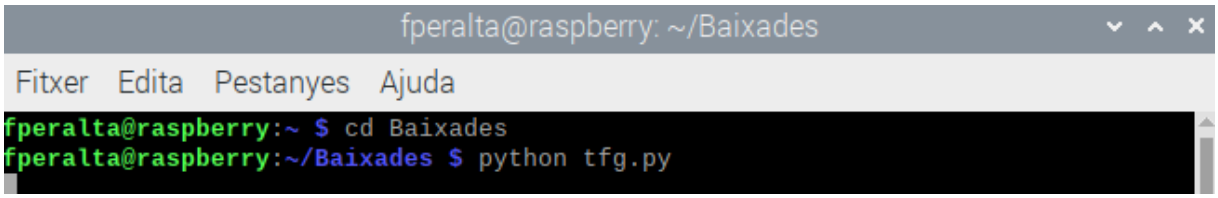

Figura 18. Exemple sobre com s'executa el programari amb el terminal de Raspberry Pi OS.

  - Amb Geany: Geany és l'editor de text que trobem en sistemes basats en Linux. Si fem doble clic damunt l'arxiu ".py", s'obre aquest editor. A la finestra superior apareix una icona d'un avió de paper, la qual al prémer-la se'ns executa el programa.

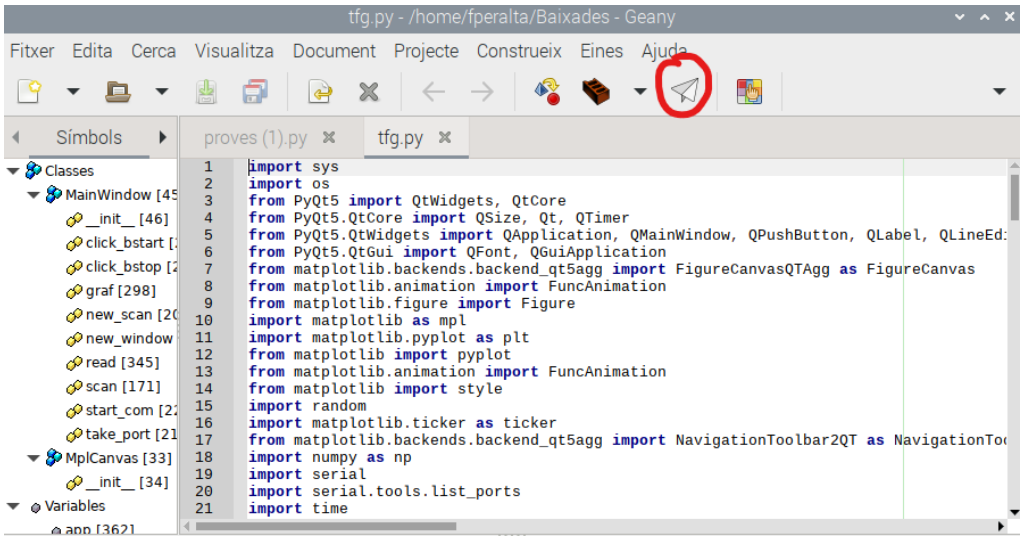

Figura 19. Arxiu .py obert amb Geany

- Windows
  - Amb la terminal de comandes: El procés és molt paregut al de Linux. Primerament, amb la comanda "cd" cal especificar el directori complet en el qual està desat el fitxer. Finalment, amb la comanda Python s'executa el programa. A continuació es deixa un exemple:

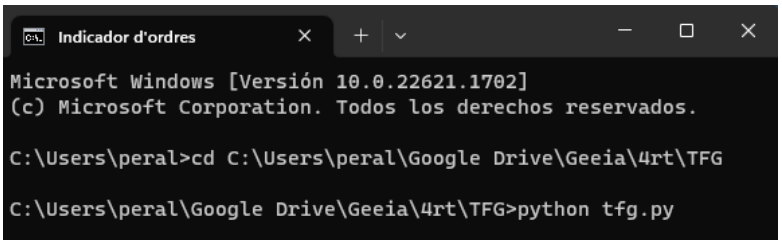

Figura 20. Exemple sobre com s'executa el programari amb el terminal de Windows

  - Amb editors de text de Python: Programes, com per exemple Visual Studio Code, permeten executar l'arxiu .py des de dins del mateix programari.

La finestra pot tardar uns segons a obrir-se, ja que realitzar l'escaneig dels dispositius connectats al port sèrie comporta un cert temps. Per tal d'informar l'usuari, s'obre una petita finestra indicant que s'està realitzant l'escaneig. Un cop finalitzat, es tanca automàticament i s'obre ja la finestra principal.

Un cop tenim oberta la finestra de control, cal donar valor als paràmetres que apareixen a la part esquerra de la pantalla. En el cas que quedi algun paràmetre buit o no estigui dins d'un rang determinat, el procés no s'iniciarà, advertint a l'usuari mitjançant una finestra d'alerta. També és necessari triar el dispositiu en el qual volem realitzar el control a la llista desplegable que es troba a la part superior esquerra. En el cas de connectar el dispositiu un cop el sistema s'ha iniciat, es pot actualitzar la llista de dispositius prement el petit botó (veure a la Figura 13) que hi ha just a l'esquerra de la llista. De la mateixa manera que quan s'inicia el programa, actualitzar la llista de dispositius tarda un instant. Per tal d'informar d'usuari, apareix la mateixa petita finestra que a l'inici.

Si l'usuari desitja realitzar diversos processos a l'hora, pot obrir una nova finestra totalment idèntica prement el botó que hi ha a la part superior esquerra, just damunt del de actualitzar la llista de dispositius. La nova finestra és totalment independent de l'anterior. A continuació es mostra un exemple d'una finestra on encara no s'ha iniciat cap procés.

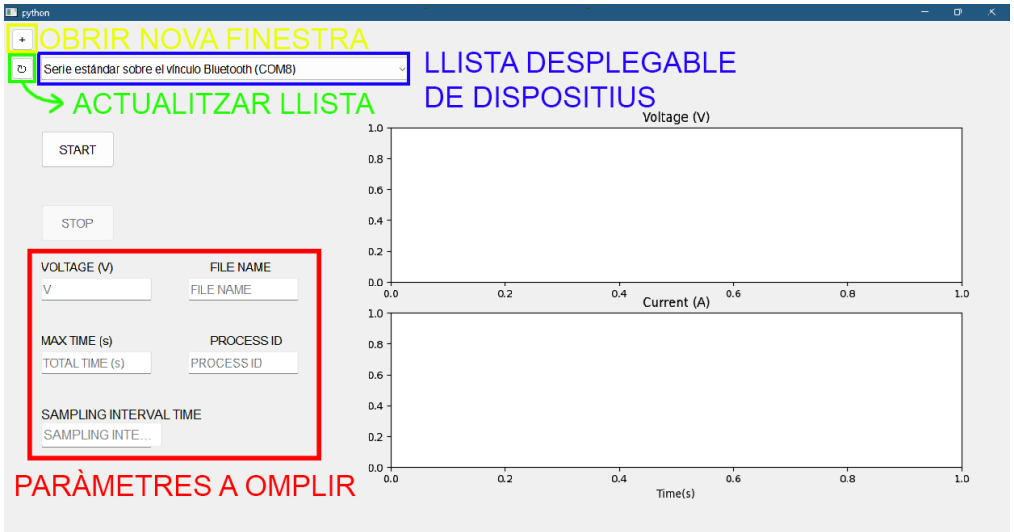

Figura 21. Paràmetres a configurar

Un cop ja s'ha triat el dispositiu i s'han assignat els valors als paràmetres, ja es pot iniciar el procés prement el botó START.

En prémer el botó 'START' s'aplica el voltatge desitjat a la cel·la electroquímica i es prenen mesures cada interval de temps, en aquest exemple cada 1 segon. Quan un procés està actiu no es permet canviar el dispositiu ni modificar els paràmetres. Solament queda habilitat el botó STOP per poder aturar el procés quan es desitgi. Durant l'execució, s'escriuran les dades mesurades en un arxiu de text el qual tindrà el nom que l'usuari hagi posat als paràmetres. També s'anirà notificant via Telegram de cada mesura que es realitza i si s'atura o finalitza el procés. A continuació es mostra una captura de pantalla sobre un procés en execució:

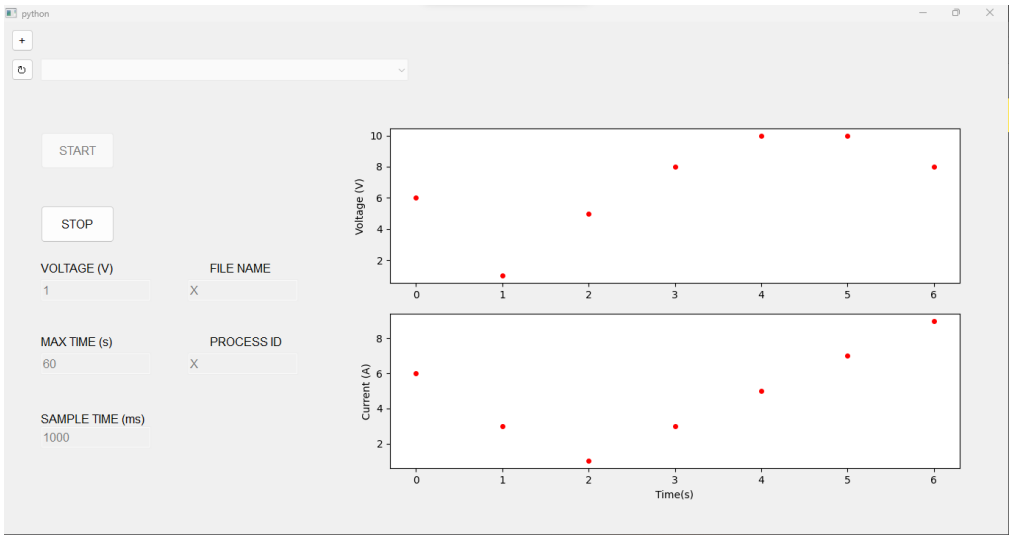

Figura 22. Exemple de procés en execució.

Quan s'assoleix el temps màxim d'execució, apareix una petita finestra per informar a l'usuari.

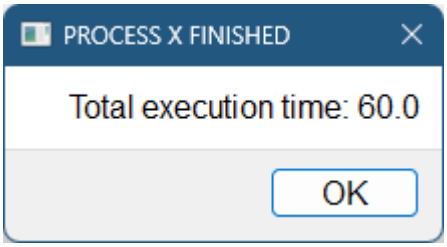

Figura 23. Alerta al usuari sobre que el procés ha finalitzat.

De la mateixa manera, també apareix una finestra quan s'atura el procés prement el botó STOP.

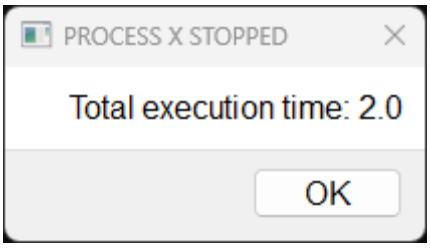

Figura 24. Alerta al usuari sobre que el procés s'ha aturat.

4.2 Com ampliar i modificar el programari

A continuació es donen indicacions sobre com es podria ampliar o realitzar modificacions al programari actual.

Primerament, és essencial tenir una sòlida base en el llenguatge de programació Python. Donat que es treballa en una interfície gràfica, també es fonamental tenir coneixements sobre PyQt6 i el seu funcionament. També cal tenir per mà biblioteques com pyserial o matplotlib. Finalment, també és molt important haver llegit i entès el contingut d'aquest treball fins aquí, on s'explica detalladament la solució adoptada.

Un cop clar tot plegat, es procedeixen a explicar algunes ampliacions o modificacions que es poden realitzar al software.

4.2.1 Establir un nou bot de Telegram

Actualment el programari incorpora un sistema de monitoratge remot mitjançant un bot de Telegram. El programa ja té assignat per defecte un bot, però es pot donar la necessitat de tenir que canviar-lo. A continuació s'explica com crear un nou bot i com incorporar-lo al software. És necessari disposar d'un compte de Telegram per fer-ho. Els passos a seguir són els següents:

1. Obrir la app de Telegram, ja sigui al ordinador o smartphone.
2. Anar a la barra de cerca i escriure "@BotFather".
3. Enviar-li "/start" per començar la creació.
4. Enviar-li la comanda "/newbot".
5. Cal contestar al bot quan pregunti sobre quin nom d'usuari se li vol posar i com s'ha d'anomenar.
6. Un cop fet, el bot ja està creat. "BotFather" enviarà un enllaç per obrir el bot i un "token" per accedir a la seva API. Aquest "token" és molt important per què és el que identifica el bot dins al programari.
7. Un cop fet tot l'anterior, cal obrir l'arxiu Python per editar-lo.
8. Cal dirigir-se a la línia de codi que hi ha just després de la incorporació de biblioteques, la qual és la següent.

```
bot=telebot.TeleBot("5996841969:AAGr28MaPupOx6ocbTjNUf9I0IROhqwxnj4”)
```

9. Per últim, simplement cal canviar el "token" actual (el que està en vermell) pel del nou bot.

   El nou bot ja estaria canviat, però també cal canviar el "chat ID". Aquest paràmetre és necessari per poder vincular el bot del programari amb el nostre compte de Telegram, ja que sinó no en podrem fer ús.

   Per obtenir-lo, hem de tornar a la barra de cerca, buscant un nou bot anomenat "Get My ID". Enviem la comanda "/start" i automàticament ens contesta amb el número. Aquest número s'ha de introduir en les funcions que permeten enviar els missatges. A continuació es mostra un exemple:

```
bot.send_message(917539506,"Process finished")
```

El "chat ID" correspondria al primer paràmetre. El segon paràmetre és el missatge que es desitja enviar.

4.2.2 Afegir o modificar elements gràfics

Si es desitja afegir o modificar algun element gràfic (botó, caixa de text, llista...) és important tenir una bona base de Python i PyQt6, tal i com s'ha comentat abans.

En quan a PyQt6, és molt recomanable fer ús del que es troba en un tutorial a la següent pàgina web: https://www.Pythonguis.com/tutorials/PyQt6-signals-slots-events/

Tots els elements que es desitgen afegir han de estar declarats dins la classe MainWindow. Un cop declarats s'han de posicionar dins la finestra, ja que en cas contrari tots es situen solapats en una mateixa posició predeterminada. També es pot definir la mida en píxels que volem que ocupi cadascun d'ells. Ambdues coses es poden fer amb el mateix mètode de la classe. A continuació es mostra un exemple on s'ajusta la posició i mida del botó START:

```
class MainWindow(QtWidgets.QMainWindow):
      ...
     self.bstart.setGeometry(50, 150, 100, 50)
```

Els dos primers paràmetres corresponen a la posició en píxels on es situa l'element, i els dos restants pertanyen a la mida que li volem donar en píxels (llargada per alçada). Es fa ús del "self" ja que el botó es troba dins la funció "__init()__" que conté la classe "MainWindow".

També es pot modificar el text que es mostra a l'element. Primer cal crear una variable que defineixi la font i mida del text que volem ajustar, tal i com es mostra a continuació.

```
font = QFont("Arial", 12)
```

Finalment, s'aplica a l'element desitjat de la següent manera:

```
self.bstart.setFont(font)
```

En el cas particular dels botons, també s'ha d'assignar una funció a executar quan són premuts, i es fa de la següent manera:

```
self.bstart.clicked.connect(self.click_bstart)
```

Com a paràmetre, es passa la capçalera de la funció a executar.

En quan a les caixes de text, es declaren de la mateixa manera que la resta d'elements, però incorporen una característica que pot ser molt útil. Quan la caixa de text està buida, es pot mostrar dins la pròpia caixa un text ombrejat on es pot indicar el paràmetre que s'ha d'introduir. A continuació es mostra un exemple sobre com s'ha fet a la caixa de text on s'introdueix el valor de tensió a aplicar:

```
self.valorV.setPlaceholderText("VOLTAGE (V)")
```

Sobre les caixes de text també és important saber com podem extraure el valor introduït en una variable. A continuació es mostra un exemple:

```
self.timemax=self.tmax.text()
```

Per guardar el valor escrit, cal fer crida del mètode de la classe "QLineEdit" que s'observa a la línia de codi anterior, el qual retorna el text emmagatzemat a la caixa de text i el desa a

la variable desitjada. D'aquesta manera obtenim el valor però en forma de cadena de caràcters. Pot passar que per determinades funcions sigui necessari tenir-ho en format numèric. En aquest cas, es pot realitzar la conversió de forma simple. Seguidament es pot veure un exemple sobre com convertir-ho a número real:

```
self.timemax=float(self.tmax.text())
```

Es faria de la mateixa manera per altres formats numèrics, però canviant "float" pel format necessari.

4.2.3 Modificar les configuracions d'inici de la font

Quan es prem el botó START, s'executa la funció "start_com()", la qual inicia la comunicació sèrie i configura la sortida de la font al nivell de voltatge desitjat per iniciar el procés. A continuació es mostren totes les comandes que s'envien per tal de configurar-ho:

```
self.ser.write(b"*RST\r")
self.ser.write(b":SENSE:FUNC:CONC OFF\r")
self.ser.write(b":SOUR:FUNC VOLT\r")
self.ser.write(b":SENS:FUNC 'CURR:DC'\r")
self.ser.write(b":SENS:CURR:PROT 0.1\r")
self.ser.write(b":SOUR:VOLT:MODE FIX\r")
self.ser.write(b":SOUR:VOLT:LEV 0\r")
self.ser.write(b":OUTP ON\r")
command = ":SOUR:VOLT:LEV "+self.valorV.text()+"\r"
self.ser.write(command.encode('utf-8'))
```

Si es desitja fer algun canvi, com per exemple, modificar el mode de funcionament, s'ha de realitzar en aquest apartat del codi, dins la funció "start_com()". Si es desitja modificar la comanda que ordenen a la font posar el voltatge escollit per l'usuari, cal modificar la penúltima línia del codi anterior. Per a més informació sobre les comandes que es poden realitzar, es recomana consultar el manual del dispositiu.

4.2.4 Editar paràmetres dels gràfics

Pot ser convenient haver de modificar certs paràmetres de la representació gràfica. A continuació es mostra com podem modificar algun d'ells.

- Afegir algun gràfic i/o modificar la seva disposició: a la solució proposada solament hi ha un element referent als gràfics, però el tenim configurat de manera que apareguin dos subgràfics. Això es configura quan creem els parells d'eixos que representen les dades de tensió i corrent en les següents línies de codi que es troben a la classe MplCanvas:

  ```
  self.axes1 = fig.add_subplot(211)
  self.axes2 = fig.add_subplot(212)
  ```

  Les dues variables anteriors van precedides del 'self' ja que formen part d'aquesta classe.

Els tres números que passem com a paràmetre corresponen a la disposició dels gràfics. El primer estableix el número de files en les que dividim l'element gràfic, en aquest cas dos, ja que tindrem un gràfic a la fila superior i un altre a la fila inferior. El següent número defineix el número de columnes, en aquest cas solament una. Finalment, el tercer número estableix la posició dels gràfics, la posició 1 o bé la posició 2.

En aquest cas, el gràfic de tensió, el qual està associat a la variable 'self.axes1', es situarà a la posició 1 (part superior), i de la mateixa manera, el gràfic de corrent estarà a la part inferior, corresponent a la posició 2.

Si es volgués incorporar un nou gràfic a sota dels ja creats, es pot fer de la següent manera:

```
self.axes1 = fig.add_subplot(311)
self.axes2 = fig.add_subplot(312)
self.axes3 = fig.add_subplot(313)
```

- Afegir títols als eixos: els títols dels eixos s'han d'afegir un cop ja s'han creat els gràfics a la finestra, és a dir, dins a la classe MainWindow. Els gràfics estan incorporats dins d'un 'canvas', el qual és l'àrea que reservem dins la finestra per incorporar els gràfics, de manera que al crear els gràfics, els eixos que representen les dades queden declarats dins del 'canvas'. El prefix 'self' indica la pertinença a la classe MainWindow. Els eixos ja estan inclosos dintre de la creació del 'canvas'. A continuació es mostra un exemple com establir títol als eixos creats:

```
self.canvas.axes1.set_title('Voltage (V)')
self.canvas.axes2.set_title('Current (A)')
self.canvas.axes2.set_xlabel('Time(s)')
```

- Modificar la mida i posició dels gràfics: com s'ha comentat abans, els gràfics estan situats dins d'un 'canvas', i el propi 'canvas' està situat dins d'un 'layout' que ens permet situar el 'canvas' dins la finestra. Per tant, si volem canviar la posició del 'canvas', i a la vegada la dels gràfics, cal fer-ho des del 'layout'. A continuació es mostra un exemple:

```
layout.setContentsMargins(400, 50, 0, 0)
```

La anterior funció no modifica la posició com a tal, sinó que estableix uns marges al 'layout' amb els límits de la finestra, de manera que serveix per situar-lo on volem. El primer paràmetre correspon al marge esquerre, el segon al dret i els dos últims amb els marges superior i inferior respectivament. Al augmentar els marges, anem ajustant l'àrea en la que volem situar el 'layout', reduint la seva superfície.

La mida dels gràfics sí es pot ajustar directament des del 'canvas'. Seguidament s'adjunta un exemple:

```
self.canvas.setFixedSize(1000, 600)
```

Com a paràmetre es passa l'amplada i l'alçada del 'canvas'. Cal tenir present que la mida del 'canvas' no pot superar l'espai marcat pels marges del 'layout'.

5. Conclusions

Tal i com queda documentat en aquest projecte, l'objectiu inicial proposat de controlar un sistema d'anodització d'alúmina nanoporosa mitjançant Raspberry Pi i programari lliure queda satisfet. S'han complit els mínims establerts al inici del projecte i un cop assolits, s'han afegit altres funcionalitats per tal de millorar el programari.

En la documentació del projecte, s'explica de manera clara com funciona el software desenvolupat. S'inclou un manual d'usuari sobre com utilitzar adequadament el programa, proporcionant una descripció detallada de les diferents funcionalitats, opcions i paràmetres del software. Aquest manual serveix com a referència per als usuaris per tal d'assegurar-se un ús eficient i correcte del programa.

Cal destacar que el projecte no queda tancat amb aquest treball, sinó que és una sòlida base per a que futurs usuaris puguin treballar sobre ell. A més de proporcionar una aplicació totalment funcional, s'han proporcionat indicacions sobre com modificar o crear noves funcionalitats utilitzant les eines ja implementades. Aquesta informació permet als usuaris ampliar i adaptar el programa segons les seves necessitats.

Així doncs, aquest projecte no només compleix els objectius inicials, sinó que també deixa oberta la possibilitat de continuar treballant en el desenvolupament i millora del sistema de control del sistema d'anodització. Amb aquesta base, altres usuaris podran construir sobre aquest treball per aprofundir en els aspectes tècnics, explorar noves característiques o abordar altres aplicacions relacionades amb el control del sistema d'anodització.

A nivell personal, durant aquest projecte he adquirit nous coneixements i habilitats. Tot i ja tenir una sòlida base de llenguatge C i C++ adquirida al llarg del grau, he hagut de aprendre a programar per primer cop amb Python i descobrir totes les eines que ofereix per poder avançar en el projecte. També he hagut d'aprendre sobre el funcionament i la comunicació de l'instrument. Gràcies al manual del dispositiu i amb el suport del tutor del treball s'ha pogut entendre perfectament.

Enfrontar-me amb problemes durant el desenvolupament del projecte era inevitable, però els vaig afrontar amb determinació. Cada obstacle es va convertir en una oportunitat per millorar les meves habilitats de resolució de problemes, un aspecte que pot ser molt positiu al futur durant meva carrera professional.

Per concloure, vull mostrar la meva satisfacció en poder ajudar a resoldre una necessitat real, en aquest cas del grup de recerca NEPHOS. No hi ha major plaer per a mi que concloure el grau aportant aquest petit gra de sorra a la universitat.

6. Annexes

6.1 Codi del programari

```
import sys
import os
from PyQt5 import QtWidgets, QtCore
from PyQt5.QtCore import QSize, Qt, QTimer
from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QLabel, QLineEdit, QVBoxLayout, QMessageBox, QHBoxLayout, QWidget, QSizePolicy, QComboBox
from PyQt5.QtGui import QFont, QGuiApplication
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.animation import FuncAnimation
from matplotlib.figure import Figure
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import pyplot
from matplotlib.animation import FuncAnimation
from matplotlib import style
import matplotlib.ticker as ticker
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as NavigationToolbar
import numpy as np
import serial
import serial.tools.list_ports
import time
import random
import telebot
import datetime

bot=telebot.TeleBot("5996841969:AAGr28MaPupOx6ocbTjNUf9I0IROhqwxnj4")

class MplCanvas(FigureCanvas):
    def __init__(self, parent=None, width=5, height=4, dpi=100):
        fig = Figure(figsize=(width, height), dpi=dpi)
        self.axes1 = fig.add_subplot(211)
        self.axes2 = fig.add_subplot(212)
        fig.set_facecolor('none')
```

```
        super(MplCanvas, self).__init__(fig)
        self.setStyleSheet("background-color:transparent;")

class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        font = QFont("Arial", 12)
        self.starting = QMessageBox()
        self.starting.setWindowTitle('STARTING PROGRAM...')
        self.starting.setFont(font)
        self.starting.show()

        self.refreshing = QMessageBox()
        self.refreshing.setWindowTitle('UPDATING LIST...')
        self.refreshing.setFont(font)

        self.canvas = MplCanvas(self, width=5, height=4, dpi=100)
        layout = QtWidgets.QVBoxLayout()
        layout.addWidget(self.canvas)
        layout.setContentsMargins(400, 50, 0, 0)
        self.canvas.setFixedSize(1000, 600)
        widget = QtWidgets.QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
        self.canvas.axes1.set_ylabel('Voltage (V)')
        self.canvas.axes2.set_ylabel('Current (A)')
        self.canvas.axes2.set_xlabel('Time(s)')

        self.bstart = QPushButton("START", self)
        self.bstart.setEnabled(True)
        self.bstart.setFont(font)
        self.bstart.setGeometry(50, 150, 100, 50)
        self.bstart.setCheckable(False)
        self.bstart.clicked.connect(self.click_bstart)

        t_sample=QLabel('SAMPLING INTERVAL TIME (ms)', widget)
        t_sample.setFont(font)
        t_sample.move(50, 520)
        t_sample.resize(300, 40)
```

```
        self.sample = QLineEdit(self)
        self.sample.setPlaceholderText("SAMPLING INTERVAL TIME (ms)")
        self.sample.setFont(font)
        self.sample.setGeometry(50, 550, 150, 30)

        self.scan()
        self.llista = QComboBox(self)
        self.llista.setEnabled(True)
        self.llista.setFont(font)
        self.llista.addItems(self.desc)
        self.llista.setGeometry(50, 50, 500, 30)
        self.llista.currentIndexChanged.connect(self.take_port)

        self.bstop = QPushButton("STOP", self)
        self.bstop.setEnabled(False)
        self.bstop.setFont(font)
        self.bstop.setGeometry(50, 250, 100, 50)
        self.bstop.setCheckable(False)
        self.bstop.clicked.connect(self.click_bstop)

        self.window = QPushButton("+", self)
        self.window.setFont(font)
        self.window.setGeometry(10, 10, 30, 30)
        self.window.clicked.connect(self.new_window)

        self.refresh = QPushButton("↻", self)
        self.refresh.setFont(font)
        self.refresh.setGeometry(10, 50, 30, 30)
        self.refresh.clicked.connect(self.new_scan)

        t_tmax=QLabel('MAX TIME (s)', widget)
        t_tmax.setFont(font)
        t_tmax.move(50, 420)

        self.tmax = QLineEdit(self)
        self.tmax.setPlaceholderText("TOTAL TIME (s)")
        self.tmax.setFont(font)
        self.tmax.setGeometry(50, 450, 150, 30)
```

```
t_id=QLabel('PROCESS ID', widget)
t_id.setFont(font)
t_id.move(280, 420)

self.id = QLineEdit(self)
self.id.setPlaceholderText("PROCESS ID")
self.id.setFont(font)
self.id.setGeometry(250, 450, 150, 30)

t_file=QLabel('FILE NAME', widget)
t_file.setFont(font)
t_file.move(280, 320)

self.file = QLineEdit(self)
self.file.setPlaceholderText("FILE NAME")
self.file.setFont(font)
self.file.setGeometry(250, 350, 150, 30)

t_volt=QLabel('VOLTAGE (V)', widget)
t_volt.setFont(font)
t_volt.move(50, 320)

self.valorV = QLineEdit(self)
self.valorV.setMaxLength(3)
self.valorV.setPlaceholderText("VOLTAGE (V)")
self.valorV.setFont(font)
self.valorV.setGeometry(50, 350, 150, 30)

self.x=[]
self.y=[]
self.z=[]

self.finished = QMessageBox()
self.finished.setFont(font)

self.stopped = QMessageBox()
self.stopped.setFont(font)
```

```
        self.error = QMessageBox()
        self.error.setWindowTitle('ERROR')
        self.error.setText('Enter a value in all the parameters')
        self.error.setFont(font)
        self.new_window = None

    def new_window(self):
        self.scan()
        self.llista.clear()
        self.llista.addItems(self.desc)
        self.new = MainWindow()
        self.new.showMaximized()
        self.new.llista.clear()
        self.new.llista.addItems(self.desc)

    def new_scan(self):
        self.refreshing.show()
        self.scan()
        self.llista.clear()
        self.llista.addItems(self.desc)
        try:
            self.new.llista.clear()
            self.new.llista.addItems(self.desc)
        except:
            pass
        self.refreshing.close()

    def scan(self):
        self.desc=[]
        self.num=[]
        self.name=''
        ports_disp = serial.tools.list_ports.comports()
        for port in ports_disp:
            try:
                ser    =    serial.Serial(port.device,    9600,    timeout=0.1,
                write_timeout=0.1)
                a = '*IDN?\n'
                ser.write(a.encode('utf-8'))
                s0 = b'0'
```

```
                name = str()
                try:
                    while s0 != b'\r':
                        s0 = ser.read()
                        name = name + s0.decode('utf-8')
                    name = name + ' (' + port.device + ')'
                    self.desc.append(name)
                    self.num.append(port.device)
                    ser.close()
                except:
                    name = port.description
                    self.desc.append(name)
                    self.num.append(port.device)
            except (serial.SerialException, serial.SerialTimeoutException):
                name = port.description
                self.desc.append(name)
                self.num.append(port.device)
        self.starting.close()


    def take_port(self):
        selected_index = self.llista.currentIndex()
        com=self.num[selected_index]
        return com

    def start_com(self):
        com=self.take_port()
        self.ser=serial.Serial(com, 9600, timeout=1)
        self.ser.write(b"*RST\r")
        self.ser.write(b":SENSE:FUNC:CONC OFF\r")
        self.ser.write(b":SOUR:FUNC VOLT\r")
        self.ser.write(b":SENS:FUNC 'CURR:DC'\r")
        self.ser.write(b":SENS:CURR:PROT 0.1\r")
        self.ser.write(b":SOUR:VOLT:MODE FIX\r")
        self.ser.write(b":SOUR:VOLT:LEV 0\r")
        self.ser.write(b":OUTP ON\r")
        command = ":SOUR:VOLT:LEV "+self.valorV.text()+"\r"
        self.ser.write(command.encode('utf-8'))
        time.sleep(1)
```

```
    def click_bstart(self):
        self.stop = 0
        self.time = 0
        self.canvas.axes1.cla()
        self.canvas.axes2.cla()
        self.i=0
        self.x=[]
        self.z=[]
        self.y=[]
               if self.sample.text() == "" or self.valorV.text() == ""  or
                self.tmax.text() == "" or self.id.text() == "":
            self.error.exec()
        else:
            self.start_com()
            self.ser.write(b":SYSTem:TIME:RESet\r")
            ruta = os.path.dirname(os.path.abspath(__file__))
            ruta = os.path.join(ruta, self.file.text() + ".txt")
            self.fit = open(ruta,  "w")
            self.fit.write("Process ID: " + self.id.text() + "\nTIME (s)
             VOLTAGE (V) CURRENT(A)\n")
            bot.send_message(917539506, str(datetime.datetime.now()) +
            "\nProcess " + self.id.text() + " started")
            self.tmax_text=float(self.tmax.text())
            self.valorV_text = int(self.valorV.text())
            self.timer = QtCore.QTimer()
            self.timer.setInterval(int(self.sample.text()))
            self.timer.timeout.connect(self.graf)
            self.timer.start()
            self.bstart.setEnabled(False)
            self.bstop.setEnabled(True)
            self.llista.setEnabled(False)
            self.id.setEnabled(False)
            self.tmax.setEnabled(False)
            self.file.setEnabled(False)
            self.valorV.setEnabled(False)
            self.sample.setEnabled(False)
            self.finished.setWindowTitle('PROCESS  '  +  self.id.text()  +  '
            FINISHED' )
```

```
            self.stopped.setWindowTitle('PROCESS  '  +  self.id.text()  +  '
            STOPPED' )

    def click_bstop(self):
        self.llista.setEnabled(True)
        self.bstart.setEnabled(True)
        self.valorV.setEnabled(True)
        self.id.setEnabled(True)
        self.tmax.setEnabled(True)
        self.sample.setEnabled(True)
        self.file.setEnabled(True)
        self.bstop.setEnabled(False)
        self.ser.write(b":OUTP OFF\r")
        command = ":SOUR:VOLT:LEV "+str(0)+"\r"
        self.ser.write(command.encode('utf-8'))
        self.ser.close()
        self.stop=1
        bot.send_message(917539506,"Process " + self.id.text() + " stopped")
        self.x=[]
        self.y=[]
        self.z=[]
        self.timer.stop()
        self.timer.deleteLater()
        time.sleep(0.5)
        self.fit.write("Process stopped")
        self.fit.close()
        self.stopped.exec()

    def graf(self):
        if self.time<=self.tmax_text:
            try:
                a,b=self.read()
                self.x.append(self.time)
                self.y.append(a)
                self.z.append(b)
                self.canvas.axes1.cla()
                self.canvas.axes1.plot(self.x, self.y, 'ro', markersize=4)
                self.canvas.axes2.cla()
                self.canvas.axes2.plot(self.x, self.z, 'ro', markersize=4)
```

```
                self.canvas.axes1.set_ylabel('Voltage (V)')
                self.canvas.axes2.set_ylabel('Current (A)')
                self.canvas.axes2.set_xlabel('Time(s)')
                self.canvas.draw()
                if self.stop == 0 :
                      self.fit.write(str(self.time) + " " + str(a) + " " +
                       str(b) + "\n")
                       bot.send_message(917539506, "Process ID: " +
                       self.id.text() + "\nExecution time: " +
                       str(self.time) +  "\nVoltage: " +str(a) + "
                       V\nCurrent: " + str(b) + " A")
                       self.finished.setText('Total  execution  time:  '  +
                       str(self.time))
                       self.stopped.setText('Total execution time: ' +
                       str(self.time))
            except (serial.SerialException, serial.SerialTimeoutException):
                self.fit.write(str(self.time) + "SERIAL ERROR\n")
                bot.send_message(917539506, "Process ID: " + self.id.text()
                + "\nExecution time: " + str(self.time) + "\nSERIAL ERROR")
        else:
            self.llista.setEnabled(True)
            self.bstart.setEnabled(True)
            self.valorV.setEnabled(True)
            self.id.setEnabled(True)
            self.tmax.setEnabled(True)
            self.sample.setEnabled(True)
            self.file.setEnabled(True)
            self.bstop.setEnabled(False)
            self.x=[]
            self.y=[]
            self.z=[]
            self.i=0
            self.timer.stop()
            self.timer.deleteLater()
            self.canvas.axes1.cla()
            self.canvas.axes2.cla()
            self.fit.write("Process finished")
            self.fit.close()
            self.ser.write(b":OUTP OFF\r")
```

```
            command = ":SOUR:VOLT:LEV "+str(0)+"\r"
            self.ser.write(command.encode('utf-8'))
            self.ser.close()
            bot.send_message(917539506,"Process  "  +  self.id.text()  +  "
            finished")
            self.finished.exec()

    def read(self):
        self.ser.write(b":MEAS?\r")
        s0 = b'0'
        meas = str()
        while s0!=b'\r':
            s0 = self.ser.read()
            meas = meas+s0.decode('utf-8')
        self.ser.flush()
        data = meas.split(",")
        volt = float(data[0])
        curr = float(data[1])
        self.time = float (data[3])
        return volt, curr

app = QtWidgets.QApplication(sys.argv)
w = MainWindow()
w.showMaximized()
app.exec()
```