

Manual de programador para el GPR de laboratorio



Universidad de los Andes

Proyecto:

Microwave Detection of Improvised Explosive Devices in Colombia (MEDICI)

Autores:

Daniel Julián González Ramírez - dj.gonzalez1203@uniandes.edu.co
Luis Eduardo Quibano Alarcón - le.quibano@uniandes.edu.co
Roberto Bustamante Miller - rbustama@uniandes.edu.co

Junio 2018

Índice

1 Pre-requisitos para el funcionamiento	2
1.1 Instalación de dependencias	2
2 Función principal del programa	3
2.1 Diagrama de flujo y tiempo	3
2.2 Variables, clases y funciones	4
2.3 Desarrollo de interfaz gráfica con QT Designer	4
3 Desarrollo de Interfaz gráfica para posicionador	5
3.1 Sección de Gráficas	5
3.2 Sección de Conexión	5
3.3 Sección configuración Posicionador y VNA	6
3.4 Sección del control de medidas	6
3.5 Sección de configuración de trayectorias	6
4 Conexión de interfaz gráfica y funciones del programa principal para el control de interfaz gráfica	7
4.1 Clase <i>Variables</i>	7
4.2 Clase <i>CloneThread</i>	7
4.2.1 Función <i>run()</i>	8
4.3 Clase <i>ExampleApp</i>	10
4.3.1 Función <i>__init__(self)</i>	10
4.3.2 Función <i>startHoming()</i>	11
4.3.3 Función <i>updateAllVarValues()</i>	11
4.3.4 Función <i>changeTypeTray()</i>	12
4.3.5 Función <i>changeConf()</i>	14
4.3.6 Función <i>drawLines()</i>	14
4.3.7 Función <i>confirmDraw()</i>	14

4.3.8 Función <i>confirmDelta()</i>	14
4.3.9 Función <i>previewSerp()</i>	14
4.3.10 Función <i>sendTraySerp()</i>	15
4.3.11 Función <i>sendTraySerpThread()</i>	15
4.3.12 Función <i>sendTrayEsp()</i>	15
4.3.13 Función <i>sendTrayEspThread()</i>	15
4.3.14 Función <i>threadConeection()</i>	15
4.3.15 Función <i>isRunning()</i>	16
4.3.16 Función <i>saveMatlab()</i>	18
4.3.17 Función <i>measures()</i>	18
4.3.18 Función <i>finished()</i>	18
4.3.19 Función <i>pause_meas()</i>	18
4.3.20 Función <i>restart_meas()</i>	18
4.3.21 Función <i>abort_meas()</i>	19
4.4 Función <i>main()</i>	19
5 Funciones para el control del VNA	20
5.1 Clase Functions_VNA	20
5.1.1 Función <i>connectVNA()</i>	20
5.1.2 Función <i>askMeasures()</i>	21
5.1.3 Función <i>updateConf()</i>	21
6 Funciones para el control de posición	22
6.1 Clase Functions_COM	22
6.1.1 Función <i>connectCOM(self)</i>	22
6.1.2 Función <i>homing(self,OBJ_S)</i>	22
6.1.3 Función <i>askPosition()</i>	24
6.1.4 Función <i>configcon(PUERTO)</i>	25
6.1.5 Función <i>gotoStartPos(OBJ_S,x,y)</i>	26

6.1.6	Función <code>tray_1(OBJ_S, x0, xf, y0, yf, dx, velocity)</code>	27
6.1.7	Función <code>pause_meas()</code>	28
6.1.8	Función <code>restart_meas()</code>	29
6.1.9	Función <code>abort_meas()</code>	29
6.1.10	Función <code>toplotCurrentPos(respuesta)</code>	30
6.1.11	Función <code>spatialDiscretization()</code>	30
7	Funciones para realizar dibujos	32
7.1	Clase <code>DrawLine</code>	32
7.1.1	Función <code>__init__(self, line)</code>	32
7.1.2	Función <code>__call__(self, event)</code>	32
7.2	Clase <code>AskInitPoint</code>	32
7.2.1	Función <code>__init__(self, line)</code>	33
7.2.2	Función <code>__call__(self, event)</code>	33
7.3	Función <code>preview_trajectory(x0, y0, xf, yf, dx)</code>	33
7.4	Función <code>createWidgetPlot(self)</code>	35
7.5	Función <code>drawPoint(self,x,y)</code>	36
8	Configuración previa y depuración de errores del controlador de posición	37
8.1	Configuración de GRBL v1.1 en la tarjeta xPro	37
8.2	Actualización del firmware GRBL	40
8.2.1	Modificación del archivo de configuración de GRBL v1.1	41
8.2.2	Instalación del firmware en la tarjeta xPro	41
8.3	Depuración del error en el final de carrera del controlador de posición	43
8.3.1	Descripción del error	43
8.3.2	Configuración inicial de la tarjeta xPro	43
8.3.3	Cambios en la configuración de la tarjeta xPro	44
8.3.4	Cambio en el montaje físico	45

1. Pre-requisitos para el funcionamiento

Este programa fue desarrollado en Python versión 3 utilizando un computador con sistema operativo Windows.

Los equipos a controlar son la tarjeta xPro que permite mover el posicionador y el analizador vectorial (VNA) Anritsu MS2026C. Si desea controlar otro equipo diferente al mencionado en este manual, por favor revise las secciones de código donde sea necesario cambiarlas.

Para el correcto funcionamiento de la herramienta GPR de laboratorio es necesario contar con los siguientes programas y dependencias:

- Python 3
- QT5 Designer
- pip → Gestor de paquetes Python.
- pyserial → Paquete para el control de equipos mediante puerto serial.
- python-vxi11 → Paquete para el control por protocolo vxi11 hacia el equipo VNA.
- pyQt5 → Paquete para el control de interfaz gráfica en QT.
- numpy → Paquete de computación para Python.
- scipy → Paquete de herramientas y algoritmos matemáticos.
- matplotlib → Paquete para el control de gráficas.

1.1. Instalación de dependencias

Al contar con el gestor de paquetes `pip` es posible instalar la mayoría de dependencias, para ello es necesario ejecutar el siguiente comando desde el terminal de Windows.

```
pip3 install pyserial pyqt5 numpy scipy matplotlib
```

Este proceso tomará un tiempo, una vez finalizado se procederá a descargar la dependencia que permite controlar el equipo VNA, para ello realice los siguientes pasos:

1. Descargue el repositorio del protocolo VXI11 en <https://github.com/python-ivi/python-vxi11>
2. Descomprima los archivos
3. Con el terminal de Windows, ingrese a la carpeta del repositorio a través del comando `cd <dir>`
4. Ejecute el comando `python setup.py install`

Con la instalación de estas dependencias ya podrá ejecutar el programa para el control del GPR de laboratorio.

2. Función principal del programa

2.1. Diagrama de flujo y tiempo

Cada una se las secciones de código del programa generan intervalos de tiempo donde interactúan los diferentes componentes del sistema; en la Figura 1 se puede ver los diferentes estados por los que pasa el programa y dependiendo el estado el tiempo promedio que gasta dicho proceso.

El proceso que más tiempo tarda es el de calibración del sistema de coordenadas, ya que el posicionador debe ajustar de forma correcta los valores de inicio en el eje X y Y de modo tal que todas las pruebas sean repetibles.

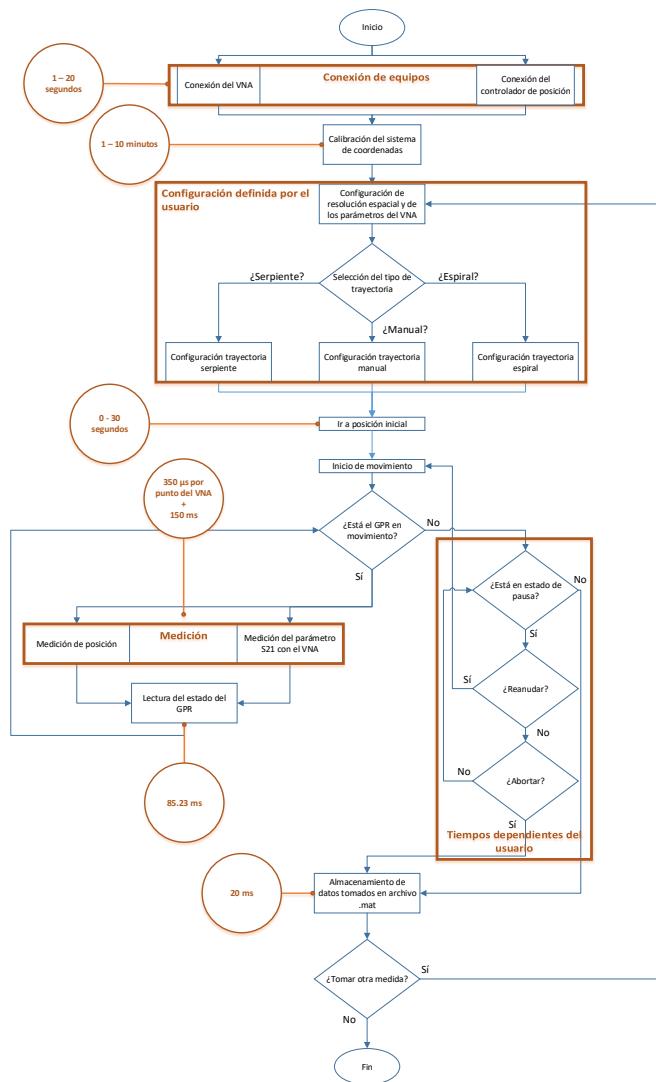


Figura 1: Diagrama de flujo y de tiempo del control de GPR

2.2. Variables, clases y funciones

2.3. Desarrollo de interfaz gráfica con QT Designer

Para trabajar con la interfaz gráfica es necesario utilizar la herramienta QT Designer, descargue esta herramienta desde la página oficial la versión open source [7]. Cuando realice la instalación asegúrese de seleccionar la versión 5.11.1 o superior, el proceso de instalación tomará tiempo.

Una vez instalado QT puede utilizar la herramienta Designer, normalmente QT es utilizado para trabajar con C++, pero como el desarrollo del control del GPR fue hecho en Python es necesario tener instalado la dependencia PyQt5.

Para utilizar la herramienta designer, diríjase a la ruta **C:\Qt\5.11.1\msvc2017_64\bin** desde el explorador de archivos y allí encontrará el ejecutable **designer.exe**

Cuando abra la aplicación puede agregar tantos elementos como desee, lo importante es asignar un valor de identificación (Ver 2, esto permitirá controlar las propiedades de cada uno de los elementos desde Python).

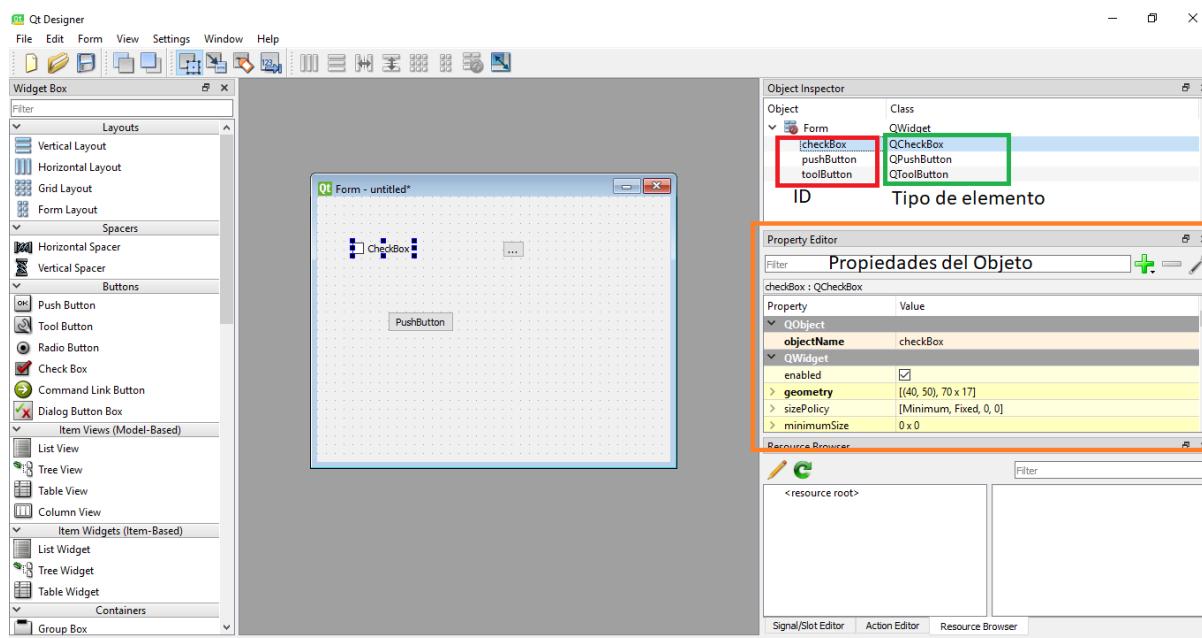


Figura 2: Estructura Básica en QT

Una vez desarrollada la interfaz gráfica debe ser exportada a Python para realizar la programación de las acciones que va a tener cada uno de los elementos; para ello se utiliza la herramienta pyuic5 que convierte el archivo .ui de QT Designer a .py mediante el siguiente código desde el terminal de Windows.

```
pyuic5 archivo.ui -o archivo.py
```

3. Desarrollo de Interfaz gráfica para posicionador

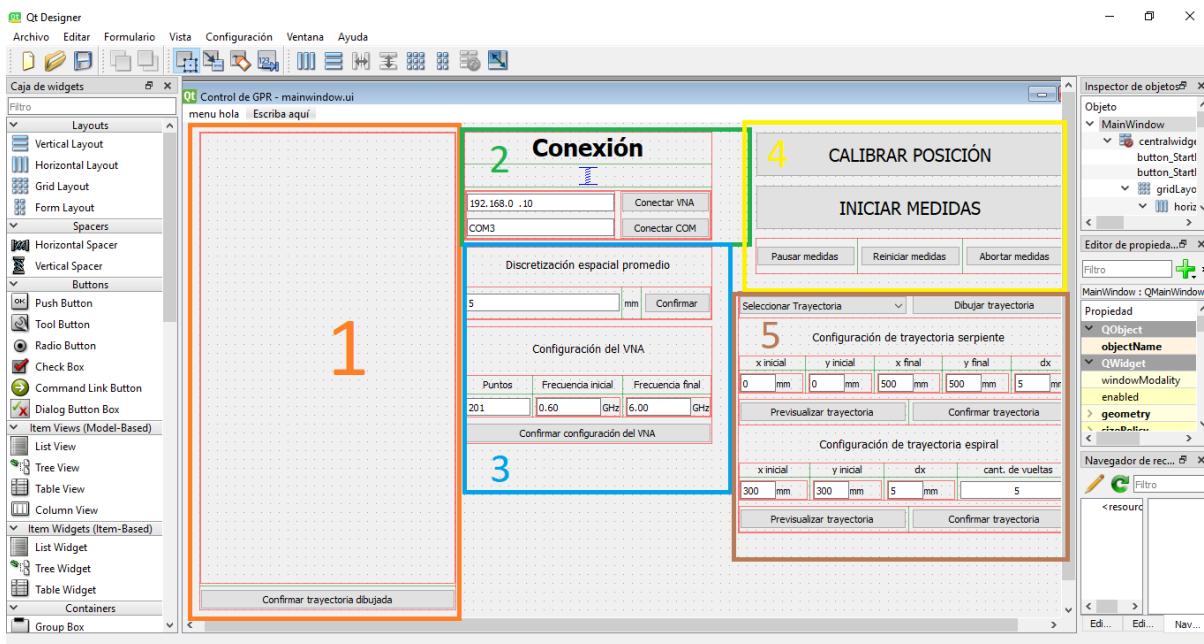


Figura 3: Interfaz Gráfica para el control del GPR de laboratorio

Para el control del sistema de GPR se diseñó la interfaz gráfica en Python mediante la herramienta QT, esta herramienta permite agregar objetos tal como entradas de texto, botones, gráficas, etc. La interfaz se encuentra dividida en cinco partes, ellas son:

3.1. Sección de Gráficas

Estas sección es utilizada para mostrar las gráficas de los diferentes trayectos que se quieran realizar, esta sección utiliza “Layouts” permitiendo agregar widgets; el widget es un elemento en QT Designer para utilizar la extensión de la librería **matplotlib** quien se encarga de realizar gráficas.

3.2. Sección de Conexión

En esta sección se encuentran los botones y entradas de texto para seleccionar la conexión del posicionador y del VNA. De acuerdo a la infraestructura actual, el posicionador se conecta mediante puerto serial y el VNA a través del puerto Ethernet.

3.3. Sección configuración Posicionador y VNA

En esta sección se encuentra la configuración de la discretización, esto se refiere al valor promedio en que el posicionador tomará una medida; por defecto el valor es de 5mm.

Por otro lado se encuentra la configuración del VNA, entre las opciones que el usuario puede modificar son:

- Cantidad de puntos que el VNA va a tomar: estos valores son de acuerdo a la marca del VNA (Anritsu), las opciones van desde 2 hasta 4001.
- Frecuencia inicial: Este valor se encuentra por defecto en 600MHz
- Frecuencia final: Es el valor de parada del VNA, por defecto se encuentra en 6GHz

3.4. Sección del control de medidas

Esta sección es utilizada para realizar el control de las medidas del sistema GPR donde se encuentran los siguientes elementos: botón para ejecutar la calibración de coordenadas del sistema GPR y el botón para iniciar las medidas. Con el botón de realizar medidas hace que el posicionador se mueva de acuerdo al trayecto seleccionado y por medio de una tarea paralela el VNA realiza medidas del parámetro S21.

3.5. Sección de configuración de trayectorias

El software del GPR permite realizar diferentes trayectos, entre ellos se encuentran: tipo serpiente; manual; y espiral.

Dependiendo del tipo de trayecto que seleccione el usuario se habilitarán las opciones necesarias para la configuración. Así mismo el usuario podrá visualizar como será el trayecto antes de iniciar cualquier medida, una vez confirmado el trayecto, se puede iniciar la medición.

4. Conexión de interfaz gráfica y funciones del programa principal para el control de interfaz gráfica

4.1. Clase Variables

Dentro del programa implementado existe una clase específica para el almacenamiento de variables globales, estas variables pueden ser consultadas por cualquier función o clase del programa

```
class Variables:
    def __init__(self):
        self.puntos = None #Puntos del VNA
        self.offset = None #Offset de las medidas
        self.measures = None #Cantidad de medidas que pueden
        self.frec_ini = None #Frecuencia Inicial configurada en el VNA
        self.frec_fin = None #Frecuencia Final configurada en el VNA
        self.descripcion = None #Descripción de la prueba
        self.instr = None #Variable que guarda el objeto donde que controla el VNA
        self.Ctrl_Pos = None #Variable que guarda el objeto donde se controla el Posicionador
        self.DatosVNAcomplex = None #Matriz donde se guardan los valores S21 real e imaginario
        self.position = None #Vector donde se guarda los valores de posicion del posicionador
        self.threatTime = None #Vector donde se guarda el tiempo que dura el thread de leer vna y el posicionador
        self.elapsed = None #Vector donde se guarda el tiempo que dura la medida del VNA (se utiliza para estimar el offset)
        self.threads = None #Número de threads

        """VARIABLES UTILIZADAS PARA EL TRAYECTO SERPIENTE"""
        self.dom_x0 = None
        self.dom_xf = None
        self.dom_y0 = None
        self.dom_yf = None
        self.dom_dx = None

        self.static_canvas=None #Variable utilizada para la grafica
        self.checkAllConnected=None #Variable para validar si se conecto el VNA y el posicionador
        self.trajectory = None #Variable donde se guarda el tipo de trayectoria
        self.discretization = None #Variable donde se guarda el valor de discretización
        self.velocity = None #Variable donde se guarda la velocidad a partir de la discretización

        """VARIABLES UTILIZADAS PARA EL TRAYECTO ESPIRAL """
        self.dom_x0_esp = None
        self.dom_y0_esp = None
        self.dom_dx_esp = None
        self.dom_qtraj_esp = None

        self.respuesta_controlador = None #Utilizado para preguntar al posicionador el estado al iniciar las mediciones

        """VARIABLES UTILIZADAS PARA GUARDAR LA POSICION DEL CURSOS EN LAS MEDIDAS MANUALES"""
        self.setXinit = None
        self.setYinit = None

        self.stateMeas=None #Estado de la medición (run, pause, abort)
        self.stateWhile=None

        """DEFINO LAS DIMENSIONES DE LA GRAFICA"""
        self.plotX=None
        self.plotY=None

        """ DEFINO LAS MARGENENES PARA COLOCAR EL RECUADRO ROJO"""
        self.plotMargin=None
        self.CurrentX=None
        self.CurrentY=None

        """ VARIABLE QUE VA A CONTENER LAS GRAFICAS"""
        self._static_ax=None

        """ VARIABLE QUE VA A SER GUARDADO EN ASK AGAIN"""
        self.askAgain=None

        """VARIABLE PARA GUARDAR EL TREAD DEL HOMING"""
        self.homingTread=None
        self.que=None
```

4.2. Clase *CloneThread*

```
class CloneThread(QThread):
    # CLASE UTILIZADA PARA REALIZAR LAS MEDICIONES DEL VNA Y DEL POSICIONADOR MEDIANTE TAREAS EN PARALELO
    signal = pyqtSignal('PyQt_PyObject') #SE CREA LA VARIABLE QUE ME INDICA CUANDO ESTE TREAD HA TERMINADO

    def __init__(self):
        QThread.__init__(self)
```

4.2.1. Función *run()*

```

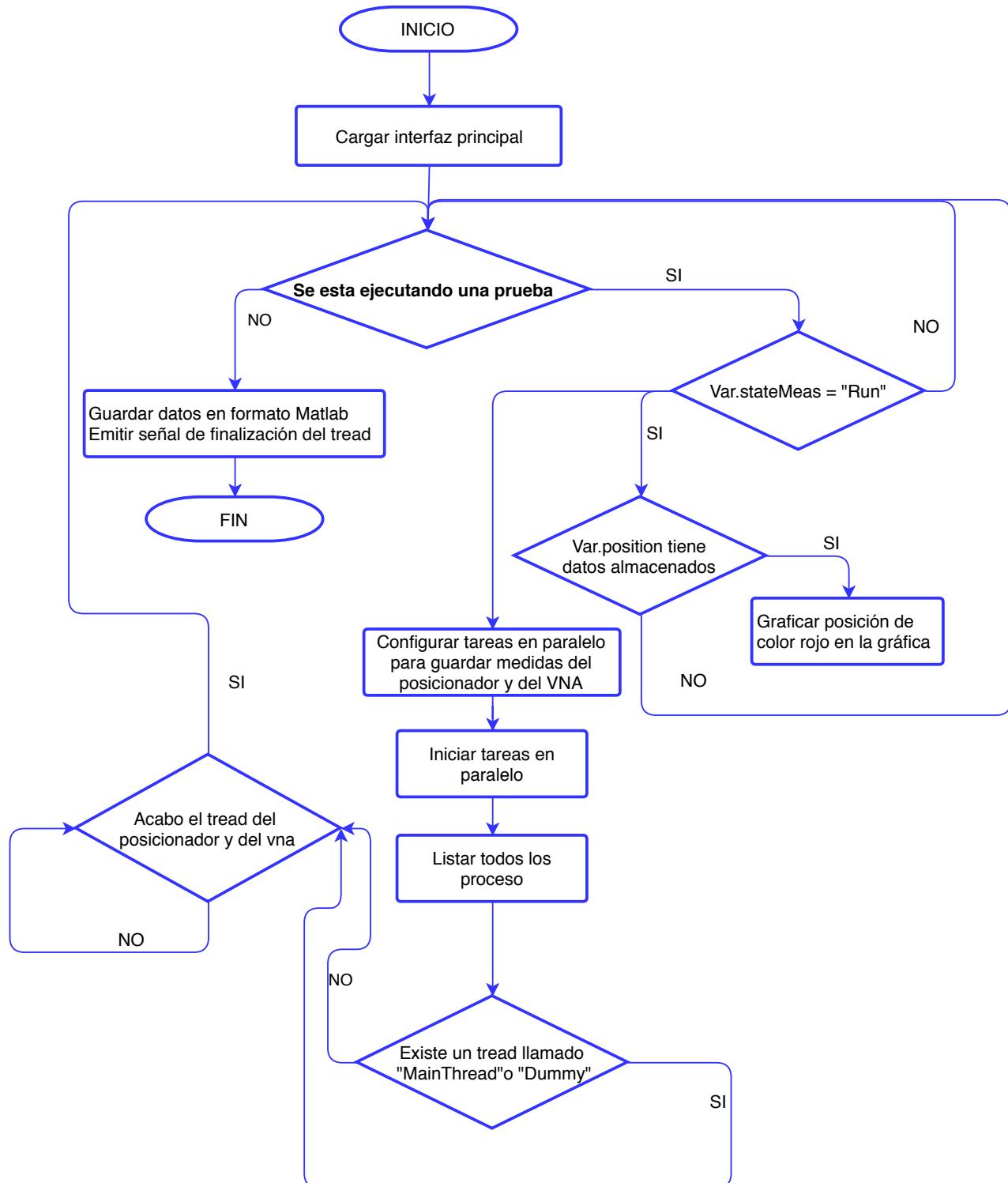
def run(self):
    #SE RE-DEFINE LA FUNCION RUN DE QThread
    self.apps = ExampleApp #LLAMO LA CLASE ExampleApp PARA CONTROLAR LOS OBJETOS Y LLAMAR LAS FUNCIONES DE ESTA CLASE
    # INICIO EL HOMING
    logging.debug("ESTA CORRIENDO EN OTRO TRHEAD")

    #CREO UN WHILE PARA SABER SI EL POSICIONADOR SE ENCUENTRA EN MOVIMIENTO
    i=0 #VARIABLE PARA SABER CUANTAS ITERACIONES SE HAN REALIZADO.
    while(self.apps.isRunning()): #LLAMO LA FUNCION isRunning DE LA CLASE PRINCIPAL Y ESTA RETORNA True/False
        if (Var.stateMeas == "Run"): #SI EL ESTADO DE LA MEDICION ESTA CORRIENDO (RUN)
            if (Var.position.shape[0]>1):
                # SI EL VECTOR DE POSICION ES >1 ES PORQUE YA TIENE DATOS ALMACENADOS
                # CREO LAS VARIABLES DE LOS VALORES ACUALES DE X y Y CONSULTANDO LA FUNCION COMFunction.toplotCurrentPos
                # DONDE SE LE ENVIA EL PARAMETRO Var.position[-1] EL CUAL CORRESPONDE AL ULTIMO DATO GUARDADO EN EL
                VETOR DE POSICION
                # ESTOS DATOS SE MANDAN A LA FUNCION drawPoint() QUE PERMITE GRAFICAR EN LA VENTANA
                # EL TRAYECTO DE COLOR ROJO CORRESPONDIENTE AL POSICIONADOR EN TIEMPO REAL, ESTO HACE QUE MIENTRAS
                SE REALIZA LA MEDIDA,
                # EL COLOR AZUL DEL TRAYECTO DIBUJADO SE IRA LLENANDO DE COLOR ROJO
                # ESTO LE INDICA AL USUARIO COMO VA EL PROGRESO DE LAS MEDICIONES
                currentx,currenty= COMFunction.toplotCurrentPos(Var.position[-1])
                #print("currentx",currentx, " currenty ",currenty)
                drawPoint(self,currentx,currenty)

            logging.debug(i) #REALIZO EL DEBUG DE LA ITERACION
            #CREO LA PRIMER TAREA QUE REALIZA LA CONSULTA DE POSICION A LA CLASE COMFunction.
            task1 = threading.Thread(target= COMFunction.askPosition, name='ask_position',
                                     args=())
            #MANDO LA TAREA COMO UN OBJETO EN COLA
            Var.threads.append(task1) # SE ENVIA EL PROCESO A LA LISTA DE PROCESOS
            #CREO LA SEGUNDA TAREA QUE REALIZA LA CONSULTA DE MEDICION S21 A LA CLASE VNAFunction.
            task2 = threading.Thread(target= VNAFunction.askMeasures, name='ask_vna', args=()) # INICIO EL PROCESO
            #MANDO LA SEGUNDA TAREA COMO UN OBJETO EN COLA
            Var.threads.append(task2)
            # INICIO LOS PROCESOS
            task1.start()
            task2.start()
            thtime = time.time() # MARCO EL TIEMPO BASE PARA SABER CUANDO DURA EL TREAD PARA REALIZAR LA MEDICION DEL
            POSICIONADOR Y DEL VNA

            for t in threading.enumerate(): # ENLISTO LOS PROCESOS
                #SI HAY UN HILO LLAMADO MainThread O Dummy, CONTINUAR CON LAS MEDICIONES
                if (t.getName() == "MainThread" ) or ( "Dummy" in t.getName() ): # SI SE ESTA EN EL PROCESO PRINCIPAL CONTINUAR
                    continue
                # logging.debug('joining %s', t.getName())
                #DE LO CONTRARIO ESPERO A QUE TODOS LOS PROCESOS FINALICEN
                #SI UNO TERMINA PRIMERO, ESPERA A QUE EL SEGUNDO ACABE Y VICEVERSA
                t.join()
            # CUANDO TODOS LOS PROCESOS HAYAN TERMINADO SE MUESTRA UN PRINT
            logging.debug("THREADS COMPLETADOS")
            Var.threatTime.extend([time.time() -thtime]) #CONCATENO EN EL VECTOR DEL TIEMPO DEL TREAD , EL TIEMPO REALIZADO
            i+=1 #INCREMENTO EN 1 LA ITERACION

        #print(Var.DatosVNAComplex)
        #print("")
        #print(Var.elapsed)
        #print("")
        #print(Var.position)
        #print("")
        #print(Var.threatTime)
        # GUARDO LOS DATOS EN UN ARCHIVO DE MATLAB
        self.apps.saveMatlab()
        #INDICO QUE TODAS LAS MEDICIONES SE HAN REALIZADO
        logging.debug("Medidas Terminadas")
        self.signal.emit(0) #EMITO LA SENAL PARA INDICAR QUE YA SE ACABO EL TREAD
    
```

Figura 4: Diagrama de Flujo para la función *run()* de la clase *CloneThread*

4.3. Clase *ExampleApp*

Esta es la clase principal de funcionamiento del software para el control del GPR, dentro de esta clase se configura cada uno de los elementos creados en QT Designer. A través de esta clase se ejecutan las diferentes funciones del control del posicionador, control del VNA y de dibujo.

4.3.1. Función `__init__()`

Esta es la función de inicio del programa, en esta función se definen como va a iniciar cada uno de los elementos de la interfaz tal como indicar qué botones iniciaran deshabilitados, que texto irá en las entradas de texto, etc.

```

def __init__(self):
    super(self.__class__, self).__init__()

    self.setupUi(self) # ESTE ES DEFINIDO POR DEFECTO EN EL ARCHIVO IMPORTADO

    ##### EN ESTA SECCION SE AGREGAN LAS SENALES Y SLOTS PARA CONTORLAR EL GUI #####
    #####
    ##### Configuracion de los botones al iniciar la ventana
    self.button_StartMeas.setDisabled(1)
    self.layout_conf_trajectory.setEnabled(1)
    self.layout_delta.setEnabled(1)
    self.layout_vna.setEnabled(1)
    self.button_draw_trajectory.setEnabled(False)
    self.button_confirm_draw.setEnabled(False)
    self.button_restart_measurements.setEnabled(False)
    self.button_abort_measurements.setEnabled(False)
    self.button_pause_measurements.setEnabled(False)
    self.combo_sel_trayectoria.setEnabled(False)
    self.button_StartHoming.setEnabled(False)
    self.button_preview_serp.setEnabled(False)
    self.button_preview_esp.setEnabled(False)
    self.button_confirm_serp.setEnabled(False)
    self.button_confirm_esp.setEnabled(False)
    self.button_confirm_delta.setEnabled(False)
    self.button_confirm_vna.setEnabled(False)
    self.button_confirm_vna.setEnabled(False)
    self.input_delta.setEnabled(False)
    self.input_x_ini.setEnabled(False)
    self.input_y_ini.setEnabled(False)
    self.input_x_fin.setEnabled(False)
    self.input_y_fin.setEnabled(False)
    self.input_dx.setEnabled(False)
    self.input_points.setEnabled(False)
    self.input_freq_ini.setEnabled(False)
    self.input_freq_fin.setEnabled(False)
    self.input_x_ini_esp.setEnabled(False)
    self.input_y_ini_esp.setEnabled(False)
    self.input_dx_esp.setEnabled(False)
    self.input_cant_vuel_esp.setEnabled(False)

    #CONFIGURAR LOS INPUTS CON LOS VALORES ASIGNADOS A LA CLASE VARIABLES
    self.input_x_ini.setText(str(Var.dom_x0))
    self.input_x_fin.setText(str(Var.dom_xf))
    self.input_y_ini.setText(str(Var.dom_y0))
    self.input_y_fin.setText(str(Var.dom_yf))
    self.input_dx.setText(str(Var.dom_dx))
    self.input_points.setText(str(Var.puntos))
    self.input_freq_ini.setText(str(Var.freq_ini))
    self.input_freq_fin.setText(str(Var.freq_fin))
    self.input_delta.setText(str(Var.discretization))
    self.input_x_ini_esp.setText(str(Var.dom_x0_esp))
    self.input_y_ini_esp.setText(str(Var.dom_y0_esp))
    self.input_dx_esp.setText(str(Var.dom_dx_esp))
    self.input_cant_vuel_esp.setText(str(int(Var.dom_qtraj_esp/2)))

    #Cuando se oprime el boton de Homming
    self.button_StartHoming.clicked.connect(self.startHoming)

    # Cuando se oprime el boton conectar COM se ejecuta la funcion para conectar el puerto COM
    self.button_COM.clicked.connect(lambda: self.threadConeection("COM"))

    # Cuando se oprime el boton conectar VNA se ejecuta la funcion connectVNA
    self.button_ConnectVNA.clicked.connect(lambda: self.threadConeection("VNA"))

    # Cuando se oprime el boton de iniciar medidas, se ejecuta la funcion measure()
    self.button_StartMeas.clicked.connect(self.measures)

    # Cuando se hace un cable a las opciones de tipo de trayectoria
    self.combo_sel_trayectoria.activated.connect(self.changeTypeTray)

    # Habilitar la opcion de dibujar

```

```

self.button_draw_trajectory.clicked.connect(self.drawLines)

# Indicar que el dibujo termino
self.button_confirm_draw.clicked.connect(self.confirmDraw)

# Confirmar la discretizacion espacial
self.button_confirm_delta.clicked.connect(self.confirmDelta)

# Previsualizar la trayectoria serpiente
self.button_preview_serp.clicked.connect(self.previewSerp)
# Correr funcion de trayectoria serpiente
self.button_confirm_serp.clicked.connect(self.sendTraySerp)

# Correr funcion de trayectoria espiral
self.button_confirm_esp.clicked.connect(self.sendTrayEsp)

# Cuando se oprime el boton call , se llama la funcion para conocer la posicion de los dibujos
#self.call.clicked.connect(self.callLine)

#ACCIONES CUANDO SE OPRIME EL BOTON DE PAUSA, REINICIAR Y ABORTAR
self.button_pause_measurements.clicked.connect(COMFunction.pause_meas)
self.button_restart_measurements.clicked.connect(COMFunction.restart_meas)
self.button_abort_measurements.clicked.connect(COMFunction.abort_meas)

self.button_confirm_vna.clicked.connect(self.changeConf)

#####
##### SE CONFIGURA EL PROCESO PARA TRABAJAR TAREAS EN PARALELO
#####

#SE ASOCIA LA CLASE CloneThread() CON LA VARIABLE git_thread
self.git_thread = CloneThread()

#Cuando se genera la senal es porque el hilo ha terminado, se ejecuta la funcion finished
self.git_thread.signal.connect(self.finished)

#####
##### CONFIGURACION PARA REALIZAR LAS GRAFICAS
#####

self.layout_grafica.setEnabled(0)
#FUNCION QUE CREA LA GRAFICA DE MATPLOTLIB COMO WIDGET EN QT
createWidgetPlot(self)

```

4.3.2. Función *startHoming()*

```

def startHoming(self):
    #SE DESHABILITA EL BOTON DE HOMMING
    self.button_StartHoming.setEnabled(False)
    #SE CAMBIA EL TEXTO DEL BOTON
    self.button_StartHoming.setText("Calibrando")
    #SE UTILIZA LA VARIABLE que PARA RECIBIR EL DATO QUE RESPONDE LA FUNCION DE HOMING EN COM_FUNCTIONS
    Var.que = Queue()

    #SE CONFIGURA EL HILO (TREAD) INDICANDO LA FUNCION A EJECUAR, EL NOMBRE QUE TENDRA DEL DEBUG Y LOS ARGUMENTOS
    # A PASAR
    # SE LE PASA SELF PARA INDICARLE QUE PUEDE CONTROLAR LA INTERFAZ GRAFUCA, Y EL OBJETO Var.Ctrl_Pos QUE SE CREA
    # UNA VEZ CONECTADO
    #EL POSICIONADOR
    Var.homingTread = threading.Thread(target=COMFunction.homing, name='startHoming',
                                         args=(self,Var.Ctrl_Pos)) # CREO EL PROCESO PARA EL HOLA

    # INICIO LOS PROCESOS
    Var.homingTread.daemon = True
    Var.homingTread.start() #SE INICIA EL TREAD PARA HACER HOMING

```

4.3.3. Función *updateAllVarValues()*

```

def updateAllVarValues(self):
    #FUNCION QUE OBTIENE TODOS LOS VALORES DE LOS INPUTS DE LA INTERFAZ Y LOS GUARDA EN LAS VARIABLES DE LA CLASE
    #VARIABLES
    Var.points = int(self.input_points.text())
    Var.frec_ini = float(self.input_freq_ini.text())
    Var.frec_fin = float(self.input_freq_fin.text())
    Var.dom_x0 = int(self.input_x_ini.text())
    Var.dom_xf = int(self.input_x_fin.text())
    Var.confirmDelta = Var.dom_y0 = int(self.input_y_ini.text())
    Var.dom_yf = int(self.input_y_fin.text())
    Var.dom_dx = int(self.input_dx.text())
    Var.dom_x0_esp = int(self.input_x_ini_esp.text())
    Var.dom_y0_esp = int(self.input_x_ini_esp.text())
    Var.dom_dx_esp = int(self.input_dx_esp.text())
    Var.dom_qtraj_esp = int(self.input_cant_vuel_esp.text())*2
    Var.discretization = int(self.input_delta.text())

```

4.3.4. Función *changeTypeTray()*

```

def changeTypeTray(self):
    #CUANDO SE REALIZA UN CAMBIO EN EL COMBO BOX
    print("SE CAMBIO EL COMBO BOX")
    #DESHABILITO EL BOTON DE INICIAR MEDIDAS
    self.button_StartMeas.setEnabled(False)

    #CONDICIONAL AL TIPO DE SELECCION
    if self.combo_sel_trayectoria.currentText() == "<->Trayectoria Manual":

        #SE BORRA EN LA PANTALLA LA TRAZA ANTERIOR Y SE GENERA UNA NUEVA
        self.layout_grafica.removeWidget(self.static_canvas)
        createWigetPlot(self)

        #SE DESHABILITAN LOS BOTONES DE INICIAR MEDIDAS, LOS BOTONES E INPUTS ASOCIADOS A LA TRAYECTORIA AUTOMATICA
        # Y A LA TRAYECTORIA EN ESPIRAL
        self.button_StartMeas.setEnabled(False)
        self.button_confirm_serp.setEnabled(False)
        self.button_preview_serp.setEnabled(False)
        self.button_confirm_serp.setEnabled(False)
        self.input_x_ini.setEnabled(False)
        self.input_y_ini.setEnabled(False)
        self.input_x_fin.setEnabled(False)
        self.input_y_fin.setEnabled(False)
        self.input_dx.setEnabled(False)
        self.button_preview_esp.setEnabled(False)
        self.button_confirm_esp.setEnabled(False)
        self.input_x_ini_esp.setEnabled(False)
        self.input_y_ini_esp.setEnabled(False)
        self.input_dx_esp.setEnabled(False)
        self.input_cant_vuel_esp.setEnabled(False)

        self.button_draw_trajectory.setEnabled(True)
        self.button_confirm_draw.setEnabled(True)

        self.combo_sel_trayectoria.setEnabled(False)
        self.button_pause_measurements.setEnabled(False)
        self.button_abort_measurements.setEnabled(False)
        self.button_restart_measurements.setEnabled(False)

        print("SE SELECCIONO TRAYECTORIA MANUAL")

        #INFORMO CON UNA VENTANA EMERGENTE QUE SELECCIONE EL PUNTO INICIAL
        buttonReply = QtWidgets.QMessageBox.information(self,
            'Trayectoria Manual',
            'Con el cursor oprima en la grafica la posicion inicial',
            QtWidgets.QMessageBox.Ok,
            QtWidgets.QMessageBox.Ok)

        self.layout_grafica.setEnabled(1) #HABILITO LA PANTALLA PARA INDICAR CON EL CURSOS
        #EJECUTO LA CLASE AskInitPoint CON LOS PARAMETROS DE LINEA PARA SELECCIONAR CON EL CURSOS EL PUNTO INICIAL
        line, = Var.static_ax.plot([0], [0], color='blue', lw=2) # empty line
        self.linebuilder = AskInitPoint(line)
        self.button_draw_trajectory.setEnabled(1)

    elif self.combo_sel_trayectoria.currentText() == "<->Trayectoria Serpiente":
        print("SE SELECCIONO TRAYECTORIA SERPIENTE")
        self.layout_grafica.removeWidget(self.static_canvas)
        createWigetPlot(self)
        self.button_StartMeas.setEnabled(False)
        self.button_pause_measurements.setEnabled(False)
        self.button_abort_measurements.setEnabled(False)
        self.button_restart_measurements.setEnabled(False)
        self.button_draw_trajectory.setEnabled(False)
        self.button_confirm_draw.setEnabled(False)
        self.button_preview_esp.setEnabled(False)
        self.button_confirm_esp.setEnabled(False)
        self.input_x_ini_esp.setEnabled(False)
        self.input_y_ini_esp.setEnabled(False)
        self.input_dx_esp.setEnabled(False)
        self.input_cant_vuel_esp.setEnabled(False)

        #HABILITO LOS BOTONES ESPECIFICOS PARA LA TRAYECTORIA TIPO SERPIENTE
        self.button_confirm_serp.setEnabled(True)
        self.button_preview_serp.setEnabled(True)
        self.button_confirm_serp.setEnabled(True)
        self.input_x_ini.setEnabled(True)
        self.input_y_ini.setEnabled(True)
        self.input_x_fin.setEnabled(True)
        self.input_y_fin.setEnabled(True)
        self.input_dx.setEnabled(True)

    elif self.combo_sel_trayectoria.currentText() == "<->Trayectoria Espiral":
        self.layout_grafica.removeWidget(self.static_canvas)
        createWigetPlot(self)

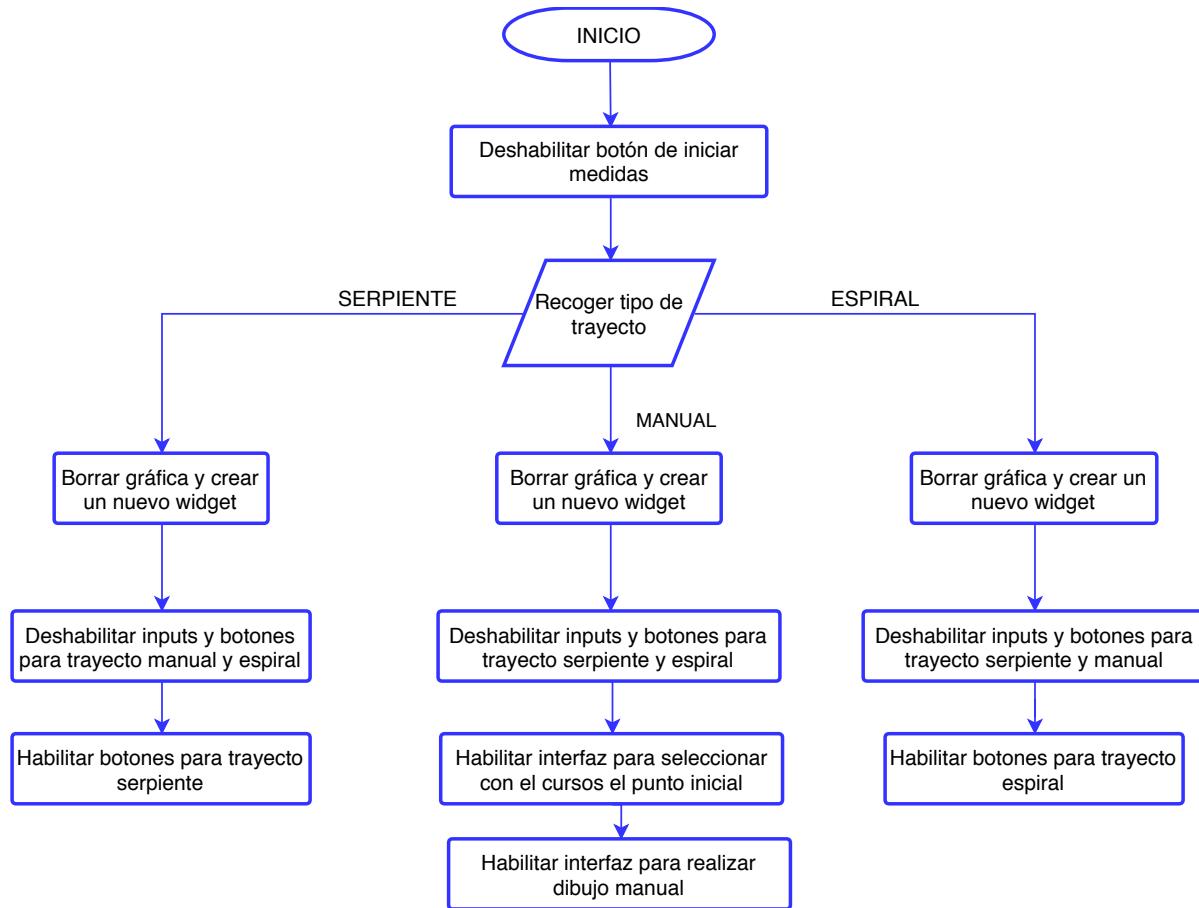
        self.button_StartMeas.setEnabled(False)
        self.button_pause_measurements.setEnabled(False)
        self.button_abort_measurements.setEnabled(False)
        self.button_restart_measurements.setEnabled(False)
        self.button_draw_trajectory.setEnabled(False)
        self.button_confirm_serp.setEnabled(False)
        self.button_preview_serp.setEnabled(False)
        self.button_confirm_serp.setEnabled(False)
        self.input_x_ini.setEnabled(False)

```

```

self.input_y_ini.setEnabled(False)
self.input_x_fin.setEnabled(False)
self.input_y_fin.setEnabled(False)
self.input_dx.setEnabled(False)
print("SE SELECCIONO' TRAYECTORIA ESPIRAL")
#HABILITO LOS BOTONES ESPECIFICOS PARA LA TRAYECTORIA TIPO ESPIRAL
self.button_preview_esp.setEnabled(True)
self.button_confirm_esp.setEnabled(True)
self.input_x_ini_esp.setEnabled(True)
self.input_y_ini_esp.setEnabled(True)
self.input_dx_esp.setEnabled(True)
self.input_cant_vuel_esp.setEnabled(True)

```

Figura 5: Diagrama de Flujo para la función *changeTypeTray()*

4.3.5. Función *changeConf()*

```
def changeConf(self):
    #FUNCION PARA ENVIAR PARAMETROS DE CONFIGURACION AL VNA
    self.updateAllVarValues()
    VNADefunction.updateConf()
```

4.3.6. Función *drawLines()*

```
def drawLines(self):
    #DESHABILITAR EL COMBO BOX Y HABILITAR LA PANTALLA PARA DIBUJAR LAS LINEAS Y EL BOTON PARA CONFIRMAR LA
    #TRAYECTORIA MANUAL
    self.combo_sel_trayectoria.setEnabled(False)
    self.button_draw_trajectory.setEnabled(False)
    self.button_confirm_draw.setEnabled(True)
    #ENVIAR UNA VENTANA EMERGENTE PARA DECIR QUE SE PUEDE EMPEZAR A DIBUJAR
    buttonReply = QtWidgets.QMessageBox.information(self,
                                                    "Dibujar Trayectoria",
                                                    "Con el cursor diuje la trayectoria",
                                                    QtWidgets.QMessageBox.Ok,
                                                    QtWidgets.QMessageBox.Ok)

    #SE EJECUTA LA FUNCION DrawLine CON LOS PARAMETROS DE LA POSICION INCIAL PARA COMENZAR A DIBUJAR
    line, = Var.static_ax.plot([Var.setXinit], [Var.setYinit], color='blue', lw=2) # empty line
    self.linebuilder2 = DrawLine(line)
```

4.3.7. Función *confirmDraw()*

```
def confirmDraw(self):
    #CUANDO SE OPRIME EL BOTON PARA CONFIRMAR EL DIBUJO MANUAL SE DESHABILITA LA PANTALLA PARA SEGUIR DIBUJANDO
    self.combo_sel_trayectoria.setEnabled(True)
    self.linebuilder2.line.figure.canvas.mpl_disconnect(self.linebuilder2.cid)
    self.button_confirm_draw.setEnabled(False)
    self.button_StartMeas.setEnabled(True)
    self.button_StartMeas.setText("Realizar Medidas")

    #SE RECIBEN TODOS LOS PUNTOS QUE DIBUJO EL USUARIO PARA CONVERTIRLOS EN CODIGO G PARA QUE EL POSICIONADOR
    #PUEDA MOVERSE
    # DE ACUERDO A LA TRAYECTORIA DEFINIDA
    for i in range(0, len(self.linebuilder2.xs)):
        # Convertir coordenadas ingresadas de centimetros a milimetros
        coord_x = round(10 * self.linebuilder2.xs[i], 2)
        coord_y = round(10 * self.linebuilder2.ys[i], 2)
        linetosend = "G1 X%d Y%d F%d\n" % (coord_x, coord_y, Var.velocity)
        Var.Ctrl_Pos.write(linetosend.encode('utf-8'))
    Var.Ctrl_Pos.write("!\n".encode('utf-8')) #SE ENVIA EL COMANDO PAUSA AL POSICIONADOR
    Var.trajecoty = "manual" #SE LE INDICA QUE LA TRAYECTORIA ES MANUAL
    Var.startMeas = "Pause" #SE CONFIGURA EL ESTADO DE LA MEDICION COMO EN PAUSA

    self.button_StartMeas.setEnabled(True) #SE HABILITA EL BOTON PARA INICIAR MEDIDAS
    self.linebuilder2.line.figure.canvas.mpl_disconnect(self.linebuilder2.cid) #DE SESCONECTA LA GRAFICA
```

4.3.8. Función *confirmDelta()*

```
def confirmDelta(self):
    #CONFIRMAR LA DISCRETIZACION ESPACIAL
    self.updateAllVarValues()
    COMFunction.spatialDiscretization()
    print("Velocidad de movimiento: ", Var.velocity, " mm/min")
```

4.3.9. Función *previewSerp()*

```
def previewSerp(self):
    #FUNCION PARA GRAFICAR EN LA VENTANA LA TRAYECTORIA DE SERPIENTE CON LOS PARAMETROS QUE EL USUARIO INGRESA
    #EN LA INTERFAZ GRAFICA

    #LIMPIAR PANALLA Y CREAR UNA NUEVA
    self.layout_grafica.removeWidget(self.static_canvas)
    createWidgetPlot(self)

    self.updateAllVarValues() #ACTUALIZAR TODOS LOS DATOS
    #print(Var.dom_x0, Var.dom_y0, Var.dom_xf, Var.dom_yf, Var.dom_dx)
    #EJECUTAR LA FUNCION CON LOS PARAMETROS INGRESADOS EN LA INTERFAZ GRAFICA, ESTE DEVUELVE UN VECTOR X y Y
```

```
#CON ESTOS DOS VECTORES SE GRAFICAN EN LA PANTALLA DE COLOR AZUL
x, y = preview_trajectory(Var.dom_x0, Var.dom_y0, Var.dom_xf, Var.dom_yf, Var.dom_dx)
line, = Var.static_ax.plot(x, y, color='blue', lw=2)
line.figure.canvas.draw()
```

4.3.10. Función *sendTraySerp()*

```
def sendTraySerp(self):
    #CREAR TREAD PARA EJECUTAR LA FUNCION DE TRAYECTO EN FORMA DE SERPIENTE
    task2 = threading.Thread(target=self.sendTraySerpThread, name='sendTraySerpThread',
                            args=()) # CREO EL PROCESO PARA EL HOLA
    # INICIO LOS PROCESOS
    task2.daemon = True
    task2.start() #SE INICIA EL HILO
```

4.3.11. Función *sendTraySerpThread()*

```
def sendTraySerpThread(self):
    #FUNCION QUE EJECUTA EL HILO CREADO EN sendTraySerp
    self.updateAllVarValues() #ACTUALIZAR TODOS LOS VALORES
    #EJECUTAR LA FUNCION gotoStartPos PARA ENVIAR AL POSICIONADOR A LA POSICION INICIAL DE LA TRAYECTORIA SERPIENTE
    confirm = COMFunction.gotoStartPos(Var.Ctrl_Pos, Var.dom_x0, Var.dom_y0)
    #UNA VEZ LA FUNCION RETORNA EL MENSAJE SE ENVÍA EL TRAYECTO AL POSICIONADOR
    if (confirm == "Ubicado en la posicion inicial"):
        COMFunction.tray_1(Var.Ctrl_Pos, Var.dom_x0, Var.dom_xf, Var.dom_y0, Var.dom_yf, Var.dom_dx, Var.velocity)
        Var.trajectory = "serpiente" #SE GUARDA EL MODO DE TRAYECTO
        Var.statMeas = "Pause" #EL ESTADO SE LA MEDICION INICIA EN PAUSA
        self.button_StartMeas.setEnabled(True) #HABILITAR EL BOTON DE INICIAR MEDIDAS
        self.button_confirm_serp.setEnabled(False) #DE DESHABILITA EL BOTON DE CONFIRMAR TRAYECTO SERPIENTE
    return #INDICARLE AL TREAD QUE ESTA FUNCION YA TERMINO
```

4.3.12. Función *sendTrayEsp()*

```
def sendTrayEsp(self):
    #ACTUALIZAR VALORES DE LOS INPUTS
    self.updateAllVarValues()
    #IR A LA POSICION INICIAL DE LA ESPIRAL
    confirm = COMFunction.gotoStartPos(Var.Ctrl_Pos, Var.dom_x0_esp, Var.dom_y0_esp)
    #UNA VEZ EL POSICIONADOR LLEGA A LA POSICION INICIAL SE EJECUTA LA FUNCION QUE CREA LOS CODIGOS G PARA HACER EL
    #TRAYECO
    if (confirm == "Ubicado en la posicion inicial"):
        COMFunction.spiral(Var.Ctrl_Pos, Var.dom_x0_esp, Var.dom_y0_esp, Var.dom_dx_esp, Var.dom_qtraj_esp,
                           Var.velocity)
        Var.trajectory = "espiral"
        Var.statMeas = "Pause"
        self.button_StartMeas.setEnabled(True)
        self.button_confirm_esp.setEnabled(False)
    return
```

4.3.13. Función *sendTrayEspThread()*

```
def sendTrayEspThread(self):
    self.updateAllVarValues()
    confirm = COMFunction.gotoStartPos(Var.Ctrl_Pos, Var.dom_x0_esp, Var.dom_y0_esp)
    if (confirm == "Ubicado en la posicion inicial"):
        COMFunction.spiral(Var.Ctrl_Pos, Var.dom_x0_esp, Var.dom_y0_esp, Var.dom_dx_esp, Var.dom_qtraj_esp,
                           Var.velocity)
        Var.trajectory = "espiral"
        Var.statMeas = "Pause"
        self.button_StartMeas.setEnabled(True)
        self.button_confirm_esp.setEnabled(False)
    return
```

4.3.14. Función *threadConeection()*

```

def threadConnection(self, connectTO):
    # FUNCION QUE DEFINE LOS TREADS PARA REALIZAR LA CONEXION, SI SE CONECTA AL VNA O AL POSICIONADOR
    if connectTO == "COM":
        task2 = threading.Thread(target=COMFunction.connectCOM, name='connectCOM',
                                args=(self,)) # CREO EL PROCESO PARA EL HOLA
        # INICIO LOS PROCESOS
        task2.daemon = True
        task2.start()

    elif connectTO == "VNA":
        task1 = threading.Thread(target=VNAFunction.connectVNA, name='connectVNA',
                                args=(self,)) # CREO EL PROCESO PARA EL HOLA
        # INICIO LOS PROCESOS
        task1.daemon = True
        task1.start()

```

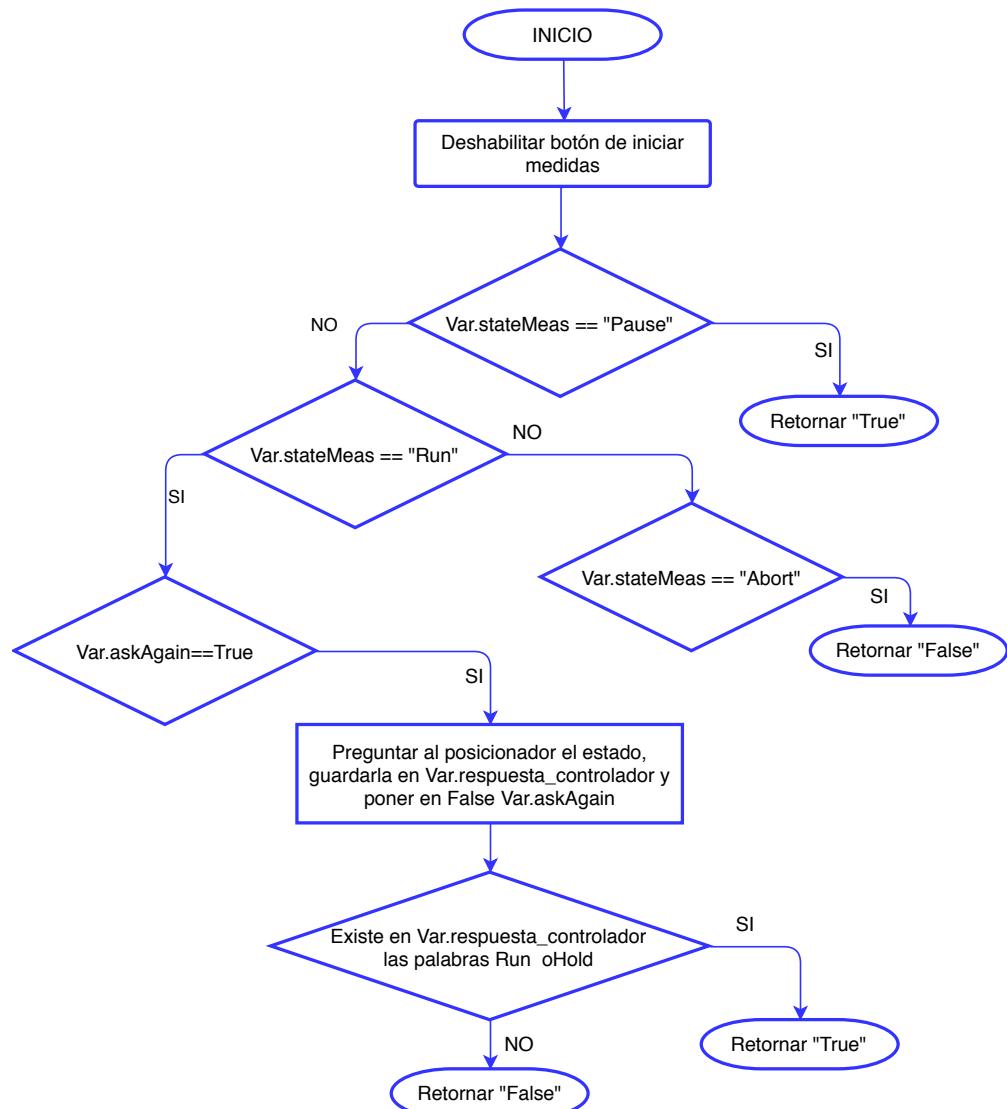
4.3.15. Función *isRunning()*

```

def isRunning():
    #PREGUNTO POR EL ESTADO
    #SI ESTA EN PAUSA ENVIAR TRUE PERO SIN ESTADO EN WHILE
    if (Var.stateMeas == "Pause"):
        Var.stateWhile = ""
        return True
    elif (Var.stateMeas == "Run"):
        #SI EL ESTADO ES RUN REALIZO SOLO UNA PREGUNTA AL POSICIONADOR PARA VALIDAR SI ESTA CORRIENDO
        if (Var.askAgain==True):
            #ENVIO EL DATO DE PREGUNTA
            Var.Ctrl_Pos.write("?.encode('utf-8')")
            #DECODIFICO LA RESPUESTA
            Var.respuesta_controlador = Var.Ctrl_Pos.readline().decode('utf-8')
            Var.respuesta_controlador = Var.respuesta_controlador.replace("\r\n", "")
            Var.askAgain=False
            #NO VUELVO A EJECURAR EL ASK AGAIN HASTA QUE SE INICIE UNA NUEVA MEDICION

            Var.Ctrl_Pos.reset_input_buffer() #LIMPIO EL BUFFER
            print(Var.respuesta_controlador)
            #SI DENTRO DE LA RESPUESTA ESTA RUN o HOLD EL POSICIONADOR ESTA REALIZANDO LA TRAYECTORIA
            if ("Run" in Var.respuesta_controlador) or ("Hold" in Var.respuesta_controlador):
                return True
            else:
                #SINO ESTAN ESAS DOS PALABRAS PUEDE QUE SE HAYA PRODUCIDO UNA ALERTA
                # SE RETORNA UN FALSE
                Var.stateMeas = "Pause"
                return False
        elif (Var.stateMeas == "Abort"):
            #SI LA MEDIDA SE ABORTA SE ENVIA UN FALSE PARA INDICARLE AL TREAD QUE NO SIGA REALIZANDO MEDICIONES DEL VNA Y
            # POSICIONADOR
            return False

```

Figura 6: Diagrama de Flujo para la función *isRunning()*

4.3.16. Función *saveMatlab()*

```

def saveMatlab():
    a = {} # CREA UN CELL CUANDO SE GUARDE EN MATLAB
    a['DatosVNAComplex'] = np.array(Var.DatosVNAComplex)
    a['elapsed'] = np.array(Var.elapsed)
    a['description'] = Var.description
    a['offset'] = Var.offset
    a['position'] = np.array(Var.position)
    a['threatTime'] = np.array(Var.threatTime)
    a['VNAPoints']=Var.puntos
    a['VNATreqIni']=Var.frec_ini
    a['VNATreqEnd']= Var.frec_fin
    a['POSVelocity']=Var.velocity

    # GUARDA EL ARCHIVO .mat EN CON LA FECHA Y HORA
    scipy.io.savemat("Measurements/" +time.strftime("%d-%m-%Y_%H-%M-%S"), a)

    #BORRAR VARIABLES QUE SE GUARDAN EN LAS MEDIDAS, PARA DEJARLAS LISTAS PARA LA SIGUIENTE MEDIDA.
    Var.DatosVNAComplex = np.array([])
    Var.position = np.array([])
    Var.threatTime = []
    Var.elapsed = []
    Var.CurrentY = []
    Var.CurrentX = []
    Var.askAgain = True

```

4.3.17. Función *measures()*

```

def measures(self):
    #FUNCION QUE SE EJECUTA CUANDO SE DA EL BOTON DE INICIAR MEDIDAS, ESTA ES UNA DE LAS FUNCIONES MAS IMPORTANTES
    #PORQUE INICIA EL PROCESO PARA TOMAR LAS MEDIDAS EN PARALELO DEL VNA Y DEL POSICONADOR

    Var.Ctrl_Pos.write("-".encode('utf-8')) # Indica al controlador de posicion que empiece la trayectoria enviada
    time.sleep(0.025) # Delay de 0.025 segundos para que las respuestas 'ok' lleguen al buffer de llegada
    Var.Ctrl_Pos.reset_input_buffer() # Elimina los datos en el buffer de entrada
    logging.debug("SE INICIO EL RECORRIDO")
    Var.stateMeas="Run" #SE INDICA QUE EL ESTADO DE LA MEDIDA ES RUN
    #HABILITAR BOTONES DE PAUSA, REINICIAR Y ABORTAR
    self.button_pause_measurements.setEnabled(True)
    self.button_abort_measurements.setEnabled(True)
    self.button_restart_measurements.setEnabled(True)
    #EL BOTON DE INICIAR MEDIDAS CAMBIA EL TEXTO
    self.button_StartMeas.setText("Realizando Medida")
    self.button_StartMeas.setDisabled(1) #SE DESHABILITA EL BOTON DE INICIAR MEDIDAS
    #COMFunction.restart_meas(self)
    self.git_thread.start() #SE INICA EL TREAD QUE CORRE LAS DOS FUNCIONES

```

4.3.18. Función *finished()*

```

def finished(self):
    #CUANDO FINALIZA LAS MEDICIONES SE AJUSTAN LOS BOTONES PARA LA SIGUIENTE MEDIDA
    logging.debug("SE ACABO")
    self.button_StartMeas.setText("Iniciar Medidas")
    self.button_StartMeas.setEnabled(False)
    self.button_pause_measurements.setEnabled(False)
    self.button_restart_measurements.setEnabled(False)
    self.button_abort_measurements.setEnabled(False)

```

4.3.19. Función *pause_meas()*

```

def pause_meas(self):
    #CUANDO SE OPRIME EL BOTON SE PAUSA
    Var.stateMeas = "Pause" #SE CAMBIA EL ESTADO DE LA MEDICION EN PAUSA
    self.button_StartMeas.setText("Medida Pausada")
    self.button_pause_measurements.setEnabled(False)
    self.button_abort_measurements.setEnabled(True)
    self.button_restart_measurements.setEnabled(True)
    COMFunction.pause_meas(self) #SE LE ENVIA AL POSICONADOR EL COMANDO G PARA PAUSAR

```

4.3.20. Función *restart_meas()*

```

def restart_meas(self):
    # CUÁNDO SE OPRIME EL BOTON SE REINICAR MEDIDAS
    Var.stateMeas = "Run" ##SE CAMBIA EL ESTADO DE LA MEDICION EN RUN
    self.button_StartMeas.setText("Realizando Medida")
    self.button_pause_measurements.setEnabled(True)
    self.button_abort_measurements.setEnabled(True)
    self.button_restart_measurements.setEnabled(True)
    COMFunction.restart_meas(self) #SE LE ENVIA AL POSICIONADOR EL COMANDO G PARA PAUSAR

```

4.3.21. Función *abort_meas()*

```

def abort_meas(self):
    # CUÁNDO SE OPRIME EL BOTON SE VA A BORTAR LA MEDICION ACTUAL
    Var.stateMeas = "Abort" ##SE CAMBIA EL ESTADO DE LA MEDICION SE ABORTA
    self.button_StartMeas.setText("Iniciar Medicas")
    self.button_pause_measurements.setEnabled(False)
    self.button_abort_measurements.setEnabled(False)
    self.button_restart_measurements.setEnabled(False)
    COMFunction.abort_meas(self) #SE LE ENVIA AL POSICIONADOR EL COMANDO G PARA PAUSAR

```

4.4. Función *main()*

```

def main():

    #DEFINIR VALORES INICIALES DE LAS VARIABLES
    Var.checkAllConnected = 0
    Var.puntos = 201 # CANTIDAD D PUNTOS EN EL SWEEP POINT Y EN EL TIEMPO DE TOMA DE MEDIDA
    Var.offset = 0.150 # OFSET DE TIEMPO, DURACION EN SEGUNDOS PROMEDIO DE ENTREGA DE DATOS DEL VNA AL PC
    Var.measures = 1000 # CANTIDAD DE MUESTRAS A TOMA
    Var.description = "Prueba de " + str(Var.measures) + " medidas con offset " + str(Var.offset) + " segundos"
    Var.instr = vxii1
    Var.setXinit=0
    Var.setYinit=0
    Var.dom_x0 = 0
    Var.dom_xf = 500
    Var.dom_y0 = 0
    Var.dom_yf = 500
    Var.dom_dx = 50
    Var.discretization = 5
    Var.velocity = 1000
    Var.dom_x0_esp = 400
    Var.dom_y0_esp = 400
    Var.dom_dx_esp = 10
    Var.dom_qtraj_esp = 6

    # VARIABLE DONDE SE GUARDAN TODOS LOS DATOS DEL VNA
    Var.DatosVNAComplex = np.array([])
    Var.position = np.array([])
    Var.threatTime = []
    # VARIABLE DONDE SE GUARDA EL TIEMPO
    Var.elapsed = []
    Var.CurrentX=[]
    Var.CurrentY=[]
    Var.threads = 2 # NUMERO DE PROCESOS
    Var.threads = list()
    Var.frec_ini = 0.6
    Var.frec_fin = 6
    Var.trajectory = "serpiente"
    Var.respuesta_controlador = "Run"

    #CONFIGURACION DE LA GRAFICA (DIMENSIONES QUE VA A TENER)
    Var.plotX=[-80,0]
    Var.plotY=[-200,0]
    Var.plotMargin=5 #MARGEN PARA REALIZAR GRAFICAS (EL RECUADRO ROJO)
    Var.askAgain=True

    app = QtWidgets.QApplication(sys.argv) # A new instance of QApplication
    form = ExampleApp() # We set the form to be our ExampleApp (design)
    form.show() # Show the form
    app.exec_() # and execute the app

if __name__ == '__main__': # if we are running file directly and not importing it
    main()

```

5. Funciones para el control del VNA

5.1. Clase Functions_VNA

Dentro de esta clase se encuentran las diferentes funciones que permitirán el control del VNA mediante la interfaz gráfica, entre ellos se encuentran cómo se realiza la conexión; cómo se guardan los datos y cómo se actualizan los parámetros del VNA.

5.1.1. Función *connectVNA()*

Esta función permite la conexión al VNA cuando se oprimen los botones de conexión, crea el objeto que contiene la información para interactuar con el VNA.

También esta función realiza la configuración inicial del VNA para dejar listo el equipo una vez comiencen las mediciones.

```
def connectVNA(self):
    # FUNCION QUE PERMITE CONECTAR AL VNA MEDIANTE EL PROTOCOLO VXI11
    # CABE RECORDAR QUE LAS DOS IPs DE LOS DOS EQUIPOS DEBEN ESTAR EN LA MISMA RED
    self.button_ConnectVNA.setText("Conectando ...") #CAMBIAR EL TEXTO DEL BOTON A CONECTANDO ...
    self.input_IP_VNA.setDisabled(1) #MIENTRAS SE TRATA DE CONECTAR, DE DESHABILITA EL INPUT DE LA IP
    self.button_ConnectVNA.setDisabled(1) #MIENTRAS SE TRATA DE CONECTAR, DE DESHABILITA EL BOTON DE CONECTAR
    Var.instr = vxi11.Instrument(self.input_IP_VNA.text()) #GENERO EL OBJETO DE LA CLASE VXI11 PARA CONECTAR LA IP INGRESADA EN LA INTERFAZ

    try:
        # INTENTAR CONECTARSE AL VNA
        Var.instr.open() #SE CREA LA CONEXION DEL VNA
        self.button_ConnectVNA.setText("Conectado") # SI SE CONECTA, CAMBIAR EL TEXTO DEL BOTON
        self.button_ConnectVNA.setDisabled(1) # DESHABILITAR EL BOTON PARA CONECTAR (PORQUE YA ESTOY CONECTADO)

        #VARIABLE DE CHECK PARA SABER SI SE CONECTO EL VNA Y EL POSICIONADOR
        #SI SE PUDO CONECTAR SE SUMA 1, SI EL VALOR SE ENCUENTRA EN 2, HABILITO EL BOTON DE INICAR HOMING
        Var.checkAllConnected += 1
        if (Var.checkAllConnected == 2):
            self.button_StartHoming.setEnabled(True)

    except:
        # SI NO SE REALIZAR LA CONEXION CONECTAR, ENVIAR MENSAJE DE ERROR, Y HABILITAR NUEVAMENTE EL BOTON DEL
        # CONECTAR Y EL INPUT
        logging.debug("NO SE PUDO CONECTAR AL EQUIPO")
        self.button_ConnectVNA.setDisabled(False)
        self.button_ConnectVNA.setText("Volver a Conectar")
        self.input_IP_VNA.setDisabled(False)
        return

    # SI EL PC SE PUDO CONECTAR AL VNA, ENTONCES EL OBJETO Var.instr SE CREO CORRECTAMENTE, POR LO TANTO YA PUEDE
    # EMPEZAR A CONFIGURAR EL VNA CON LOS PARAMETROS PARA REALIZAR LAS MEDICIONES

    Var.instr.write_raw(":SYST:PRES",encode()) # SE REALIZA UN PRESET AL VNA
    time.sleep(5) # 5 SEGUNDOS MIENTRAS HACE EL PRESET
    Var.instr.write_raw(":SENS:TRAC:TOT 2",encode()) # SELECCIONAR DOS TRAZAS
    Var.instr.write_raw(":DISP:TRAC:FORM SING",encode()) # MUESTRA UNA SOLA PANTALLA LAS DOS TRAZAS
    Var.instr.write_raw(":SENS:TRAC1:SEL",encode()) # SELECCIONA TRAZA 1
    Var.instr.write_raw(":SENS:TRAC1:DOM FREQ",encode()) # ASIGNA DOMINIO FREQ
    Var.instr.write_raw(":SENS:TRAC1:SPAR S21",encode()) # SELECCIONA S21 EN LA TRAZA 1
    Var.instr.write_raw(":SENS:TRAC2:SPAR S21",encode()) # SELECCIONA S21 EN LA TRAZA 2
    Var.instr.write_raw(":CALC1:FORM REAL",encode()) # SELECCIONA S21 EN FORMATO REAL DE LA TRAZA 1
    Var.instr.write_raw(":CALC2:FORM IMAG",encode()) # SELECCIONA S21 EN FORMATO IMAGINARIO DE LA TRAZA 2
    Var.instr.write_raw(":UNIT:POW dBm",encode()) # MEDICIONES REALIZADAS EN dBm
    Var.instr.write_raw(":SENS:FREQ:STAR " + str(Var.frec_ini) + "GHz",encode()) # FRECUENCIA DE INICIO
    Var.instr.write_raw(":SENS:FREQ:STOP " + str(Var.frec_fin) + "GHz",encode()) # FRECUENCIA DE PARADA
    Var.instr.write_raw(":SENS:SWE:POIN ",encode() + str(Var.puntos).encode()) # CANTIDAD DE PUNTO DE BARRIDO
    Var.instr.write_raw(":CALC1:LIM:LOW:POIN:X " + str(Var.frec_ini) + "GHz",encode()) # LIMITE INFERIOR EN X
    Var.instr.write_raw(":CALC1:LIM:UPP:POIN:X " + str(Var.frec_fin) + "GHz",encode()) # CALCULA EL LIMITE SUPERIOR EN X
    Var.instr.write_raw(":FORM:READ:DATA ASCII",encode()) # LOS DATOS ENTREGADOS SON EN FORMATO ASCII
    Var.instr.write_raw(":INIT:CONT ON",encode()) # BARRIDO CONTINUO
    Var.instr.write_raw(":INIT:HOLD OFF",encode())
    return Var.instr #RETORNA EL OBJETO QUE CONTIENE LA CONEXION AL VNA
```

5.1.2. Función *askMeasures()*

Esta función es ejecutada en paralelo con la función que realiza la pregunta al posicionador de la ubicación del GPR. A medida que se pregunta al VNA las mediciones, estas se guardan en una matriz; los datos obtenidos son en formato crudo por el VNA, para tener más información y entender cómo se entregan los datos del VNA , revisar [4] en la sección “Trace Data Transfer”.

```
def askMeasures():
    # FUNCION PARA GUARDAR LOS VALORES DE 21 QUE TIENE ALMACENADOS EL VNA

    logging.debug("SOLICITANDO MEDIDA AL VNA") #DEBUGGIN PARA INDICAR EN CUAL TREAD ESTA CORRIENDO
    t = time.time() # MARCO EL TIEMPO 0 COMO BASE
    # CREO UNA PAUSA, ESTA PAUSA ES DEL VNA PARA ADQUIRIR LOS n-PUNTOS
    # PARA EL VNA DE LA MARCA ANRITSU EL TIEMPO DE ADQUISICION ES 350us POR CADA PUNTO DE BARRIDO
    # A ESTO SE LE AGREGA UN OFFSET QUE ES EL TIEMPO PROMEDIO QUE TARDA EN ENVIAR LOS DATOS EL VNA AL PC
    time.sleep((350e-6) * Var.puntos + Var.offset)
    # SE GUARDAN LOS PUNTOS EN LA MATRIZ SIN PROCESAR QUE CONTIENE LOS DATOS
    # PARA MAS INFORMACION DE COMO ES EL FORMATO DE LOS DATOS, REVISAR LA HOJA DE PROGRAMACION SCPI DEL EQUIPO
    Var.DatosVNComplex = np.concatenate([Var.DatosVNComplex, [Var.instr.ask(":CALC1:DATA? SDAT")]], axis=0)
    # GAURDO EL TIEMPO QUE TARDO EN REALIZARSE LA OPERACION
    Var.elapsed.extend([time.time() - t])
    logging.debug("SE TOMO MEDIDA") #INDICO QUE YA SE ACABO LA MEDICION
```

5.1.3. Función *updateConf()*

```
def updateConf():
    # FUNCION QUE ACTUALIZA LA CONFIGURACION AL VNA CUANDO SE QUIERE CAMBIAR LA FRECUENCIA DE INICIO -FINAL
    # LA CANTIDAD DE PUNTOS
    Var.instr.write_raw(":SENS:FREQ:STAR " + str(Var.frec_ini) + "GHz").encode() # FRECUENCIA DE INICIO
    Var.instr.write_raw(":SENS:FREQ:STOP " + str(Var.frec_fin) + "GHz").encode() # FRECUENCIA DE PARADA
    Var.instr.write_raw(":SENS:SWE:POIN ".encode() + str(Var.punto).encode()) # CANTIDAD DE PUNTO DE BARRIDO
    Var.instr.write_raw(":CALC1:LIM:LOW:POIN:X " + str(Var.frec_ini) + "GHz").encode() # LIMITE INFERIOR EN X
    ("CALC1:LIM:UPP:POIN:X " + str(Var.frec_fin) + "GHz").encode() # CALCULA EL LIMITE SUPERIOR EN X
```

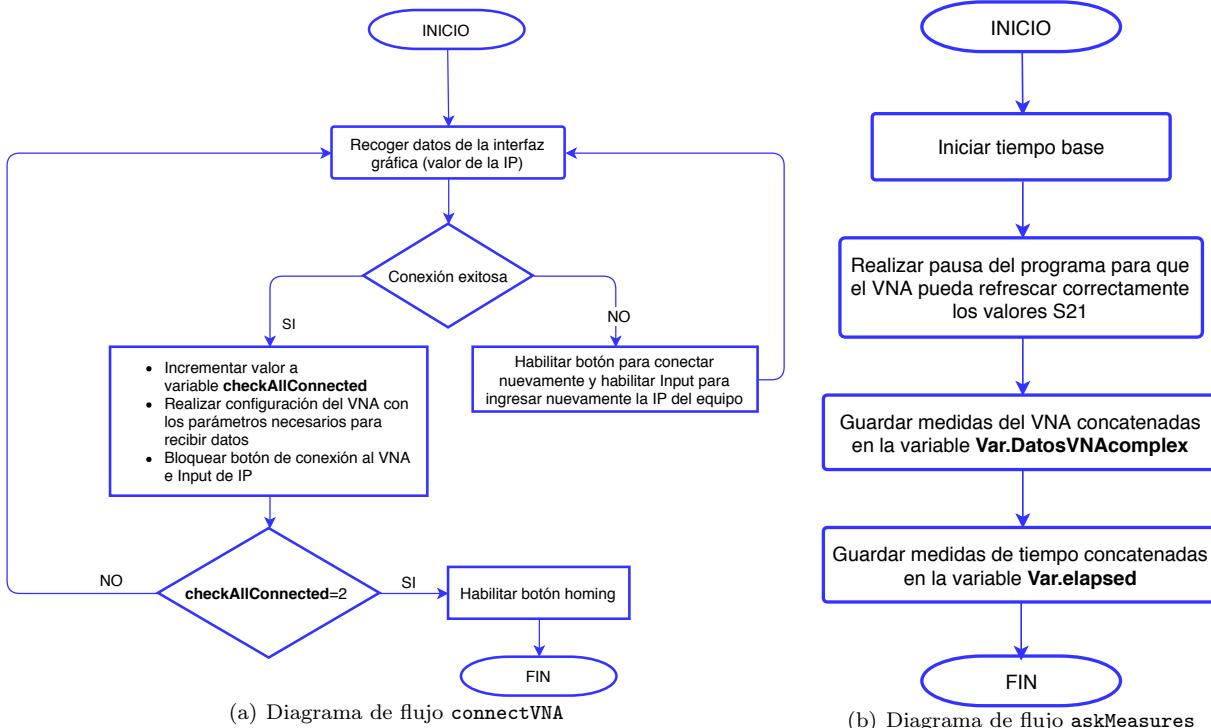


Figura 7: Diagramas de fuljo de las funciones *connectVNA* y *askMeasures*

6. Funciones para el control de posición

6.1. Clase Functions_COM

Para importar fácilmente las funciones desarrolladas para el control de posición, al programa principal de control del GPR, se implementó la clase `Functions_COM`. En la clase `Functions_COM` se encuentran todas las funciones implementadas para el control de la posición del VNA, que se describen a continuación:

6.1.1. Función connectCOM(self)

Esta función está encargada de establecer la conexión entre el PC y la tarjeta controladora xPro. La función recibe como parámetro el objeto `ExampleApp` relacionado con la interfaz gráfica. La función también se comunica con el programa principal, por medio de las variables globales definidas en la clase `Variables`. El diagrama de flujo que describe el código a continuación se presenta en la figura 8(a).

```
def connectCOM(self):
    logging.debug(self.input_COM.text())
    self.button_COM.setText("Conectando...") # CAMBIA EL TEXTO DEL BOTON DE CONEXION
    self.input_COM.setDisabled(1) # DESHABILITA LA ENTRADA DE TEXTO PARA EL PUERTO MIENTRAS ESTA CONECTANDO
    self.button_COM.setDisabled(1) # DESHABILITA EL BOTON DE CONEXION MIENTRAS ESTA CONECTANDO
    Var.Ctrl_Pos = Functions_COM.configcon(
        self.input_COM.text()) # EJECUTA LA FUNCION DE CONFIGURAR SERIAL CON EL TEXTO DE input_COM
    try:
        # INTENTA CONECTARSE AL COM
        if Var.Ctrl_Pos.isOpen():
            logging.debug("Puerto Abierto") # SI YA EXISTE UNA CONEXION EN ESE PUERTO LO REPORTA
        else:
            Var.Ctrl_Pos.open() # ABRE LA CONEXION CON EL PUERTO SERIAL
            self.button_COM.setText("Conectado") # CAMBIA EL TEXTO DEL BOTON DE CONEXION
            self.button_COM.setDisabled(1) # DESHABILITA EL BOTON DE CONEXION
            Var.checkAllConnected += 1
        if (Var.checkAllConnected == 2):
            self.button_StartHoming.setEnabled(True)
    except:
        # SI NO SE PUEDE CONECTAR, ENVIA MENSAJE DE ERROR Y TERMINA PROGRAMA
        logging.debug("NO SE PUDO CONECTAR AL EQUIPO")
        self.button_COM.setDisabled(False) # HABILITA EL BOTON DE CONEXION
        self.button_COM.setText("Volver a conectar") # CAMBIA EL MENSAJE DEL BOTON DE CONEXION
        self.input_COM.setDisabled(False) # HABILITA LA ENTRADA DE TEXTO PARA EL PUERTO
    return
```

6.1.2. Función homing(self,OBJ_S)

Esta función se encarga de calibrar las coordenadas del GPR. La función recibe como parámetros: el objeto `ExampleApp` relacionado con la interfaz gráfica y el objeto serial que se creó en la conexión del PC con el controlador de posición. La función también se comunica con el programa principal, por medio de las variables globales definidas en la clase `Variables`. Mediante el comando `$H` se indica al controlador que inicie el proceso de calibración. La función espera a que se culmine la calibración con el objetivo de conocer si el proceso de calibración fue exitoso.

El diagrama de flujo que describe el código se presenta en la figura 8(b).

```

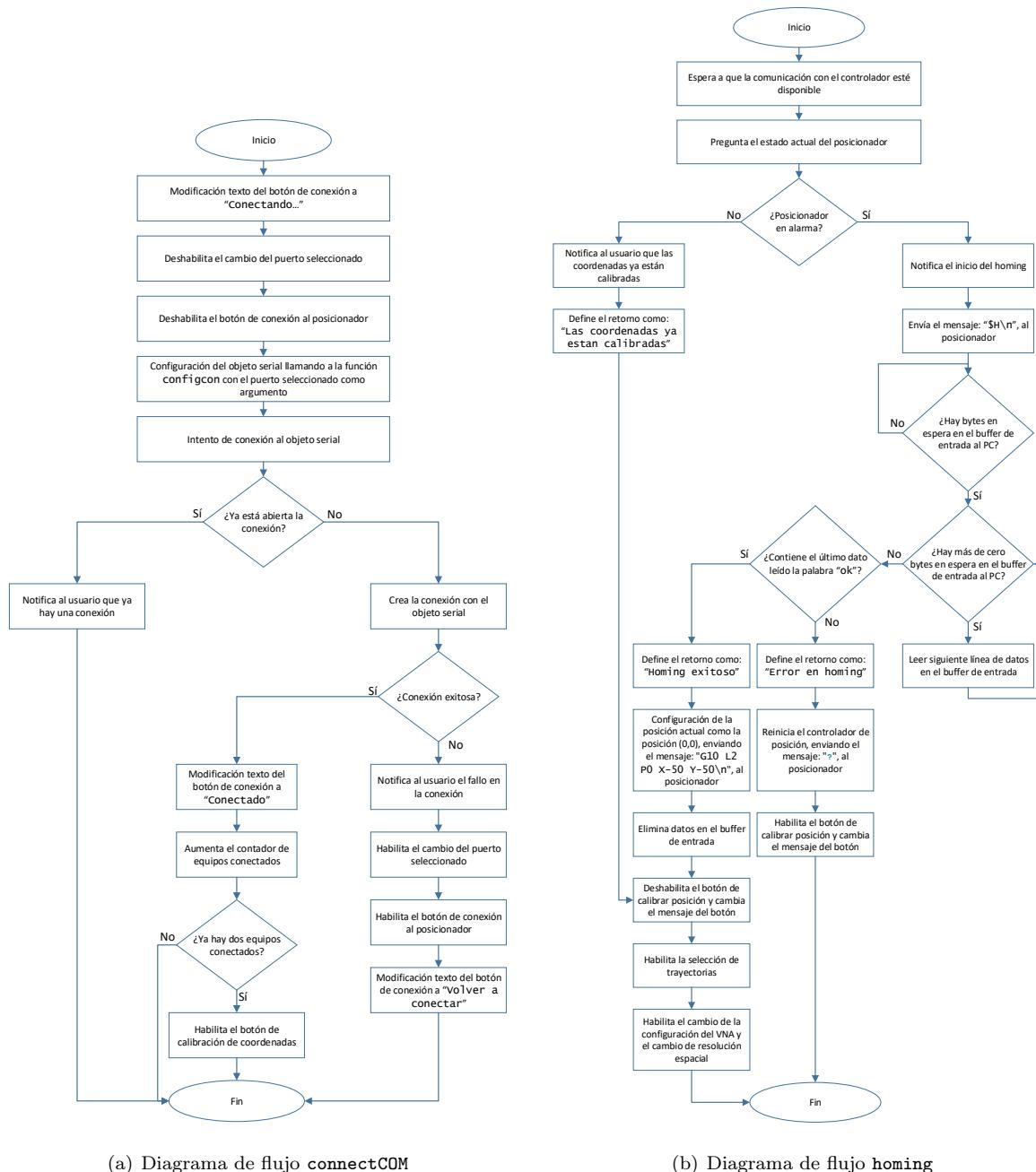
def homing(self,OBJ_S):
    # ESTA FUNCIÓN DEFINE LA RUTINA DE HOMING PARA EL CONTROLADOR DE POSICION.
    # ESPERA QUE LA COMUNICACION CON EL CONTROLADOR DE POSICION ESTE DISPONIBLE:
    iniciar = False
    while not iniciar:
        OBJ_S.write("?",encode('utf-8'))
        if OBJ_S.in_waiting >= 56:
            OBJ_S.reset_input_buffer()
            iniciar = True
    # PREGUNTA AL CONTROLADOR DE POSICION EL ESTADO ACTUAL:
    OBJ_S.write("?",encode('utf-8'))
    while OBJ_S.in_waiting == 0:
        pass
    is_homed = OBJ_S.read_until("\r\n").decode('utf-8')
    # SI SE ENCUENTRA EN ESTADO DE ALARMA ("ALARM") ES NECESARIO REALIZAR LA RUTINA DE HOMING. SI NO SE ENCUENTRA EN
    # ESTADO DE ALARMA, SE ENVIA UN MENSAJE DICENDO QUE YA TIENE CALIBRADO EL SISTEMA DE COORDENADAS:
    if "Alarm" in is_homed:
        print("Iniciando homing")
        OBJ_S.write("$H\n".encode('utf-8')) # ENVIA LA INSTRUCCION DE HOMING AL CONTROLADOR DE POSICION
        # ESPERA A QUE EL CONTROLADOR DE POSICION ACABE LA RUTINA DE HOMING ANTES DE CONTINUAR:
        while OBJ_S.in_waiting == 0:
            pass
        # ALMACENA LA RESPUESTA DEL CONTROLADOR DE POSICION DESPUES DEL HOMING:
        ok_flag = 1
        while OBJ_S.in_waiting > 0:
            ok_flag = OBJ_S.read_until("\r\n").decode('utf-8')
            print(ok_flag)
    # VERIFICÁ SI LA RESPUESTA INDICA UN HOMING EXITOSO ("OK")
    if "ok" in ok_flag:
        # SI EL HOMING ES EXITOSO, ESTO SE INDICA MEDIANTE UN MENSAJE Y SE DEFINE EL PUNTO INICIAL QUE SE TOMA
        # COMO EL CERO DEL SISTEMA COORDENADO (0,0)
        mensaje = "Homing exitoso"
        OBJ_S.write("G10 L2 P0 X-50 Y-50\n".encode('utf-8'))
        OBJ_S.reset_input_buffer()
    else:
        # SI EL HOMING FALLA, ESTO SE INDICA MEDIANTE UN MENSAJE Y SE ENVIA EL CARACTER " " PARA REINICIAR EL
        # EL POSICIONADOR
        mensaje = "Error en homing"
        OBJ_S.write(" ".encode('utf-8'))
    else:
        mensaje = "Las coordenadas ya estan calibradas"

    if mensaje == "Homing exitoso" or mensaje == "Las coordenadas ya estan calibradas":
        # SI LAS COORDENADAS ESTAN BIEN:
        # OPERACIONES CON LA INTERFAZ GRAFICA:
        self.button_StartHoming.setEnabled(False)
        self.combo_sel_trayectoria.setEnabled(True)
        self.button_StartHoming.setText("Posicionador Calibrado")

        # HABILITA LAS OPCIONES PARA CAMBIAR LA CONFIGURACION DEL VNA Y LA DISCRETIZACION DEL POSICIONADOR:
        self.input_delta.setEnabled(True)
        self.button_confirm_delta.setEnabled(True)
        self.input_points.setEnabled(True)
        self.input_freq_ini.setEnabled(True)
        self.input_freq_fin.setEnabled(True)
        self.button_confirm_vna.setEnabled(True)
    else:
        # SI LAS COORDENADAS NO ESTAN BIEN:
        # OPERACIONES CON LA INTERFAZ GRAFICA:
        self.button_StartHoming.setEnabled(True)
        self.button_StartHoming.setText("Calibrar Posicion")

    return mensaje # DEVUELVE EL MENSAJE CON LA INFORMACION QUE INDICA SI EL HOMING FUE EXITOSO

```



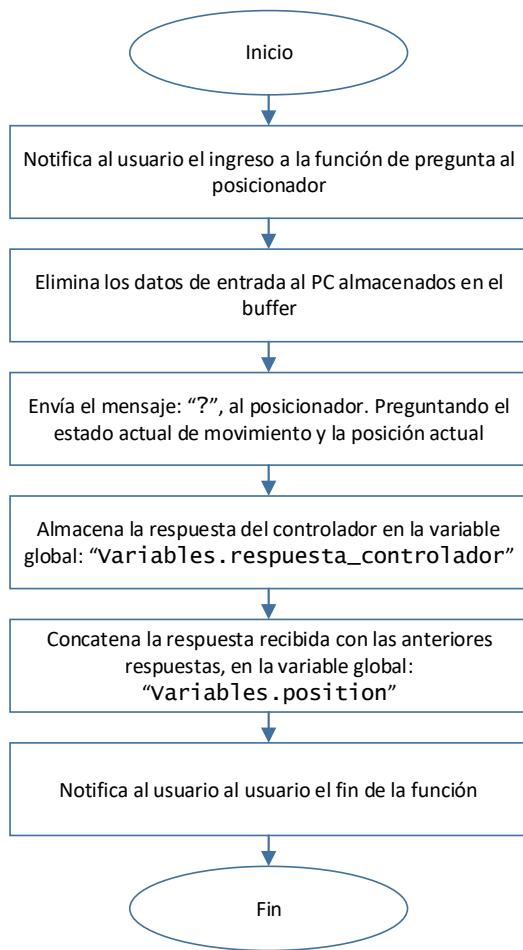
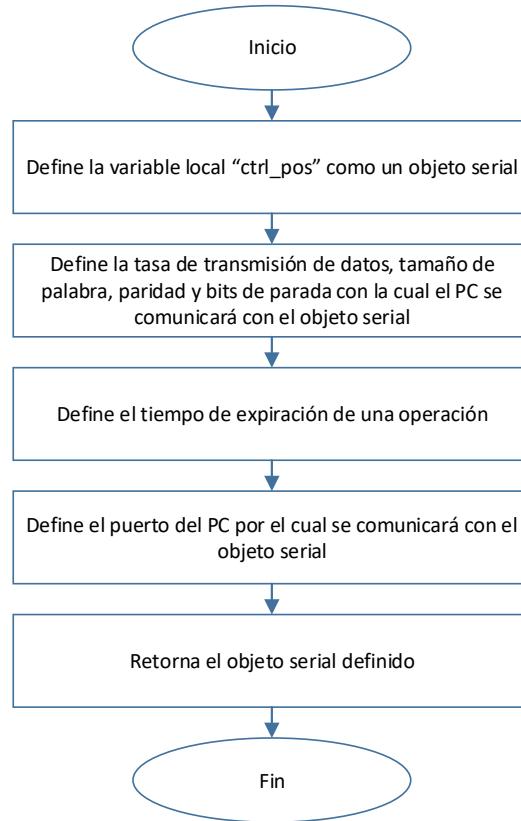
Los datos de respuesta almacenados contienen la información completa que responde el controlador de posición, en formato de arreglo de caracteres. El diagrama de flujo de la función se muestra en la figura 9(a).

```
def askPosition():
    logging.debug("PREGUNTANDO POSICION AL POSICIONADOR") # MENSAJE INFORMANDO DE LA LECTURA AL USUARIO
    Var.Ctrl_Pos.reset_input_buffer() # DEPURA LOS DATOS ALMACENADOS EN EL BUFFER DE ENTRADA
    Var.Ctrl_Pos.write("?", encode('utf-8')) # PREGUNTA LA POSICION Y ESTADO ACTUAL AL CONTROLADOR DE POSICION
    Var.respuesta_controlador = Var.Ctrl_Pos.readline().decode('utf-8') # ALMACENA LA RESPUESTA OBTENIDA
    # CONCATENA LA ULTIMA RESPUESTA CON LAS RESPUESTAS ANTERIORES EN UN ARREGLO:
    Var.position = np.concatenate([Var.position, [Var.respuesta_controlador]], axis=0)
    logging.debug("FIN POSICIONADOR") # MENSAJE INFORMANDO EL FIN DE LA LECTURA AL USUARIO
```

6.1.4. Función configcon(PUERTO)

Esta función está encargada de configurar los parámetros del objeto serial para definir la velocidad, número de bits, paridad de la comunicación, y número de puerto al que se asocia el objeto serial. Como parámetro de entrada recibe el puerto definido en la interfaz gráfica. Tiene como retorno el objeto serial. En los comentarios del código se explica en detalle la configuración definida para la conexión con la tarjeta xPro. El diagrama de flujo de la función se muestra en la figura 9(b).

```
def configcon(PUERTO):
    ctrl_pos = serial.Serial()
    ctrl_pos.baudrate = 115200
    ctrl_pos.bytesize = 8
    ctrl_pos.parity = serial.PARITY_NONE
    ctrl_pos.stopbits = 1
    # BAUDRATE: 115200 -TASA DE TRANSMISION DE BITS POR SEGUNDO. ESTA TASA ES
    # DEFINIDA EN 115200 SEGUN LO INDICADO POR EL PROTOCOLO GRBL PARA LA
    # TARJETA DE CONTROL DE POSICION UTILIZADA:
    # HTTPS://GITHUB.COM/GNEA/GRBL/WIKI/GRBL-V1.1-CONFIGURATION
    #
    # BYTESIZE: 8 -CANTIDAD DE BITS DE DATOS A TRANSMITIR. INDICA LA CANTIDAD
    # DE BITS DE DATOS EN UNA PALABRA TRANSMITIDA, ES DEFINIDA EN 8 SEGUN LO
    # INDICADO POR EL PROTOCOLO GRBL PARA LA TARJETA DE CONTROL DE POSICION
    # UTILIZADA: HTTPS://GITHUB.COM/GNEA/GRBL/WIKI/GRBL-V1.1-CONFIGURATION
    #
    # PARITY: NONE -TIPO DE COMPROBACION DE PARIDAD. PARA ESTA CONFIGURACION
    # NO SE UTILIZA COMPROBACION DE PARIDAD SEGUN LO INDICADO POR EL PROTOCOLO
    # GRBL PARA LA TARJETA DE CONTROL DE POSICION UTILIZADA:
    # HTTPS://GITHUB.COM/GNEA/GRBL/WIKI/GRBL-V1.1-CONFIGURATION
    #
    # STOPBITS: 1 -CANTIDAD DE BITS UTILIZADOS PARA INDICAR EL FIN DE UN BYTE.
    # CON UN BIT DE PARADA SE INDICA EL FINAL DE LA PALABRA SEGUN LO INDICADO
    # POR EL PROTOCOLO GRBL PARA LA TARJETA DE CONTROL DE POSICION UTILIZADA:
    # HTTPS://GITHUB.COM/GNEA/GRBL/WIKI/GRBL-V1.1-CONFIGURATION
    ctrl_pos.timeout = 5
    # TIMEOUT: 5 SEGUNDOS -TIEMPO MAXIMO PARA COMPLEATAR UNA OPERACION DE
    # LECTURA O ESCRITURA CON EL OBJETO SERIAL. CUANDO SE VENCE EL TIEMPO
    # CONFIGURADO OCURRE UN EVENTO DE TIME-OUT, AL OCURRIR UN EVENTO DE
    # TIME-OUT SE ABORTA LA OPERACION DE LECTURA O ESCRITURA REALIZADA Y SE
    # GENERA UN EVENTO DE ERROR.
    ctrl_pos.port = PUERTO # DEFINE EL PUERTO COM DEL PC AL CUAL ESTA CONECTADA LA TARJETA
    return ctrl_pos
```

(a) Diagrama de flujo `askPosition`(b) Diagrama de flujo `configcon`Figura 9: Diagramas de fuljo de las funciones `askPosition` y `configcon`

6.1.5. Función `gotoStartPos(OBJ_S,x,y)`

Esta función está encargada de ubicar el GPR en la posición inicial de la trayectoria definida. Como parámetros recibe el objeto serial, la coordenada *x* de la posición inicial y la coordenada *y* de la posición inicial. La función retorna un mensaje cuando llega a la posición inicial de la trayectoria, confirmando al programa principal que ya se encuentra disponible para iniciar la trayectoria. El diagrama de flujo de la función se muestra en la figura 10(a).

```

def gotoStartPos(OBJ_S, x,y):
    # SELECCION DE COORDENADAS ABSOLUTAS:
    OBJ_S.write("G90\n".encode('utf-8'))
    # UBICACION DEL GPR EN LAS COORDENADAS INICIALES, INDICADAS CON LOS PARAMETROS X0 Y Y0:
    ubicacion_inicial = "G0 X-%d Y-%d\n" % (x, y)
    OBJ_S.write(ubicacion_inicial.encode('utf-8'))
    time.sleep(0.025)
    OBJ_S.reset_input_buffer()

    isRunning = True
    # SE MANTIENE EL PROGRAMA EN LA FUNCION MIENTRAS NO HAYA LLEGADO A LA POSICION INICIAL:
    while isRunning:
        OBJ_S.reset_input_buffer()
        OBJ_S.write("?.encode('utf-8')")
        respuesta = OBJ_S.readline().decode('utf-8')
        if "Run" in respuesta:
            isRunning = True
        else:
            isRunning = False

    return "Ubicado en la posicion inicial"

```

6.1.6. Función tray_1(OBJ_S, x0, xf, y0, yf, dx, velocity)

Esta función se encarga de calcular los puntos que definen la trayectoria en forma de serpiente y de enviar al controlador de posición los comandos para realizar la trayectoria. Recibe como parámetro el objeto serial, la posición inicial de la trayectoria, la posición final, el cambio seleccionado en la dirección x, y la velocidad a la cual se desplazará el GPR. Una vez se calculen y se envíen todos los comandos para realizar la trayectoria, la función detiene el movimiento para que el usuario tenga la opción de iniciar el movimiento desde la interfaz gráfica. El diagrama de flujo de la función se muestra en la figura 10(b).

```

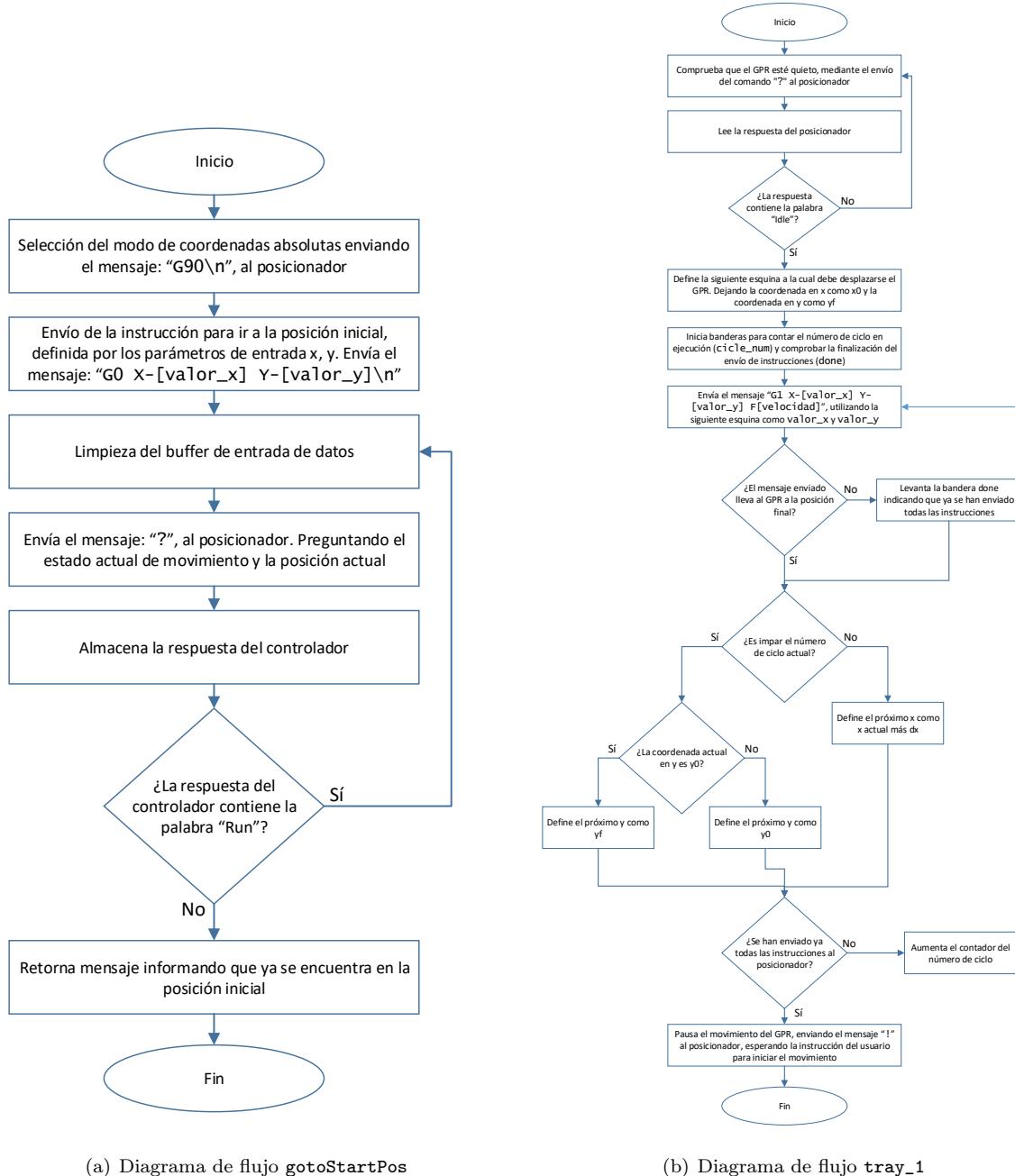
def tray_1(OBJ_S, x0, xf, y0, yf, dx, velocity):
    # LEE EL ESTADO DEL GPR, VERIFICANDO QUE ESTE QUIETO ("IDLE") PARA EMPEZAR A ENVIAR LA TRAYECTORIA:
    iniciar_trayectoria = 0
    while iniciar_trayectoria == 0:
        OBJ_S.write("?.encode('utf-8')")
        init_flag = OBJ_S.read_until("\r\n").decode('utf-8')
        #print(init_flag)
        if "Idle" in init_flag:
            iniciar_trayectoria = 1
        else:
            iniciar_trayectoria = 0
    # DEFINE EL PUNTO SIGUIENTE A DONDE DEBE MOVER EL GPR:
    next_x = x0
    next_y = yf
    # INICIA DOS BANDERAS, UNA PARA VERIFICAR EN QUE CICLO SE ENCUENTRA Y OTRA PARA SABER SI YA ACABO DE ENVIAR LOS
    # COMANDOS PARA LA TRAYECTORIA AL CONTROLADOR DE POSICION:
    cicle_num = 1
    done = False
    OBJ_S.reset_input_buffer()
    # MIENTRAS HAYA COMANDOS POR ENVIAR CALCULA EL SIGUIENTE PUNTO Y ENVIA LOS COMANDOS:
    while not done:
        # ENVIO DE LOS COMANDOS:
        linea = "G1 X-%d Y-%d F%d\n" % (next_x, next_y, velocity)
        print(linea)
        OBJ_S.write(linea.encode('utf-8'))
        time.sleep(0.02)

        # VERIFICA QUE NO HAYA LLEGADO YA AL PUNTO FINAL:
        if next_x == xf and next_y == yf:
            done = True
        # CONFIGURACION DEL SIGUIENTE PUNTO, EN LOS CICLOS PARES (2,4,6,...), LA 'SERPIENTE' SE MUEVE EN DIRECCION X.
        # EN LOS CICLOS IMPARES (1,3,5,...), LA 'SERPIENTE' SE MUEVE EN DIRECCION Y. A CONTINUACION SE DEFINE EL
        # CAMBIO DE DIRECCION PARA EL SIGUIENTE CICLO:
        if cicle_num % 2 == 1:
            next_x = next_x + dx
        else:
            if next_y == y0:
                next_y = yf
            else:
                next_y = y0

        if not done:
            cicle_num += 1

    OBJ_S.write("!.encode('utf-8')) # DETIENE EL MOVIMIENTO ESPERANDO INDICACION DEL USUARIO

```

Figura 10: Diagramas de fuljo de las funciones `gotoStartPos` y `tray_1`

6.1.7. Función `pause_meas()`

Esta función pausa el movimiento del GPR. Cuando se pausa el movimiento, el usuario tiene la opción de reanudarlo o de abortar la medición. El diagrama de flujo de la función se muestra en la figura 11(a).

```
def pause_meas(self):
    # PAUSA EL MOVIMIENTO CON EL COMANDO "!" ENVIADO:
    Var.stateMeas = "Pause"
    Var.Ctrl_Pos.write("!".encode('utf-8'))
```

6.1.8. Función restart_meas()

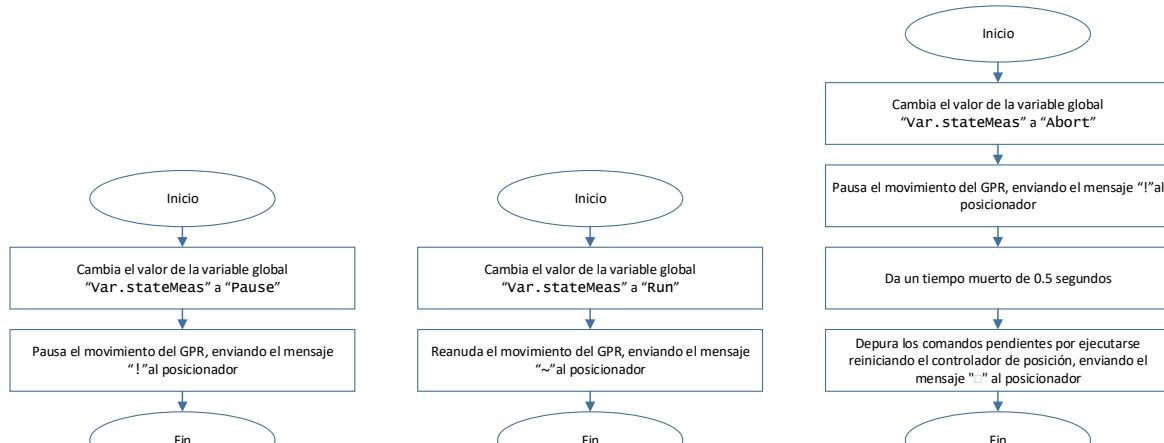
Esta función pausa el movimiento del GPR, después de haberse pausado. El diagrama de flujo de la función se muestra en la figura 11(b).

```
def restart_meas(self):
    # REANUNDA EL MOVIMIENTO CON EL COMANDO "~" ENVIADO:
    print("SE CONTINUAN LA MEDIDAS")
    Var.stateMeas = "Run"
    Var.Ctrl_Pos.write("~/".encode('utf-8'))
```

6.1.9. Función abort_meas()

Esta función aborta la trayectoria que se esté realizando con el GPR. Al abortar la trayectoria se omiten todos los comandos no ejecutados y el GPR queda disponible para realizar una nueva trayectoria. El diagrama de flujo de la función se muestra en la figura 11(c).

```
def abort_meas(self):
    # PRIMERO PAUSA EL MOVIMIENTO CON EL COMANDO "!. DESPUES DE UNA ESPERA DE 0.5 SEGUNDOS ENVIA EL COMANDO 0x18
    # PARA REINICIAR EL SISTEMA
    Var.stateMeas = "Abort"
    Var.Ctrl_Pos.write("!".encode('utf-8'))
    time.sleep(0.5)
    Var.Ctrl_Pos.write(serial.to_bytes([0x18]))
    Var.Ctrl_Pos.reset_input_buffer()
```



(a) Diagrama de flujo pause_meas

(b) Diagrama de flujo restart_meas

(c) Diagrama de flujo abort_meas

Figura 11: Diagramas de fuljo de las funciones pause_meas, restart_meas y abort_meas

6.1.10. Función toplotCurrentPos(respuesta)

Esta función lee la respuesta del GPR sobre su posición actual y obtiene los valores de las coordenadas x y y en valores numéricos en centímetros, para mostrar la posición actual en la gráfica de la interfaz. Recibe como parámetro la respuesta obtenida del controlador de posición. Retorna las coordenadas x y y en valores numéricos. El diagrama de flujo de la función se muestra en la figura 12(a).

```
def toplotCurrentPos(respuesta):
    # FUNCION PARA GRAFICAR LA POSICION ACTUAL. RECIBE COMO PARAMETRO LA ULTIMA RESPUESTA DEL CONTROLADOR DE
    # POSICION. RETORNA LAS COORDENADAS MEDIDAS.
    word_to_look = "WPos:"
    comma = ","
    # BUSCA LA PALABRA "WPos:" PARA OBTENER LA COORDENADA X DE LA POSICION ACTUAL Y "," PARA EL FINAL DE LA PALABRA:
    Wpos = respuesta.find(word_to_look)
    pos_x1 = Wpos + len(word_to_look)
    pos_x2 = respuesta.find(comma)
    # BUSCA LA PALABRA ":" PARA OBTENER LA COORDENADA Y DE LA POSICION ACTUAL Y LA SIGUIENTE "," PARA EL FINAL
    # DE LA PALABRA:
    pos_y1 = respuesta.find(comma) + len(comma)
    pos_y2 = respuesta.find(comma, pos_y1)
    # ALMACENA LOS DATOS EN VAIRABLES DE TIPO FLOTANTE EN CENTIMETROS:
    x = float(respuesta[pos_x1:pos_x2]) / 10
    y = float(respuesta[pos_y1:pos_y2]) / 10

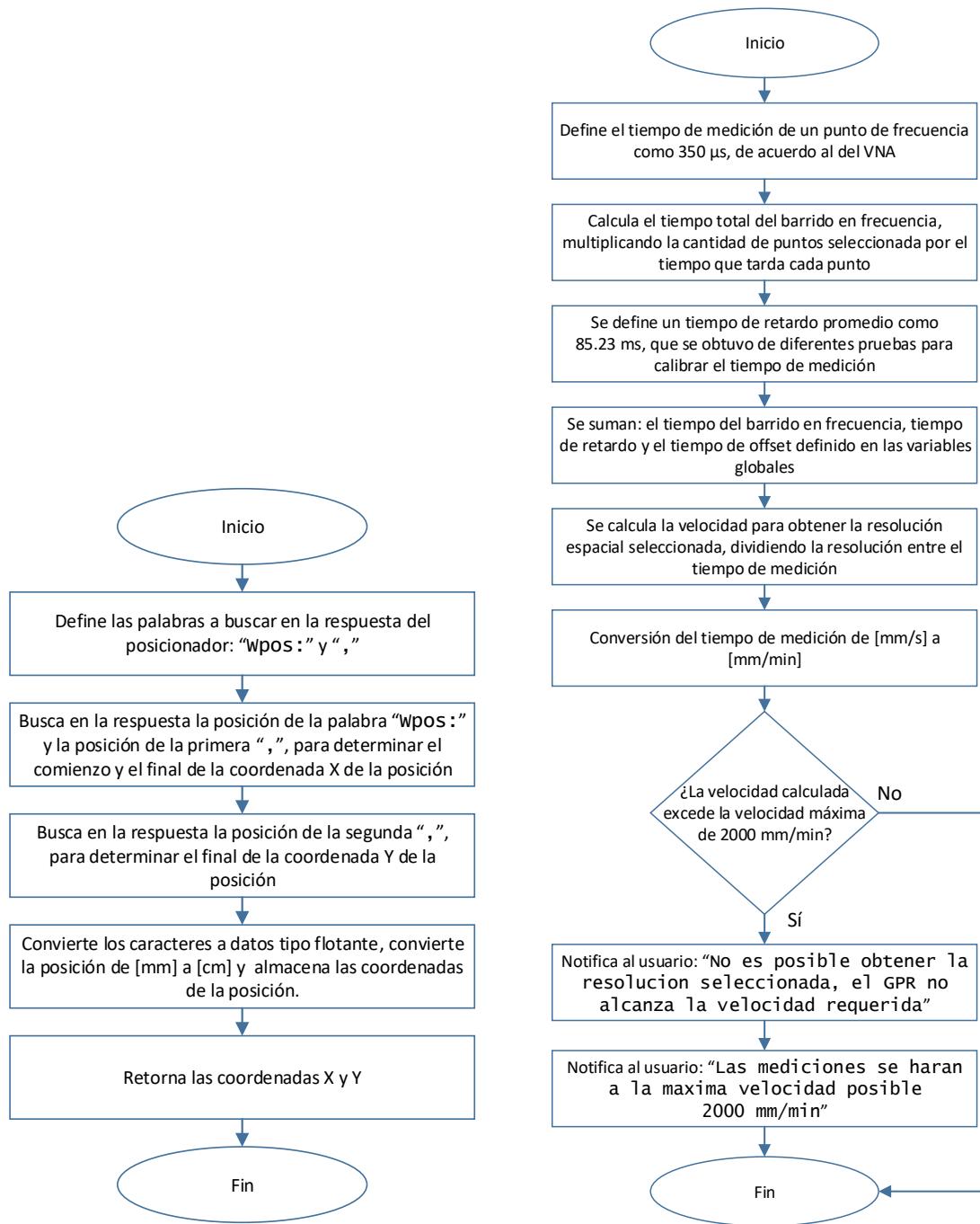
    return x, y
```

6.1.11. Función spatialDiscretization()

Esta función calcula la velocidad con la cual se debe mover el GPR para cumplir con la resolución espacial estipulada por el usuario. La velocidad calculada, se almacena en la variable global `Var.velocity`. Si la velocidad calculada, para cumplir con la resolución seleccionada, excede la velocidad máxima del GPR; se selecciona la velocidad máxima del GPR y se notifica al usuario sobre la limitación de velocidad y la velocidad seleccionada para realizar las mediciones. El diagrama de flujo de la función se muestra en la figura 12(b).

```
def spatialDiscretization():
    # CALCULA LA VELOCIDAD DE MOVIMIENTO BASADO EN LA DISCRETIZACION ESPACIAL DEFINIDA:
    t_punto = 350e-6 # TIEMPO DE MEDICION DE UN PUNTO [s]
    t_full_puntos = t_punto * Var.puntos # TIEMPO DE BARRIDO DE TODOS LOS PUNTOS [s]
    avg_delay_read = 85.23e-3 # TIEMPO PROMEDIO DE LECTURA DE DATOS [s]
    avg_meas_time = t_full_puntos + Var.offset + avg_delay_read # TIEMPO PROMEDIO DE MEDICION [s]

    vel_mm_s = Var.discretization / avg_meas_time # VELOCIDAD CALCULADA DEL GPR [mm/s]
    Var.velocity = int(vel_mm_s * 60) # VELOCIDAD DEL GPR [mm/min]
    # CONDICION Y AVISO AL USUARIO, PARA LIMITAR LA VELOCIDAD DEL GPR A LA MAXIMA PERMITIDA:
    if Var.velocity > 2000:
        print("No es posible obtener la resolucion seleccionada, el GPR no alcanza la velocidad requerida")
        print("Las mediciones se haran a la maxima velocidad posible 2000 mm/min")
    Var.velocity = 2000
```

Figura 12: Diagramas de fuljo de las funciones `topotCurrentPos` y `spatialDiscretization`

7. Funciones para realizar dibujos

7.1. Clase DrawLine

La clase **DrawLine** tiene como objetivo manejar llamados por eventos para dibujar las trayectorias manuales en los ejes de la figura definida en la interfaz gráfica. Esta clase, maneja los eventos de click generados en la figura, almacena y dibuja los puntos de la trayectoria definida manualmente.

7.1.1. Función `__init__(self, line)`

La función especial `__init__()` define la inicialización de un objeto de clase **DrawLine**. Como parámetros de entrada tiene el objeto de clase **DrawLine** y la línea con el punto inicial definida en la función principal. Esta función define las variables para almacenar las coordenadas en la clase y conecta la figura en la interfaz gráfica con eventos de obturación del ratón en los ejes de la figura.

```
def __init__(self, line):
    # INICIALIZACION DE LA CLASE. RECIBE COMO PARAMETRO UN OBJETO DE CLASE DrawLine Y EL PUNTO INICIAL DE
    # LA LINEA DEFINIDA EN LA FUNCION PRINCIPAL:
    self.line = line # EL PUNTO RECIBIDO DE LA FUNCION PRINCIPAL SE ALMACENA COMO UNA VARIABLE DE LA CLASE.
    self.xs = list(line.get_xdata()) # OBTIENE LA COORDENADA X DEL PUNTO RECIBIDO
    self.ys = list(line.get_ydata()) # OBTIENE LA COORDENADA Y DEL PUNTO RECIBIDO
    # CONECTA EL PUNTO CON LA FIGURA EN LA INTERFAZ GRAFICA MEDIANTE EVENTOS DE OBTURACION DEL RATON EN LOS EJES
    # DE LA FIGURA:
    self.cid = line.figure.canvas.mpl_connect('button_press_event', self)
```

7.1.2. Función `__call__(self, event)`

La función especial `__call__()` se encarga del manejo de los eventos de obturación del ratón. Recibe como parámetros el objeto de clase **DrawLine** y el evento del ratón. La función comprueba que el evento del ratón halla ocurrido dentro de los ejes de la figura; si el evento ocurre dentro de los ejes, se dibuja la línea uniendo el punto anterior con el actual y las coordenadas del evento son almacenadas; si el evento ocurre por fuera de los ejes, la función no hace nada.

```
def __call__(self, event):
    # CUANDO SE LLAMA A LA CLASE SE ALAMACENA LA POSICION Y SE DIBUJA:
    print('click', event) # ANUNCIA EL ENVENTO QUE LLAMA A LA CLASE
    # COMPRUEBA QUE EL ENVENTO DEL RATON HAYA OCURRIDO DENTRO DE LOS EJES DE LA FIGURA:
    if event.inaxes == self.line.axes:
        self.xs.append(event.xdata) # ALACENA LA COORDENADA X DE LA OBTURACION
        self.ys.append(event.ydata) # ALACENA LA COORDENADA Y DE LA OBTURACION
        self.line.set_data(self.xs, self.ys) # CONCATENA LAS COORDENADAS ACTUALES CON LAS ANTERIORES
        self.line.figure.canvas.draw() # DIBUJA TODAS LAS COORDENADAS HASTA EL MONETO PARA FORMAR LA LINEA
    else:
        return
```

7.2. Clase AskInitPoint

La clase **AskInitPoint** se utiliza para determinar el punto inicial de la trayectoria manual. Esta clase es muy similar a la clase **DrawLine**, se diferencia en la implementación de la función `__call__()`, ya que en este caso se desconectan los eventos del ratón después de haberse determinado el punto inicial.

7.2.1. Función __init__(self, line)

Esta función tiene el mismo objetivo y la misma implementación que la función `__init__()`, de la clase `DrawLine`.

```
def __init__(self, line):
    # INICIALIZACION DE LA CLASE. RECIBE COMO PARAMETRO UN OBJETO DE CLASE DrawLine Y EL PUNTO INICIAL DE
    # LA LINEA DEFINIDA EN LA FUNCION PRINCIPAL:
    self.line = line # EL PUNTO INICIAL DE LA FUNCION PRINCIPAL SE ALMACENA COMO UNA VARIABLE DE LA CLASE.
    self.xs = list(line.get_xdata()) # OBTIENE LA COORDENADA X DEL PUNTO RECIBIDO
    self.ys = list(line.get_ydata()) # OBTIENE LA COORDENADA Y DEL PUNTO RECIBIDO
    # CONECTA EL PUNTO CON LA FIGURA EN LA INTERFAZ GRAFICA MEDIANTE EVENTOS DE OBTURACION DEL RATON EN LOS EJES
    # DE LA FIGURA:
    self.cid = line.figure.canvas.mpl_connect('button_press_event', self)
```

7.2.2. Función __call__(self, event)

La función especial `__call__()` se encarga del manejo de los eventos de obturación del ratón. Recibe como parámetros el objeto de clase `DrawLine` y el evento del ratón. Después de recibir el evento de obturación del ratón, la función crea un cuadro de diálogo donde pide confirmación del usuario del punto seleccionado. Si el usuario confirma el punto, la función lo almacena en dos variables globales, una para la coordenada *x* y otra para la coordenada *y*. Luego desconecta los eventos del ratón de la figura y envía el comando al controlador de posición, para mover el GPR a la posición inicial.

```
def __call__(self, event):
    print('click', event)
    # ESCONDE LA VENTANA PRINCIPAL DE LA INTERFAZ GRAFICA TKINTER, UTILIZADA PARA LOS CUADROS DE DIALOGO
    root = tkinter.Tk()
    root.withdraw()

    # DESPLIEGUE DE UN CUADRO DE DIALOGO PREGUNTANDO SI LA POSICION SELECCIONADA SERA LA POSICION INICIAL DE LA
    # TRAYECTORIA:
    responder = messagebox.askokcancel("Python",
                                        "Ubicar Posicion Inicial \n x: "+str(round(event.xdata,2))+"
                                         , y: "+str(round(event.ydata,2)))
    if responder == True:
        Var.setXinit=round(event.xdata,2) # ALMACENA LA COORDENADA X DE LA POSICION INICIAL SELECCIONADA
        Var.setYinit=round(event.ydata,2) # ALMACENA LA COORDENADA Y DE LA POSICION INICIAL SELECCIONADA
        self.line.figure.canvas.mpl_disconnect(self.cid) # DESCONECTA LOS EVENTOS DEL RAON DE LA FIGURA
        # CREA LA INSTRUCCION PARA IR AL PUNTO INICIAL Y ENVIA LA INSTRUCCION AL CONTROLADOR DE POSICION PARA UBICAR
        # EL GPR EN LA POSICION INICAL DE LA TRAYECTORIA
        #linetosend = "G0 X %d Y %d \n" % (round(10 * Var.setXinit, 2), round(10 * Var.setYinit, 2))
        Var.Ctrl_Pos.write(linetosend.encode('utf-8'))
```

7.3. Función preview_trajectory(x0, y0, xf, yf, dx)

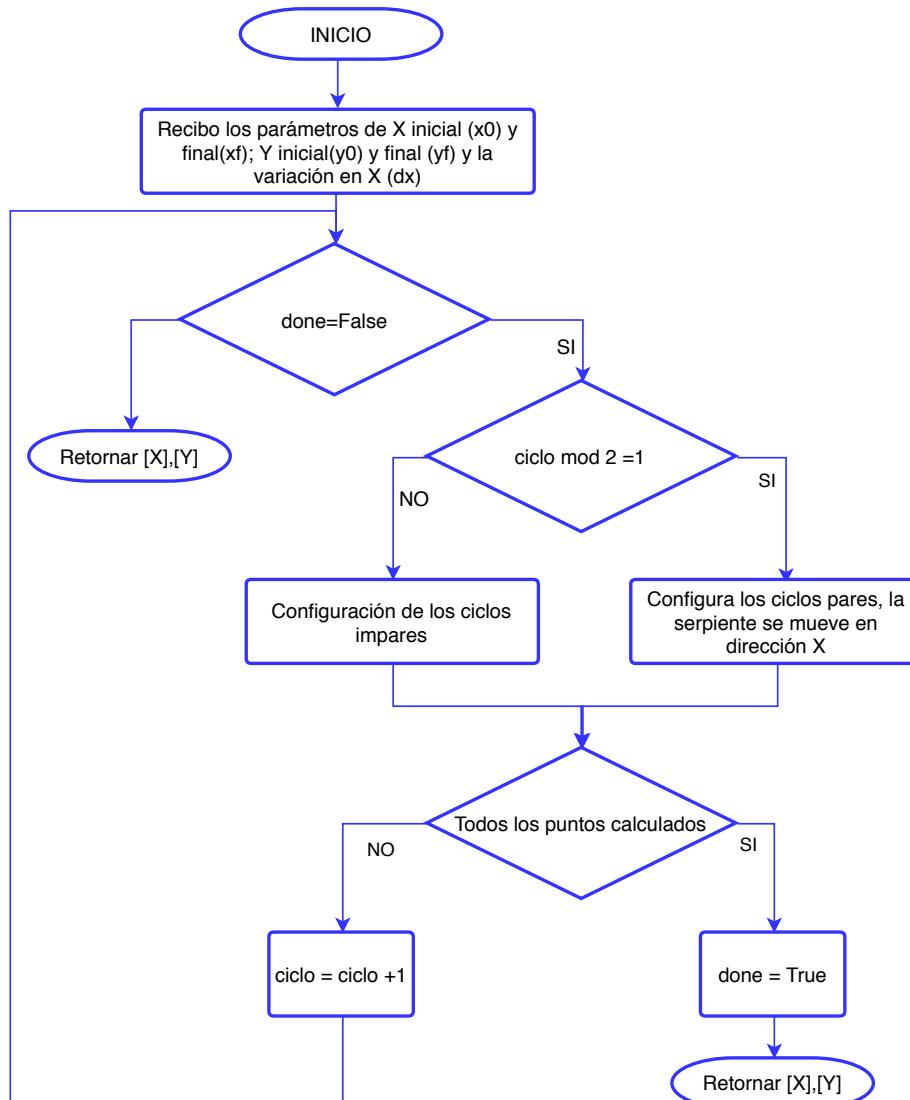
Esta función permite graficar el trayecto en forma de serpiente cuando el usuario selecciona la opción en la interfaz. Esta función recoge los valores de las posiciones iniciales y finales en las coordenadas X,Y y la separación entre las rectas de la serpiente.

```
def preview_trajectory(x0, y0, xf, yf, dx):
    # ESTA FUNCION PERMITE PREVISUALIZAR LA TRAYECTORIA EN FORMA DE SERPIENTE CONFIGURADA. RECIBE COMO PARAMETROS
    # LOS
    # DATOS DE CONFIGURACION DE LA TRAYECTORIA EN FORMA DE SERPIENTE. LA FUNCION RETORNA LOS ARRGLOS DE LAS
    # COORDENADAS
    # QUE MARCAN LA TRAYECTORIA QUE SEGUIRA EL GPR.
    x0 = x0/10; y0 = y0/10; xf = xf/10; yf = yf/10; dx = dx/10; # CONVERSION DE mm A cm PARA LA GRAFICA
    x = [-x0] # FORMA EL ARREGLO DE LAS COORDENADAS X, CON EL PRIMER DATO, QUE DEFINIRA LA TRAYECTORIA A VISUALIZAR
    y = [-y0] # FORMA EL ARREGLO DE LAS COORDENADAS Y, CON EL PRIMER DATO, QUE DEFINIRA LA TRAYECTORIA A VISUALIZAR
    # INICIA DOS BANDERAS, UNA PARA VERIFICAR EN QUE CICLO SE ENCUENTRA Y OTRA PARA SABER SI YA ACABO DE ENVIAR LOS
    # COMANDOS PARA LA TRAYECTORIA AL CONTROLADOR DE POSICION:
    cicle_num = 1
    done = False
    # MIENTRAS HAYA DATOS QUE CALCULAR PARA PREVISUALIZAR LA TRAYECTORIA COMPLETA, CALCULA Y ALMACENA LAS
    # COORDENADAS:
```

```

while not done:
    # CONFIGURACION DEL SIGUIENTE PUNTO. EN LOS CICLOS PARES (2,4,6,...), LA 'SERPIENTE' SE MUEVE EN DIRECCION X.
    # EN LOS CICLOS IMPARES (1,3,5,...), LA 'SERPIENTE' SE MUEVE EN DIRECCION Y. A CONTINUACION SE DEFINE EL
    # CAMBIO DE DIRECCION PARA EL SIGUIENTE CICLO:
    if cicle_num % 2 == 1:
        x = x + [x[-1]]
        if y[-1] == -yf:
            y = y + [-yf]
        else:
            y = y + [-y0]
    else:
        next_x = x[-1] -dx
        x = x + [next_x]
        y = y + [y[-1]]
    # SI SE HA HAN CALCULADO TODOS LOS PUNTOS SE LEVANTA LA BANDERA done PARA EVITAR LA EJECUCION DEL SIGUIENTE
    # CICLO:
    if x[-1] == -xf and y[-1] == -yf:
        done = True
    # DE LO CONTRARIO SE AUMENTA EL CONTEO DE CICLOS Y SE CALCULA NUEVAMENTE
    else:
        cicle_num = cicle_num + 1
return x,y

```

Figura 13: Diagrama de Flujo para la función *preview_trajectory()*

7.4. Función `createWigetPlot(self)`

Esta función permite limpiar la pantalla de la interfaz gráfica donde se muestra el trayecto seleccionado por el usuario y el recorrido actual del posicionador. Esta función es ejecutada cada vez que se hace un cambio en el tipo de trayecto.

```
def createWigetPlot(self):
    self.dpi = 70 # PUNTOS POR PULGADA
    # SE DEFINE LA FIGURA CON EL TAMAÑO Y EL COLOR DE FONDO:
    self.fig = Figure((5.0, 9.0), dpi=self.dpi, facecolor=(0.94, 0.94, 0.94), edgecolor=(0, 0, 0))
    self.axes = self.fig.add_subplot(111)
    # SE ASOCIA STATIC CANVAS A LA FIGURA CREADA
    self.static_canvas = FigureCanvas(self.fig)
    self.static_canvas.setParent(self)
    # A LA FIGURA SE LE DEFINE CON MARGENES PEQUEÑAS:
    self.fig.tight_layout()
    # AL LAYOUT DE LA INTERFAZ GRAFICA SE LE AGREGA EL WIDGET DE LA FIGURA EN CANVAS:
    self.layout_grafica.addWidget(self.static_canvas)
    self.layout_grafica.setStretchFactor(self.static_canvas, 1)
    self.setLayout(self.layout_grafica)

    # CREACION DE LAS MARGENES RECTANGULARES EN LOS EJES DE LA GRAFICA:
    rect = patches.Rectangle((Var.plotX[0]+Var.plotMargin, Var.plotY[0]+Var.plotMargin),
                            Var.plotX[1]-Var.plotX[0]-2*Var.plotMargin,
                            Var.plotY[1]-Var.plotY[0]-2*Var.plotMargin,
                            linewidth=2, edgecolor='r', facecolor='none')
    # Add the patch to the Axes

    # SE DEFINEN LOS VALORES Y DIMENSIONES DE LAS FIGURAS EN EL CANVAS:
    Var.static_ax = self.static_canvas.figure.gca()
    Var.static_ax.set_xticks(np.arange(Var.plotX[0], 10, 10))
    Var.static_ax.set_yticks(np.arange(Var.plotY[0], 10, 10))
    Var.static_ax.set_xlim(Var.plotX[0], Var.plotX[1])
    Var.static_ax.set_ylim(Var.plotY[0], Var.plotY[1])
    # SE ANADEN LAS MARGENES CREADAS A LA FIGURA:
    Var.static_ax.add_patch(rect)
    # SE ANADE LA GRILLA A LA FIGURA Y SE DEFINE LA RELACION DE ASPECTO ENTRE LOS EJES:
    Var.static_ax.grid()
    Var.static_ax.set_aspect('equal')
```

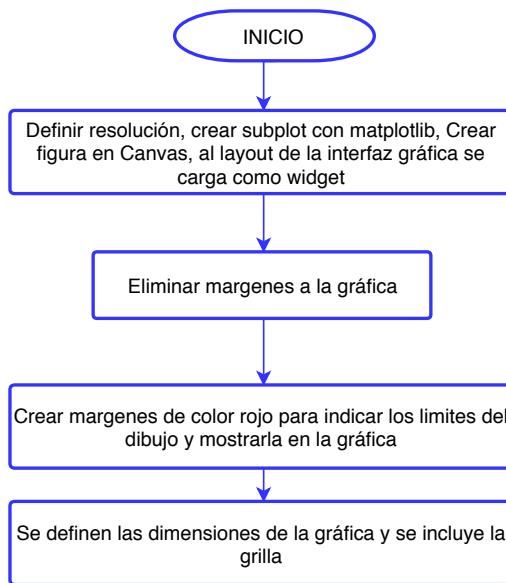


Figura 14: Diagrama de Flujo para la función `createWigetPlot()`

7.5. Función drawPoint(self,x,y)

El uso de esta función permite mostrarle al usuario el recorrido en tiempo real del posicionador, la figura dibujada en la interfaz es de color rojo para resaltar el trayecto configurado del trayecto recorrido.

```
def drawPoint(self,x,y):
    # ESTA FUNCION SE UTILIZA PARA MOSTRAR EN LA GRAFICA LA POSICION ACTUAL DEL GPR, REPORTADA EN TIEMPO REAL.
    Var.CurrentX.extend([x]) # CONCATENA LA NUEVA COORDENADA EN X CON LAS ANTERIORES
    Var.CurrentY.extend([y]) # CONCATENA LA NUEVA COORDENADA EN Y CON LAS ANTERIORES
    line, = Var._static_ax.plot(Var.CurrentX, Var.CurrentY, color='red', lw=2) # DEFINE LA LINEA A DIBUJAR
    line.figure.canvas.draw() # DIBUJA LA LINEA DEFINIDA
```

8. Configuración previa y depuración de errores del controlador de posición

8.1. Configuración de GRBL v1.1 en la tarjeta xPro

La tarjeta xPro [8] gobierna el sistema de control posición. Esta tarjeta utiliza el software GRBL [1] para enviar y recibir las instrucciones sobre la posición del GPR. La versión de firmware de GRBL que tiene instalada la tarjeta es la versión 1.1, esto se puede comprobar con el mensaje de respuesta de la tarjeta al comando \$I, como se muestra en la figura 15.

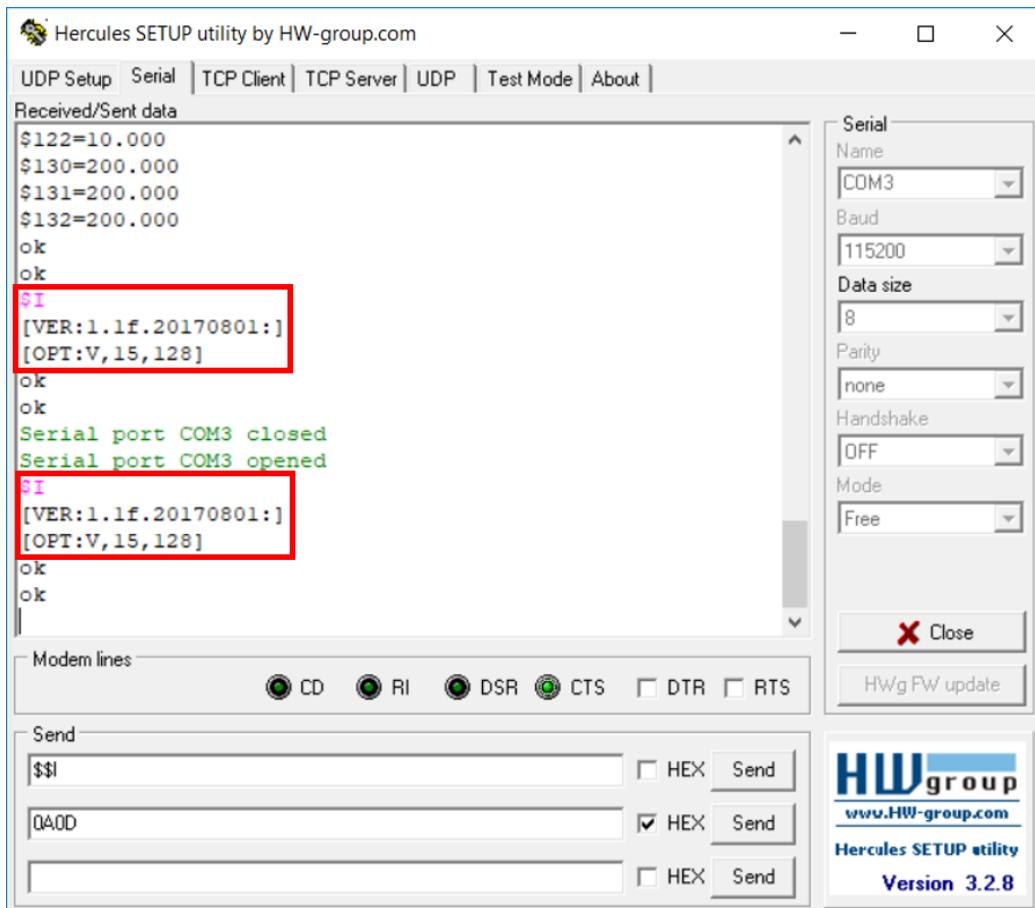


Figura 15: Comprobación de la versión de GRBL instalada

La tarjeta utiliza la configuración de GRBL de la tabla 1. En la tabla se da una descripción acerca de cada parámetro de configuración y como se afecta el funcionamiento con los valores que llegan. Para mayor información sobre la configuración, se recomienda visitar la referencia [1].

Tabla 1: Configuración de GRBL

Parámetro	Descripción
\$0=10	Tiempo de pulso, [μs]. El sistema de motores paso a paso se maneja por medio de los pulsos enviados por la tarjeta controladora a los motores. Entre más cortos sean los pulsos, mejor será el desempeño, ya que la probabilidad que se sobrepongan las instrucciones es menor. El ancho mínimo de los pulsos lo determinan los motores. Para el motor NEMA-23 del GPR, comprobamos que este parámetro configurado en 10 μs da un buen funcionamiento.
\$1=255	Tiempo de espera de pasos, [ms]. Cuando los motores completan un movimiento completo y se detienen, GRBL deshabilita los motores por el tiempo indicado en este parámetro. Si este parámetro se define como 255, los motores no se deshabilitarán nunca.
\$2=0	Inversión de pulso del puerto, Z:{0,7}. Esta variable de configuración permite cambiar el comportamiento del pulso que controla a los motores. Por defecto, los pulsos de los motores están configurados para empezar en voltaje bajo e ir a voltaje alto para después voltaje bajo por el tiempo definido en \$0. En general este parámetro no debe cambiarse. Esta variable de configuración toma valores del 0 al 7, para invertir el pulso para el motor del eje deseado. En [1] se puede encontrar una tabla para seleccionar la inversión del pulso para el motor deseado.
\$3=0	Inversión de la dirección del puerto, Z:{0,7}. Esta variable de configuración permite cambiar la dirección en la que se mueven los motores sobre el eje. Por defecto, GRBL asume que los motores se mueven en dirección positiva si se envía la instrucción sin signo menos y en dirección negativa con el signo menos. Para el GPR este parámetro se deja en cero ya que la dirección de los motores funciona de la forma deseada. Para cambiar la dirección de algún eje se debe configurar este parámetro del mismo modo que el parámetro \$2=0
\$4=0	Inversión de habilitar pulsos, Z:{0,1}. Por defecto, la terminal que habilita los motores está en voltaje alto para desactivar los motores y en voltaje bajo para habilitarlos. Este parámetro no es de mucha importancia para el GPR, ya que los motores están habilitados todo el tiempo. Para conservar la configuración por defecto, el parámetro para el GPR se define como 0.
\$5=0	Inversión de la terminal de límites, Z:{0,1}. Por defecto, la terminal a la cual se conectan los interruptores de final de carrera se mantienen en voltaje alto mediante una resistencia de pull-up interna. Cuando la terminal presenta voltaje bajo, GRBL interpreta que el límite se ha alcanzado. Para obtener el comportamiento inverso, esta variable debe modificarse a 1.
\$6=0	Inversión de sonda, Z:{0,1}. Este parámetro es útil en los equipos que manejan movimiento en el eje Z. La sonda es un equipo para calibrar con alto grado de precisión el eje Z. Con este parámetro se invierte el comportamiento de esta terminal, de forma similar al funcionamiento de \$5=0.
\$10=0	Reporte de estado, Z:{0,2}. Este parámetro determina que datos se reportan ante el requerimiento de reporte por el usuario con el mensaje "?". Por defecto, en la versión 1.1 de GRBL se reporta el estado del control de posición y la posición; los datos adicionales se reportarán solo cuando se presente un cambio con respecto al último reporte. Configurado en 0, el controlador de posición reporta la posición de trabajo es decir, la posición tomando en cuenta los offset de coordenadas estipulados con los comandos G. Configurado en 1, el controlador de posición reporta la posición de máquina es decir, la posición en coordenadas absolutas. Configurado en 2, el controlador de posición reporta la cantidad de bytes o bloques de datos que se encuentran en el buffer de entrada.

Tabla continúa en la siguiente página

Parámetro	Descripción
\$11=0.010	Desviación de la juntura, [mm]. Este parámetro determina lo utiliza el control de aceleración para lograr que los motores no pierdan pasos al realizar una curva. Si un motor pierde pasos, la posición que reporta el controlador de posición será diferente a la posición real; por esta razón es muy importante que no se pierdan pasos. Cuando este parámetro se configura con cantidades pequeñas, en las esquinas, los motores desacelerarán más suavemente hasta alcanzar una velocidad reducida donde se garantiza que no se pierden pasos. Cuando este parámetro se configura con cantidades grandes, en las esquinas, los motores realizarán el movimiento de forma rápida sin desacelerar mucho pero es posible que se pierdan pasos. Este parámetro se utiliza con el valor por defecto que da GRBL, después de comprobar que con diferentes trayectorias y velocidades no se pierden pasos. Para mayor información sobre este parámetro se puede consultar [2]
\$12=0.002	Tolerancia de arco, [mm]. Utilizando los comandos G2 y G3, es posible crear trayectorias en forma de arcos y círculos. La implementación de estos comandos en GRBL se basa en trayectoria en línea recta muy pequeñas. Este parámetro determina que tan fiel es el seguimiento de las rectas al arco. Este parámetro se utiliza con el valor por defecto que da GRBL, después de comprobar que con diferentes trayectorias y velocidades que los motores son capaces de seguir el arco requerido. Si para cierta trayectoria circular se comprueba que no se sigue el arco fielmente es recomendable hacer este valor más pequeño, teniendo en cuenta que al hacerlo la cantidad de instrucciones a incrementar notablemente.
\$13=0	Reporte en pulgadas, Z:{0,1}. Este parámetro define en que unidades se entrega la posición que se reporta y sobre qué unidades se configuran los demás parámetros relacionados con distancias. Configurado en 0, la máquina trabajará en milímetros. Configurado en 1, la máquina trabajará en pulgadas.
\$20=1	Límites suaves, Z:{0,1}. Este parámetro permite habilitar los límites suaves del controlador de posición. Los límites suaves evitan que el GPR salga del dominio definido. Cuando una instrucción indica al GPR que vaya a una posición fuera del dominio, el controlador de posición entra en estado de alarma e indica que requiere reiniciar la tarjeta controladora. La calibración de coordenadas no se pierde si se violan los límites suaves.
\$21=1	Límites duros, Z:{0,1}. Este parámetro permite habilitar los límites duros del controlador de posición. Los límites duros tienen el mismo propósito que los límites suaves: detener los motores si el GPR sale del dominio. Los límites duros están definidos por los interruptores de final de carrera. A diferencia de los límites suaves, cuando se violan los límites duros se pierde la calibración de coordenadas ya que no se puede saber si los motores perdieron pasos al encontrarse con los interruptores de final de carrera.
\$22=1	Ciclo de homing, Z:{0,1}. Este parámetro permite habilitar el homing, para calibrar las coordenadas del GPR usando el comando \$H.
\$23=0	Inversión de la dirección de homing, Z:{0,7}. Por defecto, el ciclo de homing se hace hacia la dirección positiva de los ejes. Con este parámetro se puede cambiar la dirección en la que se realiza. Para cambiar la dirección de algún eje se debe configurar este parámetro del mismo modo que el parámetro \$2=0
\$24=200	Velocidad de homing, [mm/min]. El ciclo de homing consiste en dos movimientos: primero uno rápido para buscar los interruptores de final de carrera y luego uno más lento que calibra con precisión la posición. Este parámetro define la velocidad del segundo movimiento.
\$25=400	Velocidad de búsqueda de homing, [mm/min]. Este parámetro define la velocidad de búsqueda de los interruptores de final de carrera, el primer movimiento del homing.
\$26=250	Rebote de homing, [ms]. Este parámetro define un tiempo de espera para tomar la señal de los interruptores de final de carrera. En ocasiones, los interruptores de carrera generan un rebote que son ruido para la tarjeta controladora. Para mitigar el efecto de este ruido se da un retardo a la captura de la señal, definido por el tiempo configurado en este parámetro.

Tabla continúa en la siguiente página

Parámetro	Descripción
\$27=50	Margen de seguridad del homing, [mm]. Generalmente, los límites duros y el homing comparten los mismos interruptores de final de carrera. Para no activar un límite duro sin intención, el controlador de posición utiliza un margen de seguridad en donde ubicará al GPR después del ciclo de homing. El margen de seguridad lo define este parámetro.
\$30=1000	Máxima velocidad del eje de los motores, [RPM]. Este parámetro define la cantidad máxima de revoluciones por minuto del motor. Se configuró con el valor por defecto de GRBL y se hicieron pruebas que determinaron que está bien configurado.
\$31=0	Mínima velocidad del eje de los motores, [RPM]. Este parámetro define la cantidad mínima de revoluciones por minuto del motor. Se configuró con el valor por defecto de GRBL y se hicieron pruebas que determinaron que está bien configurado.
\$32=0	Modo laser, Z:{0,1}. Este modo no es usado en el GPR así que se dejó desactivado.
\$33=0	Frecuencia del PWM, [Hz]. Al definirlo como 0, se ignora este parámetro.
\$100=26.8 \$101=26.8 \$102=26.8	Pasos por milímetro del eje [X,Y,Z], [pasos/mm]. Estos parámetros definen la distancia que el controlador entiende por cada paso del motor. La distancia por paso depende del motor, los piñones y las correas instadas. El valor 26.8 pasos/mm se tomó basado en pruebas, calibrando el movimiento con mediciones hechas con una cinta métrica y un calibrador milimétrico.
\$110=2000 \$111=2000 \$112=2000	Velocidad máxima del eje [X,Y,Z], [mm/min]. Este valor indica la velocidad máxima a la cual se desplaza el GPR sobre los ejes. La velocidad máxima limita la carga sobre los motores, los piñones y la potencia del motor. Las pruebas que se hicieron a la velocidad de 2000 mm/min mostraron resultados satisfactorios y por eso se tomó como la velocidad máxima.
\$120=100 \$121=100 \$122=100	Aceleración máxima del eje [X,Y,Z], [mm/s²]. Estos parámetros definen la aceleración máxima del GPR. Entre más bajo sea el valor será más fácil seguir la posición al controlador, ya que con menores aceleraciones es menos probable que el motor pierda pasos. La pruebas que se hicieron con este valor configurado en 100 mm/s ² dieron buenos resultados.
\$130=900 \$131=2200 \$132=200	Distancia máxima del eje [X,Y,Z], [mm]. Estos parámetro definen las dimensiones del dominio donde se moverá el GPR. Se definieron a partir de las dimensiones de la caja de arena.

La versión de GRBL instalada, tiene una modificación en la rutina de homing frente a la versión por defecto. La modificación se presenta en archivo `config.h` donde se cambia la opción para que se realice primero el homing del eje *x* y luego el homing del eje *y*. La versión por defecto se encuentra disponible en <https://github.com/gnea/grbl> y la versión modificada se encuentra en <http://bit.ly/2vHMBTc>. Más adelante en este manual, se presenta como se modificó el archivo `config.h`.

8.2. Actualización del firmware GRBL

Al trabajar con el controlador de posición, es posible encontrar que se encuentre desactualizado de la última versión de GRBL. Este fue el caso que se presentó y donde los autores tuvieron que enfrentarse antes de depurar los errores del controlador de posición. Al momento de empezar a trabajar con el controlador de posición la versión GRBL era la 1.1, al momento de escribir este documento sigue la misma versión.

Para comprobar qué versión de GRBL tiene el controlador de posición a lo hora de trabajar con este se puede utilizar el comando `$I`, como se muestra en la figura 15. La última versión de GRBL se puede consultar en <https://github.com/gnea/grbl>. Para conocer si la tarjeta es compatible con la última versión, es conveniente revisar la página de la tarjeta xPro <https://github.com/gnea/grbl>.

8.2.1. Modificación del archivo de configuración de GRBL v1.1

Para obtener el funcionamiento apropiado del controlador de posición, fue necesario modificar el archivo de configuración de GRBL. El lenguaje de programación en el cual está desarrollado GRBL es C++; al ser un lenguaje de alto nivel facilita realizar modificaciones.

Para adaptar la configuración, se modificó archivo `config.h`. En la configuración por defecto, el homing está definido de la siguiente forma: primero en el eje *z* y luego en los ejes *x* y *y* simultáneamente. Como en el caso del GPR desarrollado en la universidad no cuenta con eje *z*, solo se modificaron las dos coordenadas ya que se estaban presentando errores en el homing. Se modificó el archivo para encontrar primero la referencia del eje *x* y luego la referencia del eje *y*.

A continuación se muestra como se definieron las líneas 90 a la 115 del archivo `config.h`. No se realizó ninguna modificación de más al archivo de configuración:

```
// Define the homing cycle patterns with bitmasks. The homing cycle first performs a search mode
// to quickly engage the limit switches, followed by a slower locate mode, and finished by a short
// pull-off motion to disengage the limit switches. The following HOMING_CYCLE_x defines are executed
// in order starting with suffix 0 and completes the homing routine for the specified-axes only. If
// an axis is omitted from the defines, it will not home, nor will the system update its position.
// Meaning that this allows for users with non-standard cartesian machines, such as a lathe (x then z,
// with no y), to configure the homing cycle behavior to their needs.
// NOTE: The homing cycle is designed to allow sharing of limit pins, if the axes are not in the same
// cycle, but this requires some pin settings changes in cpu_map.h file. For example, the default homing
// cycle can share the Z limit pin with either X or Y limit pins, since they are on different cycles.
// By sharing a pin, this frees up a precious IO pin for other purposes. In theory, all axes limit pins
// may be reduced to one pin, if all axes are homed with separate cycles, or vice versa, all three axes
// on separate pin, but homed in one cycle. Also, it should be noted that the function of hard limits
// will not be affected by pin sharing.
// NOTE: Defaults are set for a traditional 3-axis CNC machine. Z-axis first to clear, followed by X & Y.
// #define HOMING_CYCLE_0 (1<<Z_AXIS) // REQUIRED: First move Z to clear workspace.
// #define HOMING_CYCLE_1 ((1<<X_AXIS)|(1<<Y_AXIS)) // OPTIONAL: Then move X,Y at the same time.
// #define HOMING_CYCLE_2 // OPTIONAL: Uncomment and add axes mask to enable

// NOTE: The following are two examples to setup homing for 2-axis machines.
// ***MODIFICACION para el proyecto de desminado humanitario. El ciclo de homing se hace primero para el eje X
// y luego para el eje Y ***
// #define HOMING_CYCLE_0 ((1<<X_AXIS)|(1<<Y_AXIS)) // NOT COMPATIBLE WITH COREXY: Homes both X-Y in one cycle.

#define HOMING_CYCLE_0 (1<<X_AXIS) // COREXY COMPATIBLE: First home X
#define HOMING_CYCLE_1 (1<<Y_AXIS) // COREXY COMPATIBLE: Then home Y
```

8.2.2. Instalación del firmware en la tarjeta xPro

Para la instalación del firmware en la tarjeta se requieren los programas de software: Arduino IDE [5] y XLoader [6].

Una vez se tiene el archivo `config.h` modificado para hacer el homing de la manera requerida, se debe compilar la versión de GRBL para obtener un archivo `.hex` que se utiliza para instalar la versión del firmware en la tarjeta xPro. La compilación se hace por medio de Arduino IDE. Para compilar la versión de GRBL se debe agregar la carpeta de GRBL modificada a la carpeta de bibliotecas de Arduino IDE (ver Figura 16). Posteriormente se debe compilar el sketch y obtener el archivo `.hex`, que se instala en la tarjeta xPro por medio del software XLoader (ver Figura 17). Una vez se sube correctamente el archivo a la tarjeta, se puede verificar que la instalación del firmware es correcta enviando el comando `$I` por el puerto serial como se muestra en la Figura 15.

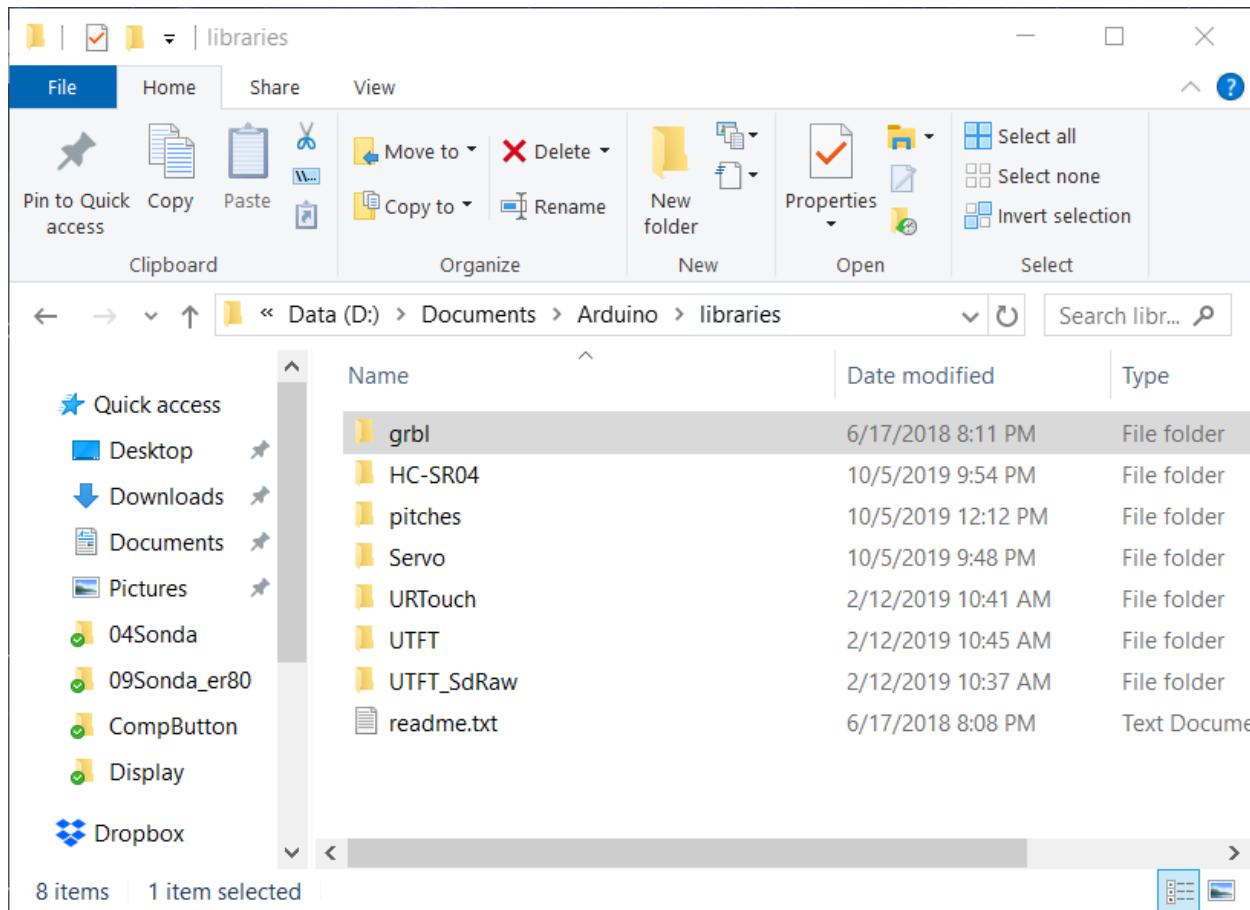


Figura 16: Biblioteca de GRBL agregada al entorno de Arduino IDE

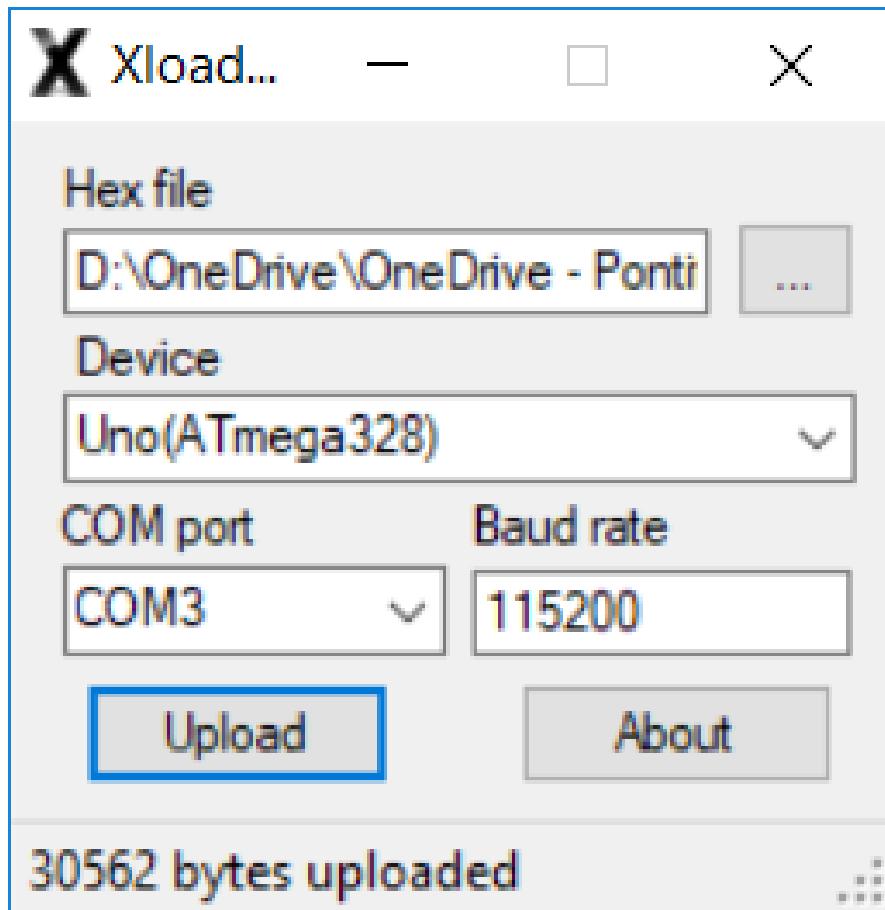


Figura 17: Interfaz XLoader que carga el frimware en la tarjeta xPro

8.3. Depuración del error en el final de carrera del controlador de posición

8.3.1. Descripción del error

Antes de realizar el programa que describe el presente manual, el controlador de posición presentaba un problema en la calibración de coordenadas en el eje y . Cuando se obturaba el interruptor de final de carrera del eje y , el controlador de posición no detenía el movimiento del GPR, sin embargo físicamente el GPR no podía seguir avanzando. Esto no solo causaba errores respecto a la calibración, sino que causaba deterioro en los motores debido a que generaban un torque muy alto al estar bloqueada su trayectoria. A continuación se describe como se mitigó el error.

8.3.2. Configuración inicial de la tarjeta xPro

Inicialmente la tarjeta xPro ejecutaba la versión 0.9j de GRBL y la tarjeta xPro se comportaba de forma extraña, motivo por el cual no fue posible verificar su configuración, debido a que mostraba errores en las respuestas sin importar como se enviaran los mensajes. El detalle de este error se muestra en la figura 18.

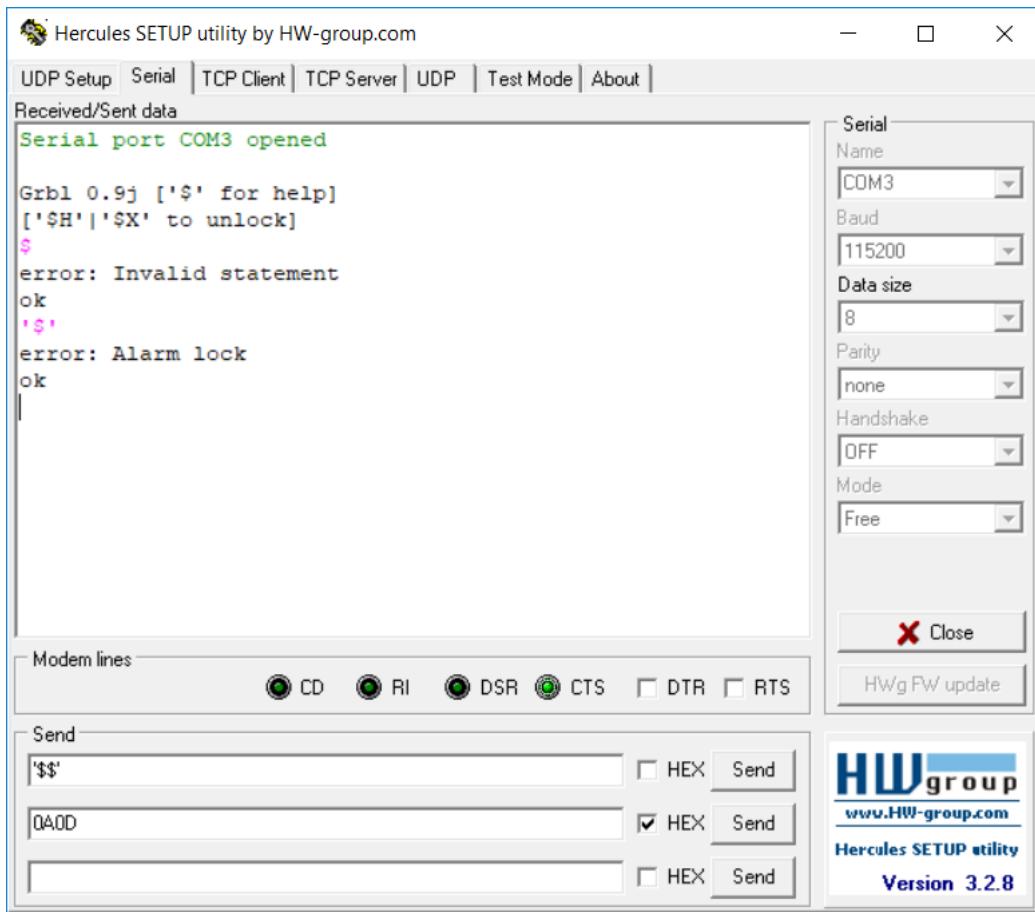


Figura 18: Errores en la configuración inicial de la tarjeta xPro

Para configurar la tarjeta xPro se decidió actualizar el firmware de GRBL a la versión 1.1. Inicialmente la tarjeta generaba un error y no respondía, así que se instaló la versión 0.9i para evaluar si desde esa versión se podía actualizar.

La configuración en la versión 0.9i era la configuración por defecto de GRBL. Con esta versión solo se hicieron pruebas básicas de movimiento, ya que el objetivo era realizar la corrección del homing desde la versión más actualizada de GRBL.

8.3.3. Cambios en la configuración de la tarjeta xPro

La tarjeta xPro se configuró con los parámetros mostrados en la tabla 1. El archivo de configuración `config.h` también se modificó, para realizar el homing con la secuencia: primero calibra el eje *x* y luego el eje *y*.

Para mitigar los errores de homing los parámetros más importantes a modificar fueron: \$24, \$25, \$26 y \$27. También fue necesario habilitar el homing con el parámetro \$23, ya que por defecto viene desactivado.

8.3.4. Cambio en el montaje físico

Una vez fue configurada la versión 1.1 de GRBL los errores en el final de carrera se redujeron notablemente, se presentó una falla a la hora de hacer pruebas. Debido a esto se implementó un filtro pasa-bajas para la conexión del interruptor de final de carrera del eje *y*, siguiendo las instrucciones de [3]. El filtro implementado tiene como frecuencia de corte 338 Hz, utilizando una resistencia de $4.7\text{ k}\Omega$ y un condensador de 100 nF. En la figura 19 se presenta el esquemático de la conexión.

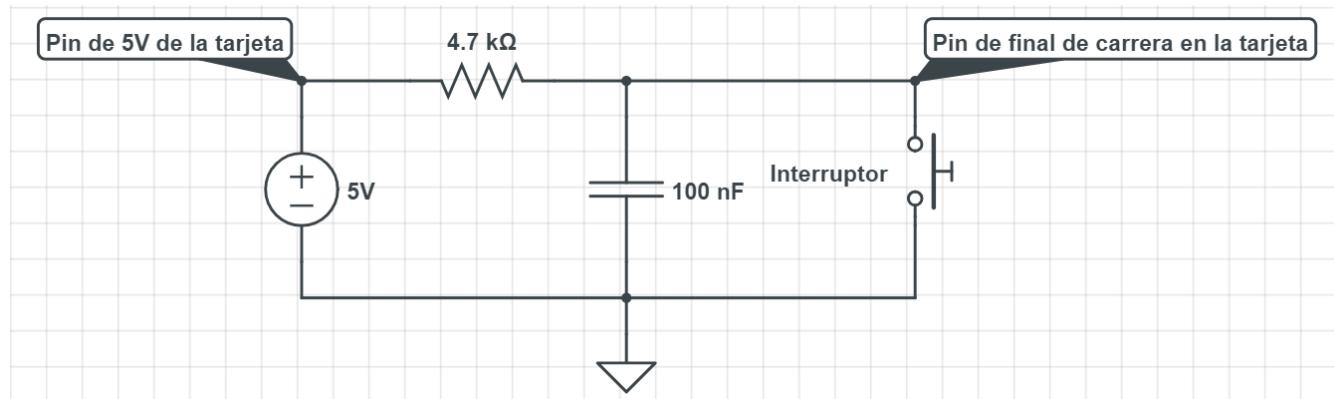


Figura 19: Esquemático para la conexión del interruptor de final de carrera

La conexión se hizo utilizando dos conectores de empalme rápido de 5 posiciones, como se muestra en la figura 20.

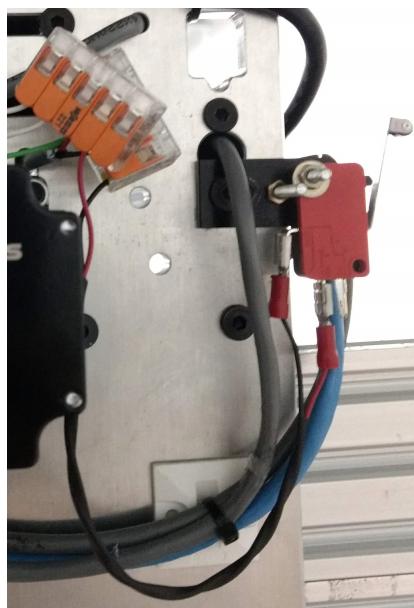


Figura 20: Interruptor de final de carrera del eje y con el filtro pasa-bajas

Con estas configuraciones y el software funcionando correctamente se da inicio al proceso de adquisición con el

sistema completo.

Referencias

- [1] Sonny Jeon aka @chamint. Grbl. <https://github.com/gnea/grbl>. Accessed: 2018-07-25.
- [2] Sonny Jeon aka @chamint. Improving grbl: Cornering algorithm. https://onehossshay.wordpress.com/2011/09/24/improving_grbl_cornering_algorithm/. Accessed: 2018-08-08.
- [3] Sonny Jeon aka @chamint. Wiring limit switches. https://github.com/gnea/grbl/wiki/Wiring-Limit-Switches/_history. Accessed: 2018-08-08.
- [4] Ansritsu. Ms20xxc vna master programming manual. <https://dlcdn-anritsu.com/en-us/test-measurement/files/Manuals/Programming-Manual/10580-00306H.pdf>. Accessed: 2018-08-01.
- [5] Arduino. Arduino ide. <https://www.arduino.cc/en/Main/Software>. Accessed: 2019-10-02.
- [6] Hobby Electronics. Xloader. <http://www.hobbytronics.co.uk/arduino-xloader>. Accessed: 2019-10-02.
- [7] QT Cross platform software development for embedded desktop. <https://www.qt.io/download>.
- [8] Spark-Concepts. xpro v3.2. <https://github.com/Spark-Concepts/xPRO>. Accessed: 2018-07-25.