CERN Summer Student Report

GUI implementation for Controlling and Monitoring of the IRRAD Shuttle System

Student: Aseel Abdulhalim

Supervisor: Blerina Gkotse

August 2021

CERN EDMS 2620441

Acknowledgment

The journey started as a student towards the professional life with the aim in mind to learn the Practical aspect of life, ended as a memorable experience.
I'm thankful for being a participant in the Summer Student Program. This opportunity provided me new experiences and knowledge which helped me to add a feather in my cap.
My special thanks to my supervisor Blerina Gkotse for providing me the most valuable guidance and the support throughout this project regardless of the remote situation. Also, I would like to thank all the IRRAD team for the support through the regular meetings. Thanks for the program coordinates and lastly thanks for all the friends all over the world for the moral support.

Table of Contents

Chapter 1

Introduction

The application is designed using the PyQt5-framework in order to generate GUI for controlling the motors of both Y and X Axis. Also monitoring the status of each motor and the security of the shuttle system. The application is written in Python including the libraries Pyserial, Telnetlib3 for the communication of X motor and Y motor respectively.
This chapter briefly explains the libraries and applications that have been used for the Shuttle-system control application.

1.1 PyQt5

The PyQt5 Library is used to generate the Graphical User Interface (GUI). PyQt is a python binding of the open-source widget-toolkit Qt, which also functions as a cross-platform application development framework [1].

1.2 Qt Designer

Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. You can compose and customize your windows or dialogs in a what-you-see-is-what-you-get (WYSI- WYG) manner, and test it using different styles and resolutions [2]. This tool provide easy visualization for Interface designing since it's a drag and drop tool, with the ability of auto-generating the Python code after finalizing the design using the following command in the Terminal: pyuic5 -x filename.ui -o filename.py

1.3 PySerial

PySerial Library is used to communicate with the X Motor. In order to control the micro-controller a binary command is sent via Ethernet port. Documentation on PySerial Library [3].

1.4 Telnetlib3

Telnet is a type of network protocol which allows a user in one computer to logon to another computer which also belongs to the same network. [4]
Telnetlib3 Library used to communicate with Y Motor. A string command is sent to the server in order to control the micro-controller and read the data. Step by step documentation on using Telnetlib3 library [5].

1.5 cx_Orcale

cx_Oracle is a Python extension module that enables Python access to Oracle Database. It is used to communicate with the database for the purpose of getting the data needed for the application functionality by sending sql-queries. [6]

Chapter 2

Design

As mentioned in the previous chapter, the designing steps have been done using the Qt designer tool. This focuses on visualizing the process of the GUI as it's going to start from the main interface and ends up with small Dialogues for displaying errors and activating the functionality of the interface.

2.1 Main Interface: User Control System Interface

2.1.1 Main Window

Main Window contains three main frames each one of them display a specific functionality in the GUI. As shown in Figure 1.

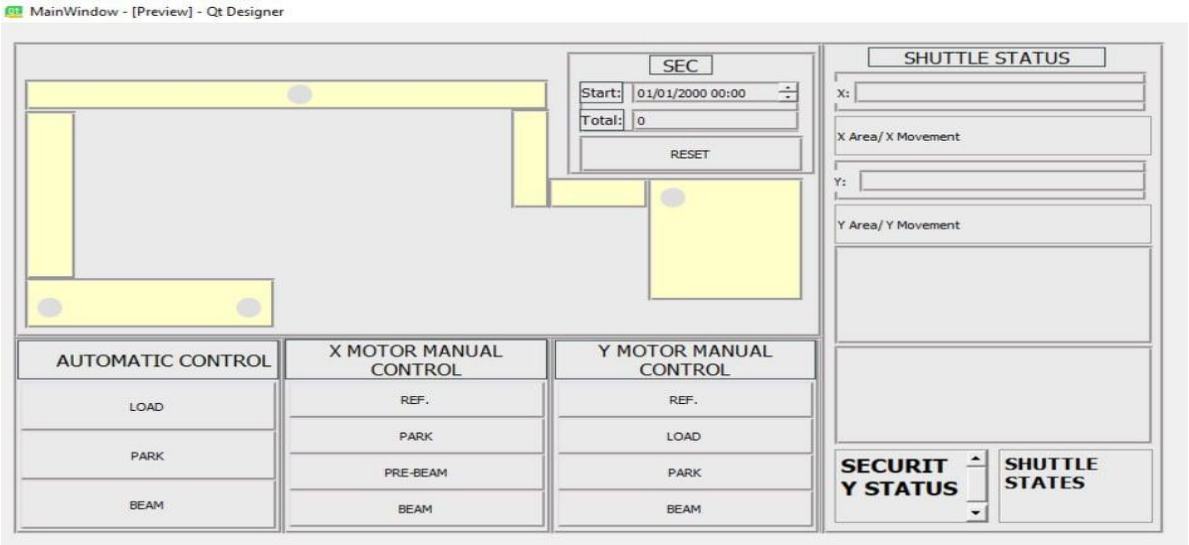


Figure 1: Main Window

2.1.2 Position Frame

Position Frame -Figure 2- designed to display the position of the shuttle system when it reaches specific point according to the function chosen from the control frame, the date time and the total whenever it reaches the position Y-beam X-beam and reset the counting button.

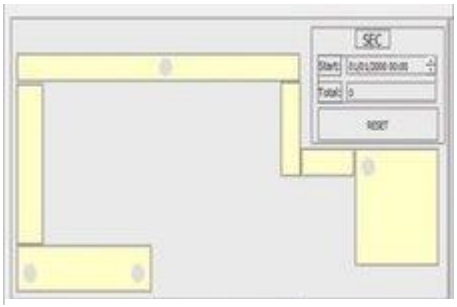


Figure 2: Position Frame

2.1.3 Shuttle Frame

Shuttle Frame -Figure 3- designed to monitor the X-Y position in mm, the area of each motor/movement and display the status of the shuttle and security.

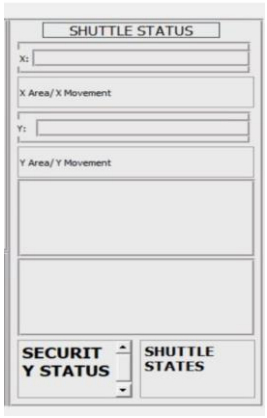


Figure 3: Shuttle Frame

2.1.4 Control Frame

Control Frame designed to control both X motor and Y motor. As shown in Figure 4, the frame is divided into three sections. Section one: **Automatic control** which automatically goes to Load/Park/Beam position. However, the movement of X-Y motor should not be performed in parallel. Section two: **X motor Manual Control** controlling only the movement of the X motor. Section three: **Y Motor Manual Control** controlling only the movement of the Y motor.

Figure 4: Control Frame

2.2 Dialogues

The main purpose of designing dialogues is to provide a sequence in the operation for the user also warning that specific operation cannot be done at specific moments.

2.2.1 Operational Dialogues

Operational Dialogues display messages to confirm the movement operation (see Figure 5).

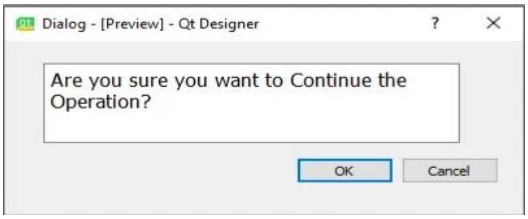


Figure 5: Operational Dialog

Chapter 3

Code Implementation

3.1 Serial Communication Module

The Serial Module manipulates the communication of the M300 microcontroller. As briefly explained to operate the micro-controllers by emitting nine bytes binary commands using PySerial Library. Inside this module Connector Class was created containing Private, Direct and Functionality commands.

3.1.1 Initialization

Initializing function -Figure 6- used to create a connection with microcontroller of a given COM with a specified timeout meaning the response of the microcontroller should be within this time otherwise the connection fail. The two bytearrays are used for debugging where the sent data and received from the command will be saved in those bytearrayes.

```
5   class Connector():
6       #Constructor
7       def __init__(self, portname):
8           self.connection = serial.Serial(portname, timeout=1)
9           self.lastRecived = bytearray(9)
10          self.lastSent = bytearray(9)
```

Figure 6: ConnectorClass __init__ method

3.1.2 Private Commands

```
#"Private" commands
def generateByteArray(self, targetAddr, instructionNo, myType, MotorNo, opb3, opb2, opb1, opb0):
def OperandsToInt(self, byteArray):
def commandSuccessful(self, array):
def GetResult(self, command):
```

Figure 7: Private Commands

generateBytearray: uses the values of the assigned bytes and calculates the checksum, after this procedure, it returns a bytearray needed to be sent to the microcontrollers.

OperandsToInt: converts the received data from the commands into integers and returns that value. When the most significant bit is 1 it calculates the two complements of the bytes.

CommandSuccessful: checks if the command is successfully being read using the received bytearray. The command is considered successful if the length of the received bytearray equals nine, the second byte equals 100, and the checksum equals the actual checksum or the result of modulo 256 of the sum in case the value is more than 256.

GetResult: is used to send the command to the microcontroller and receive the response as an integer. Only used when a numerical value is expected.

3.1.3 Direct Commands

```
#Direct Commands
def write(self, command):
def readOutput(self, NumberOfBytes=9):
```

Figure 8: Direct Commands

write : sends the command as bytearray to the microcontroller.

readOutput: returns the output array whenever the command was successfully read by the microcontroller.

3.1.4 Reading Commands

Those Commands query the microcontroller for specific information. They return the output of each command. The values will be in steps since the microcontroller operates in steps. The functions modify the steps to returns the values in millimeters/volts for particular functions.

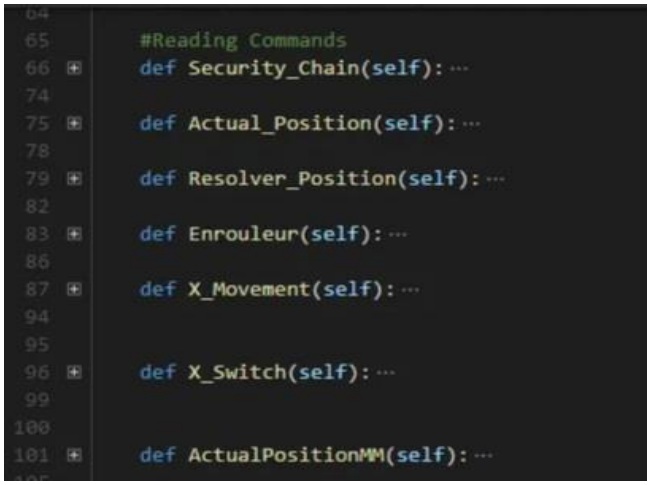


Figure 9: Reading Commands

This section discusses part of the commands as they have similar in functionality.

Security_Chain: reads the Security status where it returns Boolean value after the modification of the output. The security has two states: True when the volt is greater than 2, or False when the volt is less than 2.

Actual_Position: reads the current position of the X motor in steps.

ActualPositionMM: converts the X position from steps to millimeters and return the converted value.

X_Movement: reads the X motor state whether it's moving or not and returns a Boolean value. if the output is greater than or equal to 0 it returns False meaning the motor is not moving, or True when the output is less than 0 meaning the motor is moving.

3.1.5 Movement Commands

```
101
102     #Movement Commands
103     def x_Ref_position(self):···
106
107     def x_Park_position(self):···
110
111     def x_PreBeam_position(self):···
114
115     def x_Beam_position(self):···
118
```

Figure 10: Movement Commands

Those commands make the microcontroller move the shuttle into certain X position. The received bytearray only describes if the microcontroller received the command or not. The X-axis have three main positions Park, PreBeam and Beam. Each one has its own functions as it's shown in the Figure 10.

X_ref_poistion: moves the shuttle to the reference position of X axis. The purpose of this command is whenever we move to the park position it goes to reference then back to park to fix the error in the position.

3.2 Telnet Communication Module

The Telnet Module manages the connection using telnetlib3 which had the Telnet_Connector Class. The Class is divided into methods. Each one operates specific functionalities for controlling the Y motor or read the data by emitting a string to the driver. The emitted string corresponds to a definite command.

3.2.1 Initialization

In order to further use the class some information need to be initialized such as the Host IP address and timeout needed to define a time limit for responses. Using Telnet Library to create connection with device of defined IP address.

```
class Telnet_Connector():
    #Constructor
    def __init__(self):
        self.Host = "XXX.XXX.XXX.XXX"
        self.timeout=120
        self.tn = telnetlib.Telnet(self.Host)
```

Figure 11: Telnet_ConnectorClass __init__ function

3.2.2 Write-n-Read Commands

The methods in line 12 – 15 used to send the commands and read the output of each command.

```
11      #Write-n-Read Commands
12      def write(self, command):···
14
15      def ReadOutput(self):···
```

Figure 12: Write and Read Commands

write: send the encoded form of the string command using ASCII format for the driver to understand the command.
ReadOutput: return the decoded form of the received data from the server.

3.2.3 Reading Commands

To implement a sequence of procedure functionality some information need to be determined. Reading Commands handle collecting information form the driver.

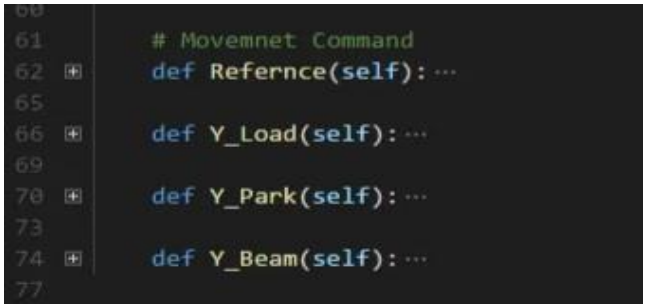
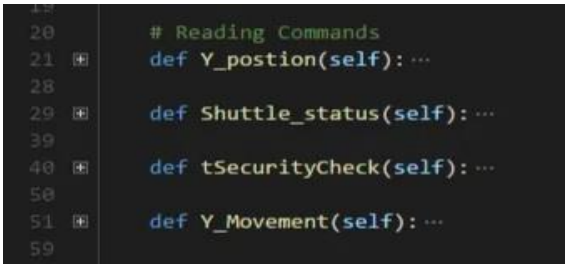


Figure 13: Reading Commands

Y_position: returns the current Y motor position after the modification.
Shuttle_status: reads the status of the shuttle and returns the Boolean value. There are two states for the shuttle, either True means the shuttle is enabled when the output is equal to one otherwise False meaning it's disabled.
tSecurityCheck: reads the security status of the Y motor then returns the Boolean value. The output equal to one whenever it is True means the Security is good, otherwise it's False.
Y_Movement: is used to check if the Y motor is moving or not and returns a Boolean value. The motor has two states moving or stopped. False when the motor is stopped and True when the motor is moving.

3.2.4 Movement Commands

These commands make the Y motor move to specific positions via sending string command. In Y axis there are three main positions Load, Park and Beam. Each position has its own specific method.



Reference: is used to evaluate the position accuracy error. Moves the motor to the reference position.

Figure 14: Movement Commands for Y motor

Y_Load: moves the shuttle to Load position in y-axis.
Y_Park: moves the shuttle to Park position in y-axis.
Y_Beam: moves to Beam position in y-axis.

3.3 Database Module

The Database Module manages the connection to the Oracle database using cx_Oracle which had the Database Class. The DatabaseClass contains get, private and read methods.

3.3.1 Initialization

Initializing function -Figure 15- used to create a connection with the database, handling errors and creating cursor object to execute queries and get results after execution.

```
class DatabaseClass():
    def __init__(self, admin=False, oracleLocation="C:\oracle\instantclient_19_11"):
        try:
            location = oracleLocation.encode('unicode_escape')
            cx_Oracle.init_oracle_client(location)
        except cx_Oracle.ProgrammingError as PE:
            print("Oracle instant client allready initialized!")
        except cx_Oracle.DatabaseError as DBError:
            print('Please download and place Oracle Instant Client on path "C:/oracle/instantclient_19_11"')
            raise Exception from DBError

        ConnectionString = '<user>/<password>@host.cern.ch:port/host.cern.ch'

        self.connection = cx_Oracle.connect(ConnectionString)
        self.cursor = self.connection.cursor()
```

Figure 15: DatabaseClass __init__ function

3.3.2 Get Methods

Those methods query the database to return the values from the specified tables according to defined conditions – Figure 16-.

get_SEC: it gets the cumulative total of proton fluence from the table.

```
# Get Methods
def get_SEC(self, timestamp):···

def get_AD6LOAD(self):···

def get_AD6PARK(self):···
```

Figure 16: Get Method

Get_AD6LOAD: executes a database query that gets the time and values of the dose rates on the Y Load position.
The same applies for **get_AD6PARK** for the Y Park position.

3.3.3 Read Methods

Those commands read the data received upon request and return a modification data to be displayed in GUI – Figure 17-.

```
# Read Methods
def Read_SEC(self,timestamp):···

def read_AD6PARK(self):···

def read_AD6LOAD(self):···
```

Figure 17: Read Methods

Read_SEC: returns the cumulative total to be displayed in the GUI.

Read_AD6PARK: returns the rate values and time where it extracts the hours, minutes and seconds from the timestamp and convert the time into seconds in purpose of finding the rate value. The exact procedure applies for **read_AD6LOAD.**

3.3.4 Private Methods

```
# Private Methods
def dictfetchall(self):···

def StringToInt(self,x):···
```

Figure 18: Private Methods

dictfetchall: it gives all the matching records in the form of list of dictionary containing key, value.

StringToInt: returns converted list of integers.

3.4 Main Project Module

This module connects all previous modules together, control the application and refresh the GUI. It contains a class that activates the main window and other specified dialogs also call the communication control modules in order to build a full function application.

Firstly importing all the needed libraries, UIs and Classes to start adding the functionality for buttons, graphs and labels. As shown in Figure 19.

```
#importing Libraries
from PyQt5 import QtWidgets, QtCore, QtGui, Qt
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5. QtCore import *
import serial, time, sys
import datetime
# Import the UI
from XMMC_Operation_Dialog import Ui_Dialog as Ui_XMMCO_Dialog
from YMMC_Operation_Dialog import Ui_Dialog as Ui_YMMCO_Dialog
from Security_Dialog import Ui_Dialog as Ui_Security_Dialog
from AC_Operation_Dialog import Ui_Dialog as Ui_AC_Dialog
from USER_CSI import Ui_MainWindow
#Importing Connector Class Functions
from ConncetorClass import Connector
from Telnet_ConnectorClass import Telnet_Connector
from DatabaseClass import DatabaseClass

```

Figure 19: Importing Libraries, UIs and Classes

Secondly creating class for each dialog in main module to call all the designed dialogs into application and initializing and adding the events of the buttons. As shown in Figure 20.

```
class XMMCOP_Dialog(QDialog):···

class YMMCOP_Dialog(QDialog):···

class ACOP_Dialog(QDialog):···

```

Figure 20: Dialogs Classes

3.4.1 Initialization

The MainWindow class contains all the functionality of the application. Firstly, calling the main window that have been designed on the first stages of the project. Secondly, setup the buttons events for X Motor Manual Control, Y Motor Manual Control and Automatic Control. Then it's needed to create a function that start a timer to refresh and update the motors position values, status of the system and the dynamics graphs -Figure 21 -.

```
45
46  class MainWindow(QMainWindow, Ui_MainWindow):
47      def __init__(self, parent=None):
48          super(MainWindow, self).__init__(parent)
49          self.setupUi(self)
50
51      # XMMC buttons for Movement
52          self.REF1.pressed.connect(lambda:self.show_XMMC_dialog(0))
53          self.xmmc_park_button.pressed.connect(lambda:self.show_XMMC_dialog(1))
54          self.pre_beam_button.pressed.connect(lambda:self.show_XMMC_dialog(2))
55          self.xmmc_beam_button.pressed.connect(lambda:self.show_XMMC_dialog(3))
56
57      # YMMC button for movement
58          self.REF2.pressed.connect(lambda: self.show_YMMC_dialog(0))
59          self.ymmc_load_button.pressed.connect(lambda: self.show_YMMC_dialog(1))
60          self.ymmc_park_button.pressed.connect(lambda: self.show_YMMC_dialog(2))
61          self.ymmc_beam_button.pressed.connect(lambda: self.show_YMMC_dialog(3))
62
63      # Automatic Control buttons for movement
64          self.Load_button.pressed.connect(lambda:self.show_Ac_dialog(0))
65          self.ac_park_button.pressed.connect(lambda:self.show_Ac_dialog(1))
66          self.ac_beam_button.pressed.connect(lambda:self.show_Ac_dialog(2))
67
68      # Reset button functionality
69          self.reset_button.pressed.connect(lambda: self.Reset_SEC())
70
71      # call timer and show graphs
72          self.Graphs_show()
73          self.start_timer(1000)
74
```

Figure 21: MainWindow __init__ function

In lines 51-54, the buttons of X Motor Manual Control (XMMC) have been setup to call the operational dialog and if the button is accepted it runs the command which moves the X motor to corresponding position. Each button is connected to the **XMoveFunction** with specific position number to define which move command should be called from the Connector Class module. As shown in figure 22.

```
# X-Axis Movement commands
def xMoveFunction(self, pos):
    if pos == 0:
        Connector("COM6").x_Ref_position()

    elif pos == 1:
        Connector("COM6").x_Park_position()


    elif pos ==2:
        Connector("COM6").x_PreBeam_position()

    elif pos ==3:
        Connector("COM6").x_Beam_position()
```

Figure 22: X Motor Movement Commands

The same concept applies for both Y Motor Manual Control (YMMC) in lines 57-60 and Automatic Control (AC) in lines 63-65.

On the same __init__ function a start timer -Figure 23- is being called for the purpose of sending signals to update the values that have been specified in **Update_Values** -Figure 24-.

```
424    # Function that trigger the update value function every one second
425    def start_timer(self,timeout):
426        self.position_timer = QtCore.QTimer(self)
427        self.position_timer.timeout.connect(self.Update_Values)
428        self.position_timer.start(timeout)
429
```

Figure 23: Start timer Function

UpdatePositionXMM: update the X-Y positon in millimeter in the UI.
Circles_pos: update the visualization of the shuttle system in the position Frame.
Circles_reset: reset the style of the origins position by checking the movement of the X motor or Y motor.
SecurityCheck: update the security state that it received from serial communication module. And enables/disables the buttons of the X Motor Manual Control - Figure 25-.

```
79
80      # update the data in the GUI
81      def Update_Values(self) :
82          self.UpdatePositionXMM()
83          self.SecurityCheck()
84          self.X_Movement_Check()
85          self.Y_Movement_Check()
86          self.tSecurityCheck()
87          self.shuttle_status_update()
88          self.circles_pos()
89          self.circles_reset()
90          self.update_DateTime()
91          self.Reset_SEC_state()
92
```

Figure 24: Update Values Function

```
133     # Security Check for the X-Axis
134     def SecurityCheck(self):
135         Security_Status = Connector("COM6").Security_Chain()
136
137         if Security_Status  == False :
138             self.Security.setText("Security! Warning")
139             self.Security.setStyleSheet("background-color: red;"
140                 "font: 16pt \"MS Shell Dlg 2\";")
141             self.xmmc_park_button.setEnabled(False)
142             self.pre_beam_button.setEnabled(False)
143             self.xmmc_beam_button.setEnabled(False)
144
145
146         elif Security_Status  == True:
147             self.Security.setText("Security! OK")
148             self.Security.setStyleSheet("background-color: green;"
149                 "font: 16pt \"MS Shell Dlg 2\";")
150             self.xmmc_park_button.setEnabled(True)
151             self.pre_beam_button.setEnabled(True)
152             self.xmmc_beam_button.setEnabled(True)
153
154         else:
155             self.Security.setText(" ")
```

Figure 25: Security Check Function

X_Movement_Check: as shown in figure 26. The function read the x movement using serial module. If True update the label to “IS MOVING”. Otherwise, it updates the label in which area in X axis using **XArea** which it read the X position then determines the area in X-axis -Figure 27-.

```
239
240     # Checking if the X Motor is moving
241     def X_Movement_Check(self):
242         X_movement = Connector("COM6").X_Movement()
243         if X_movement == False :
244             self.XArea()
245         elif X_movement == True :
246             self.area1.setText("MOTOR IS MOVING!")
247             self.area1.setStyleSheet("background-color: orange;")
248
```

Figure 26: X Movement Check Function

```
90      # function to determine and display the X position
91      def XArea(self):
92          x_pos = round(Connector("COM6").ActualPositionMM())
93          if x_pos < 3:
94              self.area1.setText("PARK")
95              self.area1.setStyleSheet("background-color: green;")
96
97          elif x_pos >= 39 and x_pos <45:
98              self.area1.setText("PRE-BEAM")
99              self.area1.setStyleSheet("background-color: green;")
100
101         elif x_pos > 60:
102             self.area1.setText("BEAM")
103             self.area1.setStyleSheet("background-color: green;")
104
105         elif x_pos == 0:
106             self.area1.setText("REFERENCE")
107             self.area1.setStyleSheet("background-color: green;")
```

Figure 27: X Area Function

The same previous procedure applies for **Y_Movement_Check, tsecurityCheck and YArea** of Y Motor using the telnet communication module.

Shuttle_status_Update: reads the state of the shuttle using telnet Communication Module. The shuttle is enabled when it's True and disabled when it's False -Figure 28-.

```
# Shuttle_Status Update
def shuttle_status_update(self):
    Shuttle_status = Telnet_Connector().Shuttle_status()
    if Shuttle_status == True:
        self.S_status.setText("Shuttle Enabled!")
        self.S_status.setStyleSheet("background-color: green;"
         "font: 16pt \"MS Shell Dlg 2\";")
    if Shuttle_status == False:
        self.S_status.setText("Shuttle NOT OK!")
        self.S_status.setStyleSheet("background-color: red;"
         "font: 16pt \"MS Shell Dlg 2\";")
```

Figure 28: Shuttle Status Update Function

circles_pos: Interactive function that activate the circle in the Interface. Each circle is define for each position. When the shuttle system reaches one of the four areas the style of corresponding circle will light green.
circles_reset: the function resetting the style of the circles when the shuttle is moving.

update_DateTime: displays the time, date, and total SEC whenever it reaches the Beam position – X-Beam, Y-Beam-. As shown in figure 29.

```
def update_DateTime(self):
    self.start.setDateTime(QtCore.QDateTime.currentDateTime())
    y_pos = Telnet_Connector().Y_postion()
    x_pos = Connector("COM6").ActualPositionMM()
    if y_pos >= 9190 and x_pos >=60:
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        sec_total = DatabaseClass().Read_SEC(timestamp)
        total = int(self.total_text.text()) + sec_total
        self.total_text.setText(str(total))
    self.start.setDisplayFormat("dd/MM/yyyy hh:mm:ss")
```

Figure 29: Update_DateTime Function

When the Reset button is pressed call **Reset_SEC:** to reset the total whenever the experiment on the beam is finished. However, the button is only activated when the shuttle is not in Beam position by using **Reset_SEC_state. CheckArea** check the area of the shuttle if it's in beam position it returns True - Figure 30-.

```
def Reset_SEC(self):
    self.total_text.setText('0')


def Reset_SEC_state(self):
    area = self.CheckArea()
    if area == True:
        self.reset_button.setEnabled(False)
    else:
        self.reset_button.setEnabled(True)

def CheckArea(self):
    y_pos = Telnet_Connector().Y_postion()
    x_pos = round(Connector("COM6").ActualPositionMM())
    if x_pos >= 60 and y_pos >=9190:
        return True
```

Figure 30: Reset Functions

Lastly, calling **Graphs_Show** function to display AD6Park and AD6Load by running **Update_AD6PARK and UpdateAD6PARK** as shown in Figure 31.
These two function get time and rate from the database then plot the rate over time – Figure 32–.

```
def Graphs_show(self):
    self.update_AD6PARK()
    self.update_AD6LOAD()
```

Figure 31: Graphs_Show Function

```
def update_AD6PARK(self):
    time,value = DatabaseClass().read_AD6PARK()
    self.AD6PARK.setBackground('w')
    pen = pyqtgraph.mkPen(color=(0, 0, 0))
    self.AD6PARK.plot(time,value, pen=pen)
    self.AD6PARK.setTitle('AD6PARK')

def update_AD6LOAD(self):
    time,value = DatabaseClass().read_AD6LOAD()
    self.AD6LOAD.setBackground('w')
    pen = pyqtgraph.mkPen(color=(0, 0, 0))
    self.AD6LOAD.plot(time,value,  pen=pen)
    self.AD6LOAD.setTitle('AD6LOAD')
```

Figure 32: Plot Functions

Chapter 4

References

1. https://www.riverbankcomputing.com/static/Docs/PyQt5/
2. https://doc.qt.io/qt-5/qtdesigner-manual.html
3. https://pythonhosted.org/pyserial/
4. https://www.tutorialspoint.com/python_network_programming/python_telnet.htm
5. https://beginnersforum.net/blog/2017/08/21/part-2-telnet-switch-using-python/
6. https://cx-oracle.readthedocs.io/en/latest/user_guide/introduction.html