

Instruction Manual

Programmable Flight Controller

Prepared for Reiland Systems

Internal Supervisor: Dr. Jason Gu

External Supervisor: Dr. Jason Rhineland

Brendon Camm
B00649176

Lucas Doucette
B00685962

Dylan Humber
B00695554

Submitted: April 10, 2017



DALHOUSIE
UNIVERSITY

This document was written in L^AT_EX. It was compiled with pdf_latex. The source of the document may be viewed at:
<https://github.com/Brendoncam/SYP/edit/master/Documents/Instruction%20Manual/InstructionMa>

Contents

1	GUI	1
1.1	Required Software	1
1.2	Basic Functionality	3
1.2.1	Home	3
1.2.2	Controller	3
1.3	Additional Resources	3
1.3.1	QtCreator Resources	3
1.3.2	PyQt5 Resources	4
1.3.3	Library and general Python Resources	4
A	GUI	5
A.1	GUI Code	5
A.2	Joystick Code	8

Abstract

The following instruction manual provides information regarding the design and usage of the flight controller. The required packages, libraries, installation procedures are outlined in the documentation. The function libraries provided are discussed in detail to ensure clarification.

1 GUI

This section gives the user instructions on how to make use of the functionalities the GUI has to offer as well as outlines how to add additional functionality if desired. It begins by explaining the required software and libraries the user must have installed in order to use the GUI. The functionality of the pushbuttons on the GUI will be explained, the section will finish off with where to find helpful resources for QtCreator, PyQt5 and the other libraries used.

1.1 Required Software

The following is a list of the required software needed to run the GUI:

- QtDesigner (Included in QtCreator5.6)
- Python3

If your system does not have QtDesigner installed follow these steps:

1. Open your preferred browser and navigate to: <https://www.qt.io/qt5-6/> and click "Download"
2. On the next page select "In-house deployment, private use, or student use" and click "Get Started"
3. On the next page select "No" and click "Get Started"
4. On the next page select "No" and click "Get Started"
5. On the next page select "Desktop/multiscreen application" and click "Get Started"
6. You should now be on a entirely different page outlining the Commercial and Open Source versions of Qt scroll down and select "Get your open source package"
7. On the next page click "Download Now" to download the installer
8. Once you have downloaded the installer run it
9. With the installer now running click "Next"
10. Enter your Qt account information, if you do not already have a Qt account you can make one within the installer enter this information and then click "Next"

11. You should now be on the Setup page on the installer click "Next"
12. Browse for an installation folder or click "Next" to use the default one provided, ensure the "Associate common file types with QtCreator" box is selected
13. On the next page click "Deselect All" at the bottom of the page and then select "Qt5.6" and "Tools" from the list. then click "Next"
14. On the next page select "I agree" and then click "Next"
15. On the next page click "Next"
16. Finally, click "Install"

If your system does not have Python3 installed follow these steps:

1. Open your preferred browser and navigate to <https://www.python.org/downloads/>
2. Click on "Python 3.x" where x is the version number. The version number is irrelevant as long as the first number is 3.
3. Scroll to the bottom of the page to "Files" and select the appropriate version for your operating system. For example, I am using a 64bit Windows OS I would click on "Windows x86-64 executable installer". Click on this to download the installer. Once it is downloaded run the installer
4. On the installer, click "Customize installation"
5. On the next page ensure every box is checked
6. On the next page ensure the following boxes are checked: "Associate files with Python", "Create shortcuts for installed applications", "Add Python to environment variables" and "Precompile standard library". Once this is done click "Install"

Once you have successfully installed QtCreator and Python3 you must install the following libraries using the "pip" command. This is done through the command window, open the command window and type, for example: "pip install PyQt5", this will install the latest version of PyQt5. Follow this process for each of the libraries listed below.

- PyQt5
- pyqtgraph
- numpy
- sip

1.2 Basic Functionality

This section outlines the functionality of the GUI. To switch pages simply click on the name of the page in the list on the right hand side of the GUI. The following sections outline the functionality of the pushbuttons on each of the two pages.

1.2.1 Home

- **Start:** Initializes communications between the Base Station and Raspberry Pi
- **Finish:** Closes the GUI

1.2.2 Controller

- **Update Axis:** Update the joystick Yaw, Pitch, Roll and Thrust inputs
- **Update Host:** Update the Host PC Name
- **Update Port:** Update the port number for the socket
- **Manual Control:** Initialize the Joystick to send manual control inputs
- **Update Connection:** Updates the Host and Port at the same time, upon completing this a new connection will be established using the new information

To see which definition is called when each of these buttons are clicked see the GUI code in Appendix A.1. How the Joystick initialization is coded can be viewed in Appendix A.2.

1.3 Additional Resources

If the user would like to add any additional functionality many excellent online resources exist to aid them through the process. Below are some of the best that were used extensively when developing the GUI. They are broken down into QtCreator resources, PyQt5 Resources and Library resources.

1.3.1 QtCreator Resources

- The best resource for QtDesigner is the one provided by Qt. This can be found at <http://doc.qt.io/qt-5/qt designer-manual.html>

1.3.2 PyQt5 Resources

- A very nice video series describing many different aspects of PyQt is provided by sentdex on YouTube. The tutorial is based on PyQt4 but the principal is still valid, the only thing to remember is that when they use the "QtGui" class in PyQt4 we use "QtWidgets" in PyQt5. To find these videos go to YouTube and type in "sentdex PyQt".
- The reference guide for PyQt5 can be found at <http://pyqt.sourceforge.net/Docs/PyQt5/>

1.3.3 Library and general Python Resources

- For additional information regarding pyqtgraph (the library that allows for live plotting) visit <http://www.pyqtgraph.org/documentation/>
- For additional information regarding anything else you're interested in for Python visit <https://docs.python.org/3/library/index.html>.

A GUI

A.1 GUI Code

Listing 1: GUI.py

```
import sys
from PyQt5 import QtCore, QtGui, QtWidgets
import numpy as np
import time
import struct
import socket
from PS4_Controller import PS4Controller as PS4
import pyqtgraph as pg
import pyqtgraph.exporters

qtCreatorFile = "GUI.ui" # Enter QtDesigner file here.
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)

class GUI(QtWidgets.QMainWindow, Ui_MainWindow, QtWidgets.QMenu):
    def __init__(self):

        super(GUI, self).__init__()
        #Qt initialization
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.page = QtWidgets.QStackedWidget()
        self.setCentralWidget(self.page)
        self.setupUi(self)
        self.setWindowTitle("Drone")
        self.setWindowIcon(QtGui.QIcon('smu.png'))

        #Networking
        self.getHost = socket.gethostname()
        self.staticPort = '1247'

        #Main Page buttons
        self.start.clicked.connect(self.connection)
        self.end.clicked.connect(self.stop)

        #Listing widget, allows for the user to select a certain page
        self.list.insertItem(0, 'Home')
        self.list.insertItem(1, 'Controller')
        self.list.currentRowChanged.connect(self.display) #Changes widget index to appropriate page

        #Controller Page
        self.axisVal.setText('1_2_3_4')
        self.hostVal.setText(self.getHost)
        self.portVal.setText(self.staticPort)
        self.axisMenu.clicked.connect(self.axisSettings) #When "Update Axis" is clicked call definition axisSettings
        self.hostMenu.clicked.connect(self.hostSettings) #When "Update host" is clicked call definition hostSettings
        self.portMenu.clicked.connect(self.portSettings) #When "Update Port" is clicked call definition portSettings
        self.updateConnect.clicked.connect(self.updateConnection) #When "Update Connection" is clicked call definition updateConnection
        self.connectPS4.clicked.connect(self.connectController) #When "Manual Control" is clicked call definition connectController

        #Live plotting Initializations
        self.initplt()
        self.plotcurve = pg.PlotDataItem()
        self.plotwidget.addItem(self.plotcurve)
        self.t = 0
        self.update1()
        self.timer = pg.QtCore.QTimer()
        self.timer.timeout.connect(self.move) # Connects a timer to the "move" definition that allows for live plotting
        self.timer.start(1000) # Poll for updates of new data ever 1000miliseconds (1 second)

    def stop(self):
        sys.exit(app.exec_())

    def connection(self):
        s = socket.socket()
        host = self.getHost
```

```

port = int(self.staticPort)
status = s.connect_ex((host,port)) #Returns 0 if connect is successful, returns errno if not
if status: # Status = errno
    self.thisworks.setText("Connection_Unsuccessful")
    self.connectionStat.setText("Communications_have_not_been_established")
else: # Status = 0
    print(status)
    self.thisworks.setText("Connection_Successful")
    self.connectionStat.setText("Communications_are_active")

def axisSettings(self):
    cont = PS4()
    #Input boxes when "Update Axis" is clicked
    text, ok = QtWidgets.QInputDialog.getText(self, 'Axis_Value[0]', 'No_Spaces')
    text1, ok = QtWidgets.QInputDialog.getText(self, 'Axis_Value[1]', 'No_Spaces')
    text2, ok = QtWidgets.QInputDialog.getText(self, 'Axis_Value[2]', 'No_Spaces')
    text3, ok = QtWidgets.QInputDialog.getText(self, 'Axis_Value[3]', 'No_Spaces')
    axis = [int(text), int(text1), int(text2), int(text3)] # Make an array of the values from the input dialogs
    self.axisVal.setText(str(axis))
    cont.axis_order = axis # Set the axis order for the controller equal to the new settings

def display(self, i):
    self.home.setCurrentIndex(i)

def hostSettings(self):
    #Input box when "Update Host" is clicked
    text, ok = QtWidgets.QInputDialog.getText(self, 'Host', 'Host_name_or_IP_address')
    newHost = str(text)
    if newHost == '':
        self.hostVal.setText(self.getHost())
    else:
        self.hostVal.setText(newHost)
    return newHost

def portSettings(self):
    #Input box when "Update Port" is clicked
    text, ok = QtWidgets.QInputDialog.getText(self, 'Port', 'Port_number')
    if ok:
        print('success')
        newPort = str(text)
        if newPort == '':
            self.portVal.setText(self.staticPort)
        else:
            self.portVal.setText(newPort)
    return int(newPort)

def connectController(self):
    new = PS4()
    cont.axis_order = self.axisSettings()
    print(str(cont.axis_order))
    new.listen() #Accept data from the manual controller, calls the listen definition from PS4Controller.py

def updateConnection(self):
    host1 = self.hostSettings()
    port1 = self.portSettings()
    s = socket.socket()
    status = s.connect_ex((host1,port1)) #Returns 0 if connect is successful, returns errno if not

    if status: #status = errno
        self.connectionStat.setText("Update_and_Connection_Unsuccessful")
        self.thisworks.setText("Update_and_Connection_Unsuccessful")
    else: #status = 0
        self.connectionStat.setText("Update_and_Connection_Successful")
        self.thisworks.setText("Update_and_Connection_Successful")

def liveData(self):
    graph_data = open('test.txt', 'r').read() # Open the text file to read in data
    lines = graph_data.split('\n') # Read in data from different lines
    xs = [] #Empty list
    ys = [] #Empty List
    for line in lines:
        if len(line)>1:
            x, y = line.split('_',') #Read in data in the form of (x ,y)
            xs.append(int(x))
            ys.append(int(y))
    return xs,ys

```

```

def initplt(self):
    self.plotwidget = pg.PlotWidget() #Initate the plotting widget field
    self.mplvl.addWidget(self.plotwidget) #Set the plotting widget field to populate the QVBoxLayout widget field
    self.plotwidget.setLabel('left', 'Altitude_[m]') # Y-Axis name
    self.plotwidget.setLabel('bottom', 'Time[s]') #X-Axis name
    self.show()

def update1(self):
    #read in the data from the liveData definition in the form of two separate lists.
    #list1 = x-values, list2 = y-values
    list1, list2 = self.liveData()
    self.plotcurve.setData(list1, list2) # Plot the data

def move(self):
    self.t+=1 #Move the data 1 spot to the right
    self.update1() # Call update1 definition to get the new data

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    main = GUI()
    main.show()
    QtWidgets.QApplication.processEvents()
    sys.exit(app.exec_())

```

A.2 Joystick Code

Listing 2: PS4Controller.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# This file presents an interface for interacting with the Playstation 4 Controller
# in Python. Simply plug your PS4 controller into your computer using USB and run this
# script!
#
# NOTE: I assume in this script that the only joystick plugged in is the PS4 controller.
#       if this is not the case, you will need to change the class accordingly.
#
# Copyright    2015 Clay L. McLeod <clay.l.mcleod@gmail.com>
#
# Distributed under terms of the MIT license.

#TODO:
#  rewrite connection for new server
#  test

import os
import pprint
import pygame
import socket
import struct
import sys

if sys.version_info[0] < 3:
    raise Exception('Lucas', 'not_compatible_with_Python_version_2')

class PS4Controller(object):
    """Class representing the PS4 controller. Pretty straightforward functionality."""

    controller = None
    axis_data = None
    button_data = None
    hat_data = None

    def __init__(self, axis_order=[1, 2, 3, 4], hostname='raspberrypi', port=2222, limits = [10, 10, 1, 10]):
        """Initialize the joystick components"""

        pygame.init()
        pygame.joystick.init()
        self.controller = pygame.joystick.Joystick(0)
        self.controller.init()
        self.hostname = hostname
        self.port = port
        self.limits = limits
        if isinstance(axis_order, list):
            self.axis_order = axis_order # For changing how controller axes are bound
        else:
            raise Exception(TypeError, 'axis_order_must_be_list.')

    def update_axes(self, axis_order):
        self.axis_order = axis_order

    def listen(self):
        """Listen for events to happen"""

        if not self.axis_data:
            self.axis_data = {0: float(0),
                              1: float(0),
                              2: float(0),
                              3: float(0),
                              4: float(-1),
                              5: float(-1)} # Added explicit number of axes to avoid waiting for input

        if not self.button_data:
            self.button_data = {}
            for i in range(self.controller.get_numbuttons()):
                self.button_data[i] = False

        if not self.hat_data:
```

```

self.hat_data = {}
for i in range(self.controller.get_numhats()):
    self.hat_data[i] = (0, 0)

# host = '192.168.2.19' #ip of Server (PI)
host = socket.gethostbyname(self.hostname) # if fails install samba on pi and reboot

while True:
    for event in pygame.event.get():
        if event.type == pygame.JOYAXISMOTION:
            self.axis_data[event.axis] = round(event.value, 2)
        elif event.type == pygame.JOYBUTTONDOWN:
            self.button_data[event.button] = True
        elif event.type == pygame.JOYBUTTONUP:
            self.button_data[event.button] = False
        elif event.type == pygame.JOYHATMOTION:
            self.hat_data[event.hat] = event.value

    # Insert your code on what you would like to happen for each event here!
    # In the current setup, I have the state simply printing out to the screen.

    # Defining Variables to send through the socket to the RPi, need to be strings

    # axis_data=str(self.axis_data)
    # button_data = str(self.button_data)
    # hat_data = str(self.hat_data)

    # Sending Data over a socket to the RPi
    # print(str(self.axis_data))
    # Isolate desired Axes

    axes_data = [self.axis_data[self.axis_order[0]] * self.limits[0],
                  self.axis_data[self.axis_order[1]] * self.limits[1],
                  self.axis_data[self.axis_order[2]] * self.limits[2],
                  self.axis_data[self.axis_order[3]] * self.limits[3]]
    byte_data = [] # To hold the axes data serialized to bytes
    for axis in axes_data:
        byte_data.append(struct.pack("f", axis)) # F for float

    # Send the control input data in byte form over the to be sent over the socket
    xmission_bytes = bytes().join(byte_data)
    connection = socket.socket()
    connection.connect((host, self.port)) # Make the connection to the RPi
    connection.send(xmission_bytes) # sending the controller data over the port
    connection.close() # Whenever no control inputs, close socket
    # print(xmission_bytes)

    # os.system('cls')
    # break
    # s.send(button_data)
    # s.send(hat_data)
    # s.close()

if __name__ == "__main__":
    ps4 = PS4Controller()
    # ps4.init()
    ps4.listen()

```