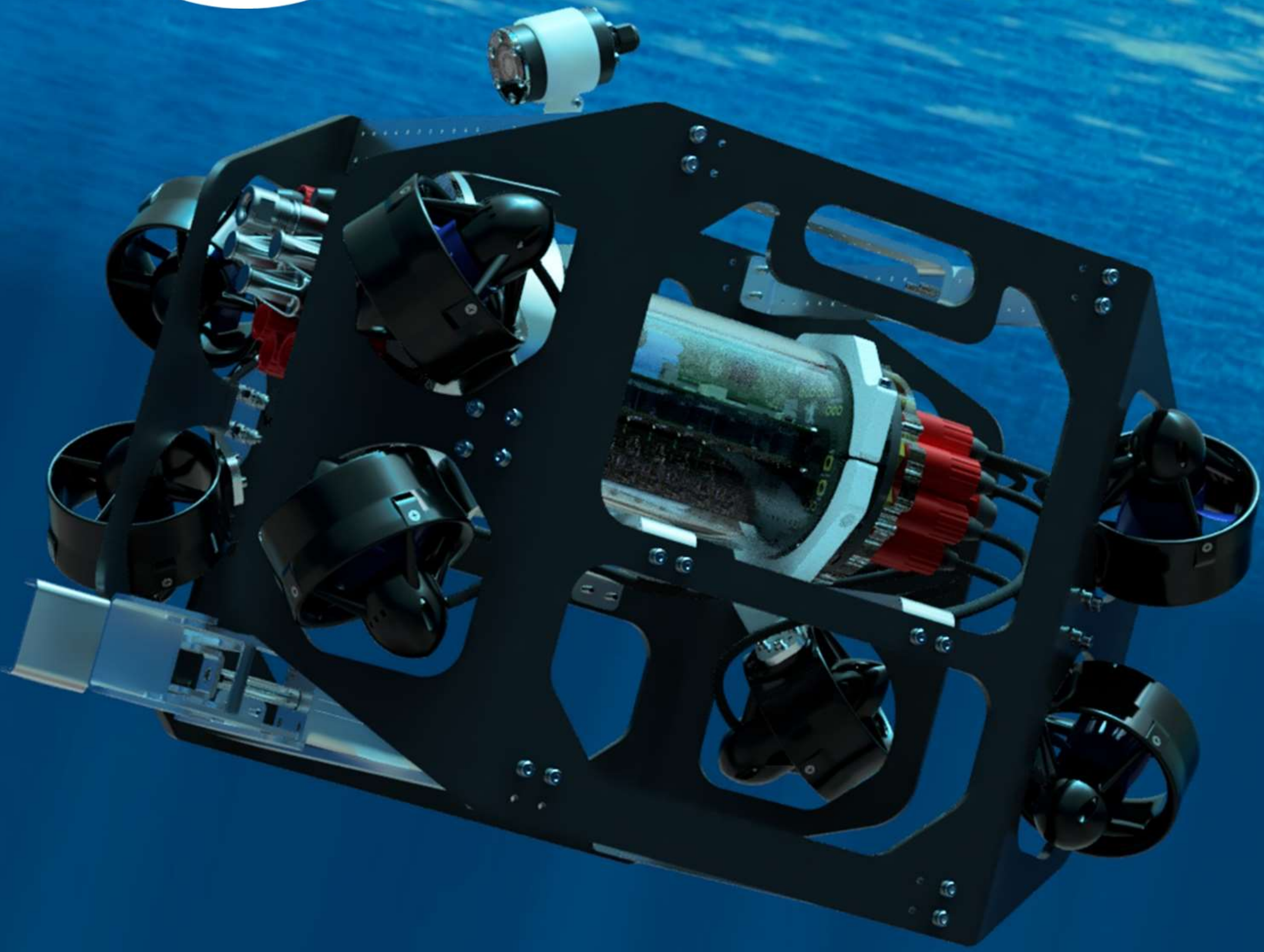
Pilot Control Program

User Guide



Written by

Benjamin Griffiths

Introduction

The Avalon ROV pilot control program is written in Python, and primarily uses the PyQt5 library (a Python binding of the popular Qt application framework) to create the graphical user interface widgets and layouts.

The user interface in split into a *control panel* and *configuration* tab, shown below.

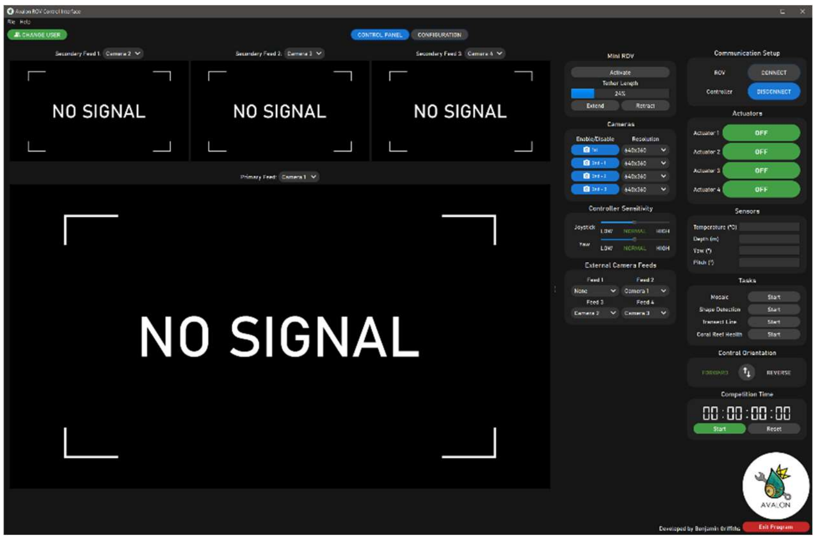
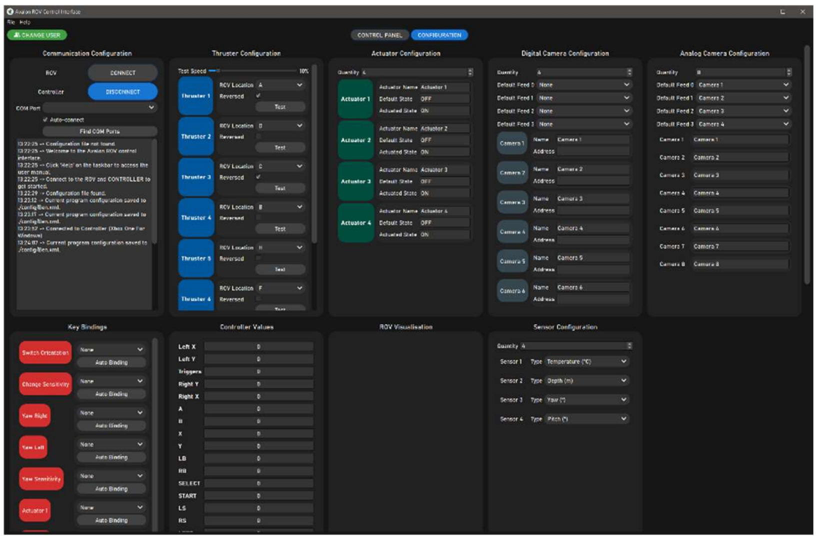


Figure 2: Control panel tab.



```
13:22:25 -> Configuration file not found.
13:22:25 -> Welcome to the Avalon ROV control interface.
13:22:25 -> Click 'Help' on the taskbar to access the user manual.
13:22:25 -> Connect to the ROV and CONTROLLER to get started.
13:22:29 -> Configuration file found.
13:23:12 -> Current program configuration saved to ./config/Ben.xml.
13:23:17 -> Current program configuration saved to ./config/Ben.xml.
13:23:52 -> Connected to Controller (Xbox One For Windows)
13:24:07 -> Current program configuration saved to ./config/Ben.xml.
```

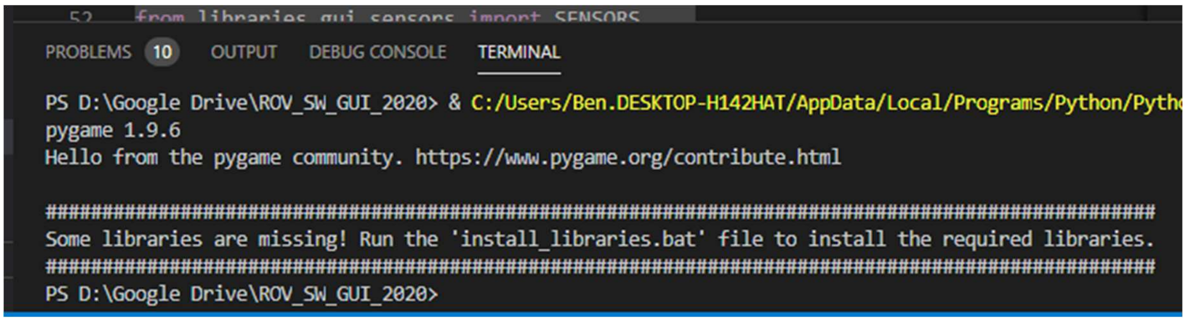
Figure 1: Configuration tab.

The control panel tab contains all the essential controls the pilot requires to operate the ROV during the competition, such as viewing live camera feeds, initiating serial communication links, toggling actuators and viewing sensors readings.

The configuration tab is where the program is configured to match the pilot's preferences and ROV setup. For example, the number of actuators, the address of the camera feeds, the location of each thruster and the preferred controller key bindings can be custom configured and saved to a user profile. Multiple user profiles can be set up to match different ROV configurations or different pilot key binding preferences.

Using the program

Python 3+ must be installed to run this program, and you may not be able to run it initially, instead receiving this error message on the terminal, which is a warning that not all the required dependencies are installed on your system.



```
from libraries.gui.sensors import SENSORS
```

```
PS D:\Google Drive\ROV_SW_GUI_2020> & C:/Users/Ben.DESKTOP-H142HAT/AppData/Local/Programs/Python/Pytho
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html

##########################################################################################
Some libraries are missing! Run the 'install_libraries.bat' file to install the required libraries.
##########################################################################################
PS D:\Google Drive\ROV_SW_GUI_2020>
```

Figure 3: Uninstalled libraries warning message.

To fix this, run the install_libraries.bat file, which will sequentially install each Python library listed below:

- PyQt5
- OpenCV
- PyGame
- PySerial

When the program is launched, a pilot profile selector window will appear, which will list all the currently available profiles that are stored in the 'config' folder in the programs directory. The pilot can load a profile simply by clicking on it, and the program will read the file and the configure the program accordingly. Additionally, each profile can be renamed or delete, and new profiles can be added.

To load a configuration file that is not listed, i.e. not stored in the 'config' folder in the programs directory, go to File -> Load Configuration File.

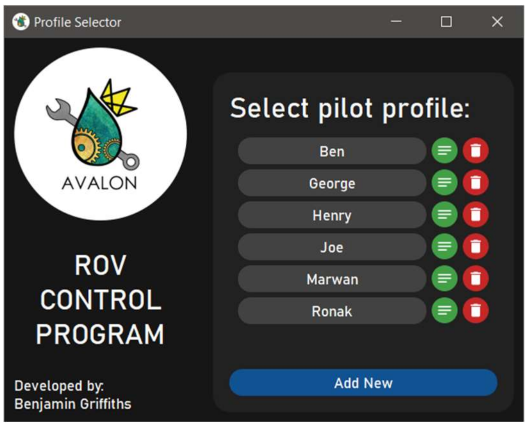


Figure 4: Pilot profile selector window.

Alternatively, if the window is closed, the program will be in an unconfigured state, and the profile will have to be saved if it wants to be loaded in the future. To save a profile, either use the Ctrl+S shortcut, or go to File -> Save Configuration. Changes are not saved automatically, so the configuration must be saved before closing the program.

The program can be reset to an unconfigured state at any time by going to File -> Reset.

The program is launched in full screen mode. To transition to maximised mode, hit the ESC key, and to transition back to full screen, hit the F11 key.

The default program theme is dark, but an alternative light theme is available by going to File -> Toggle Theme. The selected theme is saved to the configuration file.

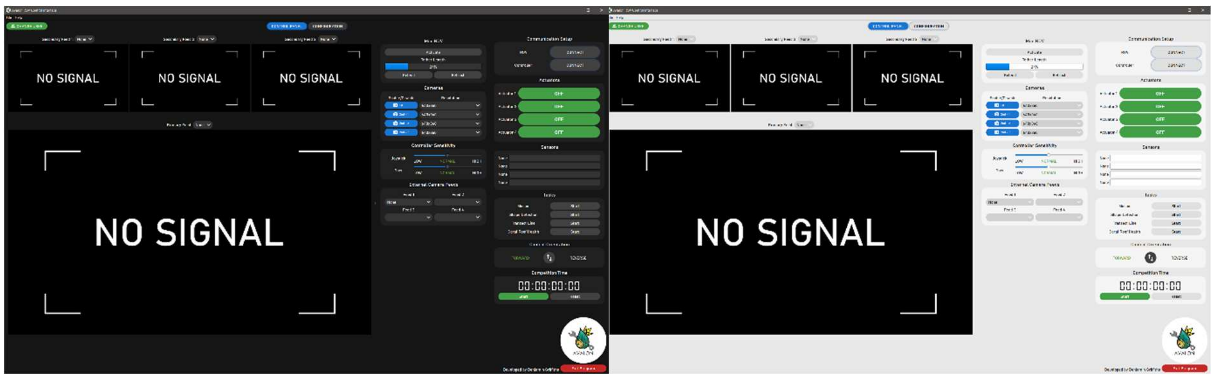


Figure 5: Dark and light program themes.

To toggle between the *control panel* and *configuration* tabs, uses the navigation buttons at the top of the screen.

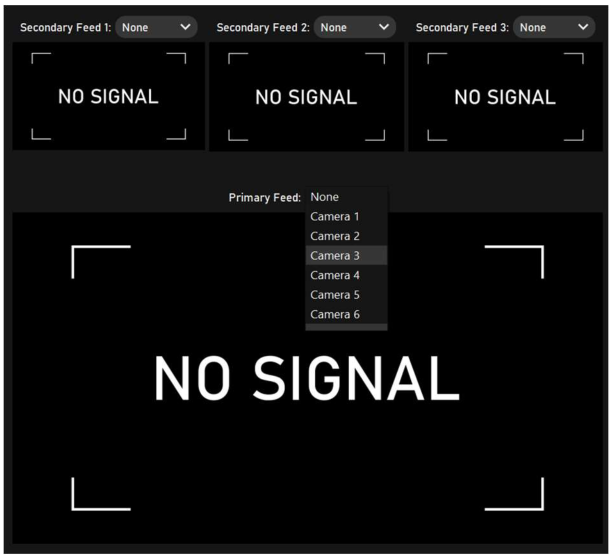


Figure 6: Camera feeds.

There are four live camera feeds available to help navigate the ROV, with one large primary, and three smaller secondary camera feeds. The camera source shown on each feed can be changed with the drop-down menus. Alternatively, any of the secondary feeds can be displayed on the large primary feed by clicking anywhere in the frame.

The controls required to operate the ROV are shown on the right-hand side of the screen, and are clearly labelled.

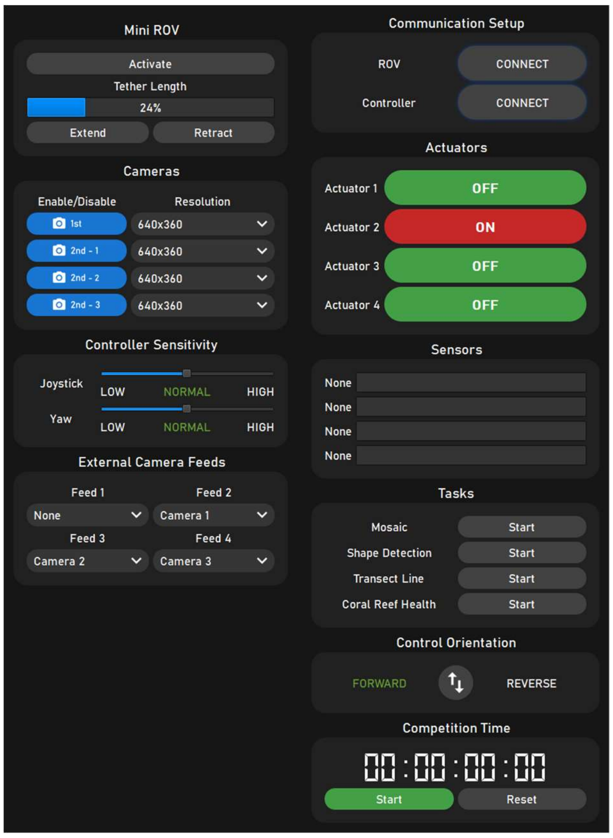


Figure 7: ROV controls.

To get started, click the ROV connect button in the communication setup tile, which will automatically check the identity of every device connected to the computer, and initiate a serial communication interface on the correct COM port.

To connect to the controller, click the Controller connect button, which will automatically search for an XBOX One controller plugged in via USB.

When a serial communication link is successfully established between the program and the ROV, the program will immediately start requesting sensors readings, which will be displayed on the sensors tile.

Each camera feed can be enabled or disabled by toggling the button in the cameras tile. Additionally, the display resolution of each camera feed can be selected from the drop-down menus.

Warning: Using multiple camera feeds on a high resolution may cause the USB bus to saturate and cause unexpected program crashes. It is recommended to keep the secondary camera on low resolutions since they are small.

Each actuator connector the ROV can be toggled using the buttons displayed in the actuators tile. The buttons will turn red when an actuator is activated/turned-on.

The different machine vision tasks that are to be completed at the competition can be activated using the start buttons in the tasks tile. When a task is activated, a new window will slide out which contains the necessary controls to complete the task.

The button in the control orientation tile toggles the control direction of the thrust vectoring algorithm, so that the pilot can drive the ROV in reverse as if it was being controlled forwards. This is especially useful if there are different actuators on the front and back of the ROV.

The two sliders in the controller sensitivity tile allow the pilot to change the sensitivity of the thruster vectoring algorithm and yaw movements. The yaw of the ROV is controlled using buttons on the controller, so settings this gain to a low value will make fine movements easier. Alternatively, setting the joystick gain to its maximum value will unlock the full power of the thrusters for lifting heavy objects.

The drop-down menus in the external camera feeds tile select which analogue cameras are transmitted up the ROVs tether. These analogue cameras are only able to be displayed in the program by using an RTSP communication protocol via an external DVR.

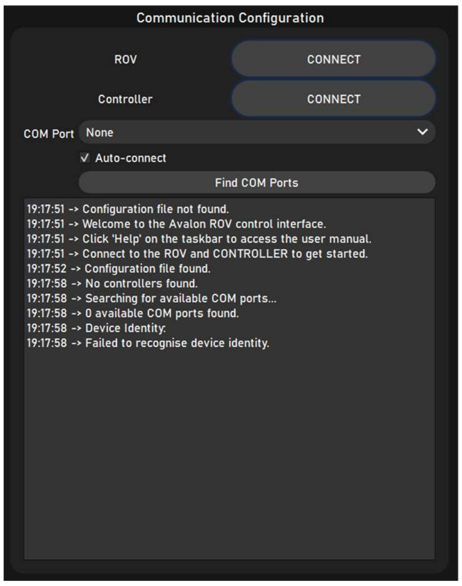
A convenient timer is also available so that the pilot can keep track of progress during the competition.

The GitHub repository for this program, and the Doxygen generated code documentation can be accessed under the Help menu.

Configuring the program

Almost every aspect of the program is configurable, allowing the program to adapt to different ROV design and pilot preferences.

Communication



COM Port: None

✓ Auto-connect

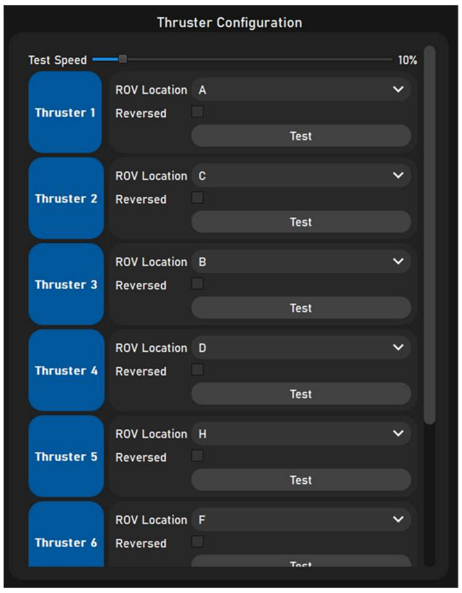
```
19:17:51 -> Configuration file not found.
19:17:51 -> Welcome to the Avalon ROV control interface.
19:17:51 -> Click 'Help' on the taskbar to access the user manual.
19:17:51 -> Connect to the ROV and CONTROLLER to get started.
19:17:52 -> Configuration file found.
19:17:58 -> No controllers found.
19:17:58 -> Searching for available COM ports...
19:17:58 -> 0 available COM ports found.
19:17:58 -> Device Identity:
19:17:58 -> Failed to recognise device identity.
```

Figure 8: Communication configuration tile.

The communication configuration tile contains connect button for the ROV and controller, just like on the *control panel* tab. By default, the program will automatically search for the ROVs COM port, but if desired, the Auto-connect checkbox can be un-checked, and the COM port can be manually selected from the drop-down menu.

Additionally, a terminal is included which displays error messages and the connection status.

Thrusters



Test Speed 10%

Thruster 1 — ROV Location: A; Reversed: ☐

Thruster 2 — ROV Location: C; Reversed: ☐

Thruster 3 — ROV Location: B; Reversed: ☐

Thruster 4 — ROV Location: D; Reversed: ☐

Thruster 5 — ROV Location: H; Reversed: ☐

Thruster 6 — ROV Location: F; Reversed: ☐

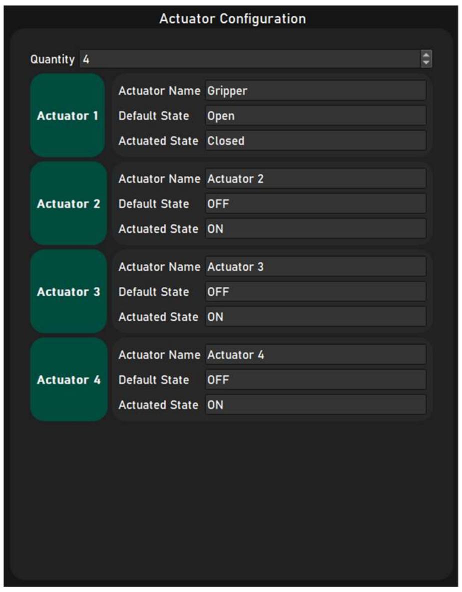
Figure 9: Thruster configuration tile.

There are eight thrusters on the ROV, in a configuration where each thruster is offset to the X, Y and Z axis by 45°. For each thruster, its location on the ROVs chassis can be selected from a drop-down menu, which must be set correctly for the thrust vectoring algorithm to work.

Each thruster can also be reversed depending on how the 3 phase windings of the thrusters are connected.

A convenient test button allows each thruster to be individually spun up to verify the position and rotational direction is correct. The thruster test speed can be changed using the slider.

Actuators



Actuator Configuration

Quantity 4

Actuator 1
Actuator Name: Gripper
Default State: Open
Actuated State: Closed

Actuator 2
Actuator Name: Actuator 2
Default State: OFF
Actuated State: ON

Actuator 3
Actuator Name: Actuator 3
Default State: OFF
Actuated State: ON

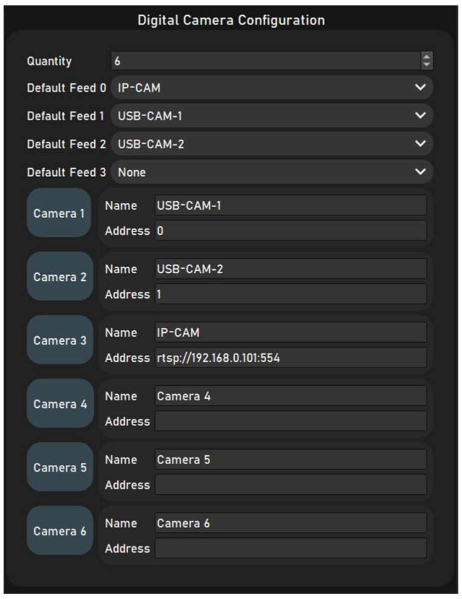
Actuator 4
Actuator Name: Actuator 4
Default State: OFF
Actuated State: ON

Figure 10: Actuators configuration tile.

The number of actuators on the ROV can be set, up to a limit of 10 (can be increased in software if required).

For each actuator, its name, default state and actuators state labels can be changed, and will update the buttons on the *control panel* automatically.

Digital Cameras



Digital Camera Configuration

Quantity 6
Default Feed 0: IP-CAM
Default Feed 1: USB-CAM-1
Default Feed 2: USB-CAM-2
Default Feed 3: None

Camera 1
Name: USB-CAM-1
Address: 0

Camera 2
Name: USB-CAM-2
Address: 1

Camera 3
Name: IP-CAM
Address: rtsp://192.168.0.101:554

Camera 4
Name: Camera 4
Address:

Camera 5
Name: Camera 5
Address:

Camera 6
Name: Camera 6
Address:

Figure 11: Digital cameras configuration tile.

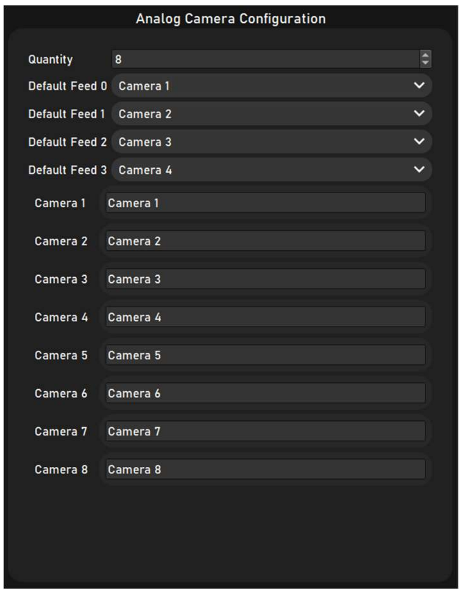
The number of digital cameras can be set, and the default camera sources that are displayed on the primary and secondary feeds upon program launch can be selected using the drop-down menus.

For each digital camera, the display name and address can be changed.

The program supports USB and network video streams. For USB webcams, the address is typically an integer (0, 1, 2 etc). Network streams from IP cameras will typically use an RTSP link (eg: rtsp://192.168.0.101:554), which enables the ROVs analogue camera to be viewed inside the program via the DVRs on-board server.

If no address is set, the camera will display the no-signal placeholder.

Analogue Cameras


Analog Camera Configuration

Quantity	8
Default Feed 0	Camera 1
Default Feed 1	Camera 2
Default Feed 2	Camera 3
Default Feed 3	Camera 4
Camera 1	Camera 1
Camera 2	Camera 2
Camera 3	Camera 3
Camera 4	Camera 4
Camera 5	Camera 5
Camera 6	Camera 6
Camera 7	Camera 7
Camera 8	Camera 8

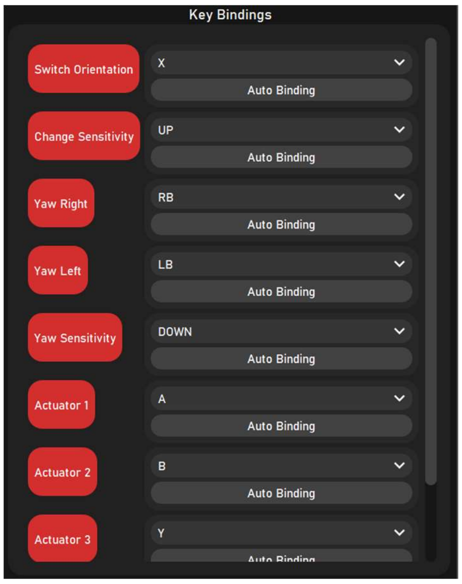
Figure 12: Analogue camera configuration tile.

The number of analogue cameras that are connected to the ROVs electronics control system can be set, and the default camera signals that are transmitted up the ROVs tether can be changed using the drop-down menus.

For each camera, the display name can be changed.

(Not yet implemented on the ROVs embedded software.)

Key Bindings


Key Bindings

Switch Orientation	X
Change Sensitivity	UP
Yaw Right	RB
Yaw Left	LB
Yaw Sensitivity	DOWN
Actuator 1	A
Actuator 2	B
Actuator 3	Y

Figure 13: Key bindings configuration tile.

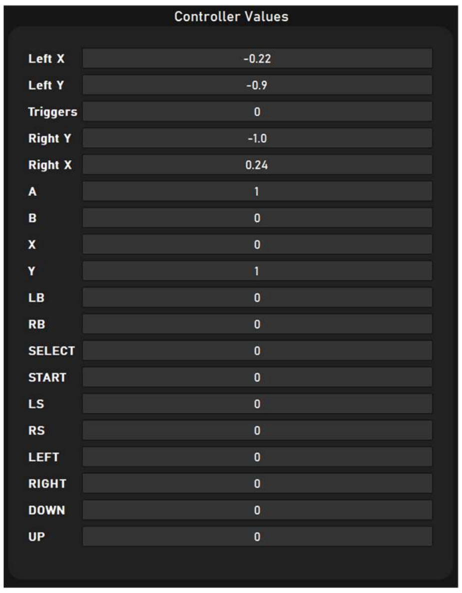
The key bindings allow certain program and ROV controls to be changed by the pilot using the XBOX controller. The standard set of controls are to toggle the control direction, increment the joystick/yaw sensitivity and yaw left/right. An additional control is then included for each actuator.

Each control can be bound to a button on the XBOX controller via the drop-down menu, which lists all the possible buttons.

Alternatively, the auto-binding button can be pressed, and the program will automatically detect which button is pressed and set the key binding.

If a button is already assigned to a control, and the user attempt to assign it to another control, the original control will be un-bound.

Controller Values


Controller Values

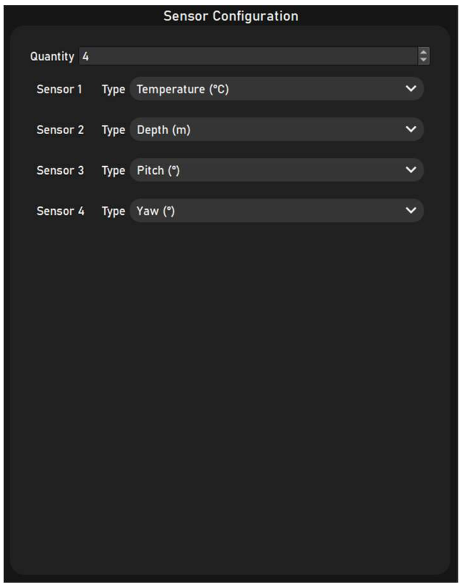
Left X	-0.22
Left Y	-0.9
Triggers	0
Right Y	-1.0
Right X	0.24
A	1
B	0
X	0
Y	1
LB	0
RB	0
SELECT	0
START	0
LS	0
RS	0
LEFT	0
RIGHT	0
DOWN	0
UP	0

Figure 14: Controller values configuration tile.

The current value of each joystick and the state of each button on the XBOX controller is listed (with a 60Hz refresh rate).

This can be used to check the controller is functioning correctly.

Sensors


Sensor Configuration

Quantity 4

Sensor 1	Type	Temperature (°C)
Sensor 2	Type	Depth (m)
Sensor 3	Type	Pitch (°)
Sensor 4	Type	Yaw (°)

Figure 15: Sensors configuration tile.

The number of sensors connected to the ROV can be changed, and for each sensor, the type can be changed using the drop-down menus (changes the sensor display label on the *control panel*).

Code Structure

The program is made up of several folders and files, outlined in the table below.

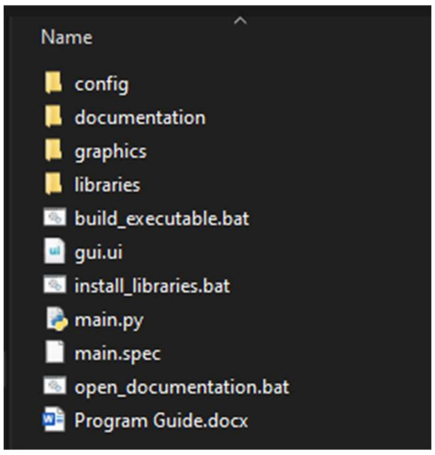

Name
config
documentation
graphics
libraries
build_executable.bat
gui.ui
install_libraries.bat
main.py
main.spec
open_documentation.bat
Program Guide.docx

Figure 16: Programs file structure.

File/Folder	Description
config	A folder containing all the pilot profile XML files.
documentation	A folder containing all the files generated by Doxygen for the documentation.
graphics	A folder containing all the images required by the program, such as camera feed placeholders, logos and button icons.
libraries	A folder containing all the custom Python libraries that the program imports.
build_executable.bat	A batch file that prompts Py-Installer to build an executable of the program, so that it can run on computers without Python installed.
gui.ui	The Qt user interface file generated by Qt Designer. This file defines the basic structure of the user interface, such as parent layouts, as most of the widgets are added in code.
install_libraries.bat	A batch file that uses PIP to install all the required external libraries one by one such as Open CV and PyQt5.
main.py	The main Python file that imports all the libraries and launches the program.
main.spec	A file that contains the settings for Py-Installer to use, such as which files and folders to include in the executable.
open_documentation.bat	A batch file that points to the index.html file in the *documentation folder*, opening the Doxygen documentation in the browser.

The Python modules contained in the libraries folder perform the bulk of the programs functionality. The main Python file simply imports all these modules and links them together where appropriate, and performs other functionality such as scaling, theme changing, user loading user profiles etc.

A description of each library is shown below.

Library	Description
configurationFile	Contains a class of functions to read and write the data to a configuration XML file. Uses the Python ElementTree library.
rovComms	Contains all functions relating to serial communication with the ROV, such as automatic COM port detection, and functions to format the commands to send to the ROV.
xboxController	A thread running at 60Hz that reads joystick and button values from an XBOX controller. These values run through filtering functions before being emitted as a signal to the main program.
cameraCapture	A thread that captures frames from connected cameras, runs them through processing algorithms where required, and emits them as a signal. Other functions include changing the source address of the camera feed, and handling cameras connecting/disconnecting.
visualEffects	Contains all the style sheets that determine the colours and styles of the widgets in the program, as well as functions to change the program theme.
timerWidget	A timer widget used by the pilot at the competition.
thrusters	Contains widgets for the configuration tab, and the thrust vectoring algorithm.
actuators	Contains widgets for the control panel and configuration tab, with functions to toggle each actuator.
analogueCameras	Contains widgets for the configuration tab, and functions to change camera settings.
digitalCameras	Contains widgets for the configuration tab, and functions to change camera source addresses.
keyBindings	Contains widgets for the configuration tab, and functions to automatically detect control button presses to set key-bindings.
controllerDisplay	Contains widgets for the configuration tab to visually indicate the values of the controller joysticks and buttons.
sensors	Contains widgets for the control panel and configuration tab, with functions for filtering the incoming data.
profileSelector	Popup window to load/add/rename/delete pilot profiles.
slideAnimation	Handles the slide animation between the *control panel* and *configuration* tabs, as well as the vision task windows.
computerVision	Folder containing the OpenCV algorithms for the different tasks.

Table 1: Python libraries.

The modules communicate with the main Python file and with each other using the Qt *signals* and *slots* convention, where a module will emit a signal, which will send data to a slot it is connected to. The signals from each module are connected to the appropriate slots in the main Python file.

A highly simplified flow diagram that explains the programs operation is shown below.

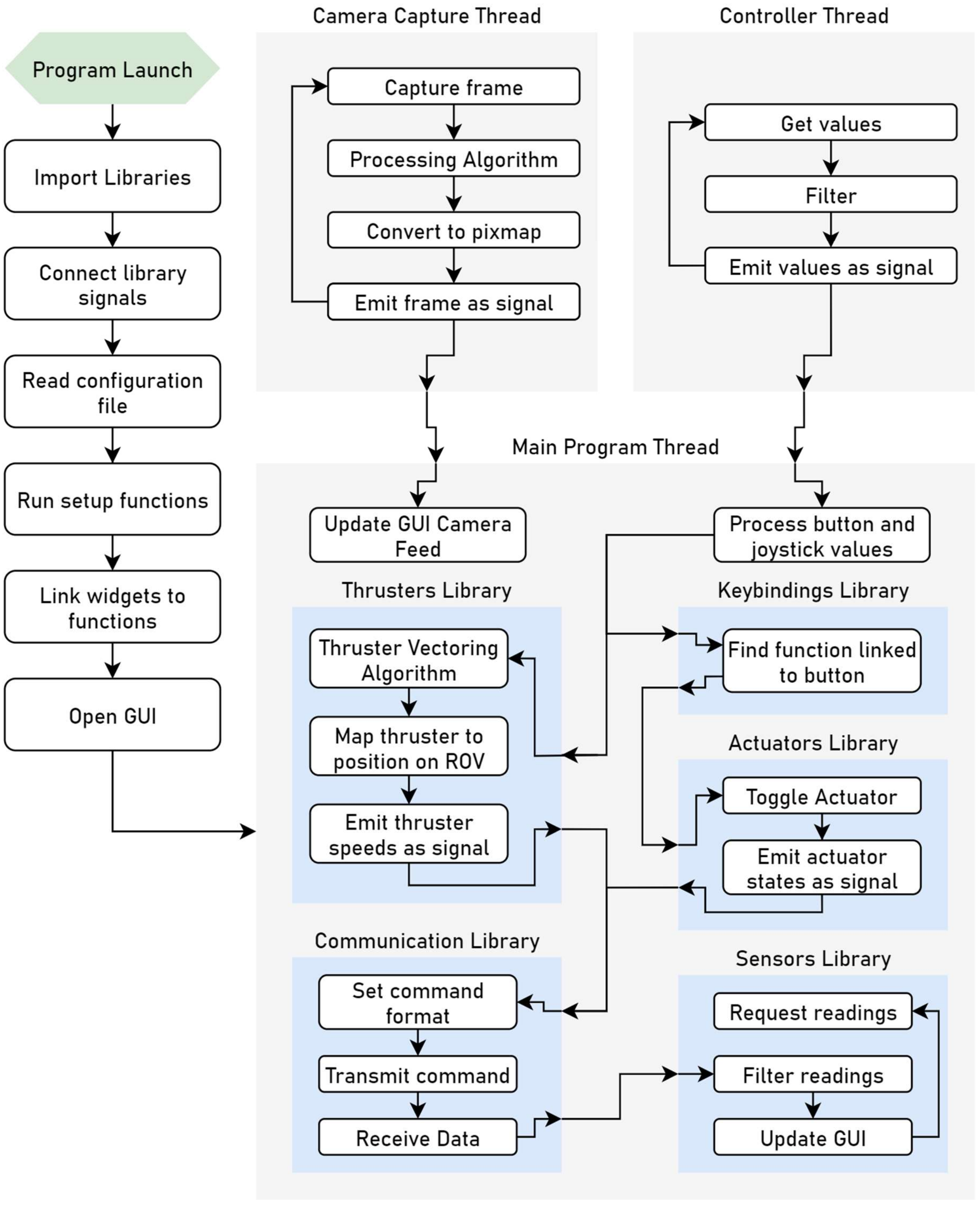
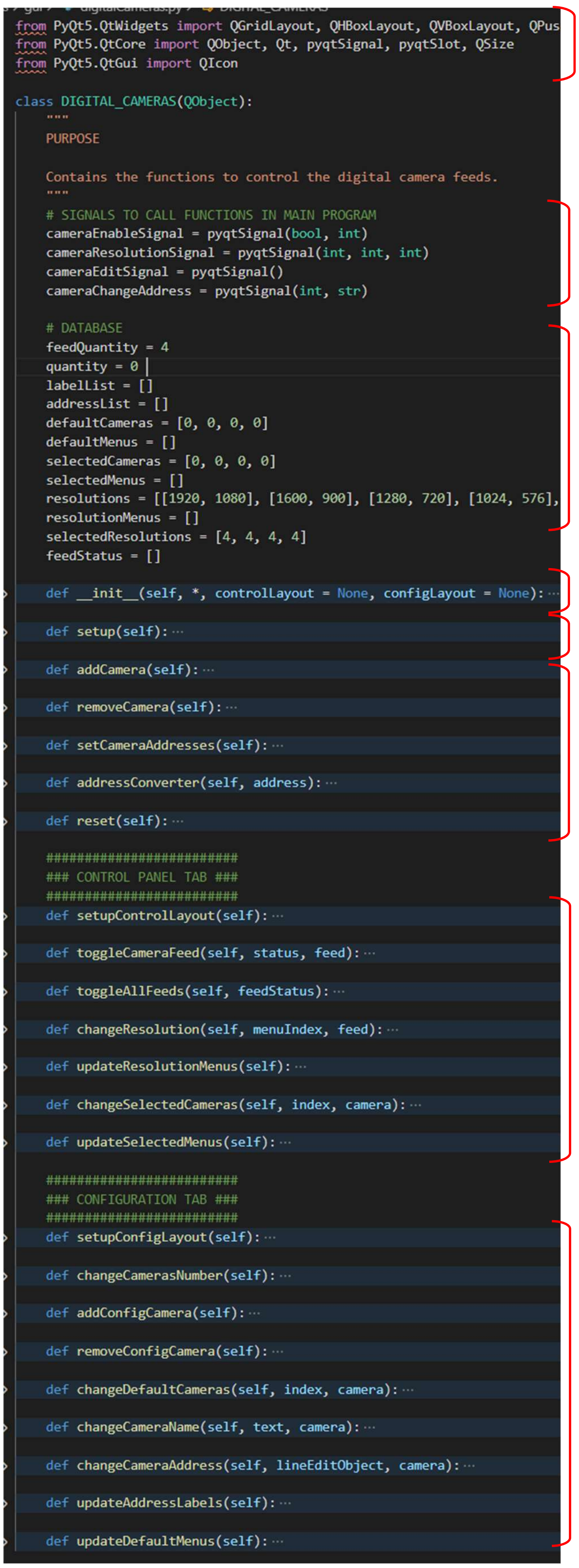


Figure 17: Program operation flow diagram.

The modules that control parts of the GUI, such as the digitalCamera and actuators library, have a standardised structure. An explanation of the digitalCamera libraries structure is shown below.



```
from PyQt5.QtWidgets import QGridLayout, QHBoxLayout, QVBoxLayout, QPus
from PyQt5.QtCore import QObject, Qt, pyqtSignal, pyqtSlot, QSize
from PyQt5.QtGui import QIcon

class DIGITAL_CAMERAS(QObject):
    """
    PURPOSE

    Contains the functions to control the digital camera feeds.
    """
    # SIGNALS TO CALL FUNCTIONS IN MAIN PROGRAM
    cameraEnableSignal = pyqtSignal(bool, int)
    cameraResolutionSignal = pyqtSignal(int, int, int)
    cameraEditSignal = pyqtSignal()
    cameraChangeAddress = pyqtSignal(int, str)

    # DATABASE
    feedQuantity = 4
    quantity = 0
    labelList = []
    addressList = []
    defaultCameras = [0, 0, 0, 0]
    defaultMenus = []
    selectedCameras = [0, 0, 0, 0]
    selectedMenus = []
    resolutions = [[1920, 1080], [1600, 900], [1280, 720], [1024, 576],
    resolutionMenus = []
    selectedResolutions = [4, 4, 4, 4]
    feedStatus = []

    def __init__(self, *, controlLayout = None, configLayout = None):···

    def setup(self):···

    def addCamera(self):···

    def removeCamera(self):···

    def setCameraAddresses(self):···

    def addressConverter(self, address):···

    def reset(self):···

    ########################
    ### CONTROL PANEL TAB ###
    ########################
    def setupControlLayout(self):···

    def toggleCameraFeed(self, status, feed):···

    def toggleAllFeeds(self, feedStatus):···

    def changeResolution(self, menuIndex, feed):···

    def updateResolutionMenus(self):···

    def changeSelectedCameras(self, index, camera):···

    def updateSelectedMenus(self):···

    ########################
    ### CONFIGURATION TAB ###
    ########################
    def setupConfigLayout(self):···

    def changeCamerasNumber(self):···

    def addConfigCamera(self):···

    def removeConfigCamera(self):···

    def changeDefaultCameras(self, index, camera):···

    def changeCameraName(self, text, camera):···

    def changeCameraAddress(self, lineEditObject, camera):···

    def updateAddressLabels(self):···

    def updateDefaultMenus(self):···
```

Imports: Where the required libraries are imported.

Signals: Initialise the signals required to be emitted from this object. For example, the changeCameraAddress signal could be emitted, which is connected to the cameraCapture library to change the camera source.

Database: Set the default configuration parameters that will loaded upon program launch. These variables will be overwritten when data is read from a configuration file.

Constructor: The object is passed a layout for the control panel and configuration tabs to add widgets to.

Setup: Uses the database variables to add/remove/modify widgets where necessary. This function is called to setup the program once a configuration file has been read.

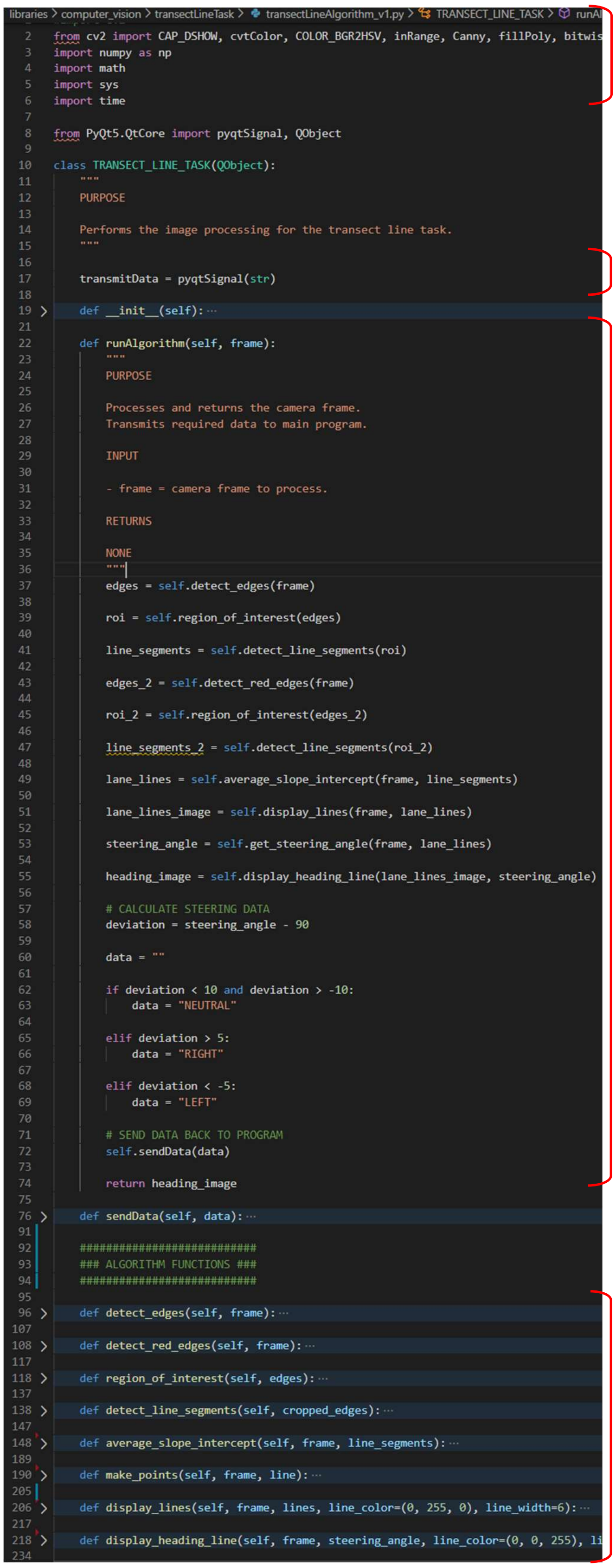
High level functions: Functions that either control elements on both the control panel and configuration tabs, or perform miscellaneous tasks that are not related to adding or removing widgets.

Control panel tab: Functions that control widgets for the layout on the control panel tab. There could be functions to add, remove or modify the widgets, or emit signals back to the main program.

Configuration tab: Functions that control widgets for the layout on the configuration tab. There could be functions to add, remove or modify the widgets, modify widgets on the control panel or emit signals back to the main program.

By using this standard format for all modules, it is very easy to test individual components of the program and implement new features by simply importing the module and connecting the signals.

Similarly, there is a standard format for implementing OpenCV algorithms into the GUI, shown below:



libraries > computer_vision > transectLineTask > transectLineAlgorithm_v1.py > TRANSECT_LINE_TASK > runAl

```python
from cv2 import CAP_DSHOW, cvtColor, COLOR_BGR2HSV, inRange, Canny, fillPoly, bitwis
import numpy as np
import math
import sys
import time

from PyQt5.QtCore import pyqtSignal, QObject

class TRANSECT_LINE_TASK(QObject):
    """
    PURPOSE

    Performs the image processing for the transect line task.
    """

    transmitData = pyqtSignal(str)

    def __init__(self): ...

    def runAlgorithm(self, frame):
        """
        PURPOSE

        Processes and returns the camera frame.
        Transmits required data to main program.

        INPUT

        - frame = camera frame to process.

        RETURNS

        NONE
        """
        edges = self.detect_edges(frame)

        roi = self.region_of_interest(edges)

        line_segments = self.detect_line_segments(roi)

        edges_2 = self.detect_red_edges(frame)

        roi_2 = self.region_of_interest(edges_2)

        line_segments_2 = self.detect_line_segments(roi_2)

        lane_lines = self.average_slope_intercept(frame, line_segments)

        lane_lines_image = self.display_lines(frame, lane_lines)

        steering_angle = self.get_steering_angle(frame, lane_lines)

        heading_image = self.display_heading_line(lane_lines_image, steering_angle)

        # CALCULATE STEERING DATA
        deviation = steering_angle - 90

        data = ""

        if deviation < 10 and deviation > -10:
            data = "NEUTRAL"

        elif deviation > 5:
            data = "RIGHT"

        elif deviation < -5:
            data = "LEFT"

        # SEND DATA BACK TO PROGRAM
        self.sendData(data)

        return heading_image

    def sendData(self, data): ...

    ###########################
    ### ALGORITHM FUNCTIONS ###
    ###########################

    def detect_edges(self, frame): ...

    def detect_red_edges(self, frame): ...

    def region_of_interest(self, edges): ...

    def detect_line_segments(self, cropped_edges): ...

    def average_slope_intercept(self, frame, line_segments): ...

    def make_points(self, frame, line): ...

    def display_lines(self, frame, lines, line_color=(0, 255, 0), line_width=6): ...

    def display_heading_line(self, frame, steering_angle, line_color=(0, 0, 255), li
```

Imports: Where the required libraries are imported.

Signals: Initialise the signals required to be emitted from this object.

Run Algorithm: This function is called by the cameraCapture library when a vision task is active. The latest camera frame is passed to this function, where it can be processed and modified by all the required algorithms.

Calculations can be made such as changes to the ROVs position, which can then be communicated to the main program by emitting the suitable signal.

Processing Functions: Contains all the functions/algorithm to send the camera frame through.