**REMOTE FLIGHT CONTROLLER**

**A screenshot of a computer

Description automatically generated**

Text

Description automatically generated with low confidence**SOFTWARE USER GUIDE**

By

Huw Williams

**DISCLAIMER**

This remote flight controller user manual is intended for educational purposes and is designed for use in conjunction with a simple flight simulator. The creator of this project is not liable for any damage or malfunctions resulting from the use of the controller. Users are encouraged to thoroughly understand the manual and seek assistance if needed.

Copyright © 2023 by Huw Williams 30051129 & University of South Wales.

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher. For permission requests, write to the publisher, addressed “Attention: Permissions Coordinator,” at the address below.

Huw Williams

[30051129@students.southwales.ac.uk](mailto:30051129@students.southwales.ac.uk)

[willowsayswhat@gmail.com](mailto:willowsayswhat@gmail.com)

Contents

[1 PREFACE 4](#_Toc153556839)

[1.1 Description of the User 4](#_Toc153556840)

[2 Description of the Software 5](#_Toc153556841)

[2.1 Purpose of the Software 5](#_Toc153556842)

[2.2 Software Elements 6](#_Toc153556843)

[2.3 Understanding the User Interface 7](#_Toc153556844)

[2.4 Operating Panels 8](#_Toc153556845)

[3 Operation/Use 10](#_Toc153556846)

[3.1 How to Run/Execute the Software 10](#_Toc153556847)

[3.1.1 Operational Environment 10](#_Toc153556848)

[3.1.2 Starting/Stopping the Application’s Operation 12](#_Toc153556849)

[3.2 Software Documentation 13](#_Toc153556850)

[3.2.1 Setting up the Environment 13](#_Toc153556851)

[3.2.2 Technical Data 15](#_Toc153556852)

[3.2.3 The Code 16](#_Toc153556853)

[3.2.4 Adding Components in the Software 16](#_Toc153556854)

[3.2.5 Writing the Code 16](#_Toc153556855)

[4 Maintenance and improvements 19](#_Toc153556856)

[4.1 How to Maintain the Software 19](#_Toc153556857)

[4.2 Future Improvements 20](#_Toc153556858)

[5 Troubleshooting and Testing 21](#_Toc153556859)

[5.1 How to Identify and Solve Problems 21](#_Toc153556860)

[5.1.1 Troubleshooting Use Case - 1 21](#_Toc153556861)

[5.1.2 Troubleshooting Use Case - 2 21](#_Toc153556862)

[5.1.3 Troubleshooting Use Case - 3 22](#_Toc153556863)

[5.1.4 Troubleshooting Use Case - 4 22](#_Toc153556864)

[5.1.5 Frequently Asked Questions - more 22](#_Toc153556865)

[5.2 Testing 23](#_Toc153556866)

[5.2.1 Connect 23](#_Toc153556867)

[5.2.2 Control 23](#_Toc153556868)

[Appendix A. 24](#_Toc153556869)

# PREFACE

## Description of the User

***Operator***

The Remote Fight Controller is straightforward, although the Operator will need to connect the controller to the Flight Simulator requires technical knowledge. This manual includes a step-by-step guide for connecting the Controller to the Fight Simulator.

***Supervisor/Developer***

The remote Fight Controller is designed to be operated with the guidance of a novice C# programmer. The Operator or Supervisor should possess a basic understanding of event-driven programming. While the Controller utilises delegates and events for scalability, a comprehensive grasp of these concepts is not mandatory, as they will be explained in this manual.

Expectation of competency will be reflected within this manual. Certain code snippets, concepts and functionalities will be highlighted, but not explained. It is at the readers discretion to understand the fundamentals of C# .Net.

# Description of the Software

## Purpose of the Software

The Remote Flight Controller is a simple interface that allows you to control a basic flight simulator via Internet Protocol (IP) connection. It is crafted for educational purposes and can be used as an example of event-driven and socket programming, demonstrating scalable techniques in beginner software development.

**Key Features**

* **User-friendly Interface:** Navigate through the software effortlessly with a recognisable interface.
* **Real-time Control:** Adjust flight parameters in real-time, including altitude, speed, and direction.
* **Open-Source:** Tailor the software or use features in your own software. Customise the Controller to suit your needs.
* **Scalable:** The Remote Flight Controller is built so more features can be added without altering the source code.
* **Built for Novices:** An easy project for C# programmers to delve in open-source contribution.

Whether you are a novice event-driven programmer or someone wanting to experience the Remote Flight Controller, this manual will guide you through the installation process, provide detailed instructions on using each feature, and help you make the most of your requirements of the Remote Flight Controller.

## Software Elements

The application has two distinct graphical user interfaces: the Remote Flight Controller and the original Flight Simulator. Developed using Unity and C#, the Flight Simulator provides a realistic and immersive virtual environment for flight experiences.

These Graphical User Interfaces communicate seamlessly through socket networking. This requires the specification of the localhost IP address and a port number – which is automatically provided by both Graphical User Interfaces. This enables the exchange of information between the Remote Flight Controller and the Flight Simulator.

For detailed instructions on connecting and operating the simulator with the controller, please refer to the "Operation/Use" section. To enhance your experience, consider placing the Flight Simulator and Remote Flight Controller side by side for a more immersive and enjoyable simulation.

A screenshot of a device

Description automatically generated

*Figure 1: The Flight Simulator & Remote Flight Controller.*

## Understanding the User Interface

**Remote Flight Controller**

A screenshot of a computer

Description automatically generated

*Figure 2: Labelled parts of the Remote Fight Controller.*

|  |  |  |
| --- | --- | --- |
| **ALPHA** | **NAME** | **FUNCTION** |
| A | IP Address Input  Port Input | Visualises the localhost ip address and the port needed to connect to the Flight Simulator. The port is always pre-filled with the correct port. |
| B | Dis/connect Button | Connects and disconnect controller from the Flight Simulator. |
| C | Close Button | Closes the Remote Flight Controller. |
| D | Telemetry Table | Displays telemetry from the Flight Simulator. |
| E | Warning Panel | Displays the warnings. No Warning, Stalled, Low Altitude. |
| F | Controls Table | Displays any adjustments made to the controls. |
| G | Flight Controls | Controls the Throttle and Elevation speed of the Simulator. |
| H | Telemetry Gauges | Visualises the Flight Simulator telemetry with guages. |
| I | Telemetry Values | Displays telemetry value in numerical form. |

*Table 1 Explanation of the Remote Flight Controller GUI.*

## Operating Panels



*Figure 3: Important Operating Panels.*

Figure 3 illustrates the important panels on both graphical user interfaces, the Controller connection panel, the Flight Simulator connection panel, and the Control Panel.

While a chapter is dedicated to describing the step-by-step operation of the remote flight controller, a quick overview of the important operation panels will help to clarify and reenforce the steps necessary to successfully fly the plane.

To assist in simplifying the use of the remote flight controller, the above panels should be separated into two steps of operation: Connect, and Control.

**Hint:** Notice that the Flight Simulator has a Light theme, and the Remote Flight Controller has a Dark theme.

***Connect***

The first step of operation is to connect the Remote Flight Controller to the Flight Simulator. To do this we must first identify the Flight Simulator’s connection panel, which holds the ‘Open Port’ button – labelled ‘D’ in figure 3. This can also be verified in figure 1. This button tells the Flight Simulator to prepare for connection.

Next, the Operator will be asked to find and execute steps on the remote Flight Controller’s connection panel. This panel holds the ‘IP Address’ input field (A), and the ‘Port’ input field (B). The ‘Connect’ (C) and ‘Disconnect’ (D) buttons are located under the aforementioned, and the ‘Close’ (E) and ‘Help’ (F) buttons adjacent.

Recognising the connection panels will help in the coming Operation/Use chapter.

***Control***

The second step of operation will only require the use of the Controls Panel, which consists of two controls: a throttle control and an elevation control. This allows the Operator to adjust the aircraft’s pitch and acceleration.

The Controls Panel is located to the centre-right of the Remote Flight Controller (figure 2).

Finally, to start the flight simulation, the Operator will need to locate the ‘Start Sim’ button. It can be found on figure 1 by using the following prompts:

1. Locate the Flight Simulator GUI (Light theme).
2. Locate the far-right gauge labelled ‘Vertical Speed’.
3. Locate the scenario dropdown box below the gauge labelled ‘None Selected’.
4. The ‘Start Sim’ button is directly below the dropdown box.

Prior knowledge of the control’s location will assist in minimising wait times during take-off.

# Operation/Use

The Remote Flight Controller is a learning tool for novice C# programmers; therefore, no traditional installation method has not been created. All software files are available for use. This means that a level of competency is needed to setup and run the Controller. The operator will need to check whether the NuGet package is installed. The instructions for this can be found in Software Documentation.

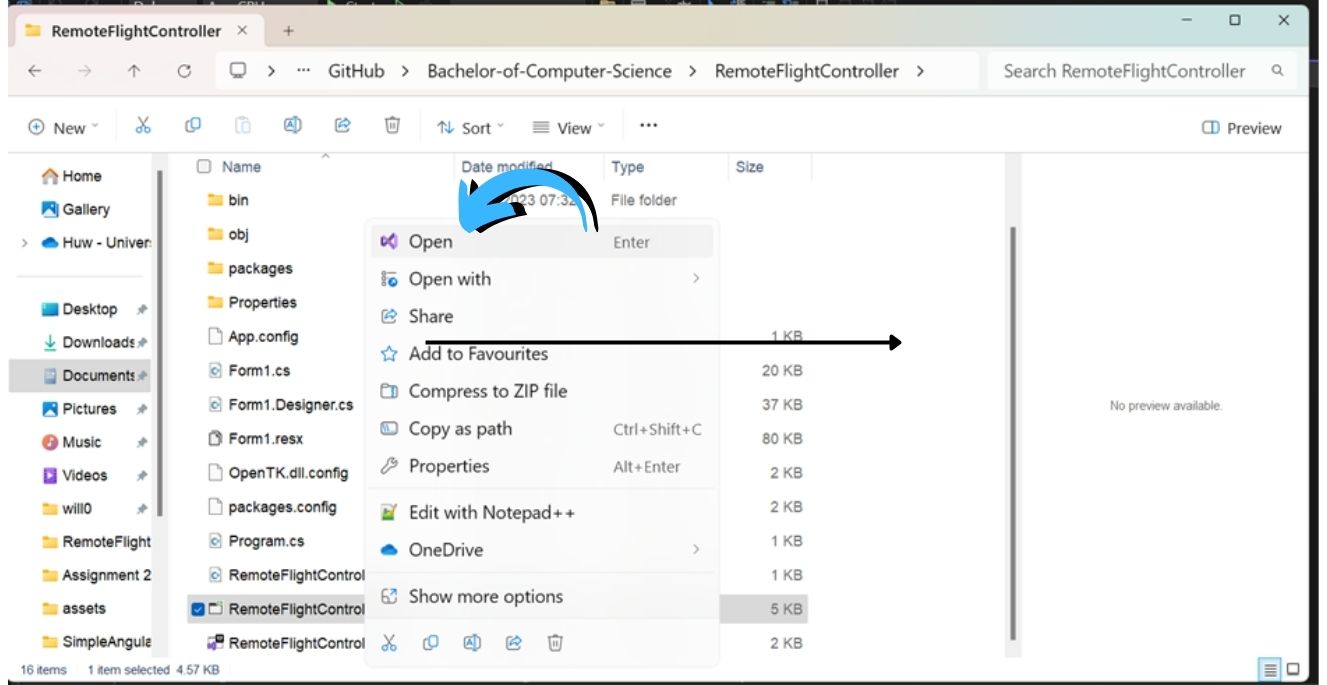
## How to Run/Execute the Software

### Operational Environment

The remote Flight Controller was built using Visual Studio and it is recommended that operators and programmers use Visual Studio when running the program. To simplify the coming instructions, open the folder that holds the Remote Flight Controller

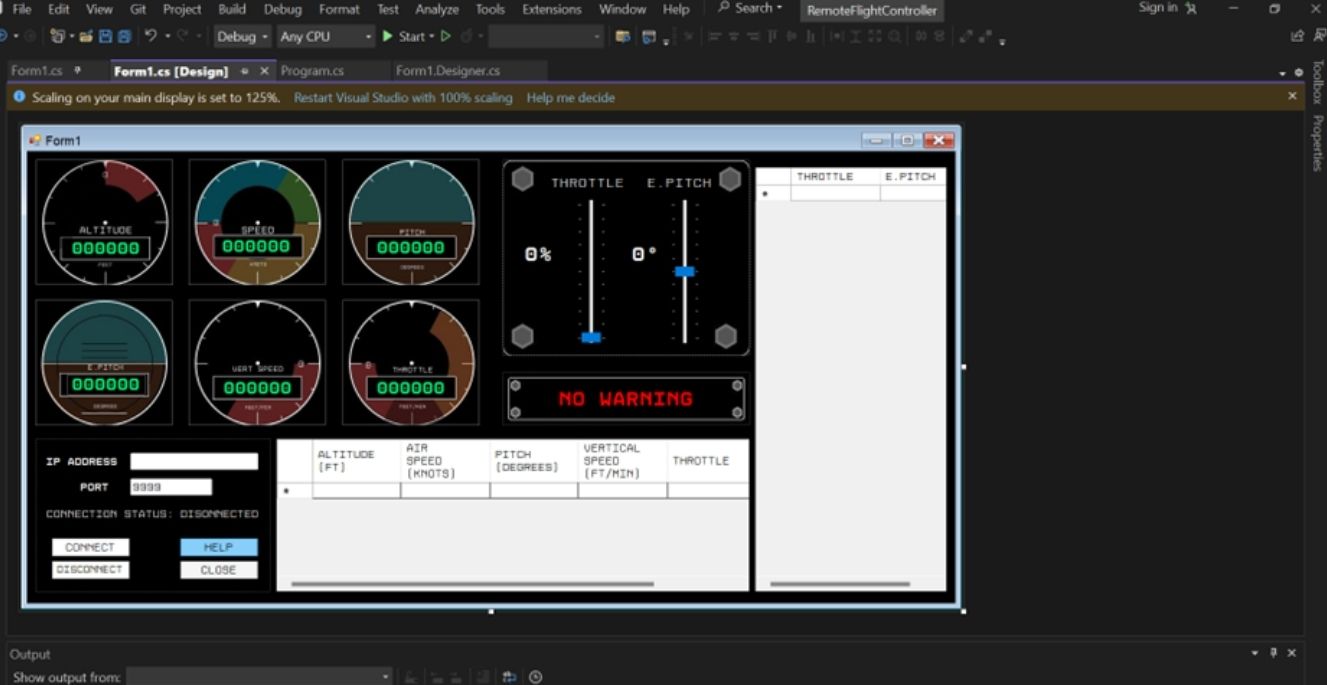
**Open in Visual Studio:**

1. Right-click on RemoteFlightController.csproj (figure 4).
2. Click on **Open.**
3. Right-click on **Form1.cs** in the Solution Explorer.
4. Select **View Design** (figure 5).
5. Developers can view the code by repeating step 3 and selecting **View Code** instead (figure 6).

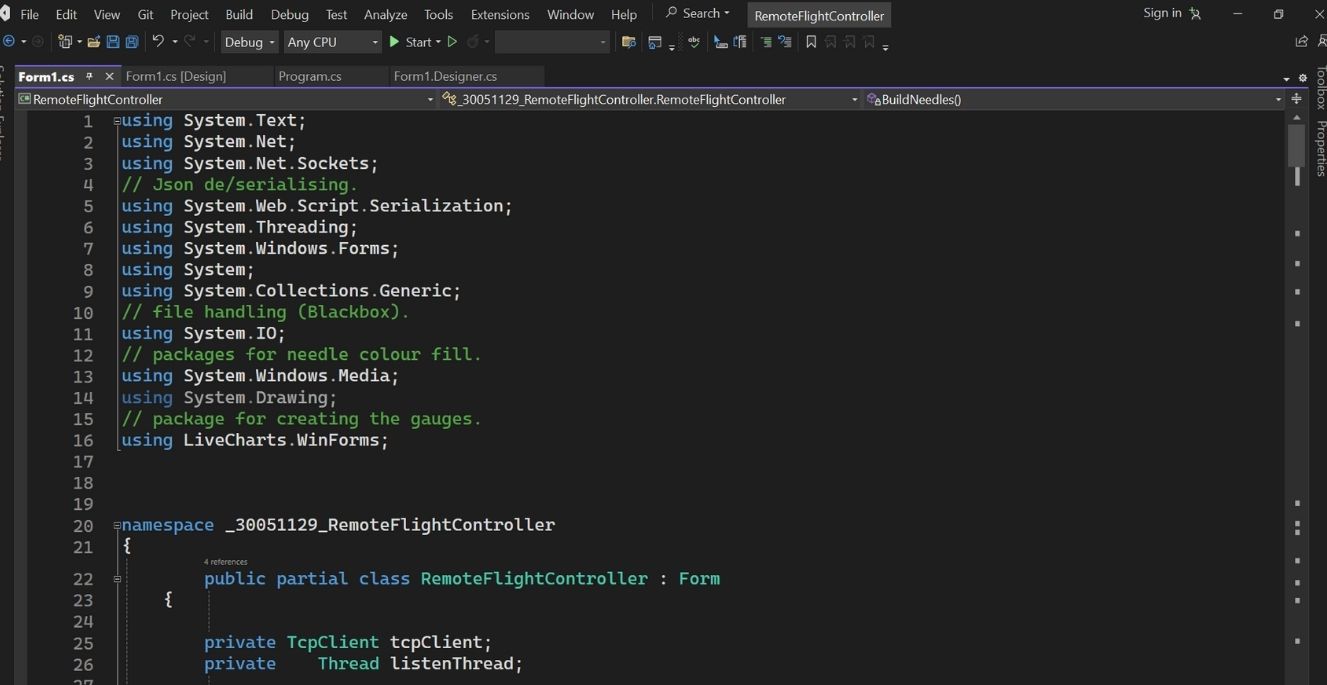


*Figure 4: Right-click on project file and open.*

Successful completion of these steps should have the Remote Flight Controller project open and visible on the operators screen as demonstrated in figure 5.



*Figure 5: Visual Studio in Form1.cs Design View*



*Figure 6: Visual Studio in Form1.cs Code View*

If the code shows any error messages, then a package installation may need to be performed. These packages are called NuGet Packages. The installation instructions are found in “Setting Up the Environment”.

### Starting/Stopping the Application’s Operation



*Figure 7: Important components to Connect.*

**Start the Remote Flight Controller:**

1. Click on the green “Play” button on Visual Studio’s menu bar.
2. Click the “Open Port” Button on the Flight Simulator ( E ).
3. Click the “Connect” button on the Remote Flight Controller ( C ).
4. Confirm Connection by closing the pop-up message box.
5. Click the “Start Sim” button on the Flight Simulator.
6. Use the “Help” Button for simple instructions.

**Take off.**

To take-off, adjust the Throttle and Elevator Pitch to full and wait for the aircraft to climb before levelling off. There is a dropdown menu that allows you to select an event. The description will remain vague so that the operator can test each event for themselves.

**Stop the Remote Flight Controller:**

1. Click the “Disconnect” button to stop receiving telemetry.

1. To close the Remote Flight Controller, either press the “x” in the top right corner or click the close button.
2. The Flight Simulator could close automatically upon disconnecting, however, the “x” in the top-right is also available to exit the program.

## Software Documentation

### Setting up the Environment

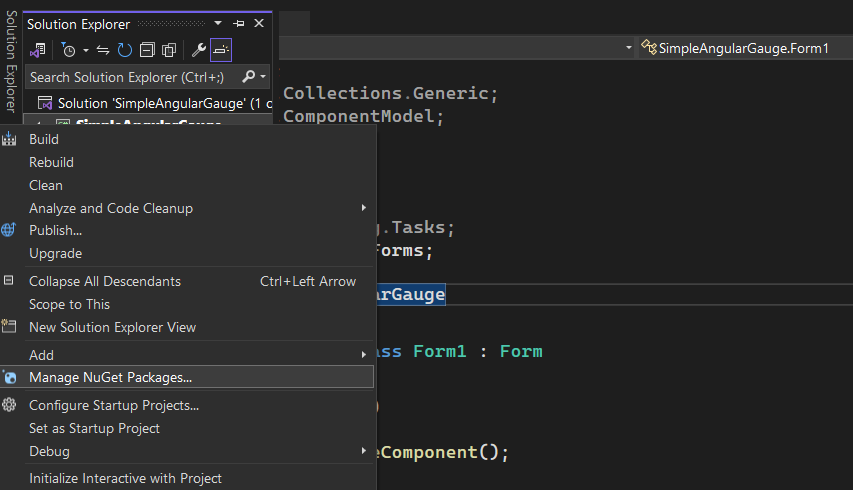
**Set Up the Environment for the Software:**

To run the Remote Flight Controller. Visual Studio will need to be installed. Instructions for the installation of Visual Studio can be found [here](https://learn.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2022).

Further installations may need to be completed for the Remote Flight Controller to work. For example, the Live Chart package should be present in the project, however, installation may be required.

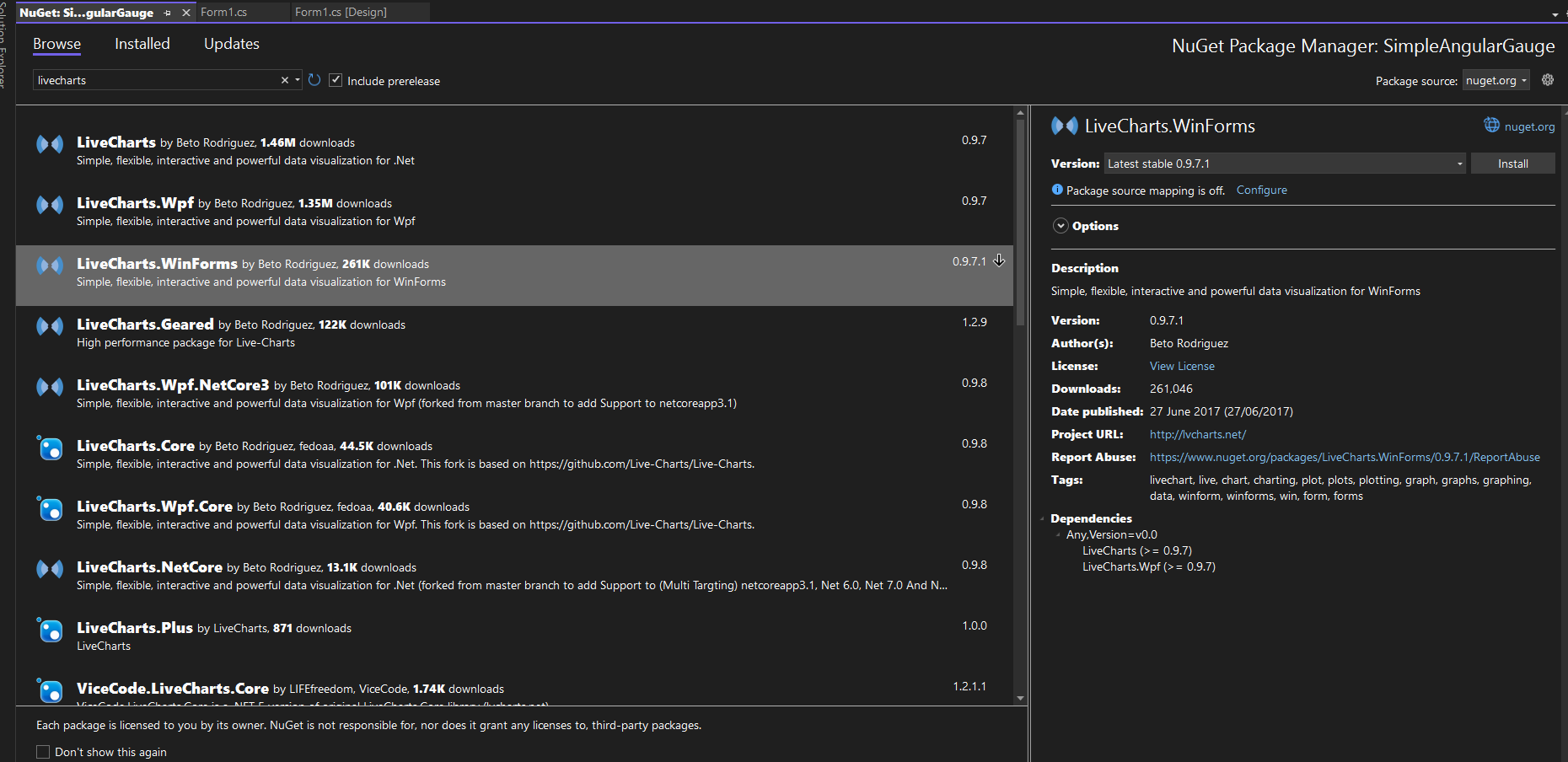
Follow these steps to install the Live Charts NuGet package.

1. To find your package manager window, right click on the name of your solution in the solution explorer. You can see in the image below that I have highlighted the correct selection.



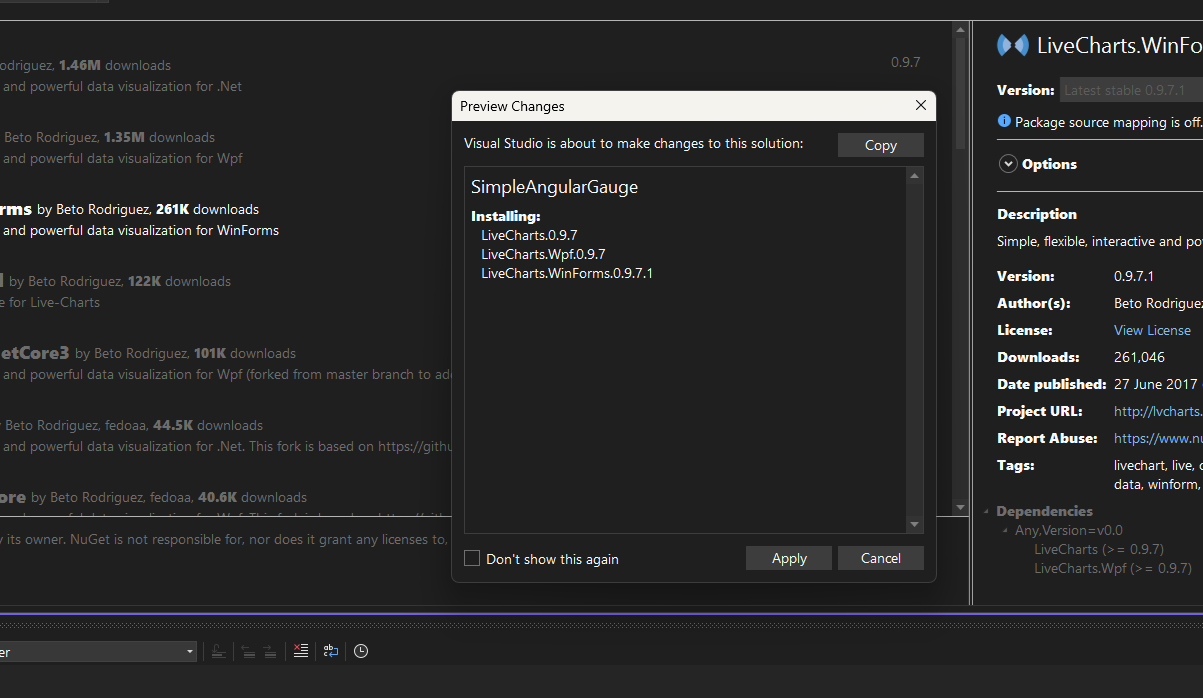
*Figure 8: NuGet Package Manager*

1. The package manager screen will open in the "Installed" tab.
2. You will need to change it to the "Browse" tab and search for 'LiveCharts'. The search bar is under the "Browse" tab.



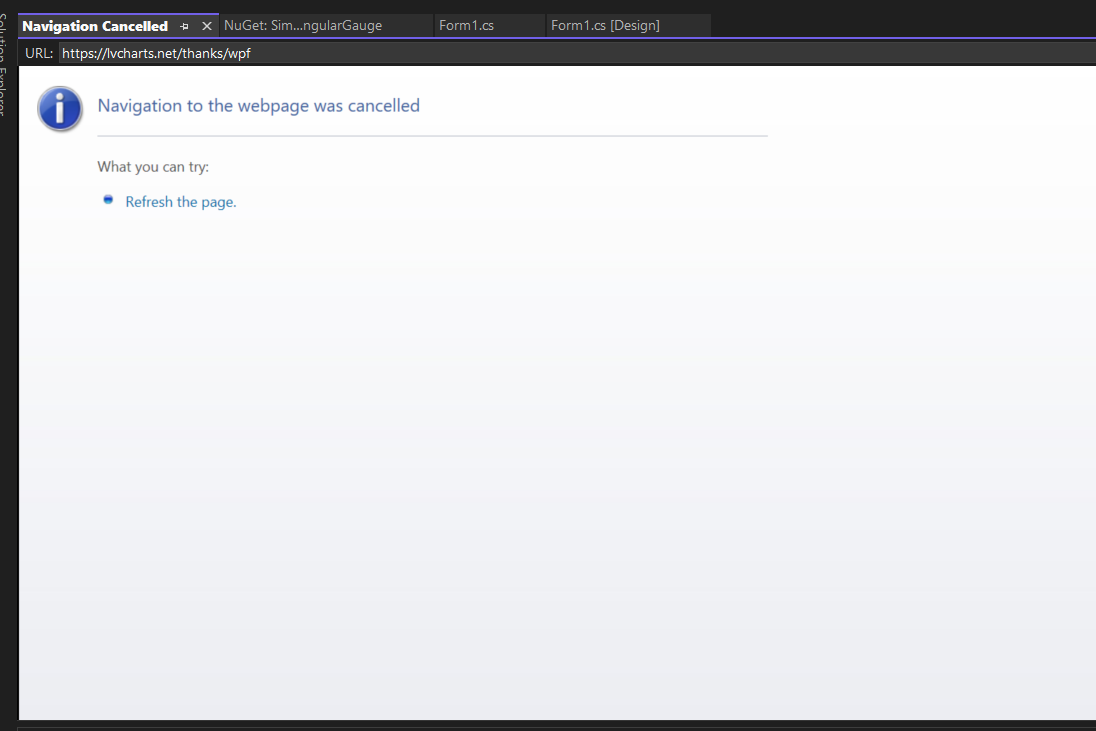
*Figure 9: Live Charts WinForms Selected*

1. Ensure you are installing the correct version of LiveCharts for your the project. The Remote Flight Controller uses LiveCharts.Winfroms.
2. The "Install" button is located on the right side of the packages window.
3. This single selection will download all the dependencies.



*Figure 10: The 3 packages in one selection.*

1. Ignore this webpage. just close it



*Figure 11: Broken Hyperlink - Close it.*

The Live Charts NuGet package is now installed. This should solve any errors associated with the Angular Gauges on the Remote Flight Controller.

### Technical Data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PARAMETER | TYPE | NAME | VALUE | COMMENT |
| TcpClient | Object | tcpClient | (IPAddress,int) |  |
| IPAddress | Object | Ip | int |  |
| AngularGauge | Object | GaugeAltitude  GaugeSpeed GaugePitch GaugeEpitch GuageVspeed GuageThrottle | Location(int) ToValue(int) FromValue(int) Width(int) Height(int) NeedleFill(int) |  |
| UpdateTelemetry | Object | telemObj, t  telemetryUpdate | Altitude(double) Pitch(double) ElevatorPitch(double) Speed(double) VerticalSpeed(double)Throttle(double) |  |
| ControlsUpdate | Object | Controls, C | Throttle(double) ElevatorPitch(double) |  |
| byte[] | Array | buffer | byte |  |
| NetworkStream | Object | Stream | tcpClient.getStream() |  |
| Integer | Int | bytesRead | (Array, int, int) |  |
| JavaScriptSerialiser | Object | Serialiser | JavaScriptSerialiser |  |

### The Code

For a snapshot of Form1.cs, please see Appendix A.

### Adding Components in the Software

**Add Components in the Software:**

The Remote Flight Controller is completely open source therefore, improvements and new features are encouraged. However, implementation will need to follow Event-Driven programming paradigms. Here is a list of examples where programming rules may need to be adhered to:

1. ***Creating a loop*** to continuously monitor data will need to be executed via a Thread. For example, The Telemetry Object is created and populated via a while loop. If this was executed on the main thread, it would freeze the GUI. To prevent this, the program creates a new thread so that it can loop concurrently while the main thread continues to serve the user and any buttons and trackbars they may interact with.
2. ***Any new feature*** that requires the manipulation of the Angular Gauges, gauge labels, or data tables will need to be executed via a delegate and an event. Delegates and Events will be explained in the sub-chapter “Writing the Code”.
3. ***Both classes are*** structured to send and receive data from the Flight Simulator. These are not to be modified. The Flight Simulator will not receive new information. New objects can only be used within the Remote Flight Controller.

New visual features added to the Graphical User Interface are welcomed. Therea are no restrictions on the positioning of new buttons and labels. The repositioning and optimisation of the current Graphical User Interface feature is open to interpretation and any improvements will be committed to the main repository.

### Writing the Code

#### Threads

Threads are an important part of graphical user interface design and event-driven programming. Without the option to create multiple threads, a GUI would suffer continuous inactivity. This would appear as if the graphical user interface has frozen. Threads are used to prevent this.

**Note:** The Remote Flight Controller has a thread. It is executed withing the click handler of the connect button. It runs the method that starts a loop that listens for incoming JSON packages from the Flight Simulator.

The while loop in the Remote Flight Controller is a good example of this situation. Without the ability to create a thread to run the continuous loop, the while GUI would freeze. With two concurrent threads executing separate task without interfering with the other, the program can use one thread for GUI interaction and the other for receiving data from the Flight Simulator.

Thread creation is simple. The program will need System.Threading to create threads.

|  |
| --- |
| using System.Threading; |

Initialise the thread.

|  |
| --- |
| Private Thread listenThread; |

The thread execution can be thought of as a function call. It will start a new threat that runs a function, but will not wait for, or interfere with any other processes already running. once the thread has a value, thisThread.Start() will execute the thread.

|  |
| --- |
| listenThread = new Thread(new ThreadStart(ReceiveData));  listenThread.Start(); |

The thread will then run the function, in this case, named ReveiveData.

#### Delegates and Events

A Delegate can be thought of as a contract or agreement that specifies what kind of task needs to be done. For example, a program may need to do a range of math with two integers.

|  |
| --- |
| delegate int DoMathWithTwoInts(int a, int b); |

Interestingly, you can see that this delegate above does not agree to do addition, subtraction, or multiplication. It agrees to do math. This means that any function we create that requests a and b as an argument can be used with this delegate. But first we need to create the functions.

|  |
| --- |
| void addIntegers(int a, int b)   {    Console.WriteLine(a + b);   }   void minusIntegers(int a, int b)   {      Console.WriteLine(a - b);  }   void multiplyIntegers(int a, int b)   {       Console.WriteLine(a \* b);   } |

These functions all execute different tasks but use the same arguments. This allows a single delegate to execute all three functions and allows for further events to be created using the delegate. Maybe a future feature will divide both integers.

Next, subscribe all three functions to the Event. The subscription is best placed within the constructor.

|  |
| --- |
| event DoMathWithTwoInts addthis += new DoMathWithTwoInts(addIntegers);  event DoMathWithTwoInts minusThis += new DoMathWithTwoInts(minusIntegers);  event DoMathWithTwoInts multiplyThis += new DoMathWithTwoInts(multiplyIntegers); |

If there was no need to execute these sums individually. A single event could be initialised, and all delegates added to it. This would execute and print all results simultaneously. See below.

|  |
| --- |
| event DoMathWithTwoInts mathWithAandB += DoMathWithTwoInts(addIntegers);                          mathWithAandB += DoMathWithTwoInts(minusIntegers);                          mathWithAandB += DoMathWithTwoInts(multiplyIntegers); |

Finally, The Events are invoked. The invoke is used within a function or method with the arguments necessary for the correct invocation of the three functions; add, subtract, and multiply.

|  |
| --- |
| void invokeThis()  {     addthis?.Invoke(1,2);    minusThis?.Invoke(1,2);     multiplyThis?.Invoke(1,2);  } |

# Maintenance and improvements

**NOTICE:** Maintenance should be done by a programmer or software developer.

## How to Maintain the Software

The Remote Flight Controller is a simple application for novice programming; therefore, no maintenance is necessary. However, Visual Studio will require periodic updates. These notifications are found in the bottom-right corner of the editor. Updates are visualised by a bell icon with a red notification alert.

A screen shot of a computer

Description automatically generated

*Figure 12: Visual Studio update notification*

The Live Charts NuGet package will also need to be monitored for any patches. The “Updates” tab is found to the right od the “Installed” tab. Updates are highlighted on the tab by a blue rectangle-shaped notification icon. Instructions to find the NuGet Package management window can be found in Software Documentation chapter.

*A screenshot of a computer

Description automatically generated*

*Figure 13: NuGet update notifications.*

## Future Improvements

There are two major features that are to be added soon.

1. Blackbox recorder
2. Autopilot.

***Blackbox Recorder***

The Blackbox will record the telemetry data to a CSV file that will be in the “bin” folder of this project. Each entry will have the date, time, and each item of telemetry data. The Blackbox feature will not overwrite data history but append only.

*Update: the Blackbox has been implemented.*

***Autopilot***

The button for the Autopilot is already visible on the Controller, however it is disabled until the autopilot can be fixed. It is currently crashing the aircraft. This will be improved in the coming updates.

# Troubleshooting and Testing

## How to Identify and Solve Problems

### Troubleshooting Use Case - 1

|  |  |  |
| --- | --- | --- |
| Error | Cause | Solution |
| Program Crashes upon execution | **Unknown** | Pull again from GitHub repository |

***Frequently Asked Question: Why doesn’t the Remote Flight Controller run?***

It can be difficult to troubleshoot this problem. The program was tested before pushing it to its repository. Download it and try again. If problems persist, please contact me at [*Email withheld ]* or via other social media platforms.

### Troubleshooting Use Case - 2

|  |  |  |
| --- | --- | --- |
| Error | Cause | Solution |
| Missing NuGet Packages | Missing Live Charts Package for WinForms | Install Packages – see FAQ below. |

***Frequently Asked Question: Why am I getting a NuGet Package Error? It’s at the top of Form1.cs!***

A package is needed to support the Angular Gauges on the Graphical User Interface. See “Software Documentation” Chapter for an in-depth guide to installing the Live Charts NuGet Package.

A speedometer with a white needle

Description automatically generated

### Troubleshooting Use Case - 3

|  |  |  |
| --- | --- | --- |
| Error | Cause | Solution |
| Controller throws error when connecting. | “Open Port” button not activated. | Restart application |

***Frequently Asked Question: Why did an error box pop up when I clicked connect?***

The most common fix for this error is to restart the application – including exiting the Flight Simulator – and restarting the Remote Flight Controller and Flight Simulator. Now start the connection process by clicking the “Open Port” button first. It will be disabled after the first click, so it is impossible to accidently click it twice. Now click the connect button on the Remote Flight Simulator. The Controller should now connect to the Flight Simulator.

For a better walkthrough, see “Operation/use” Chapter.

### Troubleshooting Use Case - 4

|  |  |  |
| --- | --- | --- |
| Error | Cause | Solution |
| Controller crashes on disconnect. | Unknown | Restart application |

***Frequently Asked Question: Why did it crash when I clicked “Disconnect”?***

This problem is under review. It is believed that it is caused by the Flight Simulator, however, this is not confirmed and is under investigation.

### Frequently Asked Questions - more

***What does the Blackbox button do?***

The Backbox is currently disabled and will be implemented in future updates.

Update: it is fixed now.

## Testing

### Connect

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Test Method | Expected Result | Actual Result | Comment |
| 1 | Run Application. | Remote Flight Controller Opens. | Remote Flight Controller Opens. | Successful. |
| 2 | Connect To Flight Simulator . | Message box popup confirms connection. | Message box popped up. | Successful. |
| 3 | Connection label change. | Connection label changes to “Connected”. | Label changed to “Connected”. | Successful. |

### Control

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Test Method | Expected Result | Actual Result | Comment |
| 1 | Starts receiving data. | Start Sim button clicked – data visualised in data grid view, labels, and gauges. | Data grid was populated, labels update and gauges move. | Successful |
| 2 | Track Bars activate and send data. | Gauges move, controls data grid view populates. | Message box popped up. | Successful |
| 3 | Track bars control Flight Simulator. | Flight Sim telemetry adjust according to track bar movement. | Telemetry adjusts accordingly. | Successful. |
| 4 | Autopilot controls aircraft | Label changes to Enabled. Autopilot controls plane. | Aircraft crashes | Fail. |
| 5 | BlackBox record data | A Blackbox CSV file is create on the desktop and is populated with telemetry data | Data grid view stops receiving data. | Fail.  Update: Successful. |

# Appendix A.

Code starts on next page. Double click on the code to enter the OpenDocument and view the complete file.

