Teensy++ UFCP user's manual

Synonyms used in this document

Synonym	Description
UFCP	Up Front Control Panel. It is a keyboard that sits directly under Your HUD
HUD	Heads Up Display. A see-through display at eye level showing basic information about Your aircraft
Autopilot	The Autopilot (or Robot pilot) is a system that flies Your aircraft according to preset values like heading, altitude or VVI without any additional pilot interaction.
VVI	Vertical Velocity Indicator. An instrument that shows how fast and in which direction your craft is changing its altitude. Readout is in feet per minute.
GA	General Aviation. A term that refers to airplanes and systems intended to be used by the public and Corporate Sectors.
MCP	Mode Control Panel. This is the interface found on Commercial Airliners for changing the Autopilot presets and for engaging certain Autopilot functions as needed.
DCS	Digital Combat Simulator. It is a simulator for fighter aircraft
IDE	Integrated Development Environment. A program that lets you write, edit, build or compile codes in a given programming language. If it is microcontroller related then usually it comes with built in functionality to upload your programs (in this case they are called sketches) to your microcontroller. Usually this is done with an expensive proprietary hardware that costs you a few hundred bucks.
POE	Power Over Ethernet. A solution that lets you power low consumption hardware (like switches, hubs or microcontrollers) through an Ethernet cable without the need of an external power adapter.

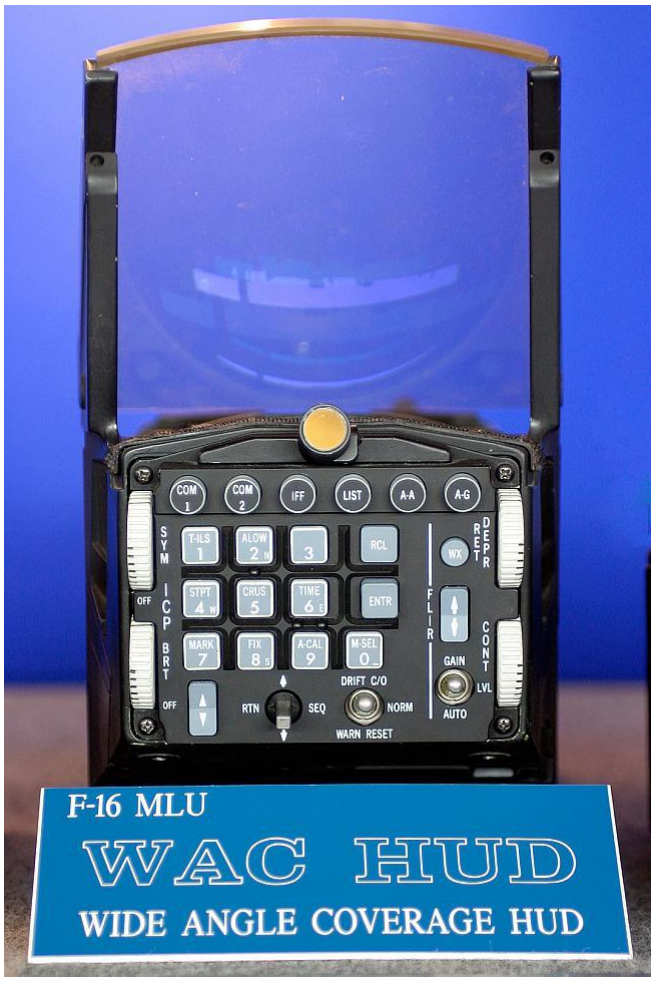
1. Basic Introduction

The Up Front Control Panel (or UFCP for short) is a specialized Autopilot interface intended to be used in applications where the traditional interfaces found in GA Applications would not be useful due to size considerations. Traditionally the MCP would have a separate display for each Autopilot preset and an associated knob for changing that value. In addition to the knobs it also has a pushbutton with an LED illumination that informs the pilot if the function is enabled or not. On large jets the MCP has two sets of controls, as it has a Pilot / Copilot side. Below you can see the MCP panel of a Boeing Airliner

While this type of panel is very easy to use the dimensions of it make it impossible to fit this inside anything but a Commercial Airliner. The major factor here is the width of the panel, as it is wider than most cockpits found in GA airplanes. So in order to fit this into a GA airplane some functions were dropped and the whole layout was redesigned. Below you see an Autopilot panel that you can expect to see in a smaller airplane, like a Cessna 172

Please note the size difference. This Garmin manufactured panel is less than one third of the width of the commercial one, and has a lot less features. You will also be expecting to see the settings on your primary flight instrumentation, as there are no displays in this one.

After taking a look at the commercial and GA planes we see that the Autopilot – even if it is only the most basic one possible – is generally considered an integral part of a modern aircraft. High performance aircraft however have even more limited space, and generally they tend to prefer to use even more compressed controls. This is most obvious in fighter planes, where you would be hard pressed to find a traditional Autopilot control for single seat models. They prefer to have a generic keypad, where instead of switches, knobs and levers you simply type in the appropriate command, and that gets translated to the Autopilot. This poses a problem tough: Traditional GA flight simulators, like the X-Plane or the Microsoft Flight Simulator do not have the required programming to translate for these highly specific systems, so model builders for these programs have to cheat, and use that space to try and hide conventional controls that you can manipulate with a mouse. And if you want to build a home cockpit, which is accurate to your plane then you will need to cheat. Specialized simulators, like DCS or Falcon 4.0 have the programming to translate button clicks on your virtual cockpit, but they lack the capability of simulating GA aircraft. On the left you can see a HUD and UFCP from an F16 fighter.



2. UFCPs and traditional controllers in GA simulators

While most of the time a standard off-the-shelf joystick is enough for most users, there are cases when they are not. Yes, you can always go and buy a new joystick with a quadrillion extra switches, but it will do you no good if the problem is not in the number of switches available.

To start off, PC-s and simulators in general hate toggle switches on joystick inputs. Why? Because they send the switch position continuously. They are dumb controllers, designed to be used with pushbuttons. One button for one function, and if you need it, just press the button and it is all good. Or not, as for home cockpits the positions of those switches are as important, as the command they do. And it is the same with the UFCP. The main problem is not getting a button push event to register in your simulator; the problem is doing something based on a set of commands in the appropriate order.

To show an example: On a traditional MCP to select a heading of 260 degrees you would rotate the heading knob (or press the heading set button and rotate the thumbwheel) until the new heading is set. While the first one is not a problem, you cannot do the second one with off the shelf controllers. The Garmin panel we saw earlier as a GA panel can be simulated in X-Plane, but you would need to use non-momentary pushbuttons and a custom electronic panel to interpret the signals as UP – DOWN commands. And you would need 2 inputs for each function of your autopilot plus a dual pole switch to connect it to the wheel. So for the 4 basic functions you need 8 inputs.

Also, joystick buttons are not reliable sources, as they can point to different functions. Every time you start your program you have to check that each and every button is linked to the appropriate command. Not very fun thing to do if you have about a hundred switches right?

The only other solution is to use a microcontroller, and try to talk to your simulator directly. While this involves getting used to the programming language of the microcontroller, getting a good basic understanding of how electronics work, and learning high quality soldering techniques (or paying top dollar to someone to design and build your circuit boards based on your sketches) it is still considered the best approach.

All in all, we can state, that although these days microcontrollers are cheaper than dirt they are still generally unavailable as PC interfaces for the general public. So our interfaces are still back in the ‘90s.

Interfacing a microcontroller to the X-Plane simulator

While it is relatively easy to tell people to plug in a microcontroller in their PC there are a few issues in this. First, common microcontrollers come in a myriad of shapes, sizes and functions. Second, almost all of them use some kind of a proprietary programming language that is either relatively unknown or hard to master. For example: Most microcontrollers use a C derivative programming language, a dedicated programmer and a dedicated IDE for writing and compiling the codes. Personally I do not think it is fun to toggle pins with bitmasks and low level coding. Maybe I have a problem with my head.

Fortunately people started to realize that they need to give a microcontroller to the masses that is relatively easy to use, and can be programmed in a language that is acceptable for elementary school kids in science clubs. The result of this is the Arduino programming language and the related Arduino microcontrollers. The programming language itself is a derivate of the highly popular and easy to understand C language, but includes a set of BASIC like commands for directly controlling the pins on the microcontrollers. Since Arduino only has about a hundred instructions it is relatively easy to learn. Especially, if you factor in the fact, that you only need to learn around 30 commands – that have talkative names by the way – to start using the language. And once you can use the basics you can choose from dozens of official libraries to help you connect to and use different external hardware's like servos, stepper motors, encoders, LCDs…

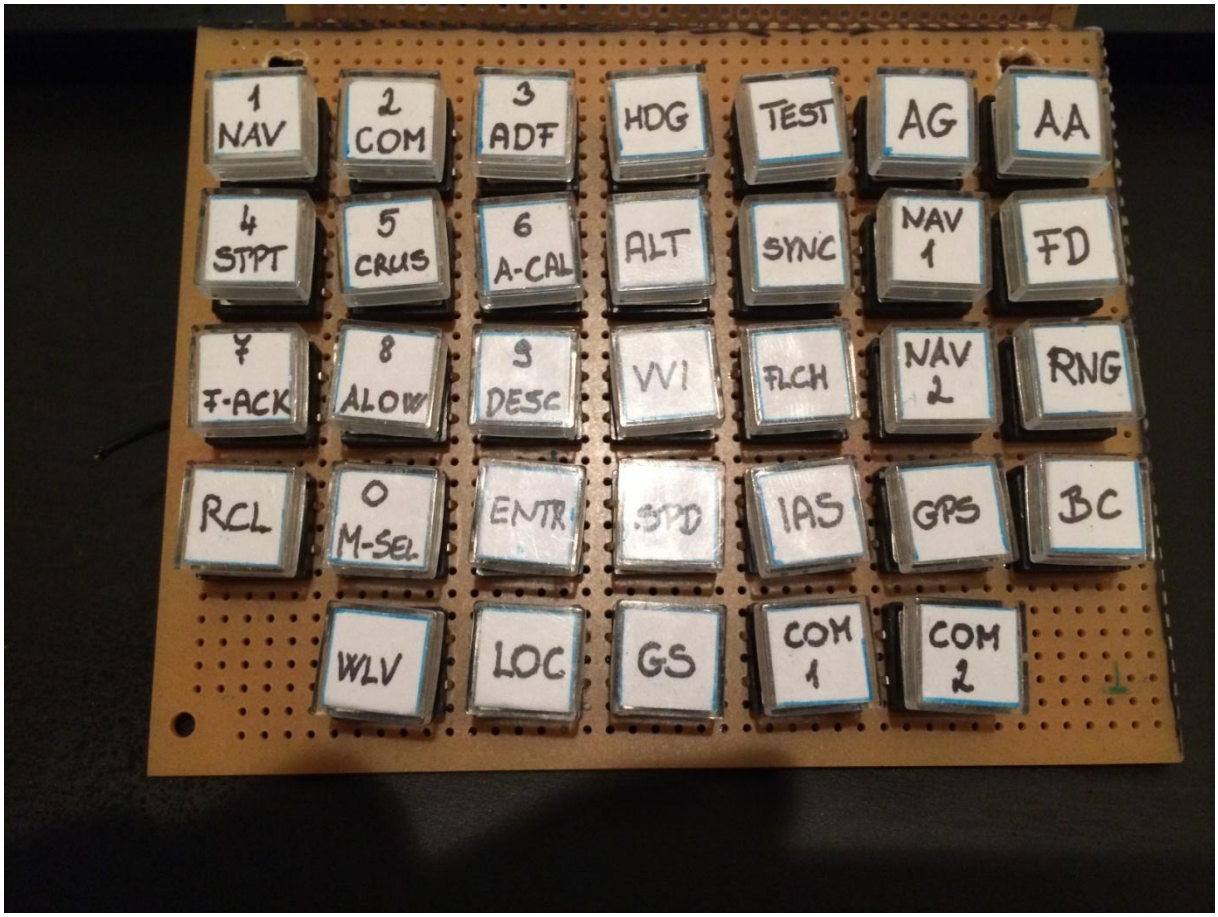
It can also access flight simulator data using an Internet shield, and can be powered using POE. So in theory it can connect to any flight simulator using an UDP protocol, receive the data you work with, and then send back the command inputs you need. It is also a reliable controller, so it will always send the desired input for a button switch, even if it was disconnected or you made modifications to the programming. Arduino is also extremely easy to program, as you only need to connect it to your computer with an USB cable. Unfortunately setting up the UDP data references is time consuming, and requires a lot of research into the output format of your simulator.

There is a solution to this problem, and it is using an officially recognized Arduino derivative, the Teensy board. The Teensy can access more data from the simulator than what is available through UDP output, and it is a lot easier to set up. Unfortunately it accesses it through an USB plug, so you will need USB 2.0 switches if you want to connect more of them at the same time. But it can use all of the Arduino libraries, and can run standard Arduino codes without any modification to them. It is also cheaper than any Arduino controller with comparable features.

Setting up the Teensy controller as an X-plane interface

The basic setup is relatively easy. After you download the Arduino IDE you need to download and install the Teensy specific files. Once you are done, you have a universal IDE that can support any official Arduino board, and the Teensy boards. To use the Teensy as an X-Plane interface you need to select USB Type as "Flight Sim Controls". Unfortunately you cannot use the "Flight Sim controls" USB mode with any of the official Arduino boards. Paul has a good tutorial about interfacing with the X-plane (available here), and we have a small community to help you with your questions about interfacing Arduino and Teensy controllers to X-Plane that you can access here.

From this point on I will show you how to make a generic UFCP with a Teensy++ microcontroller and some basic hardware to be used in X-Plane. It help a lot, especially if you fly with a controller, as course change inputs are a lot faster than using the virtual cockpit and a mouse. Not to mention the fact, that you do not have to release the joystick while flying manually to grab the mouse and click the buttons. Here is a picture of roughly what we are aiming to build



There is very little to be sad about the hardware requirements. You will need a Teensy++ microcontroller ($30 plus shipping), 33 pushbuttons (I use 12 by 12 mm momentary pushbuttons. They cost around $5 for a pack of 40 including shipping if you get them from eBay. The buttons can be mounted on a prototyping board. That is around another $2 including shipping on eBay. You also need to get a pack (100 pieces) of clear plastic caps for the buttons, and that is an extra $2. So your basic hardware is about the price of a lunch at a fast-food restaurant, or less. Fairly affordable I reckon.

Setting up the hardware is a bit trickier. The buttons need to be arranged in a matrix keyboard layout, since we wouldn't want to check 33 inputs one by one, would we? That would be silly, as on the UFCP only one button is pushed at any time. This also means that you do not have to worry about handling multiple switches at the same time, or ghosting problems. This will save us a lot of trouble.

Also, since we expect only one button to be pressed at any given time we can block the program execution after the key press is registered until the button is released. Since we will only be using 12 pins from the total 45 available on the microcontroller we should have free pins to connect a small 20 by 4 character LCD (about $5 on eBay including shipping from china). This is the first part that requires caution when purchasing. The Teensy 2.0++ controllers run on 5 volts, while most character LCD-s expect an input of 3.3. So you have to be careful to buy one that is designed for +5V operation. You can also go for the Teensy 3.x series, as they are cheaper, faster and run on 3.3V like most Arduino hardware. For this documentation we will use the 2.0++.

How to use the keypad, or what do those buttons do?

The keypad has two basic regions. On the left side there is a 3 by 4 numerical keypad with RCL and ENTR buttons at the sides of the "0" key. This is used for programming the onboard systems. The rest of the keys are used for sending simple commands to the simulator (with the exception of the RNG key, more on that later). All modes require the Autopilot to be fully on, if the servos are not engaged then the Autopilot will simply show you a visual reference on your PFD, but will not fly the plane.

The buttons refer to the following Autopilot commands (buttons are listed top to bottom and in a left to right order):

WLV	Wing Level Mode. The Autopilot will hold the plane's wings level, keeping you fly at your current altitude and heading.
LOC	VOR/LOC mode toggle. The autopilot will fly to the selected directions. If you select GPS as the VOR/LOC source the Autopilot will fly the route from the Flight Management Computer.
HDG	Heading Select Toggle. The autopilot will fly the plane in the selected heading.
ALT	Altitude Hold Mode. The Autopilot keeps the plane at the selected altitude. Airplane speed is not monitored with this mode.
VVI	The autopilot will climb or descend at this rate. Airplane speed is not monitored.
SPD	Auto throttle. The Autopilot will use whatever throttle setting is required to keep the indicated airspeed at the pre-set value. Mode will not change altitude or climb settings if the maximum thrust is inadequate.
GS	Glide Slope mode toggle. The Autopilot will attempt to capture and fly the glide slope (the vertical component) of the ILS beacon dialed in to the NAV radio. Only works after the LOC mode actually captured the ILS beacon, and if you are UNDER the Glide slope when engaging it.
TEST	Tests the Autopilot warnings.
SYNC	Toggles the Altitude Hold mode, and also sets your current altitude as the target.
FLCH	The Autopilot will change altitude to the preselected settings while keeping the airspeed at the preselected values. Useful for high altitude flight, where the engines do not have a lot of extra thrust. Climb rate is selected by adjusting the throttle manually to the desired setting. Disengages VVI and SPD modes.
IAS	The autopilot will keep the plane at the preselected airspeed by climbing or descending. Auto throttle is disabled; the throttle has to be adjusted manually.
COM1	Listen in to the COM1 channel you dialed into the Radio.
AG	Selects Air to Ground weapons. Currently it is pointing to weapon select down

NAV1	Sets Autopilot HIS source as NAV1
NAV2	Sets Autopilot HIS source as NAV2
GPS	Sets Autopilot HIS source as GPS receiver
COM2	Listen in to the COM2 channel you dialed in to the Radio
AA	Selects Air to Air weapons. Currently it is pointing to weapon select up.
FD	Autopilot mode change. If the Autopilot is not in AP mode, then it will turn it on in AP mode (servos disconnected). If the Autopilot is in AP mode then it will turn it on in FD mode (servos connected).
RNG	Changes the Radar range, or the Displayed Map range. It is evaluated as a command string.
BC	Engages VOR/LOC back course mode. You can fly the ILS approach in a back course mode, but you will have no GS information.
0 – 9	Numerical keypad buttons. Used to input command strings
RCL	Clears the current command string. Use only if you made an error and need to start over.
ENTR	Tells the autopilot to parse and evaluate the current command string. After evaluation the string is cleared.

Using command strings with the UFCP

The UFCP can accept 11 types of command strings.
The basic syntax is [Function selector][Optional parameter][Desired setting][ENTR]

Detailed syntax and what they do:

[1 NAV][1 or 2][frequency][ENTR]	Sets the Nav1 or Nav2 active frequency to the new setting. Frequency is expected in the 118.120 format, but without the dot. If you do not enter a frequency, then the active and standby frequencies are flipped.
[2 COM][1 or 2][frequency][ENTR]	Sets the Com1 or Com2 active frequency to the new setting. Frequency is expected in the 129.135 format. If you do not enter a frequency, then the active and standby frequencies are flipped.
[3 ADF][1 or 2][frequency][ENTR]	Sets the ADF1 or ADF2 active frequency to the new setting. Frequency is expected in the 118.500 format. If you do not enter a frequency, then the active and standby frequencies are flipped.
[4 STPT][heading][ENTR]	Sets the Autopilot Heading selector to the new value. Expects heading as a 3 digit number with no decimals, like 008
[5 CRUS][IAS in knots][ENTR]	Sets the Auto throttle speed to the new value. Speed is expected as a 3 digit number with no decimals, like 080.
[6 A-CAL][hg In][ENTR]	Sets the barometer to the new setting. Expects a 4 digit number without decimals (like 2992). If the new barometer is not present it resets the Barometer to the standard value of 29.92 inches of Mercury. Will not accept millibars!
[7 F-ACK] [3 digit number][ENTR]	Sets the Autopilot altitude setting to the new value. Expected parameter is an FL value, tough it is possible to enter any FL – even 0 is accepted as long as it is in a 000 format.

[8 ALOW][ENTR]	Terrain following mode toggle. This is not a true command string in the sense, that it does not accept any parameters.
[9 DESC][0 or 1][2 digit number][ENTR]	Sets the Autopilot VVI to the new value. After the mode selector a 0 is uses to indicate a negative number, while a 1 is used to indicate positive numbers. Resolution is 100 fpm.
[0 M-SEL][3 digit number][ENTR]	Sets the Autopilot Decision height to the new value.
[RNG][1 to 6][ENTR]	Sets the Map / Radar range to the new value. Ranges are 1, 4, 10, 20, 40, 80, 160

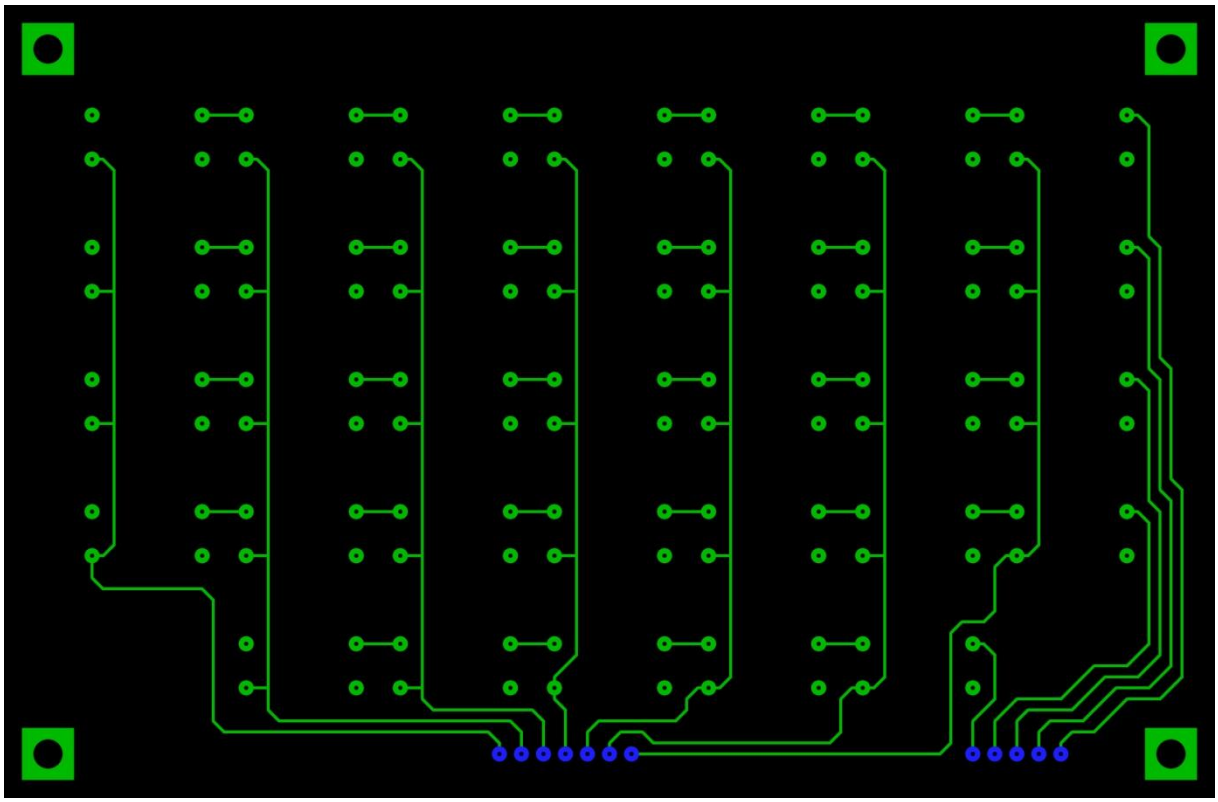
The UFCP will not function (meaning no buttons are registered and no Flight Sim data updates are pulled if the simulator is not running OR the supply volts are too low for the instrument in the flight sim. So if your aircraft experiences total electrical power loss, or if power is cut to the instrument in the sim then the UFCP will do nothing, and the LCD will go black. You have been warned.

Background information, or how does it work?

The code, layout and all information regarding the controller is Open Source. You are free to make modifications to it to suite your needs, add additional functionality and so on. You are required to mention the fact that the basic design is from me and the modifications tough. All derived products must also be open source. If you want to use this project as the basis of a commercial product then you need to negotiate the terms and conditions. Selling the project as-is is requires written permission, and is only accepted when you only charge for shipping, parts and labor costs from work hours required to assemble the finished product.

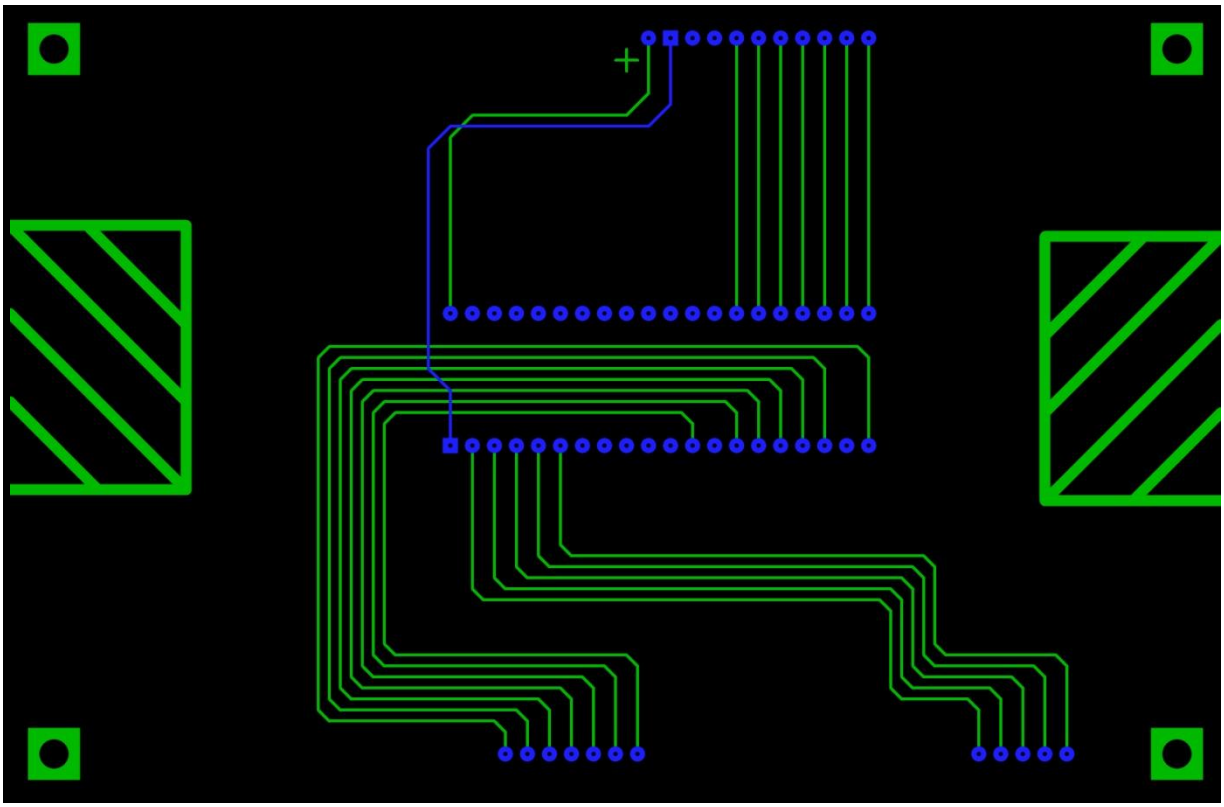
The electrical boards are easy, and can be manufactured at a relatively low cost. The switches are tied into a matrix connection. This means, that they are arranged in a rows – columns formation, where the left side pins are connected together in the rows, and the right side pins are connected together in the columns. The microcontroller has a pin connecting to each row / column. If you press a switch it will connect its row and column wires, and the button press will be registered in the microcontroller. While theoretically it is possible to detect simultaneous key presses as long as they are on different rows and columns the panel has no need for such features.

In my example the pushbuttons had 4 pins. The Pins Top Left and Top Right were always connected. Also the Pins Bottom Left and Bottom Right were always connected. Pushing the button connected the top and bottom pins together. This made things a lot easier in terms of PCB creation, resulting in a single sided PCB layout.



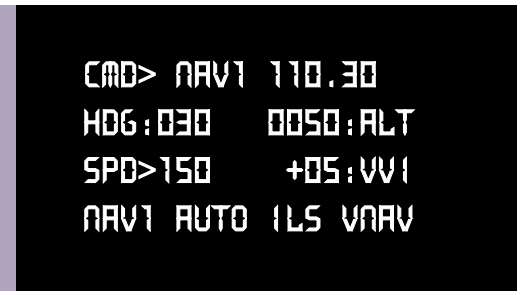
The only place where the double side is needed is the pin connections, as the single side layout would place them on the same side as the buttons, pointing up. If you use thin wires instead of pins, then it can be done with a single sided PCB as those will fit under the face plate (use spacers in this case).

The next thing that is required is to actually plug in your microcontroller into something. Unfortunately this is not as easy as it sounds. The basic idea here is to have a series of shields on top of each other. If you stack the electronics into layers, then the overall size of the controller goes from two dimensions into three dimensions. And that is good since you need a box to fit this into, and then you might want to place that box in a cockpit frame. So the microcontroller PCB is the same size as the keyboard, and can be mounted directly on top of it, extending it towards the back. Since PCBs in general are not too strong I would recommend installing a clear acrylic sheet between the keyboard and microcontroller boards. A 6 mm or ¼" sheet should be strong enough to give support to your keyboard. The screw holes on the sides are marked for 3mm diameter, but a 1/8"screw will fit, if you drill them a bit larger and have no easy access to Metric screws. The microcontroller board is a two sided layout, as I did not like the idea of using a wire bridge on a single sided layout. The top connectors are there to provide a header for a character LCD expansion. The first 2 wires on the top header are for powering the LCD, while the rest is for data communication. Since the project uses the FastLCD Arduino library any compatible Character LCD-s can be used. For best results I recommend using a 4x20 layout. This will give you a top line for showing the actual command as it is being input into the controller, and a bottom line for displaying Autopilot related data. The two middle lines can be used to display the 4 standard autopilot presets: Heading, Altitude, Vertical Speed and Autothrottle. The two marked areas on the sides are cutouts, as the microcontroller will face towards the keypad, and you need space for the USB cable (and possibly to the external 5V power supply if needed).



Optional Character LCD expansion for the microcontroller

The UFCP can be expanded with a 4x20 Character LCD for easy data overview, or for in use in Home cockpits. While this is not strictly required, it is an advisable expansion, as it will add a lot to the functionality of the UFCP. The following data is displayed on the LCD screen by default (you are more than welcome to change the programming, should you feel an urge to do so).



Upon entering an accepted command the Command line will change to the text of ACCEPTED, and display that message for roughly 5 seconds, or until the next a key is pressed.

Entering a command that is not recognizable by the panel will result in the command line changing to show the text ERROR, and displaying this for roughly 5 seconds or until a key is pressed.

The two middle lines contain information about the 4 standard autopilot settings. The general syntax is a 3 letter identifier on the side of the screen followed by a : if the mode is not currently active or a > if the mode is currently active. After the mode display character you will generally see a set of numbers. HDG is a 3 digit number with all leading zeros shown, and displays the preselected magnetic heading for the Heading Select mode. SPD is a 3 digit number with all leading zeros, and displays the preselected speed for the various Speed Hold modes the autopilot can fly. ALT is a 4 digit number,

and displays the preselected flight altitude. The number displayed is in Flight Levels, so it has a unit of a hundred feet. The VVI is the preselected Vertical Speed in 100 feet per minute. A + or – sign will indicate if the preselected value is a climb or a dive. **Due to current autopilot limitations no value higher than 9900 feet per minute is accepted. If you need faster descent for any reason, then you have to fly that leg manually.**

The bottom line is a status bar for the currently selected and active Autopilot options. It consists of 4 Blocks. Each of them can display a single value out of a preset number of choices.

Starting from left to right:

A. Block A displays the currently active HSI/Autopilot source. It can display 3 possible values: NAV1 NAV2 or GPS .
B. Block B displays information about the Autopilot being enabled or not. It can display 3 values: OFF ON or AUTO. OFF means the Autopilot is turned off. ON means it is in Flight Director Mode but without the servos being engaged. AUTO means the autopilot is currently flying the aircraft.
C. Block C displays information about the route (horizontal) navigation mode. If the autopilot is turned off, or if no route navigation is selected then it will be empty. Otherwise it will show one of the following values: WLV is displayed if the Autopilot is in Wings Level mode. ILS is displayed, if you are in ILS approach mode. NAV is displayed if the Flight Management Computer is navigating the aircraft according to the preprogrammed waypoints. HDG is displayed, if the Autopilot is in Heading Select mode.
D. Block D displays information about the Vertical navigation mode. If the autopilot is turned off, or if no vertical navigation mode is selected it will be empty. Otherwise it will show one of the following values: VNAV means, that the autopilot is setting the altitude based on the preprogrammed settings in the Flight Management Computer. Please keep in mind that you can change the Vertical speed in this mode, but you can only change the preselected altitudes by changing the flight plan stored in the Flight Management Computer. This setting is currently NOT available through the UFCP interface. GS indicates, that the autopilot is currently flying the Glide Slope from an active ILS approach. GS will only show AFTER the autopilot has locked on to the glide slope of the approach, so simply pressing the GS button on the UFCP will not result in a change of the displayed value in Block D. Finally ALT indicates, that the autopilot is currently in Altitude hold mode. Usually this is the setting displayed between arming the GS mode (pressing the button), and the autopilot actually locking on to the vertical segment of the approach.