Imperial College London

BEng Individual Project

Imperial College London

Department of Electrical & Electronic Engineering

Cost Effective Intelligent Platform Management Interface for PCs

Author:
Raymond R. Zhu

Supervisor:
Dr. Benjamin Hou

Second Marker:
Dr. Thomas Clarke

June 18, 2021

Contents

1 Abstract **4**

2 Introduction **5**

2.1 Motivation . . . 5

2.2 Report Structure . . . 5

3 Background **7**

3.1 Prior Solutions . . . 7

3.1.1 Dell Remote Access Controller . . . 7

3.1.2 Raritan Dominion KX IV . . . 8

3.1.3 Pi-KVM . . . 8

3.1.4 TinyPilot . . . 9

3.2 Stakeholders . . . 9

4 Requirements Capture **10**

5 Analysis & Design **11**

5.1 Single Board Computer . . . 11

5.2 Video Capturing Device . . . 12

5.3 ATX Controller . . . 12

5.4 Web UI . . . 14

6 Implementation **16**

6.1 TinyPilot . . . 16

6.2 Adding features to the TinyPilot . . . 17

6.2.1 Booting up the Motherboard . . . 17

6.2.2 Power Out Detection . . . 18

6.2.3 Watchdog Timer . . . 18
6.2.4 Environmental Variables Logging . . . 19

7 Testing **21**
7.1 Power Cut Simulation . . . 21
7.2 Network Disconnection Simulation . . . 21
7.3 Peripheral Input Test . . . 22
7.4 Power Loss Logging Test . . . 22
7.5 General Results . . . 22

8 Evaluation **23**
8.1 Physical Deployment . . . 23
8.2 Comparison to the Original TinyPilot . . . 24
8.3 Comparison to the Pi-KVM . . . 24
8.4 Change in Project Direction . . . 25
8.5 Legal Considerations . . . 25
8.6 Known Bugs . . . 26
8.6.1 Video Input Black Bar . . . 26
8.6.2 Temperature Sensor Notification . . . 26

9 Conclusion **27**
9.0.1 Overall Thoughts . . . 27
9.0.2 Future Work . . . 27
9.0.3 Personal Reflections . . . 28

10 User Guide **29**
10.1 Hardware Setup . . . 29
10.2 Software Setup . . . 31

11 Appendix A - Testing Results **32**

Chapter 1

Abstract

The KVM-over-IP switch is a device that is able to relay keyboard and mouse inputs, and video output between two devices. It allows users to access headless computer (no peripherals or monitor) via the web browser of another device.

This goal of this project is to produce a budget-friendly KVM-over-IP solution to enable remote control of desktop devices on a hardware level for the education sector and small/medium sized businesses.

The switch will be based on the Raspberry Pi 4B, and the **main** functionalities implemented are: keyboard and mouse emulation, video output streaming, emulation of physical power button pressing, and an email notification system for whenever there is a power cut.

Although there are many other similar products available in the market, all of them are for different groups of target consumer which has different budgets. This project is based on the TinyPilot, another open-source KVM-over-IP solution, and additional features were added to meet the specific requirements from Department of Computing and Department of Electrical and Electronic Engineering in Imperial College London. The final product is lightweight, user-friendly, and has shown great reliability - successfully passing all the different test cases.

Chapter 2

Introduction

2.1 Motivation

During the COVID-19 pandemic, work from home has become the norm for most of the society. By connecting to the company/university local area network using a Virtual Private Network (VPN), users are able to access their work computers remotely via Secure Shell Protocol (SSH) and conduct their work as if they were in the office/lab.

At Imperial College London, students and faculty alike are relying on this measure to work remotely as well. However, unscheduled and unexpected power outages can cause disruption to individual college issued computers, where physical presence is needed to bring them back online. This remains an unsolved problem, hence the need for an automated solution.

Many other home lab users or Small and Medium-sized Enterprises (SME) also experience similar issues when their server freeze and needs to be restarted, and they are forced to connect mouse, keyboard, and monitor to the device in order to reboot and reconfigure the device.

Although there are already similar Intelligent Platform Management Interface (IPMI) solutions on the market, most of these tend to be very expensive as they are loaded with complex features and sometimes even comes with a licensing fee since their target audience are mostly large enterprises who are able to afford it. However, these solutions will unfortunately not be suitable for the education sector and SME as it is way above their price range.

The aim of this project is to develop an affordable, easy to use KVM-over-IP switch using the Raspberry Pi 4 that meets the requirements of Department of Computing and Department of Electrical and Electronic Engineering, to be deployed on the departments servers in the future.

2.2 Report Structure

Chapter 3 (Background) will go through what existing solutions there are, as well as the pros and cons of each product, and how the existence of this project could affect different stakeholders.

Chapter 4 (Requirements Capture) will list all the requirements given by the Imperial College faculty, as what they expect to see in this project since it will be used to manage the servers on campus.

Chapter 5 (Analysis & Design) and Chapter 6 (Implementation) will discuss how the components were picked for this project, along with some of the design decision made over the development of the product, as well as the system architecture and software implementation of specific features.

Chapter 7 (Testing) will explain how the product was tested for its reliability.

Chapter 8 (Evaluation) will compare this implementation to the other hobbyist products in the market, and the known bugs existing in the final prototype.

Finally, chapter 9 (Conclusion) sums up this paper and shares some overall thoughts and personal reflections, as well as future work to improve the product.

Chapter 3

Background

The KVM-over-IP switch is a device that is able to relay keyboard and mouse inputs, and video output between two devices. It allows users to access headless computer (no peripherals or monitor) via the web browser of another device.

In this chapter, we will be comparing existing commercial and homemade KVM-over-IP solutions, as well as discussing their feasibility in the context of problem we are tackling. Additionally, the stakeholders who will benefit from a product as such existing will also be considered here.

3.1 Prior Solutions

3.1.1 Dell Remote Access Controller

Dell Remote Access Controller [1] is Dell's proprietary platform for accessing and configuring servers remotely. The platform comes in two formats, as a separate (DRAC) expansion unit usable on any supported hardware, or as an integrated (iDRAC) functionality in their PowerEdge product line for servers. The former is no longer manufactured, with the DRAC 5 as its last model being released in 2006.

Since then, Dell has moved to the integrated platform, providing support for their server grade computers exclusively. The platform is based on Linux and Busybox, with the source code available, however, without a build environment.

Apart from the basic remote console and power management functionalities, iDRAC also supports virtual drive. This allows the server administrator to virtually load a disk-image to the server as if it was physically inserted, which could prove to be a valuable feature for those who are looking to remotely install operating systems.

The downside of iDRAC is that it only supports Dell's PowerEdge servers, and there is no way to port the solution on a non-proprietary device. These servers are priced relatively steeply depending on the hardware specification, with their top selling general purpose mid-range level model [2] (PowerEdge T640 with a 10 core / 20 thread Xeon CPU) priced at £2,937.32 per device. Additionally, an one time licensing fee of $300.00 is required to use iDRAC per machine.

A solution as such to replace the DoC machines, which are mostly custom built PC for specific purposes, is not viable both from the budget and functionality aspect.

Finally, HP also has a similar product named Integrated Lights-Out (iLO) [3] which is very similar to Dell's iDRAC implementation. It is also only available on HP's ProLiant servers, and for the same reason is not a solution we can use for our problem.

3.1.2 Raritan Dominion KX IV

The flagship product in Raritan's Dominion product line, the KX IV [4], is packed with features that makes it an ideal KVM over IP switch for many large enterprises. Unlike the iDRAC, all of the Dominion switches are stand-alone devices that servers plug into.

The switch is able to allow up to 8 users at once to remotely control up to 64 servers, making it ideal for industries such as TV broadcasting centers, large scale cloud services, or even government operations. It claims to have "military grade security and encryption with its use of Advanced Encryption Standards (AES) and Federal Information Processing Standard Publication (FIPS) 140-2" [4].

The KX IV is able to encode and stream up to three 4K video sources at once, whilst maintaining 30 frames per seconds and a low latency of 50ms. It is also equipped with the virtual disk drive feature to allow remote OS installation. Finally, it has incredible reliability with a fail-safe measure of two power and Ethernet ports, ensuring that the switch never comes offline unexpectedly.

As the product comes with top-of-the-line specifications, it has a steep price point as the main target consumer are large enterprises. The Dominion IV pricing starts at $679.00 for the 101 model (supports one client and one server) and goes up to $10,299.00 for the 864 model (supports 8 clients and 64 servers). Additionally, Computer Interface Module [5] would be needed to connect the server to the switch, with each module priced at $176.00. As can be seen from these numbers, SMEs and education institutions simply cannot afford to employ an entire infrastructure using these switches.

3.1.3 Pi-KVM

One of the most notable open-source DIY KVM over IP solution is the Pi-KVM [6], developed by the GitHub user Mdevaev (Max Devaev). The project started off as his personal hobby, however due to the large success of it, he is currently working on the project full time as the lead developer and commercialized the production of these devices on his website.

The first iteration of the project, Pi-KVM v0, was designed for the Raspberry Pi 2 and 3. The downsides of those devices the lack of an USB Type-C port, hence in order to emulate peripheral input, an additional Arduino Pro Micro was required to interface with the device. Despite the inconvenience, the product gained enough popularity in the home lab users community to encouraged him to push the Pi-KVM further, implementing more requested features and increasing its reliability.

The Pi-KVM v2 was based the Raspberry Pi 4, and there are some drastic improvements for the previous iteration. It now supported on board USB peripheral emulation thanks to the USB Type-C port. Although the port is meant for charging, it is possible to use it to emulate the USB peripherals using a USB-C Data/Power Y-splitter. Additionally, during the development of the v2, he also wrote his own video encoding library specifically for the Pi-KVM project, μStreamer [7], as opposed to using mjpg-streamer [8].

The benefit of μStreamer comes from its specialization for the purpose of streaming the video output for the Pi-KVM. In terms of hardware utilization, μStreamer supports multi-threaded MJPG encoding and OpenMax IL hardware acceleration on the Raspberry Pi, reducing the encoding time by utilizing all the hardware performance available on the Pi. On the software side of things, μStreamer allows the option of changing the streaming resolution on the fly, as well as the ability to skip frames when streaming static images to reduce HTTP traffic. The library is open-source as well, and many DIY Raspberry Pi KVM projects have reported an increase in performance both in terms of latency and quality of the video stream after switching to μStreamer.

Currently, Max is working on the 4th iteration of the Pi-KVM, supporting even more premium features such as virtual disk image and mass storage drive emulation that usually only commercial KVM switches include. Additionally, the pre-order price is merely $130.00, making it stand out in the competition in both the DIY market in terms of features and the enterprise-grade market in terms of pricing.

3.1.4 TinyPilot

Another popular alternative to Pi-KVM in the hobbyist market is the TinyPilot [9], developed by Michael Lynch. In terms of functionalities, it is very similar to the Pi-KVM, fulfilling all the basic requirements such as USB device emulation and low-latency video streaming (it also uses μStreamer). However, since Michael is only working on the project part time, it lacks some features such as ATX control and virtual disk mounting.

The benefit of this project over the Pi-KVM is however that the open-sourced codebase is significantly more lightweight, with the back-end written purely in Python, and the front-end written with a mixture of HTML, CSS, and Javascript. All the code is executed on runtime and there is no compilation required. This makes the codebase easier to work with and does not require much refactoring when adding features to it compared to the Pi-KVM.

TinyPilot makes the ideal candidate for the project to be based on, as it allows adding missing features and carry out testing with ease. Since the codebase is open-sourced, this will not implicate any legal issues, and I have even been in touch with Michael and gotten his approval and support for this project.

The TinyPilot is also available to purchase, with the hobbyist kit being priced at $169.99, while the Voyager, which comes with a custom 3D printed case and a cooling fan, is available for $299.99.

3.2 Stakeholders

Large corporations with dedicated server rooms consisting of hundreds of server racks will either seek out the integrated solution such as Dell or HP, or build an entire remote access infrastructure with standalone KVM switches like the Raritan Dominion KX product line.

In contrast, a budget friendly KVM-over-IP switch could benefit the home-lab users, SME, and the education sector. These stakeholders will often be satisfied with having full hardware level remote access to their smaller server clusters.

As aforementioned in the introduction, the product developed for this project will be specifically catering to the requirements of the Imperial College London Department of Electrical and Electronic Engineering and Department of Computing. Hence, some of the products functionalities will be less applicable for regular users, as they for example are less likely to experience irregular power surges. However, having extraneous features that is less useful some users but crucial for other users is often important in a product as such.

Another group of users who could benefit from this project are custom PC builders. Personally, inconveniences has arises when working on custom-built PC projects, where all the USB peripherals and display cables would need to be connected when configuring the BIOS and installing the OS. This often implies that the current setup would have to disconnect since most users only have one set of keyboard and mouse, and plugging in a HDMI cable on a wall-mounted monitor could sometimes also be a nuisance.

With the KVM switch, however, all the PC builder needs to do is by connecting the HDMI cable and an USB Type-C cable to the Pi. Doing so, they will instantly have access to the headless machine from boot, including the BIOS, from any device that is capable of running the web UI.

Chapter 4

Requirements Capture

This projects aims to bridge the gap between the laborious manual reboots and the overwhelming price tag on enterprise grade KVM over IP switch, providing users with the ability to reboot and access their servers on a hardware level remotely while maintaining budget friendly. After consulting the stakeholders (Dr. Benjamin Hou for DoC and Mr. Richard Stephenson EEE), the solution is aiming to meet the following set of requirements, in descending order of priority:

- Boot up the PC from a turned off state.
- Reboot the PC from a frozen state.
- Emulating hardware level keyboard and mouse input.
- Receiving real time video output from boot incl. BIOS / Safe-mode.
- Notify system administrator via email whenever the power goes out.
- Logging the duration for each power outage.
- Logging environmental variables (temperature, humidity, etc.)

Additionally, some additional features that was not requested but will be able to support the user experience are:

- Fully functional web UI that supports full screen for immersive workflow.
- Change host-name and port to support any network infrastructure.
- Configurable video resolution in case of bandwidth limitations.
- Clipboard access to paste text from client to server.
- Screenshot the server video output and save to the client machine.
- Displaying a short keystroke history queue for confirmation purposes.

The goal is to produce a device that is self-sufficient with minimal need for post-deployment support, without compromising reliability or user-friendliness.

Chapter 5

Analysis & Design

This chapter will give an insight of the design decisions made behind each components chosen for the implementation, and how these choices support the requirements given by the departments.

5.1 Single Board Computer

The core of the KVM over IP switch is the Single Board Computer (SBC). It is responsible for everything from receiving and relaying the mouse and keyboard inputs from the client, to encoding and streaming the video output feed from the server computer.

Naturally, there are multiple suitable SBCs to be used for the switch, and each of them comes with their advantages and disadvantages. There are a few criterion that the decision of which SBC to use will be based on: pricing, USB Type-C availability (for peripheral emulation), performance, and power consumption.

There are four main SBC options that stood out during the preliminary search, and here are their specifications.

SBC	Processor	USB-C	TDP	Pricing
PROFIVE NUCV-1202B [10]	Duo-Core 2.3 - 3.2GHz	No	12 - 25W	£148.59
ODROID-N2+ [11]	Quad-Core 1.8 - 2.4GHz	No	Up to 6.2W	£96.75
LattePanda Delta 432 [12]	Quad-Core 1.1 - 2.4GHz	Yes	Up to 6W	£181.92
Raspberry Pi 4 [13]	Quad-core 1.5GHz	Yes	Up to 5.1 W	£49.99

Table 5.1: Specifications on the four selected single board computers

The ideal processor would actually not be the fastest, since the main CPU bound task is video encoding and streaming. Having a faster CPU is most likely going to lead to higher pricing on the SBC as well as higher power dissipation. The lower frequency limit and the ability to potentially underclock [14] the device is more important in order to increase stability if undervolting is taking place [15].

Finally, another factor to consider that was not put in as criterion - the availability of online resources. Since this is a DIY style project, information is valuable whenever running into unforeseen issues. The Raspberry Pi 4 should have the most available resources but in order to confirm this, the date from Google's search trend of the different devices.

Combining all the factors of consideration, the Raspberry Pi 4 makes the ideal candidate. It's pricing beats out the rest of the competition by a large margin, and boasts good enough hardware specification to carry out the necessary task. Furthermore, it comes with the USB Type-C port so there is no need for an add-on board to emulate USB peripheral input. Finally, there is an

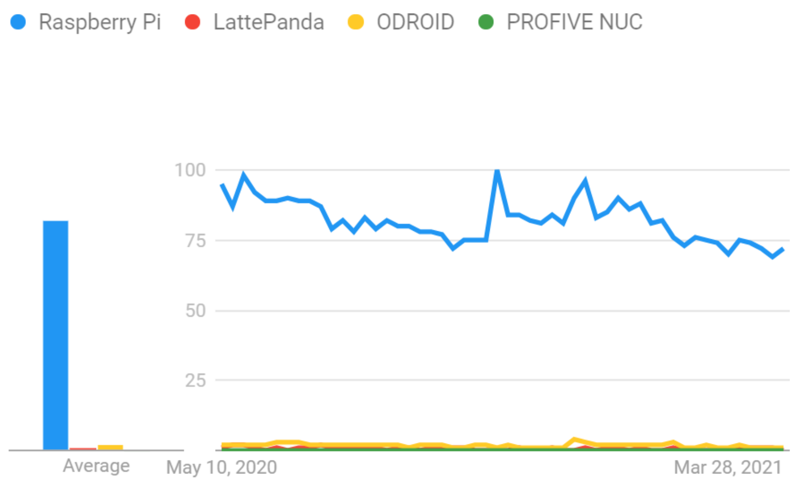


Figure 5.1: The trend of using the keywords on Google Search in the past 12 months

active online community to seek help from and even has complete, open-sourced projects of similar nature like the TinyPilot as aforementioned will be used as the skeleton of the project.

5.2 Video Capturing Device

There are two solid avenues to explore in terms of feeding the HDMI signal from the server into the Pi. Either a HDMI-to-CSI2 bridge [16] can be used to interface with the Raspberry Pi's CSI port, or a HDMI-to-USB capture card [17] can be used to plug into the USB port. Both methods work fine as Max Devaev (developer of Pi-KVM) has confirmed on his GitHub repo, however the capture card will have slightly higher latency [6].

The benefit of using the capture card over the bridge adapter is that the latter is considered an uncommon component that needs to be shipped from abroad. The delivery time is at minimum a whole month long and the pricing is around 50% higher than the capture card. Additionally, the capture card is more flexible since it plugs into the USB port, which makes the overall device neater and can easily be assembled/disassembled from scratch.

Since it is unlikely that any latency sensitive tasks such as gaming or Voice-over-IP will take place on the server machine, latency is not a big factor of consideration here. Hence, following the overarching objective of the project - to produce a cost-effective device, and the USB capture card is the best value option.

5.3 ATX Controller

The design options available for this section was one of the highlights of this project, as it required combined knowledge of electronic engineering and software engineering.

Since one of the requirements is to be able to start up and reboot the server on a physical level, the device needs to be able to interface with the motherboard directly. On the motherboard, the front panel connectors often has a layout as on figure 2.2.

The motherboard is powered on and restarted when #PWRBTN and #RSTCON are grounded respectively [18], as this is what happens when the power or reset button on the chassis is being pressed.

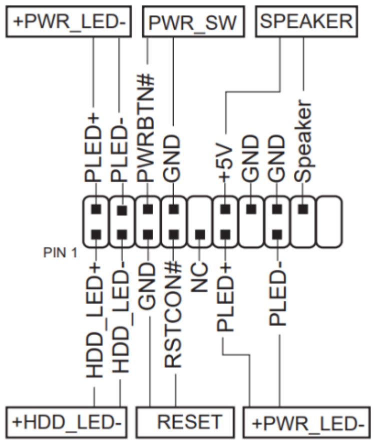


Figure 5.2: A typical motherboard front panel connector layout.

The initial idea was to create a breadboard circuit, to incorporate the motherboard front panel IO pins, chassis front panel connectors, and the Pi's GPIO pins. All of the pins will simply be relayed on the breadboard and connected between the motherboard and chassis, with the exception of the #PWRBTN and #RSTCON, which will have parallel circuit attached to it with a MOSFET controlled by the GPIO pins in order to ground it on command. This circuit will ensure that all of the other other pins will be usable for features such as the power on LED, HDD LED, and audio connectors to remain functional, while giving the Pi the ability to ground the #PWRBTN and the #RSTCON pins. The circuit design is better illustrated in the figure below.

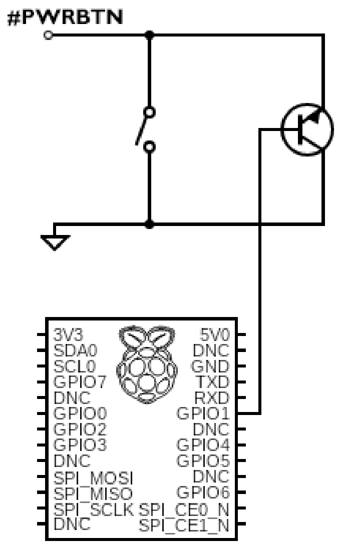


Figure 5.3: Parallel circuit for ATX controller.

The GPIO pin is directly connected to the #PWRBTN pin, and by setting the GPIO pin to the same voltage as the PWRBTN# pin, then quickly reducing the voltage to 0V for around 100 milliseconds, then back to high, the server boots up at 100% success rate. The simplicity of the solution was unexpected, but it so happens that the PWRBTN# at a logic high state sits at 3.3V, and the Raspberry Pi GPIO pin's voltage is also 3.3V when it's at a logic high state. Hence, by connecting the two while the GPIO is high, it's equivalent to as if the #PWRBTN pin is insulated. In contrast, when the GPIO pin drops down to 0V, it's as if the #PWRBTN pin has been grounded, which boots up the PC.

This implementation is significantly more lightweight than the breadboard circuit solution, however there is still the problem of maintaining the functionality of the rest of the front panel connectors. In order to achieve this, a coil was formed with the wire leading from the Pi, and slotted it beneath between the pins and the connectors. Although this solution might seem overly minimalist, after extensive testing, the server has not failed to boot up a single time. Following the principle of Occam's Razor, the simplest solution might indeed be the best one. A circuit diagram representation of the implementation can be seen below.

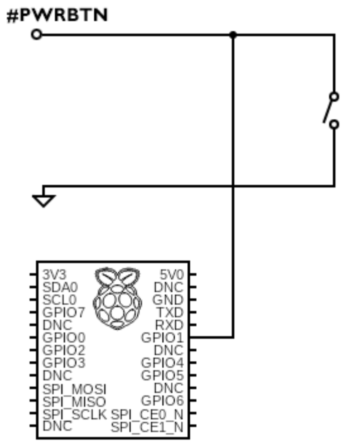


Figure 5.4: Simplified implementation of the ATX Controller

The #RSTCON pin can also be connected in a similar manner, however since the test-bench is a Small Form Factor (SFF) device, it did not have a reset pin. Hence, an additional wire connected to the Pi has been left inside the chassis.

An important thing to note about this implementation is that this only works because the #PWRBTN pin uses 3.3V as it's logic high level on the motherboard on the test-bench device. Since there were no definitive sources whether the voltage is different between every motherboard, three more motherboards have been tested: Gigabyte Aorus Z370 Gaming 7, Gigabyte Aorus X570 Xtreme, and ASUS ROG STRIX B450M, and they all had 3.3V on the #PWRBTN pin.

5.4 Web UI

The plan for this project was always to have a minimal yet functional web UI that is easy to navigate for users. In the beginning of the project, a custom web UI was developed which was was able to stream the video feed from the server as required. However, after discussing with my supervisors, a design choice to be made whether or not use an existing open-source project as the foundation for this project. There are many pros and cons with this approach, and it completely changes the nature of the project from a complete DIY solution to hacking another project.

The biggest benefit of hacking an open-source project is that a majority of the allocated project time could be saved and instead used to implementing the important requested features that the original project did not have, since the USB peripheral interaction would already be complete. Additionally, the entire full stack structure will already be provided.

The downsides of doings so however is that some of the open-source code has terrible quality. Since most of these projects are developed by hobbyists, there are no fixed standard for quality. Sometimes, understanding an existing project could be more time-consuming than developing your own. This was definitely the case for the Pi-KVM project, as the code base is heavily compartmentalized and rather difficult to work with. For example, the installation comes with a custom Arch Linux ARM based OS, and flashing the SD card is required. Thankfully, TinyPilot was available as an open-source project as well, and it was more lightweight and could run natively in the default Rasbian OS. The code was easy to get started with and had clear comments. There was an issue regarding system permission when implementing bash commands, the overall progress working on the TinyPilot was smooth and the developer, Michael, provided his support for a very specific bug.

In terms of the web UI of the TinyPilot, it had all the elements that of a good design would be have - menu bar on top with drop down sub-menus when hovering, expanding into the individual settings and options. The screen is centralized, taking up roughly 70% of the browser's horizontal screen real estate. Some small UI adjustments were made to improve the user experience. The original design had a virtual keyboard that emitted keystroke inputs when pressed with the mouse, however, a feature as such was quite redundant as the system administrator is most likely using a client PC with a keyboard. The final UI can be seen in the figure below.

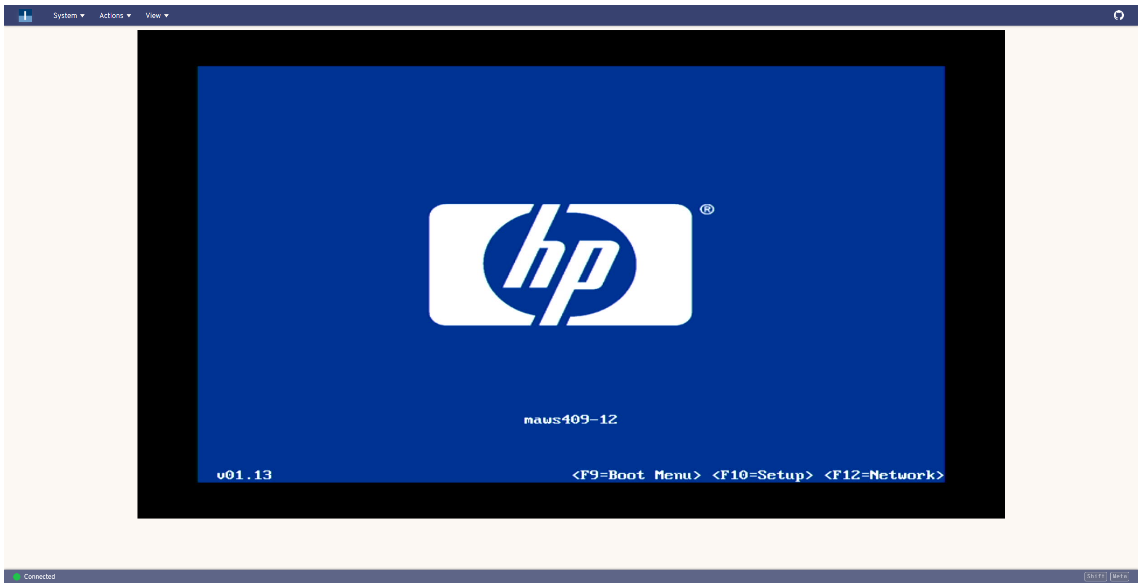


Figure 5.5: The User Interface for the Web Application

Chapter 6

Implementation

This chapter will include most of the software side for the project development. Due to the fact that the TinyPilot codebase is used as the foundation of the project, full stack architecture will need to be briefly described before moving into the custom features implemented.

6.1 TinyPilot

As described in the background section, TinyPilot is a lightweight program that runs its front end without React, Angular, or Vue.js etc. It is purely written in HTML, CSS, and Javascript, packed with only the necessary features. As a result, it was easy for me to edit the interface and add my own functionality. The front end is also responsible for forwarding keyboard and mouse input to the back end. To achieve this, HTML Custom Elements [19], which is supported on all modern browsers natively without the need to install any extensions.

The back end for the TinyPilot is based on Flask, and it is used to handle three types of requests, page requests, REST requests, and WebSockets requests. Page request is the most straight forward one as TinyPilot uses Flask to pre-render a template when loading a page like the main page. REST requests are used when the front end needs to interface with the back end, in order to query server state or perform an action based on an user input (menu buttons being pressed). Flask Blueprint will be used here to allow the Python functions in the *api.py* to be called by the Javascript when a custom event has been emitted in the front end. Lastly, WebSockets requests are used for keyboard and mouse inputs, as the WebSockets channel is able to pass the input to the back end faster than regular REST requests.

The back end is also responsible for emulating USB peripheral inputs, which TinyPilot handles by using the Linux USB Gadget API [20]. The API can be run on any Linux device that has an USB port that operates in device mode, or supports USB OTG (On-the-Go), meaning that it can switch between host mode and device mode). Since the Raspberry Pi 4's USB-C port is capable of both supplying the device with power as well as USB OTG, the TinyPilot makes use of it using a data/power splitter. By using this library, the TinyPilot presents itself to the server device as a USB Multifunction Composite Gadget, or put in lament terms, an USB hub with mouse and keyboard attached. This allows the TinyPilot to send both keyboard and mouse inputs simultaneously via a single USB connection.

Nginx [21] is being used in the TinyPilot to listen to incoming HTTP requests. It also serves all of the web UI's static resources, as opposed to the server-side processed resources which is served by Flask.

Finally, the video stream is handled by uStreamer [7]. Since it has already been described in detail in the background section, this section will not go into the details of how it works, but rather, discuss its limitations. Since uStreamer uses is MJPEG as its encoding format, which is equivalent to sending each frame as a JPEG image, the video stream could sometimes become bandwidth limited, as 30 JPEG images per second could be heavy on the HTTP traffic. In order to reduce the impact of this, the TinyPilot has an in-built video resolution slider which ensures that users can tweak the quality of the stream to ensure a smooth stream.

The high-level architecture can be illustrated as in the figure below.

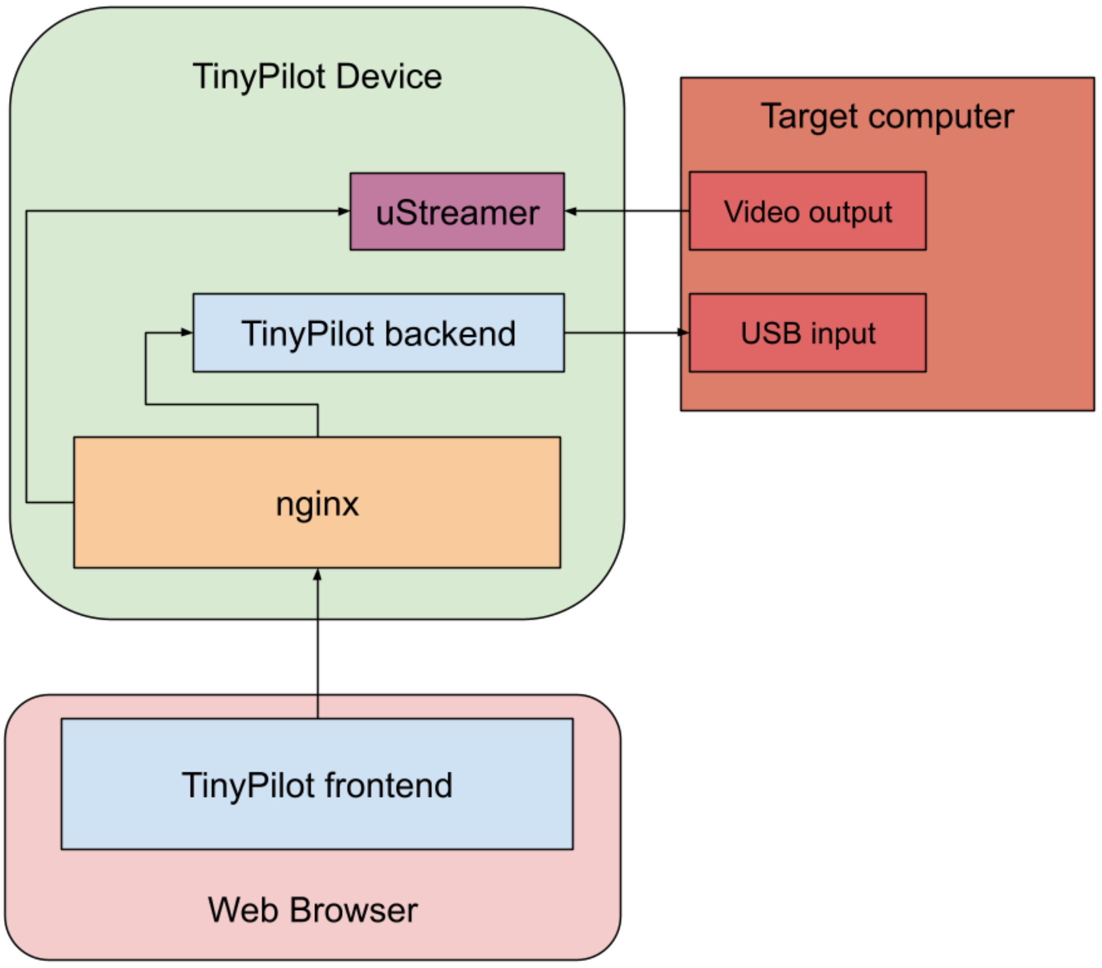


Figure 6.1: The TinyPilot Architecture

6.2 Adding features to the TinyPilot

Since the TinyPilot source code had a very clear structure, it was very easy to inject new components into the code. All that needed to be done was to create a new button which emits a new custom event, writing the Javascript that listens to the event and calls a Python function from *api.py* using Flask Blueprint.

6.2.1 Booting up the Motherboard

The hardware implementation of this task has already been described in the previous chapter, however the software implementation of this was more difficult than imagined. The problem was that the Raspberry Pi's GPIO Python library was only accessible when superusers ran the script, and the TinyPilot did not have superuser privileges.

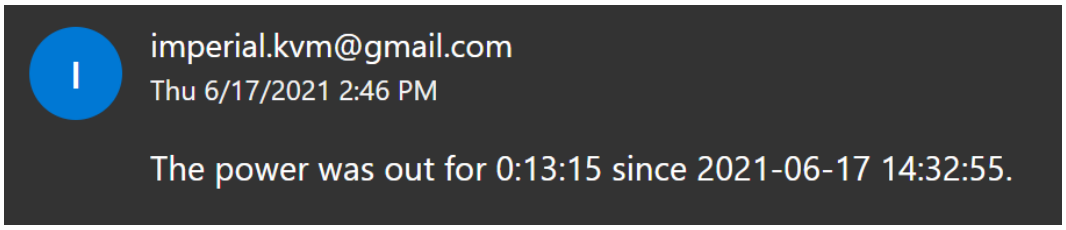
At first, a naïve approach was taken to upgrade the TinyPilot's permissions. Although this solution partially worked, it is bad practice to give all of superuser privileges to a program, which means that an alternative must be taken. To attempt this, a BASH script to execute the Python script with sudo permissions was created and add to the $PATH variable. This worked well when ran in Visual Studio Code, but when it was called from the web UI, it did not work and there was no error message, as if that command was never issued.

This error remained unsolved for nearly half a week, before reaching out to Michael, the developer of TinyPilot. He pointed out that it was highly likely that since the BASH script had sudo commands inside, it required a password input, which was not experienced in VSCode since the password was already entered when calling **"sudo -i tinypilot"** to switch profile to TinyPilot. However, when the script is ran, the back-end opens the terminal natively with the TinyPilot profile, the superuser password was never entered, thus issuing a teletypewriter (TTY) prompt, which is not being picked up by the std_out that the debug logs checks for [22].

The issue was exactly that, and by edit the **/etc/sudoers** file to add the script to the NOPASSWD section, the TTY was bypassed and the script successfully ran .

6.2.2 Power Out Detection

This feature was explicitly requested in the requirements capture, as it will help the system administrator to keep track when the power goes out. The implementation of the power out detection was straight forward, and a parallel process is created, using the multiprocessing library, running a **while True** loop that updates a .txt file every minute with a datetime object converted into string whenever the Pi is powered on. On boot, the Pi will check the time delta between the last recorded datetime and the current datetime, send out an email to the system administrator, as well as append the outage time and duration in the log file. This log file will also be accessible in the web UI.


imperial.kvm@gmail.com
Thu 6/17/2021 2:46 PM

The power was out for 0:13:15 since 2021-06-17 14:32:55.

Figure 6.2: The email sent out by the Pi on boot up

An interesting problem here was that the script would work perfectly whenever the device rebooted through **"sudo reboot now"**, but whenever the power connector was disconnected for an extended period, the time delta would always be showing a negative value. This was very confusing as the current datetime object would be one of different time on startup, but whenever the script was ran directly, everything worked flawlessly. After further investigation, the culprit has been exposed and it lies in that the script is ran on start up, which may sometimes be executed before the system clock has time become synced with the internet. This was then easily solved by adding a 20 seconds delay to the script at start up, which fixed everything.

6.2.3 Watchdog Timer

A feature which was suggested, however not implemented, was a watchdog timer. A watchdog timer is a system feature which automatically reboots the device when it becomes unresponsive. Since the KVM switch is an "always on" device, having a reset feature in the event of a system failure could prove itself useful at times. The way it works is that the system will "pet" the watchdog once every fixed period of time, and if the watchdog has not been "pet" within the set duration, it will "bark" and reset the device.

The Raspberry Pi has an internal watchdog timer module on its BCM2835 System-on-Chip (SoC) that powers the Pi, and it has a 20 bit counter with 16 microseconds per tick, giving a WTO (maximum amount of time the watchdog timer can count) of roughly 16 seconds. The timer starts on system start up and if for any reason the watchdog has not been "pet" within 16 seconds, the system will reboot. This feature was omitted due to the other processes that needs to be executed on start up (sending email, launching the web server, etc.) and if these services take over 16 seconds to launch, the system will end up rebooting, causing an infinite reboot loop. Another shortcoming of using the internal watchdog timer is that if you attempt to shutdown the system with "sudo shutdown -h now", the system will never be able to boot up again [23]. Finally, another flaw is that the timer is actually not very reliable, sometimes when the system resources are overloaded and becomes unresponsive, the "petting" stills sometimes goes through and a reboot is not initiated.

With these consideration in mind, the internal watchdog timer was not feasible to implement. However, having an external watchdog timer card which is interfaced with the GPIO pins is an alternative option, and this could be left as an extension for future work.

6.2.4 Environmental Variables Logging

One of the lower priority requirements for this project is to include a temperature monitoring feature which can detect if the chassis temperature exceeds a certain threshold. This could be caused by prolonged period of high CPU/GPU activity such as rendering or training a machine learning model, or if the server room air conditioning is malfunctioning causing a rise in overall room temperature. Both of these scenarios could be damaging to the servers if the temperature remains high for a long time, and usually for most PCs, only the CPU and GPU temperature is able to be monitored using programs such as CPU-Z and MSI Afterburner, hence, having an external method to monitor the overall chassis temperature is a desirable feature.

The hardware used for monitoring the environmental variables is the Adafruit BME688 [24]. It is able to sense the temperature, humidity, and the pressure of its surroundings. The sensor is connected to the Raspberry Pi's GPIO pins using I2C as the protocol, and has its own dedicated Python library which allows the program to access the variables with ease.

The detection mechanism comes in two forms, a warning system which sends out an email notification whenever the temperature exceeds the threshold, as well as a monitoring feature embedded in the web UI to check at anytime. The former is implemented as a very simple synchronous finite state machine (FSM) with a clock tick every one minute.

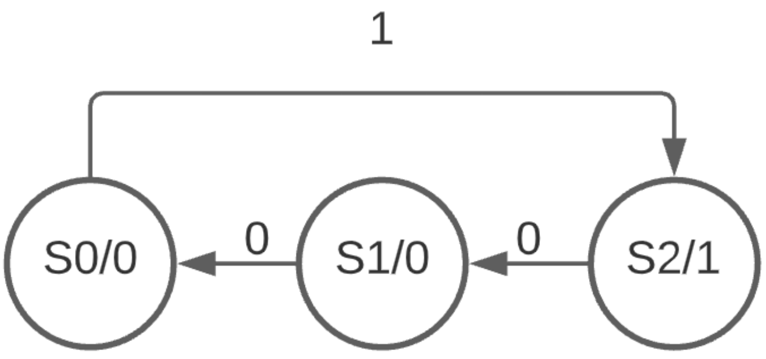
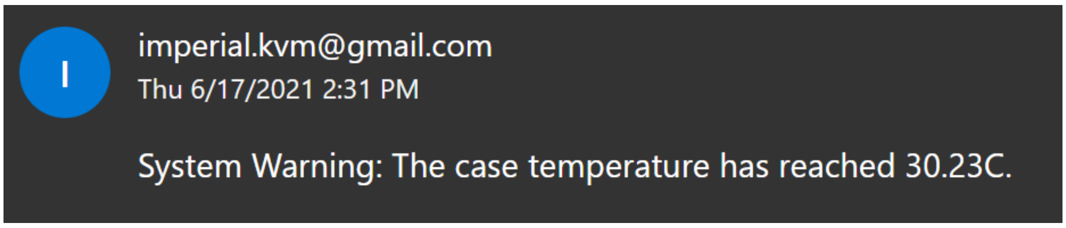


Figure 6.3: The Moore diagram of the FSM

The input represent whether the temperature exceeds the threshold or not, and when the output is 1, an email will be sent out. As can be seen on the diagram, the FSM allows the device to only send out an email the first time the temperature exceeds the threshold, and as long as it remains above the threshold, it will not continuously attempt to notify system administrator.

The threshold temperature should be set to 60 degrees Celsius as temperatures above that has the potential to cause permanent damage to some less heat resistant components inside the PC. For testing purposes though, this threshold has been set to 30C as it allowed the notification event to be triggered when putting a finger on the sensor.


imperial.kvm@gmail.com
Thu 6/17/2021 2:31 PM

System Warning: The case temperature has reached 30.23C.

Figure 6.4: Example email sent out by the Pi when reaching temperature threshold

Note that this is the intended implementation, but in reality, there is a bug which will be discussed later that causes unintended behaviour in the brownout detection mechanism. Since the latter is more an more important feature, the temperature notification FSM will be left out in the prototype and could potentially be added in the future.

Chapter 7

Testing

In order to ensure the solution meets the standard of the DoC and EEE Department, extensive testing needs to be done. It is arguably the most important aspect of this entire project, and a full range of test cases have been created to guarantee is robustness once deployed.

The device will be attached to an old server machine acquired from DoC. The specifications of the server is insignificant as the KVM switch is device agnostic, and the only important thing to note is that the GPU output port is DMS-59 (Dual Monitor Solution w/ 59 Pins). In order to get an HDMI output, an DMS-to-HDMI adaptor was used.

Unlike other projects, not too many evaluation metrics are not required to display the quality of the solution as the most important criterion is reliability. To test for unintended behaviours and bugs, potential situations where the device could be at risk for becoming unresponsive have been simulated and tested.

7.1 Power Cut Simulation

The most simple test case is when the power goes out in the building for a random period of time. To simulate this,both the server and the KVM switch were connected onto the same extension lead, and then toggled the socket off simultaneously. When the power comes on, the intended behaviour is that the server will stay shut down, but the Pi will power itself on and by using the ATX controller, to turn on the server again. The "Boot after Power Loss" option in the BIOS on the test bench had to be turn off as this would cause the device to boot twice, leading to a shut off state.

This procedure was replicated with different time frames of power outages, from five minutes up to five hours, and tried to record any point of failure. The criterion to achieve a success in a run means that the server device needs to be successfully booted, and the web UI for the Pi needs to be accessible. A few of the longer duration runs were skipped and marked in yellow as N/A, and all of the other tests succeeded, and the device has proven itself to have good reliability when it comes to power cuts. This is not surprising as this scenario was the reason why the device was built in the first place. The results for the tests can be found in the appendix.

7.2 Network Disconnection Simulation

Another case for the switch to become unresponsive is if the network disconnects. Since the switch will be controlled with a web client, as soon as the network goes out, the user will be able to tell. When the network eventually comes back up, the Pi needs to be able to establish connection with the client again.

To simulate this scenario, the Pi was disconnected temporarily using my router configuration software, and reconnect it after a random time frame similar to above. The criterion to achieve a success in a run means that the Pi's web UI needs to be accessible after the connection is restored. Once again, the device has proven itself to be reliable in this simulation. The results for the tests can be found in the appendix.

7.3 Peripheral Input Test

The emulated keyboard and mouse inputs will be sent to the Pi over the network, there is a risk for package loss. Although losing a key-press every now and then might not be enough to deem the product broken, it will certainly be annoying for the users. Hence, it will also be a test to be carried out.

A virtual keyboard program was used on the client computer to run a macro to enter a string of text into the web client to control the server. Then the string from the client and the server will be compared, and if there are any differences, it means that there are package loss.

The resulting **.txt** file on the server perfectly matched up with the macro that was set, indicating that there is no package loss between the client and the Pi. The file is added to appendix for reference.

7.4 Power Loss Logging Test

This was partially already done during the implementation stage, however, a full log of the power out duration during the power cut simulation will be produced. This will ensure that the logging is also working as intended. In the appendix, the logs of during the power cut simulation can be found.

7.5 General Results

The device works as intended and has not failed any test cases listed above, hence the core functionality of the KVM switch (keyboard, video, mouse, power on) can be guaranteed to work flawlessly. The web UI sometimes had minor bugs such as freezing which was not reproducible, but they were easily resolved by refreshing the web page.

Chapter 8

Evaluation

8.1 Physical Deployment

One of the concerns when creating this device was that it could **not** be assumed that the device and the target desktop will be permanently stationary. There could be requirements to move the server rack between rooms or even building, hence the device is modular to prevent potential permanent damage caused by sudden movements. All components and wires can be easily detached and reattached to the Pi so even if the device is accidentally "yanked" from the desktop, the cables will simply come loose from the GPIO instead of potentially breaking soldered joints.

There were considerations whether or not to put the Pi inside the chassis, as this would save space if the server needs to be put on a server rack. additionally, it would also be alternative solution to ease the mobility of the device. However, after further experimentation, the way to put the Pi inside the chassis is by slotting it between the PSU and the GPU (on the SFF testbench used) which are both components with high heat dissipation. The Raspberry Pi does not have a built in fan and relies purely on passive cooling, and if the ambient temperature is high, it could cause the Pi to overheat. Hence this approach was not taken.

An example of how the device could be deployed can be seen below.

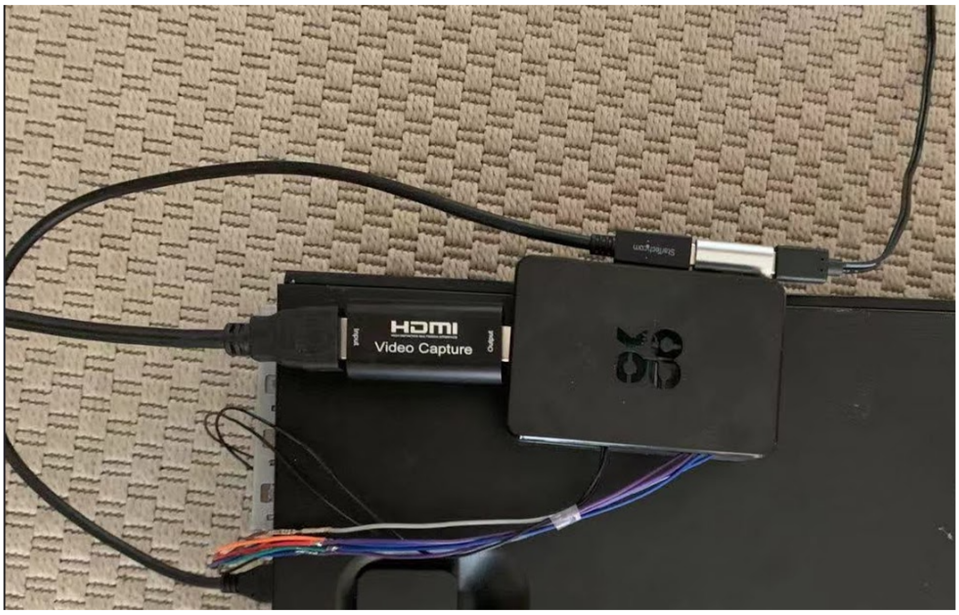


Figure 8.1: The device put on top of a server (The top cover of the case is optional)

Additionally, even with device outside the case, there were considerations made around the overall temperature and potential for overheating since the device will always be on. Potting compounds was one of the potential solutions if the temperature becomes an issue, as most potting compounds such as silicon adhesive has a thermal conductivity of 0.6W/mK [25] as opposed to air in room temperature which only has 0.026W/mK [26] thermal conductivity. Hence by applying the adhesive on top of integrated heat spreader (IHS) of the CPU, it could increase the rate of heat dissipation and reduce the CPU temperature. However, during the testing, the CPU never exceeded 60 degrees, which is well inside the safe range of temperatures for the Pi to be continuously ran in. Hence, this step was not taken.

8.2 Comparison to the Original TinyPilot

Although the original TinyPilot is a great product, it lacked crucial features to be widely deployed in Imperial College. The lack of ATX control being the biggest factor, as the main motivation behind the development of this project is solve the issue of power outage affecting remote users, and removing the manual step of needing the system administrator to physically be in the server room and rebooting all the servers after each outage.

Additionally, the TinyPilot had some unnecessary UI features such as the virtual keyboard that was removed entirely as the users who is accessing the web UI is nearly guaranteed to have a keyboard. Other features removed includes the shutdown and restart of the TinyPilot itself, which made no sense to be included as a feature on a KVM switch.

The complete custom setup costs £87.46 (ex.VAT) to create, while the TinyPilot goes for $169.99, and that does not include the import duty of up to 12% that consumers might have to pay as the product is shipped from the U.S. Although it is understandable that the developer is aiming to make a profit with his products, the price discrepancy will most likely lead more users to create their own, with or without his source code.

8.3 Comparison to the Pi-KVM

The Pi-KVM is the other main competitor in the hobbyist KVM-over-IP switch market, however its implementation of the ATX controller is overly complicated. The Pi-KVM requires an entire breadboard to issue to boot command, which itself takes up more space than the Raspberry Pi. This causes the deployment of the switch very inconvenient, as it will consist of two loose boards held together only by a few wires. Additionally, since the breadboard is acting as a intermediary between the chassis front panel connector and the motherboard front panel IO pins, the wires will need to exit the chassis, plugged into the breadboard, and then routed back into the chassis. It makes the solution look very messy and the installation process will also be very tedious, as this process needs to be done for every machine.

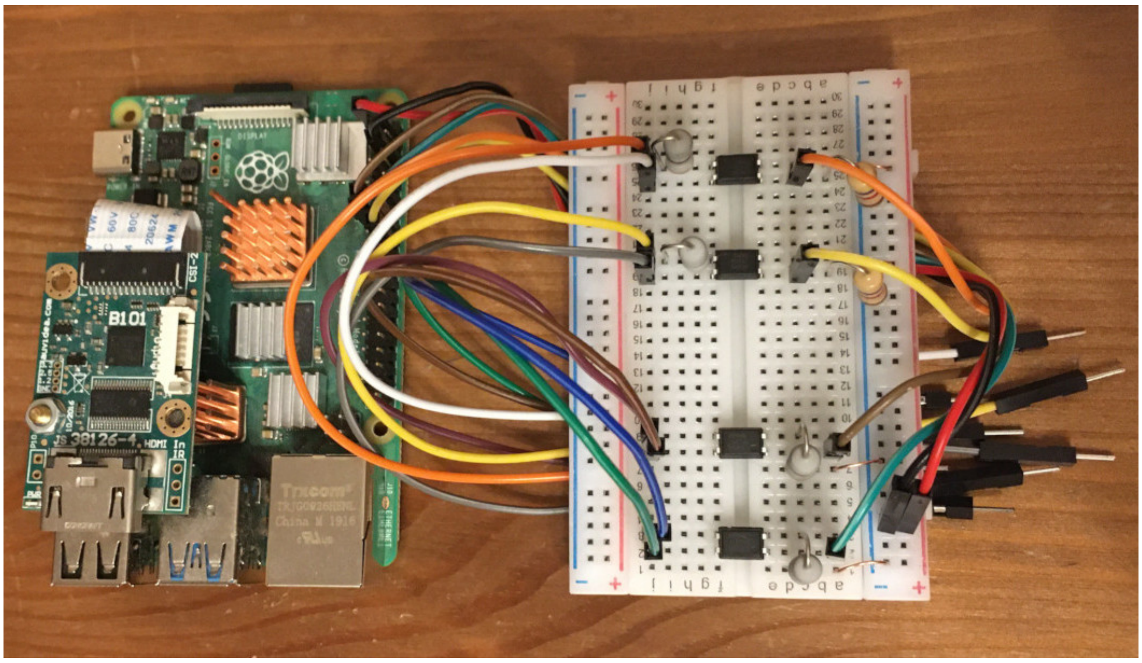


Figure 8.2: Breadboard implementation for the PiKVM

With this project however, all that needs to be done is to route a single wire directly from the Pi to the motherboard. Although it might be seen as unreliable at first glance, the only point of failure with this method is if the wire is not connected with the pin, or if it shorts with another pin. These two issues can easily be solved with a custom made wire with a small coil in the end that connects to the pin, as for the purpose of this project, only a proof of concept was made.

8.4 Change in Project Direction

The project objective was clear from the beginning and the final product is functioning exactly as planned. What did change though was the approach taken to complete it. Since it became apparent that the timeline was becoming relatively tight, and perhaps the time it would take to implement all the basic features was underestimated,the TinyPilot source code was used as the shell instead to compensate. This was most likely the best decision made during this project, as it allowed spending all the time focusing on the specific requirements requested by the Imperial College faculty. If that wasn't the case, the project might've barely finished on time for the video, mouse, and keyboard functionalities, and would've pushed the power loss notification and logging as an extension. Instead, all of the requirements from the requirements capture was completed.

8.5 Legal Considerations

The TinyPilot codebase is licensed under the MIT license, which permits commercial use, modification, distribution, and private use. This means that this project has not violated any legalities by using the open-sourced project as its basis. If this product becomes mass produced in the future, it will also fall under protection of the distribution clause, hence there is nothing to be worried about.

8.6 Known Bugs

During the experimentation of the limitations of the device, a few bugs occurred. Some of these bugs are minor and some are more major, however, none of these bugs breaks the core functionality of the device.

8.6.1 Video Input Black Bar

The video output is misaligned with the allocated area on the web UI for the video player, causing there to be black bars around it. Additionally, since the mouse location is based on the pointers relative location inside the allocated area, this causes the mouse on the client to be misaligned with the mouse on the target device. This can be solved by removing the cursor visibility when inside the area by navigating on the top menu bar (View -> Cursor -> None). Although this is not a major issue, it could cause annoyance for the user, hence a "None" option was added.

8.6.2 Temperature Sensor Notification

When the temperature notification was incorporated in the always-on multiprocessing script, it causes the power out detection script to not execute properly, but only when it was started from boot. If the application was launched from the command line manually, everything works as intended. The culprit behind this bug is still unknown, and methods such as adding a delay like how earlier issues were solved did not work. Hence, this feature will not be included in the final product as the power out detection notification is more important than the temperature notification.

Chapter 9

Conclusion

9.0.1 Overall Thoughts

This project was overall successful as a fully functional KVM-over-IP device has been developed has met all of the requirements given within the short time span of 4 weeks. The testing showed great, reliable results without any signs of inconsistent or unexpected behaviours. As aforementioned, the main reason this was possible was due to using the TinyPilot source code and developed on top of that; hence, I would like to show some appreciation to Michael Lynch for allowing the use of his project codebase as well as helping the development of this project when the progress was stuck and sharing his knowledge.

The main flaw of this project is that the organization of the files and dependencies. Since the code was not written from scratch, some of the functions code needed to be replaced while maintaining the function signature in order to make sure that the code would still be able to run without changing the signature in all of the other files. This caused some inefficiency later on as it was difficult keeping track of all the edited functions when working on the code and it was easy to get lost. Additionally, as little content as possible was deleted from the original source code, so even if an element in the web UI has been removed, the corresponding scripts was not deleted nor commented out. This was originally due to unfamiliarity with the program's architecture and to ensure that it would not break, but in hindsight, the code base should've been cleaned up more thoroughly to ease the process for future developers.

9.0.2 Future Work

The most impactful extension is to implement a method to switch between multiple servers, as this would allow a single Pi to control multiple devices and cut the cost of deployment. However, due to the limitations of time and resources, this opportunity remained unexplored in this project. However, this could be easily done with an HDMI splitter for the video output and USB switch for the peripheral input. The difficult part is to integrate those components into the KVM web UI, but that could be left for future students.

Another extension that could be done is the external watchdog timer. The internal watchdog timers had too many situational unreliable properties as discussed earlier in this report, and the trade off between its usefulness and the risk it brought was not worth its implementation. However, an external watchdog timer does not have any inherent risks for bootlooping as one could always just remove the module from the Pi.

Additionally, the current implementation of the power cut notification is that it will send an email notification and log the outage after the power has been restored after an outage. However, a possible extension could be to send out the notification as soon as the outage takes place. This would require the Pi to be powered by an uninterruptible power supply (UPS), as well as having a detection mechanism for when the power outages occurs. This should not be a difficult task but some architectural changes needs to be made.

Lastly, some of the bugs listed in the evaluation section could be fixed in the future with more time and resources, however, due to the testing of those bugs often require a restart, it was very difficult to find a solution when working alone within the given time.

9.0.3 Personal Reflections

I learnt so much from working on this project, as this was a very nice hybrid between software and electronic engineering. I was also able to familiarize myself more with the Linux OS, mostly the Unix Shell, as I have never worked with it before. I feel significantly more comfortable editing system permissions and editing the $PATH variable for different users, a lesson learnt from when I accidentally deleted the entire $PATH and bricked the Pi during the developement stage.

The most difficult part of the project was to design the ATX controller, which required circuit analysis and design knowledge that I needed to revise since I have not worked with circuits since Year 1 of university. Although there was a circuit diagram that the Pi-KVM used that I could just copy, I decided to be experimental and tried different work arounds since I did not like the breadboard solution. The exploration paid off and I was able to uphold the same functionality but with a more minimalist approach.

Finally, I would like to give a huge thanks to Dr. Benjamin Hou and Dr. Thomas Clarke for being my supervisors and supporting me during the course of the project.

Chapter 10

User Guide

This section will list in detail how to setup the device from a brand new Raspberry Pi. The procedure is straight forward to follow and easy to replicate.

10.1 Hardware Setup

The list of components required to produce this device and their respective pricing can be found below:

Component	Price (ex.VAT)
Raspberry Pi 4B 2GB Kit	£49.98
HDMI-to-USB Capture Card	£10.82
Type-C Data/Power Y-Splitter	£5.00
BME688 4-in-1 Air Quality Breakout	£14.25
USB Type-C to Type-A Cable	£5.00
Wires	£2.41
Total	£87.46

Table 10.1: Components required to build to the device and the cost

Please refer to the top level diagram in the figure below for the setup guide.

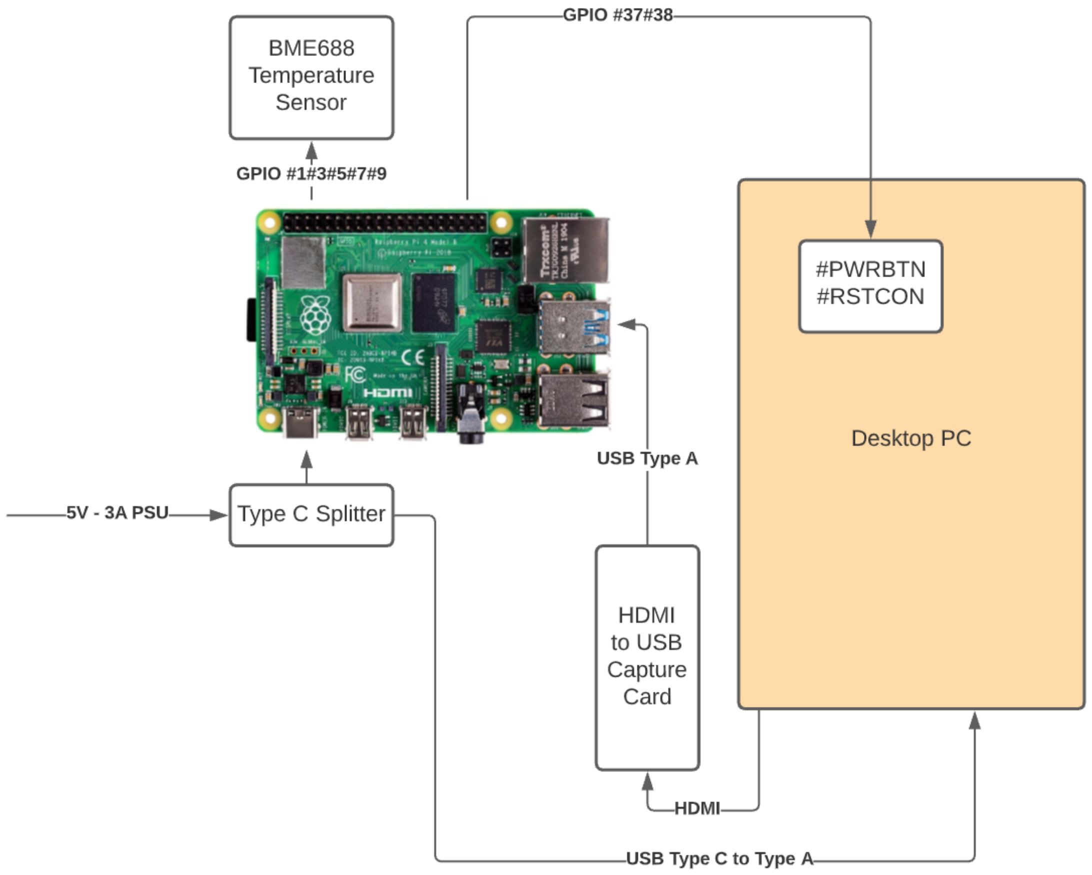


Figure 10.1: Top level diagram of the hardware implementation

Note that the output (USB) of the HDMI capture card needs to be put in the upper USB 3.0 port as the video input is pre-configured in uStreamer. As for the exact pin assignments, please refer to the table below.

GPIO	Destination
#1	2-6V (BME688)
#3	SDA (BME688)
#5	SCL (BME688)
#7	X (BME688)
#9	GND (BME688)
#37	#PWRBTN
#38	#RSTCON

Table 10.2: GPIO Pin Assignments

10.2 Software Setup

Since the device may need to be produced in a larger scale (10-100 units), a disk image of the prototype will be supplied. This image can then be flashed onto each Pi. The reason for this installation method over a bash script installation is due to using the TinyPilot as the shell, and some of the permissions changes might cause conflict and result in unintended behaviour. By using a disk image, the exact copy of the SD card on the prototype will be created, and all the changes made to the system ($PATH, /etc/sudoers, etc) will carry over to each Pi when the image is flashed onto the SD card.

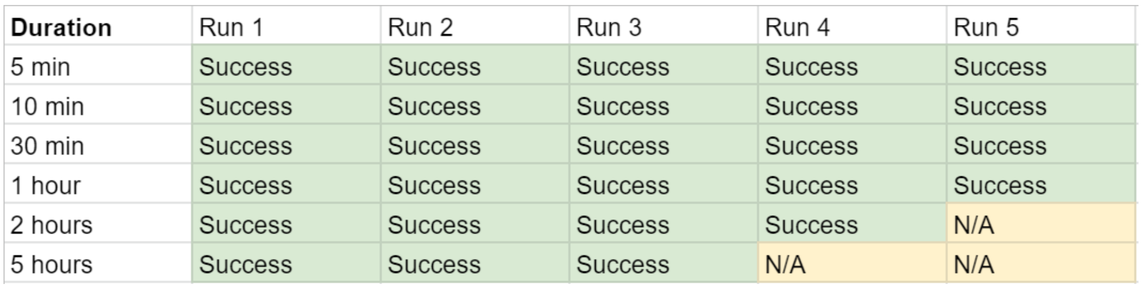
Once the Pi has been flashed and an internet connection is established, any device on the local area network should be able to connect to it by entering either the IP address or the hostname of the Pi, which can be found by typing the command "hostname" or "hostname -I" in the terminal. The hostname can be later changed in the web UI by navigating the top menu bar (System -> Hostname).

The email for receiving notifications about power outages and temperature anomalies can be changed by editing the **mailing.py** file under **/opt/tinypilot/app/brownout**. Ideally, this should've been editable directly in the web UI, however the due to bugs during the development, it has been left in the state it is currently and can potentially be updated as a piece of future work.

When all of the above steps for both the hardware and software setup have been followed, the device should be ready to be used. Start with navigating the top menu bar (System -> Power -> Power Button) to boot up the system, and the HDMI feed should be visible in the web UI. Once it shows up, the keyboard and mouse connection should become activate allowing access to the BIOS if needed by pressing F9, F10, or F11 depending on the motherboard manufacturer.

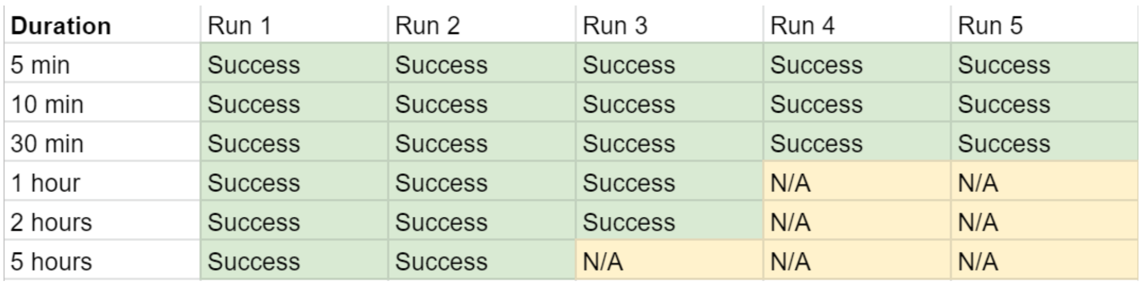
Chapter 11

Appendix A - Testing Results



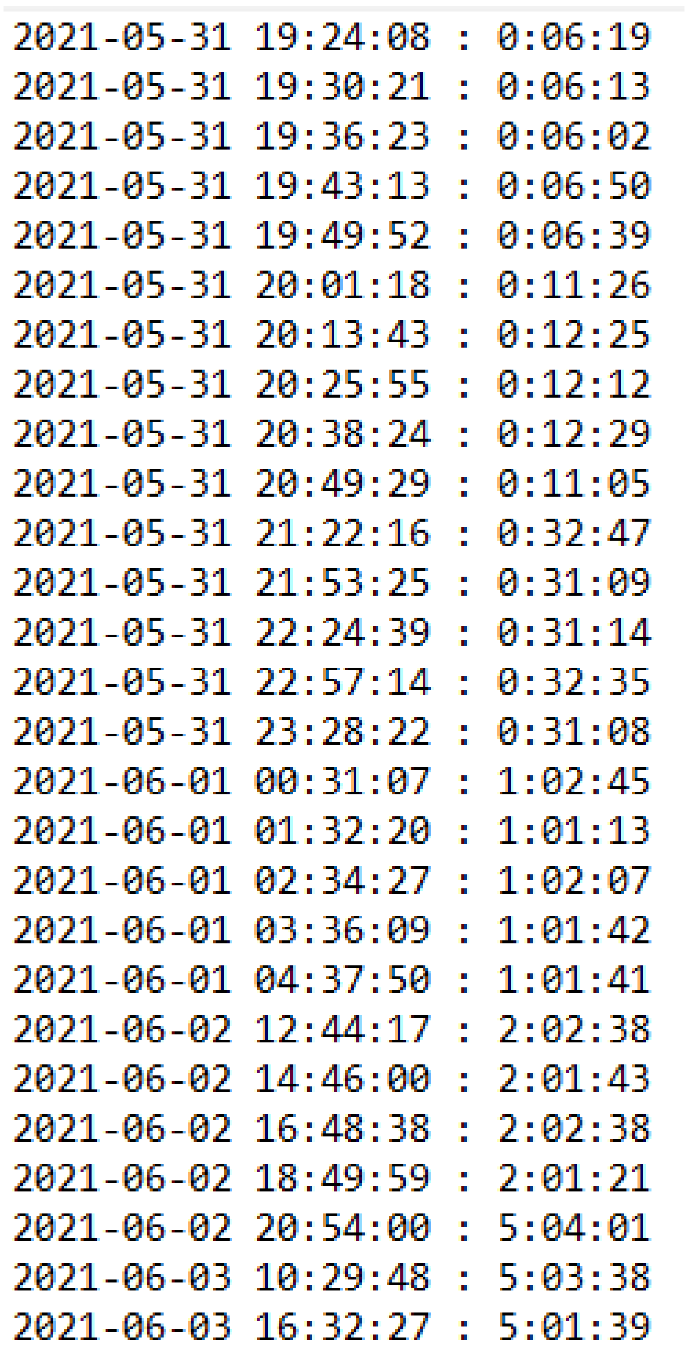
Duration	Run 1	Run 2	Run 3	Run 4	Run 5
5 min	Success	Success	Success	Success	Success
10 min	Success	Success	Success	Success	Success
30 min	Success	Success	Success	Success	Success
1 hour	Success	Success	Success	Success	Success
2 hours	Success	Success	Success	Success	N/A
5 hours	Success	Success	Success	N/A	N/A

Figure 11.1: The results from carrying out the power cut simulation.



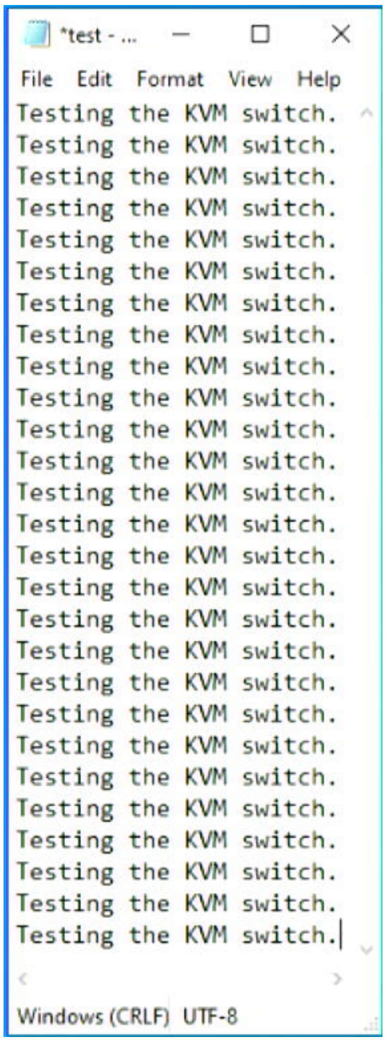
Duration	Run 1	Run 2	Run 3	Run 4	Run 5
5 min	Success	Success	Success	Success	Success
10 min	Success	Success	Success	Success	Success
30 min	Success	Success	Success	Success	Success
1 hour	Success	Success	Success	N/A	N/A
2 hours	Success	Success	Success	N/A	N/A
5 hours	Success	Success	N/A	N/A	N/A

Figure 11.2: The results from carrying out the net cut simulation.



```
2021-05-31 19:24:08 : 0:06:19
2021-05-31 19:30:21 : 0:06:13
2021-05-31 19:36:23 : 0:06:02
2021-05-31 19:43:13 : 0:06:50
2021-05-31 19:49:52 : 0:06:39
2021-05-31 20:01:18 : 0:11:26
2021-05-31 20:13:43 : 0:12:25
2021-05-31 20:25:55 : 0:12:12
2021-05-31 20:38:24 : 0:12:29
2021-05-31 20:49:29 : 0:11:05
2021-05-31 21:22:16 : 0:32:47
2021-05-31 21:53:25 : 0:31:09
2021-05-31 22:24:39 : 0:31:14
2021-05-31 22:57:14 : 0:32:35
2021-05-31 23:28:22 : 0:31:08
2021-06-01 00:31:07 : 1:02:45
2021-06-01 01:32:20 : 1:01:13
2021-06-01 02:34:27 : 1:02:07
2021-06-01 03:36:09 : 1:01:42
2021-06-01 04:37:50 : 1:01:41
2021-06-02 12:44:17 : 2:02:38
2021-06-02 14:46:00 : 2:01:43
2021-06-02 16:48:38 : 2:02:38
2021-06-02 18:49:59 : 2:01:21
2021-06-02 20:54:00 : 5:04:01
2021-06-03 10:29:48 : 5:03:38
2021-06-03 16:32:27 : 5:01:39
```

Figure 11.3: The logs generated from carrying out the power cut simulation.



```
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
Testing the KVM switch.
```

Figure 11.4: The results from running a keyboard macro from the client.

Bibliography

[1] *Integrated Dell Remote Access Controller (iDRAC): Dell EMC US.* URL: https://www.delltechnologies.com/en-gb/solutions/openmanage/idrac.htm.

[2] *Dell PowerEdge T640 Tower Server : Servers: Dell UK.* URL: https://www.dell.com/en-uk/work/shop/dell-servers-storage-networking/smart-value-poweredge-t640-server-standard/spd/poweredge-t640/pet6402a?view=configurations.

[3] *HPE Integrated Lights Out - iLO Remote Server Management Tools.* URL: https://www.hpe.com/us/en/servers/integrated-lights-out-ilo.html.

[4] *4K KVM over IP - Dominion KX IV-101.* URL: https://www.raritan.com/products/kvm-serial/kvm-over-ip-switches/4k-kvm-single-port-ip-switch.

[5] *Computer Interface Modules: CIMs.* URL: https://www.raritan.com/products/kvm-serial/accessories/computer-interface-modules.

[6] Max Devaev. *PiKVM - Open and cheap DIY IP-KVM on Raspberry Pi.* URL: https://pikvm.org/.

[7] Max Devaev. *µStreamer.* URL: https://github.com/pikvm/ustreamer.

[8] Liam Jackson. *mjpg-streamer.* URL: https://github.com/jacksonliam/mjpg-streamer.

[9] Michael Lynch. *The Modern, Open-Source KVM over IP.* URL: https://tinypilotkvm.com/.

[10] *SBC PROFIVE® NUCV (x86).* Apr. 2021. URL: https://www.eepd.de/en/boards/single-board-computer/sbc-profiver-nucv-x86/.

[11] *Odroid N2 - 4GB RAM with Case [77301].* URL: https://www.odroid.co.uk/index.php?route=product/product&path=246_239&product_id=868.

[12] *LattePanda Delta 432.* URL: https://www.lattepanda.com/products/lattepanda-delta-432.html.

[13] Raspberry Pi. *Raspberry Pi 4 Model B specifications.* URL: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/.

[14] Mike Chin. *CPU Undervolting Underclocking: A Primer.* Aug. 2002. URL: https://silentpcreview.com/cpu-undervolting-underclocking-a-primer/.

[15] Peyton Murray. *Undervolting drastically reduces CPU power draw.* July 2019. URL: https://peytondmurray.github.io/coding/undervolting-massive-power-reduction/.

[16] *HDMI TO CSI-2.* URL: https://wiki.geekworm.com/HDMI_TO_CSI-2.

[17] *The Best Capture Cards for 2021.* Apr. 2021. URL: https://www.digitaltrends.com/gaming/best-capture-cards/.

[18] Steve Burke. *How to Jump a Motherboard Without a PWR_SWButton.* July 2015. URL: https://www.gamersnexus.net/guides/2011-jumping-a-motherboard-without-power-switch-button.

[19] Caleb Williams onMar 20 and Caleb Williams. *Creating a Custom Element from Scratch: CSS-Tricks.* Aug. 2020. URL: https://css-tricks.com/creating-a-custom-element-from-scratch/.

[20] *USB Gadget API for Linux¶.* URL: https://www.kernel.org/doc/html/v4.13/driver-api/usb/gadget.html.

[21] *High Performance Load Balancer, Web Server, amp; Reverse Proxy*. May 2021. URL: https://www.nginx.com/.

[22] Lucas F. Costa. URL: https://lucasfcosta.com/2019/04/07/streams-introduction.html.

[23] SwitchDocLabs. *Raspberry Pi and Arduino: Building Reliable Systems With WatchDog Timers*. Oct. 2017. URL: https://www.instructables.com/Raspberry-Pi-and-Arduino-Building-Reliable-Systems/.

[24] *EYE on NPI – Bosch BME688 EyeOnNPI digikey Adafruit @DigiKey @Adafruit @boschglobal*. Apr. 2021. URL: https://blog.adafruit.com/2021/04/29/eye-on-npi-bosch-bme688-eyeonnpi-digikey-adafruit-digikey-adafruit-boschglobal/.

[25] *Acc Silicones 740010410 Black Silicone Potting Compound 75 ml*. URL: https://uk.rs-online.com/web/p/potting-compounds/4480286/.

[26] *Air - Thermal Conductivity*. URL: https://www.engineeringtoolbox.com/air-properties-viscosity-conductivity-heat-capacity-d_1509.html.