Buddy The Broker

Critical Design Review
Embedded Systems Development Lab
October 3rd 2022

Samuel Mehalko

Samuel Mehalko
Embedded Systems Development Lab
EN.525.743.8VL.FA22
Critical Design Review
October 3rd 2022

Table of Contents

Samuel Mehalko
Embedded Systems Development Lab
EN.525.743.8VL.FA22
Critical Design Review
October 3rd 2022

Table of Figures

Samuel Mehalko
Embedded Systems Development Lab
EN.525.743.8VL.FA22
Critical Design Review
October 3rd 2022

1 Abstract

Each year thousands of hedge funds and brokerages with armies of some of the most educated individuals in finance each compete to make money in the stock market. These individuals comb through heaps of data using some of the most complex strategies and spend countless hours in an attempt to beat the market by even a couple percentage points. These individuals will now have a new competitor on the horizon; the goal of my project is to put Wall Street's best and brightest up against my cat, Buddy.

Each day, with the help of an embedded system to communicate his stock picks Buddy will go paw to toe with investors and traders the world over. When presented with a variety of stock tickers Buddy will be able to buy/sell stocks on the open market.

Samuel Mehalko
Embedded Systems Development Lab
EN.525.743.8VL.FA22
Critical Design Review
October 3rd 2022

2 Project Description

2.1 General Overview

A raspberry pi 4 will be positioned upon Buddy's feeder equipped with a screen and a camera (all he will need to make his stock selections). The screen attached to the raspberry pi 4 will cycle through potential stocks, displaying a new stock ticker at a consistent interval. When a stock ticker is displayed that Buddy finds to be a sound investment he will sit in view of the attached camera for at least half of the interval in which that stock ticker is displayed on the screen. The raspberry pi will determine whether or not Buddy is sitting in front of the camera using computer vision. If the camera has recognized Buddy as in view for half of the trading interval that the stock ticker is displayed then the system will make note his stock selection and open a buy order. Once his account depletes its purchasing power and he no longer has enough funds to execute his next stock pick the first stock he purchased will be sold to allow for him to continue making purchases. This first in first out nature will continue as he makes more and more stock picks. To not get Buddy's account flagged as a pattern day trader he will be limited in the amount of buys that he can execute each day.

How does buddy know where to stand? Buddy’s feeder is set up on timer giving him constant meals each day. Due to this consistency each day he lines up in font of his food dish a fair bit of time before his food will be dispensed. Some meals he is front and center of his food bowl well over an hour before its time for him to be feed. Other meals he may only show up five minutes before hand. Some times he will stand in front of his bowl just hoping that it is feeding time and give up after a while. This inconstant behavior can be leveraged as mechanism for his stock picks.

Samuel Mehalko
Embedded Systems Development Lab
EN.525.743.8VL.FA22
Critical Design Review
October 3rd 2022

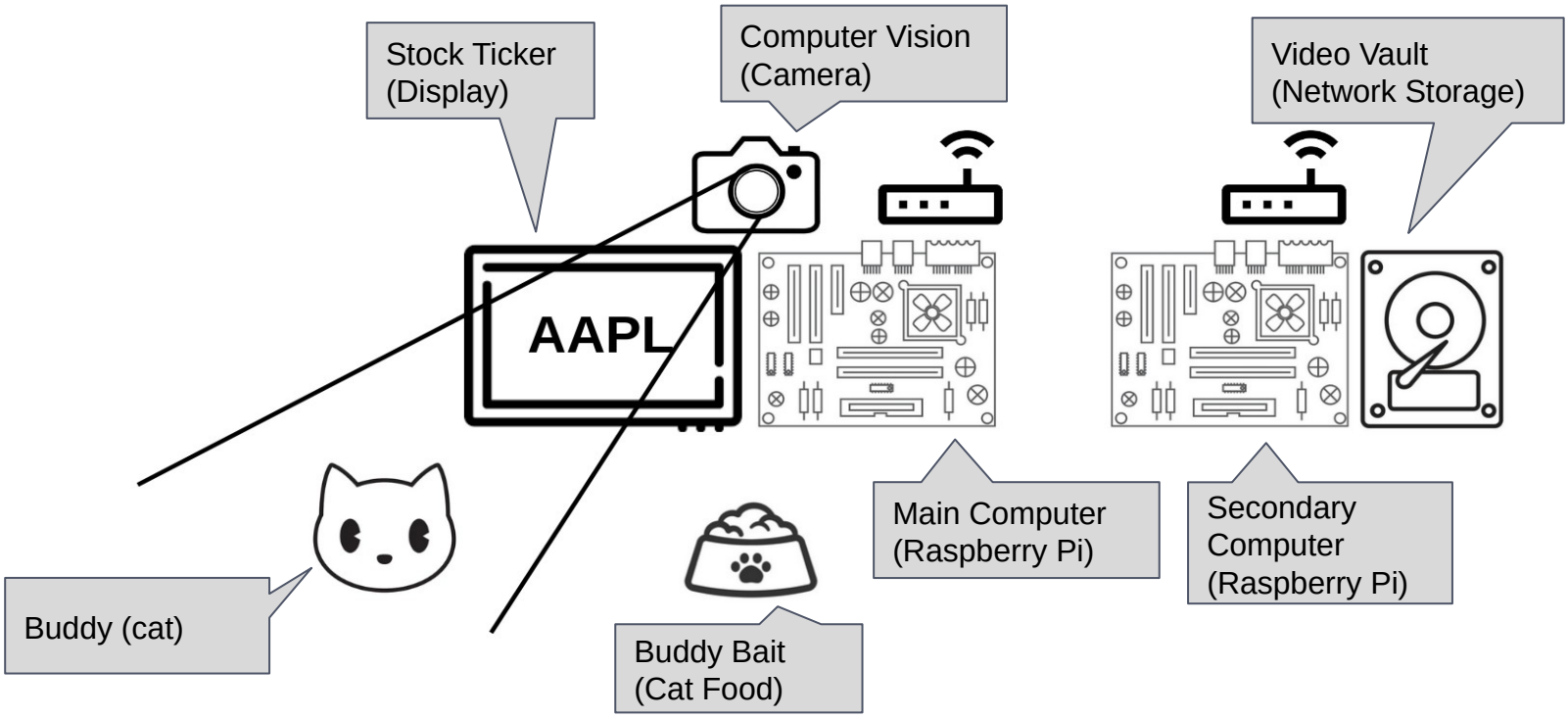

Figure 1: OV-1 Diagram With Call Out Boxes

The operation view one (OV-1) diagram, Figure 1: OV-1 Diagram With Call Out Boxes, illustrates all of major components of the system. It can be seen that the system is broken into two separate physical pieces.

The first physical piece of the system will contain the main computer which will be interfacing with the camera and display. The camera will detect Buddy using computer vision. The display will be used to display the ticker that is currently available for purchase. As described previously Buddy will be drawn to the system due to its placement above his food dispenser.

The second physical piece of the system will contain another raspberry pi used for video storage. This raspberry pi will be connected to a hard drive in a remote location, able to communicate with the first raspberry pi over the local area network (LAN) using each raspberry pi's WiFi capabilities.

At this point it should be noted that several diagrams have been chosen to use the Department of Defense Architecture Framework (DoDAF) methodology due to its ability to easily convey intricacies of a system.1 Several operations views have been omitted, due to the relative simplicity of the system those generated are sufficient in properly explaining the system.

Note: Any and all gains make from this project will be used to purchase toys for Buddy as a reward for all of his hard work.

2.2 Similar Work

Before anyone goes doubting Buddy's trading prowess and thinks an embedded project in which money is burn via a remote control flamethrower would yield the same results it is worth noting that this is not the first project of its kind and there is a level of scholarly thought on this subject. In 1973, Princeton University professor Burton Malkiel claimed in his book 'A Random Walk Down Wall Street' that, *"A blindfolded monkey throwing darts at a newspaper's financial pages could select a portfolio that would do just as well as one carefully selected by experts." 2* As it would turn out Malkiel was wrong and monkey often do a much better job than the experts. *2* The Wall Street Journal has replicated this experiment each year by choosing 100 random stocks from the S&P 500 and interestingly enough it will often bet the S&P 500. 3 Even the man arguably most famous for investing has given credence to this idea. Warren Buffett is quoted as saying, *"A patient and sensible monkey, who builds a portfolio by throwing 50 darts at a board that includes the entire S&P 500 Composite Index could increase his capital." 4*

At this point it is worth pointing out that a monkey throwing darts is really a tongue and cheek analogy for picking stocks at random. Some YouTube personalities have taken this more literally. Graham Stephen in his video, *"I Spent $100,000 On A Stock Picking Monkey"*, allowed for an actual monkey to chose ten stocks which would comprise a $100,000 account. 5 Several months later in his video titled, *"How To Make Easy Money In The Stock Market"*, Mr. Stephen goes on to show that his monkey account did in fact perform better than comparative indexes. 6 It is worth noting that others have conducted similar experiments as well with one YouTube personality allowing for his fish to trade stocks, which again outperformed the market. A link to said YouTube personality will not be provided due to the inappropriate humor in said video.

The point to be made is that this is by no means a new idea nor is it one I would expect to fail miserably. The goal of this project is to simply put a new spin on the idea and perhaps get my cat a new scratching tower in the process.

2.3 Capabilities And Limitations

2.3.1 Capabilities

Capabilities will be covered at various levels of detail throughout this document. The main sections to point out would be the following:

- Libcamera
- Libstock
- Libtrading
- Libdisplay
- Remote Storage (Open Media Vault Use Case)

Essentially the main capabilities will include a camera application programming interface (API) with the ability to detect cats, a display API with the ability to display stock tickers, a stock API with the ability to fetch various stock information, a trading API with the ability to buy/sell stocks, and the ability to store video on a separate remote drive. These separate capabilities when combine make up the indented use case for this project.

2.3.2 Limitations

Given the specificity of the goal of this project the design should enable it the end product to achieve its desired goal without too many draw backs. The main two that I see with the current design are:

1. Lack of processing power on the Raspberry Pi 4

While the Raspberry Pi 4 is an impressive little computer, computer vision is no easy task. For this reason the frame rate or even quality may be severally limited. Since the use case of this design is to only ensure that a cat is present for several minutes at a time this should be a trade off that is easy to live with. If this project needed to detect a multitude of different objects with millisecond precision this would be an issue, but for the purpose of this project a Raspberry Pi 4 should be more than enough.

2. Using a static list of stocks

The current planned implementation uses a pre-downloaded list of stocks to form the data frame used for stock data. The current list of stocks includes the top 1000 U.S. stocks sorted by market capitalization. Should this order change in the near future the order in which the stocks are presented could be slightly unaligned with the current market. Or if stock number 1000 is having a bad month and stock number 1001 is having a good month the placements could swap leaving my system with a

stale data set. For the purpose of this project the easy of using a pre-downloaded data set outweigh these drawbacks. The order in which the stocks are presented is not incredibly important so long as they are presented in an even distribution giving each a likely chance to be bought. Similarly, if stock number 1001 is bought over stock number 1000 it doesn't seem it would terribly effect the overall functioning of this project given that these stocks will likely be bought and sold relatively frequently.

3 Interface Description

Further breaking down the system we can see the interactions between each of the physical modules. The operation view two (OV-2) diagram, Figure 2: OV-2 Diagram With Call Out Boxes, illustrates these interactions. In the center of the diagram the it is shown that the Raspberry Pi which will interface with the camera and display will be a Raspberry Pi 4. A Raspberry Pi 4 is required here as computer vision software is computationally intensive. As shown in the diagram this Raspberry Pi will interface with the both the display, showing the stock tickers, and the camera, determining if a cat is in view. In theory Buddy would see the stock ticker and position himself in front of the camera (perhaps while eating) enabling the system to purchase the desired stock.

All video captured from the camera will to sent to a Raspberry Pi Zero in another room. A Raspberry Pi Zero should be more than sufficient as this Raspberry Pi will simply be interacting with a hard drive as well as with the local area network (LAN). This remote storage will allow for the stock picks to be verified and also show the fun computer vision cat identification boxes.

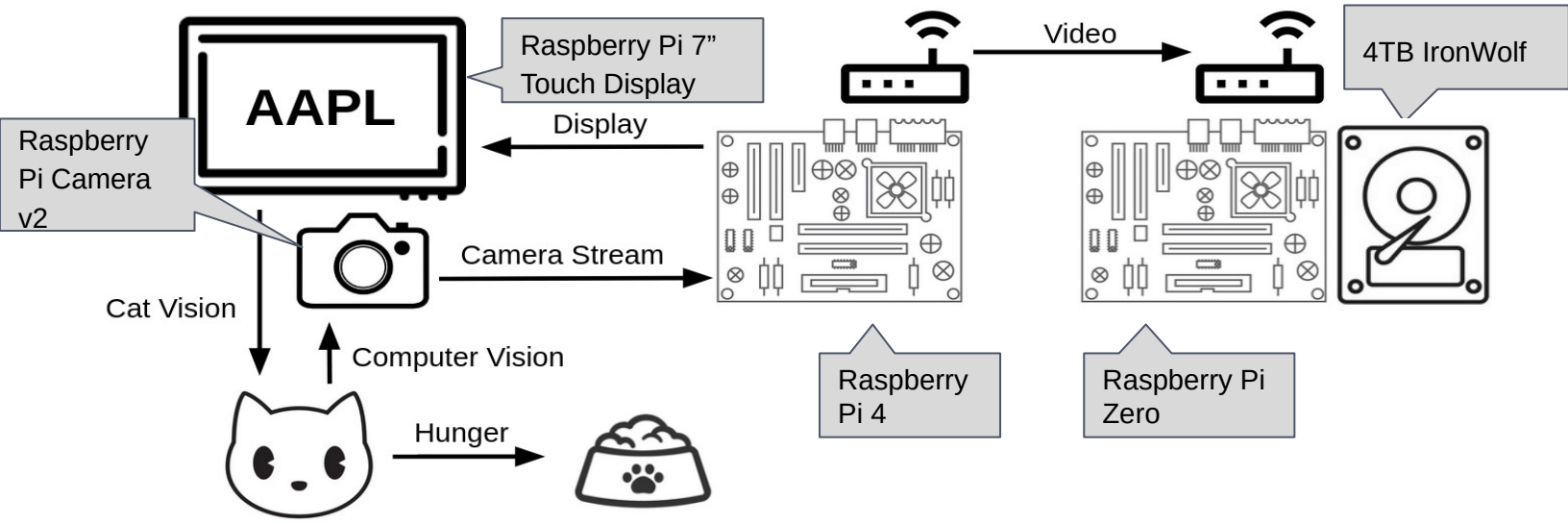

Figure 2: OV-2 Diagram With Call Out Boxes

Notably OV-2 diagrams show the "*what*" not the "*how*" of interactions. If this diagram still leaves some to be desired in terms of interface description these interfaces can be expanded upon calling out the actually protocol being used. In Figure 3: OV-2 Diagram - Interfaces Expanded it can be seen that Display Serial Interface (DSI) is the protocol used for the interface between the Raspberry Pi 4 and the display, Mobile Industry Processor Interface (MIPI) is the protocol used for the interface between the Raspberry Pi 4 and the camera, and Server Message Block (SMB) is used for the interface between the Raspberry Pi 4 and Raspberry Pi Zero. OpenCV is also specified as the method in which computer vision will be used though it should be noted that this is a library rather than a protocol.

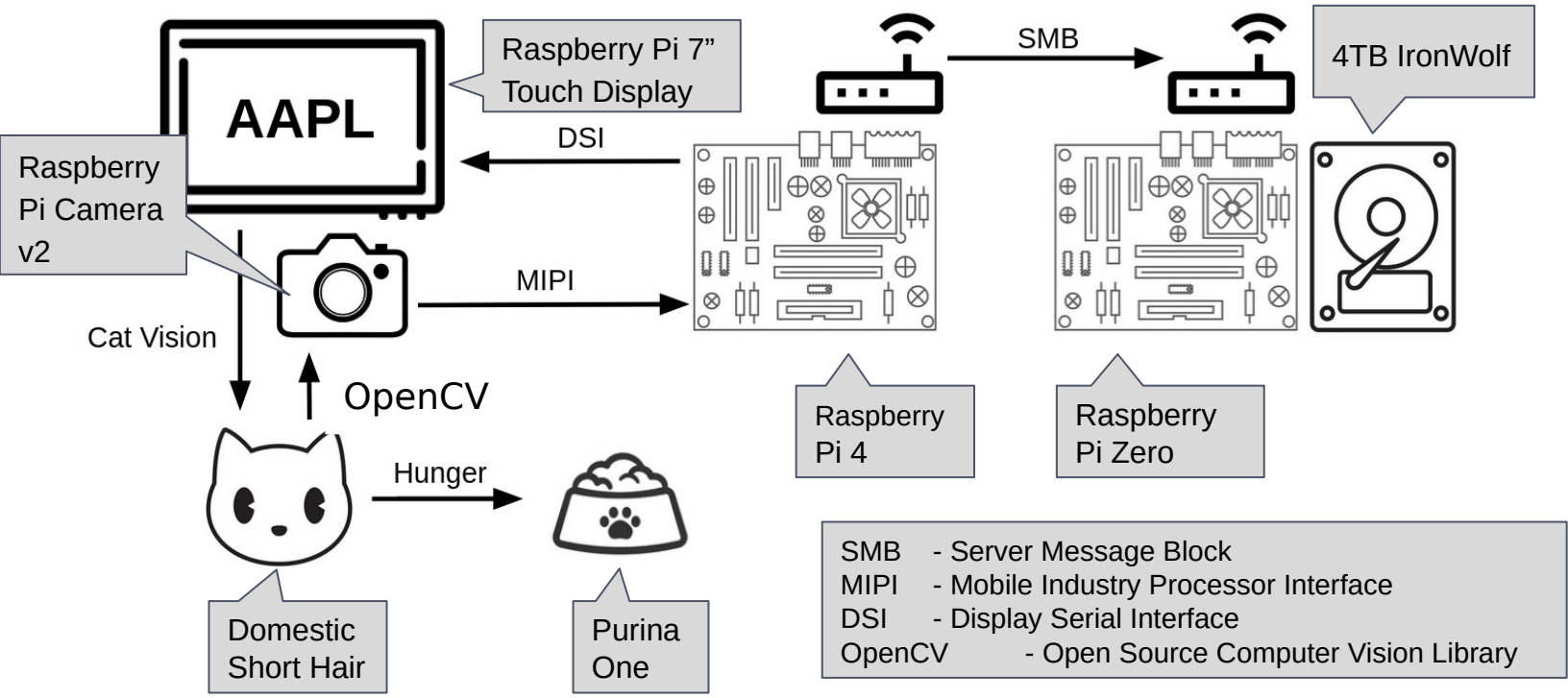

Figure 3: OV-2 Diagram - Interfaces Expanded

4 Functional Description

4.1 Project Hierarchy

Breaking down the hierarchy of this project there will be two physical subsystems each comprised of their own hardware. Following Figure 4: OV-4 Diagram, the first subsystem will be denoted as subsystem A which will be comprised of the Raspberry Pi Display, Raspberry Pi 4, and Raspberry Pi Camera. The second subsystem will be denoted as subsystem B which will be comprised of the hard drive and Raspberry Pi Zero.

The next layer down shows the software level for each subsystem. Subsystem A will use four custom libraries written for this project, libcamera will deal with any camera interactions, libdisplay will deal with any display interactions, libstock will provide stock tickers, and libtrading will enable stock trading. This modularized architecture was chosen to allow for separation of responsibilities and a cleaning overall software architecture. Staying on this software level of the diagram subsystem B will be using Open Media Vault. Which is an open source program allowing for network mounted storage.

Following the diagram to its final, bottom layer are libraries leveraged to create the four necessary libraries for this project's use case. Each library is given a quote summarizing what it does as well as a link to each library in the later section Python Libraries Leveraged but the relation of every library can clearly be seen in the diagram.

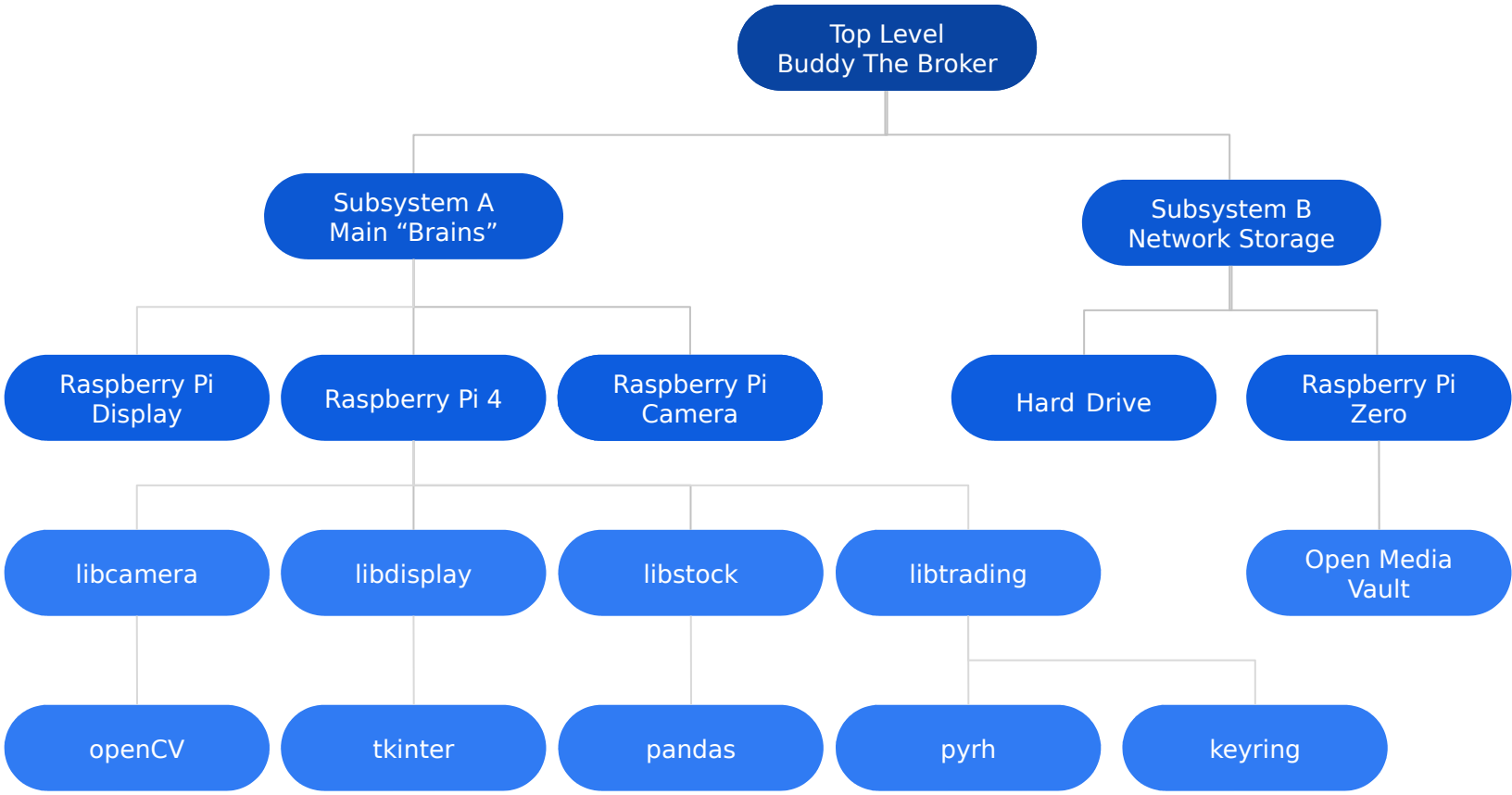

Figure 4: OV-4 Diagram

4.2 Subsystem A: Main “Brains”

To get a more realistic depiction of what subsystem A will look like Figure 5: Subsystem A is provided. It can be seen that the camera, display, and Raspberry Pi will all fit together neatly in one case. This assembly will be mounted atop Buddy’s food dispenser. This case an pivot allowing for the optimum camera angle to get the best view of Buddy. The Python logo on both the cartoon Pi and the “real” Pi depicts that this subsystem will be using Python for its various libraries.

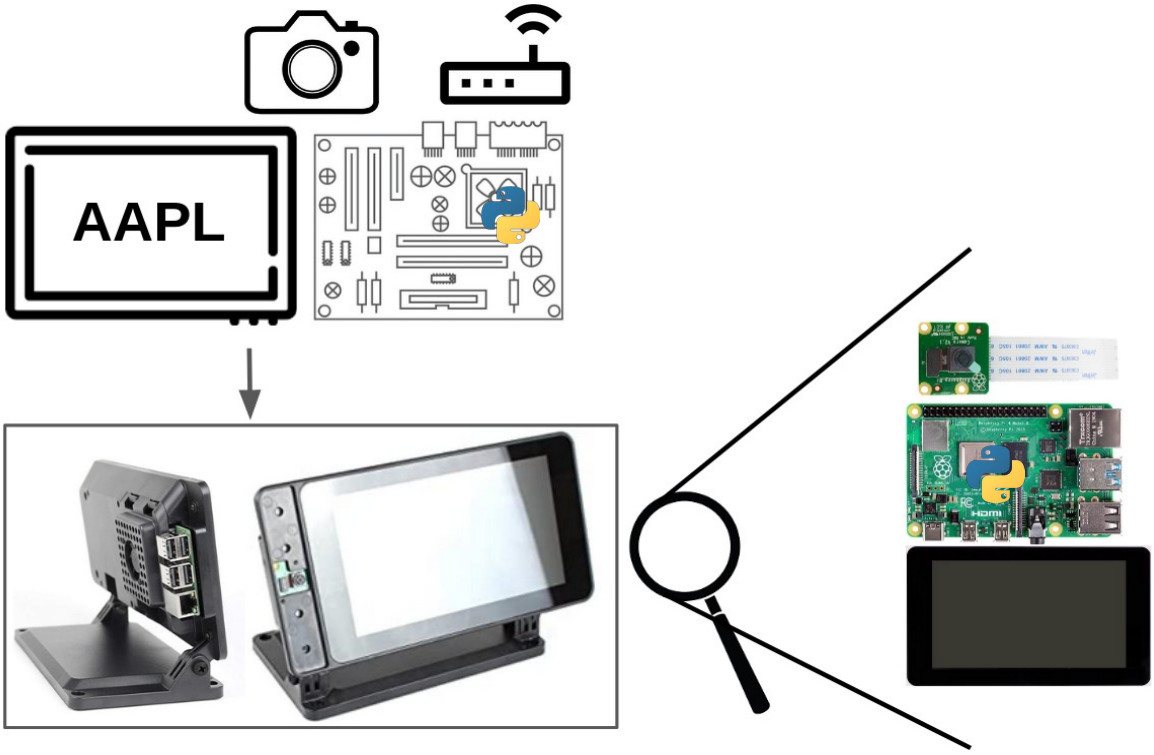

Figure 5: Subsystem A

4.2.1 Libcamera

This library will use computer vision to locate any cats (Buddy) in frame and return the coordinates. This library also will save off the recorded video to a remote hard drive, see sections Subsystem B: Network Storage and Remote Storage (Open Media Vault Use Case).

4.2.2 Libstock

This library will return stock information given an index. This library will use the top 1000 stocks ordered by market capitalization.

4.2.3 Libtrading

This library provides a means to interact with some brokerage account in order to allow for automated buying and selling of stocks. There are several trading accounts that support this but I've chosen Robinhood as I already have an open account, the API is free to use, and the app would allow me to easily monitor Buddy’s stock picks while at work.

Moreover, the system will need the smarts to keep track of the order I'm which the stocks had been picked in which order to know which to sell first, how much money is left in the account, the number of trades made each day for each stock to prevent the account as being flagged as a pattern day trader, and buffering of any stock picks (if made while the market is closed) until market open.

4.2.4 Libdisplay

Display the ticker of the current stock that Buddy could buy. This will display a new stock ticker on a regularly interval.

4.3 Subsystem B: Network Storage

To get a more realistic depiction of what subsystem B will look like Figure 6: Subsytem B is provided. It can be seen that Raspberry Pi will be connected to a hard drive docking station via a USB cable. This hard drive docking station will house a 4TB hard drive. The Open Media Vault logo on both the cartoon Pi and the "real" Pi depicts that this subsystem will be running Open Media Vault for its operations.

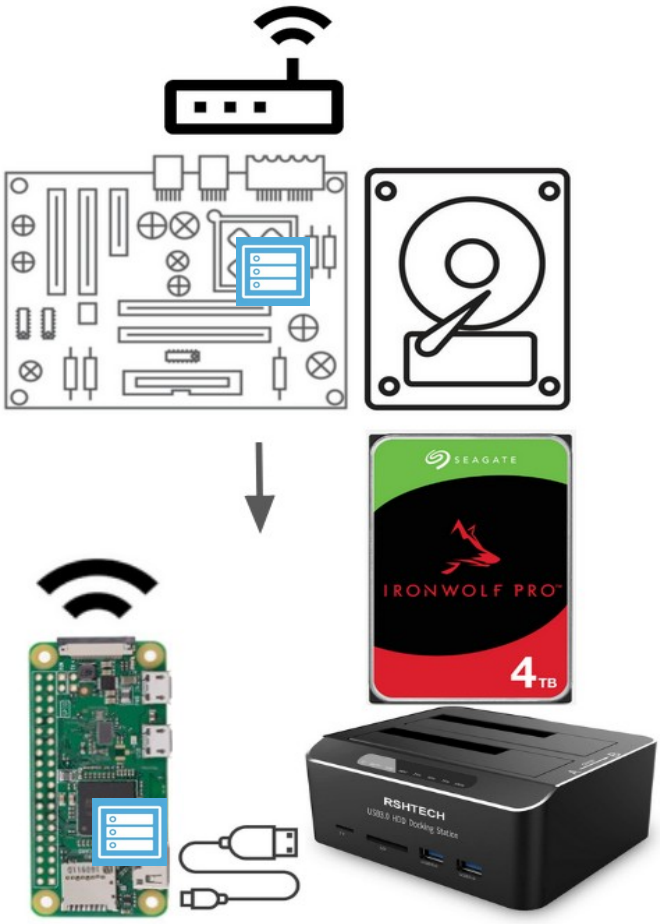

Figure 6: Subsytem B

4.3.1 Remote Storage (Open Media Vault Use Case)

What is a computer vision project without video and boxes drawn around identified objects? All video recordings of Buddy should be keep to ensure all of his stock trades are being faithfully executed.

Why not use local storage on the Raspberry Pi 4 from subsystem A? The microSD slot on the pi 4 would only allow for so much video play back and while one could connect a hard drive directly to the Pi from subsystem A that would make for a bulky setup and the HDD could easily be damaged by Buddy. Moreover, it would just be more fun to send the footage off to be saved in another room and this way it can easily accessed by any device on the network.

4.4 Unified Modeling Language (UML) Diagrams

The general concept of operations (CONOPS) of this project is again to show various stock tickers on a display and should Buddy be in the field of view of a mounted camera for a period of time while the ticker is displayed then the stock will be purchased. To better follow this use case sequence diagrams have been created. These sequence diagrams are still very high level and abstract much detail but do make the flow of this project fairly clear.

It should be noted that to read the diagrams some amount of zoom may be necessary.

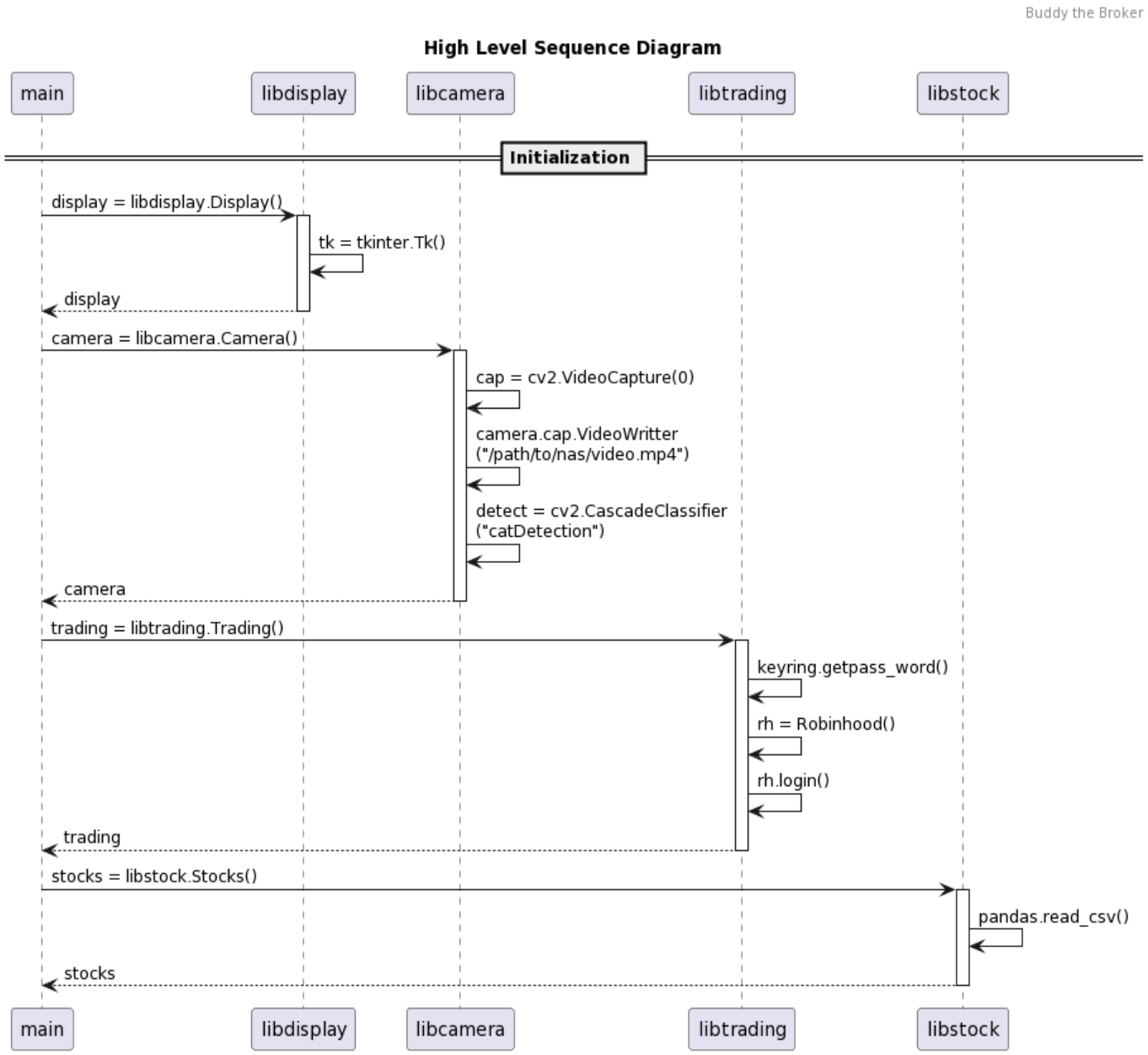

Figure 7: OV-6C Diagram Initialization

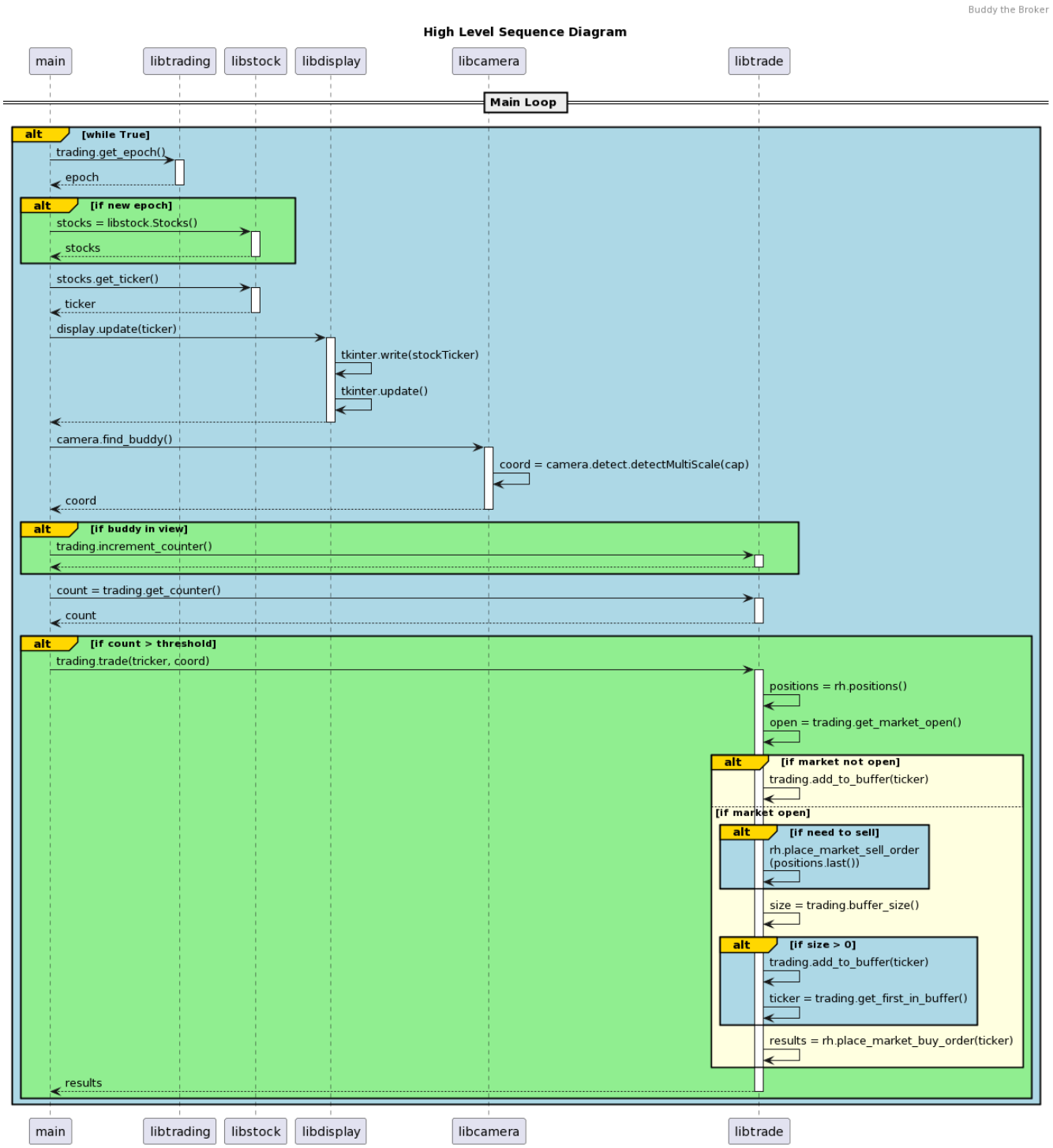

Figure 8: OV-6C Diagram Main Loop

Samuel Mehalko
Embedded Systems Development Lab
EN.525.743.8VL.FA22
Critical Design Review
October 3rd 2022

5 Material and Resource Requirements

5.1 Material Requirements

5.1.1 Main Processor: Raspberry Pi 4

"*The Raspberry Pi 4 offers ground-breaking increases in processor speed, multimedia performance, memory, and connectivity compared to the prior-generation boards, while retaining backwards compatibility and similar power consumption. The Raspberry Pi 4 provides desktop performance comparable to entry-level x86 PC systems. The Raspberry Pi 4 comes in three on-board RAM options for even further performance benefits: 2GB, 4GB and 8GB.*

Figure 9: Raspberry Pi4

This product's key features include a high-performance 64-bit quad-core processor, dual-display output via two Micro HDMI ports, up to 4K resolution, hardware video decoding at up to 4Kp60, up to 4GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0, and PoE capability." - Vendor description.

Link to device:

https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

5.1.2 Camera: Raspberry Pi Camera Module 2

"The Raspberry Pi Camera Module 2 replaced the original Camera Module in April 2016. The v2 Camera Module has a Sony IMX219 8-megapixel sensor (compared to the 5-megapixel OmniVision OV5647 sensor of the original camera).

Figure 10: Raspberry Pi Camera Module 2

The Camera Module 2 can be used to take high-definition video, as well as stills photographs. It's easy to use for beginners, but has plenty to offer advanced users if you're looking to expand your knowledge. There are lots of examples online of people using it for time-lapse, slow-motion, and other video cleverness. You can also use the libraries we bundle with the camera to create effects." - Vendor description.

Link to device:

https://www.raspberrypi.com/products/camera-module-v2/

5.1.3 Display: Raspberry Pi Touch Display

"Raspberry Pi OS provides touchscreen drivers with support for ten-finger touch and an on-screen keyboard, giving you full functionality without the need to connect a keyboard or mouse.

The 800 x 480 display connects to Raspberry Pi via an adapter board that handles power and signal conversion. Only two connections to your Raspberry Pi are required: power from the GPIO port, and a ribbon cable that connects to the DSI port on all Raspberry Pi computers except for the Raspberry Pi Zero line." - Vendor description.

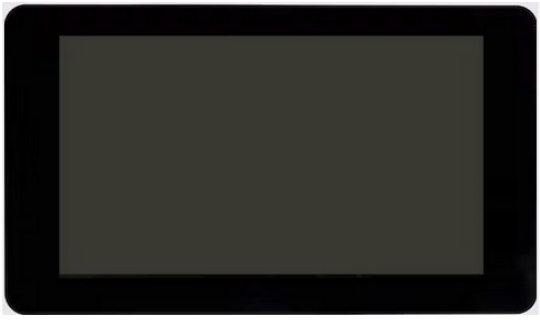

Figure 11: Raspberry Pi Touch Display

Link to device:

https://www.raspberrypi.com/products/raspberry-pi-touch-display/

5.1.4 Case: SmartiPi Touch 2

"The SmartiPi Touch 2 is a case for the Official Raspberry Pi Display, Raspberry Pi and Raspberry Pi camera V1 or V2. Not compatible with the Raspberry Pi HQ camera." - Vendor description.

Link to device:

https://smarticase.com/products/smartipi-touch-2

Figure 12: SmartPi Touch 2

5.1.5 Secondary Processor: Raspberry Pi Zero W

"*The ultra-small and ultra-slim Raspberry Pi Zero W is the smallest form factor Raspberry Pi on the market now incorporating WiFi and Bluetooth connectivity on board. It is 40% faster than the original Raspberry Pi but measures only 65mm long by 30mm wide and 5mm deep. The Raspberry Pi Zero W supports mini connectors to save on space and the 40 pin GPIO is unpopulated providing the flexibility to use only the connections your project requires.*" - Vendor description.

Figure 13: Raspberry Pi Zero W

Link to device:

https://www.raspberrypi.com/products/raspberry-pi-zero-w/

5.1.6 Storage: IronWolf 4TB Hard Drive

"IronWolf® Pro drives are engineered to deliver 24×7 performance, reliability, and dependability in multi-bay, multi-user commercial and enterprise RAID storage solutions. Total peace of mind with five-year limited warranty, complimentary three-year Rescue Data Recovery Services, and IronWolf Health Management." - Vendor description.

Figure 14: IronWolf 4TB Hard Drive

Link to device:

https://www.seagate.com/products/nas-drives/ironwolf-hard-drive/

5.1.7 Storage Docking: RSHtech Docking Station

"Hard Drive Docking Station Supports Standard 2.5"/3.5" SATA SSD and HDD, supports hard drives up to 16TB. Tool-free installation, plug & play, no driver required." - Vendor description.

Figure 15: RSHtech Docking Station

Link to device:

https://www.rshtech.com/products/hard-drive-docking-station-rsh-ds01

5.1.8 Where/When Is it Coming?

A majority of the needed materials were already in my possession. As of September 20th 2022 all materials that were not already in my possession that are necessary for this project have arrived.

Materials already owned:

Main Processor: Raspberry Pi 4

Secondary Processor: Raspberry Pi Zero W

Storage: IronWolf 4TB Hard Drive

Storage Docking: RSHtech Docking Station

Materials ordered and already arrived:

Camera: Raspberry Pi Camera Module 2

Display: Raspberry Pi Touch Display

Case: SmartiPi Touch 2

5.2 Tool Requirements

5.2.1 Python3

Python3 will need to be installed on the Raspberry Pi in subsystem A in order for the software to run as expected. I am using python3.8 so python3.8 or greater should be used to ensure compatibility.

Several python libraries will also be required. These libraries are listed in Python Libraries Leveraged.

5.2.2 Secure Shell (SSH)

A means to interface with both Raspberries Pi's will be necessary to install and run required software. The SSH protocol is one such method but there are other equally valid ways to interface with these embedded computers.

5.2.3 Open Media Vault

Open Media Vault is required for subsystem B. The software is discussed further in sections Subsystem B: Network Storage and Open Media Vault.

6 Development Plan and Schedule

The table in section Tentative (Planned) Milestones/Schedule outlines the capability based development approach. The order of development will be week by week with major accomplishments set forth for each week.

6.1 Tentative (Planned) Milestones/Schedule

Week	Goal
1 (10/10)	• Display Library (libdisplay) written • Raspberry Pis flashed and all necessary hardware connected • Network storage setup
2 (10/17)	• Stock class (libstock) written • Trading library (libtrading) can buy and sell stocks
3 (10/24)	• Camera library (libcamera) can identify cats and return coordinates
4 (10/31)	• Camera library (libcamera) draws box and send video to remote storage • Trading library (libtrading) algorithm started
5 (11/7)	• Trading library (libtrading) algorithm completed
6 (11/14)	• Main written connecting all libraries
7 (11/28)	• Acceptance testing & polish product
8 (12/5)	• Prepare final design report/demo

6.2 Potential Risks

Below in sections Week 3 and Week 5 are outlined potential risks I see in the project schedule. These risk capabilities are spread to some extent across two week boundaries as a risk mitigation effort. The week of Thanksgiving (11/21) has not been accounted for and work can bleed into this should need be. Moreover, time has been accounted for to enable acceptance testing and polishing of the product (thorough testing, documentation, reduction of technical debt, general cleanup, etc). Should the base functionality be at ask the polishing activities could be reduced.

6.2.1 Week 3

Camera library (libcamera) can identify cats and return coordinates -

Never using any computer vision software/libraries in the past I expect a decent learning curve for this weeks work. Moreover, the raspberry pi's limited computation power will add an additional layer of complexity to this problem. I fully expect that the frame rate will be single digit at best and the detection boundaries may not always be spot on. For this project that yields no larger issues as the computer vision module only needs to determine if a cat is in view for an extended period of time.

6.2.2 Week 5

Trading library (libtrading) algorithm complete -

Due to the complexity of this algorithm and limited testing time (stock market is only open week days from 9:30 a.m. - 4 p.m. Eastern time) I expect this week will contain quite a bit of work. For that reason I choose to start working on the algorithm in week four and to have the ability to buy and sell stocks by week three. Looking into the pyrh library I plan on leveraging there is no available function calls for fractional stock buys/sells. This will means that I will either have to implement fractional stock buys/sells or account for the different prices of different stocks in my algorithm. For example the algorithm may need to sell of several other stocks before it can by the newly desired stock.

7 References

7.1 Python Libraries Leveraged

7.1.1 Keyring

"The Python keyring library provides an easy way to access the system keyring service from python. It can be used in any application that needs safe password storage." - Library description

Link:

https://github.com/jaraco/keyring

7.1.2 OpenCV

"OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. " - Library description

Link:

https://github.com/opencv/opencv

Figure 16: OpenCV Logo

7.1.3 Pandas

"pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way towards this goal." - Library description

Figure 17: Pandas Logo

Link:

https://github.com/pandas-dev/pandas

7.1.4 Pyrh

"Python Framework to make trades with Unofficial Robinhood API." - Library description

Figure 18: Pyrh Logo

Link:

https://github.com/robinhood-unofficial/pyrh

7.1.5 Tkinter

"The tkinter package ('Tk interface') is the standard Python interface to the Tcl/Tk GUI toolkit." - Library description

Link:

https://docs.python.org/3.8/library/tkinter.html

7.2 Open Source Programs Leveraged

7.2.1 Open Media Vault

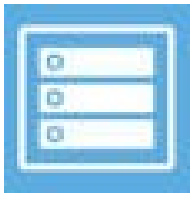

Figure 19: Open Media Vault Logo

"openmediavault is the next generation network attached storage (NAS) solution based on Debian Linux. It contains services like SSH, (S)FTP, SMB/CIFS, DAAP media server, RSync, BitTorrent client and many more. Thanks to the modular design of the framework it can be enhanced via plugins.

openmediavault is primarily designed to be used in small offices or home offices, but is not limited to those scenarios. It is a simple and easy to use out-of-the-box solution that will allow everyone to install and administrate a Network Attached Storage without deeper knowledge." - Tool description

Link:

https://www.openmediavault.org/about.html

7.3 Miscellaneous References

1. DODAF - DOD architecture framework version 2.02 - DOD deputy chief information officer. (n.d.). Retrieved September 17, 2022, from https://dodcio.defense.gov/library/dod-architecture-framework/

2. Ferri, R. (2022, April 14). Any monkey can beat the market. Forbes. Retrieved September 24, 2022, from https://www.forbes.com/sites/rickferri/2012/12/20/any-monkey-can-beat-the-market/?sh=4f3443dd630a

3. Kueppers, A. (2001, June 5). Blindfolded monkey beats humans with stock picks. The Wall Street Journal. Retrieved September 24, 2022, from https://www.wsj.com/articles/SB991681622136214659

4. Yahoo! (n.d.). How a monkey beats the stock market. Yahoo! Finance. Retrieved October 1, 2022, from https://ca.finance.yahoo.com/news/monkey-beats-stock-market-172803113.html

5. Stephen, G. (2020). *I Spent $100,000 On A Stock Picking Monkey*. *YouTube*. Retrieved October 2, 2022, from https://www.youtube.com/watch?v=TA-P5ilI_Vg

6. Stephen, G. (2021). *How To Make Easy Money In The Stock Market*. *YouTube*. YouTube. Retrieved October 2, 2022, from https://www.youtube.com/watch?v=5_3Ra-Q6vK4

7. Vaughan, A. (n.d.). Planttext UML editor. PlantText UML Editor. Retrieved September 17, 2022, from https://www.planttext.com/

8 Appendix

8.1 Source configuration management

For source configuration management I will be using github. I plan to commit somewhat frequently to ensure no work would be lost should my local hard drive storing my work fail.

Link:

https://github.com/SamuelDonovan/BuddyTheBroker

8.2 Documentation Website

Using Sphinx and ReadTheDocs (no listed as requirements as they are not needed for base functionality) a website is being hosted with documentation on all code written as part of this project. This auto-documentation can be leveraged throughout the development of this project and be used as a form of documentation for the final project document.

Link:

https://buddythebroker.readthedocs.io/en/latest/

8.3 Code Formatting and Linting

To follow PEP-8 standards the code formatter “Black” is being used. This again is not listed as a requirement for this project as its not needed for base functionality but rather improved code readability.

Link:

https://github.com/psf/black

8.4 Unified Modeling Language (UML) Source

Below is the source code used to generate the diagrams for section Unified Modeling Language (UML) Diagrams.

```
@startuml

header Buddy the Broker
title High Level Sequence Diagram

== Initialization ==

main -> libdisplay: display = libdisplay.Display()
activate libdisplay
libdisplay -> libdisplay: tk = tkinter.Tk()
return display

main -> libcamera: camera = libcamera.Camera()
activate libcamera
```

```
15  libcamera -> libcamera: cap = cv2.VideoCapture(0)
16  libcamera -> libcamera: camera.cap.VideoWritter\n("/path/to/nas/video.mp4")
17  libcamera -> libcamera: detect = cv2.CascadeClassifier\n("catDetection")
18  return camera
19
20  main -> libtrading: trading = libtrading.Trading()
21  activate libtrading
22  libtrading -> libtrading: keyring.getpass_word()
23  libtrading -> libtrading: rh = Robinhood()
24  libtrading -> libtrading: rh.login()
25  return trading
26
27
28  main -> libstock: stocks = libstock.Stocks()
29  activate libstock
30  libstock -> libstock: pandas.read_csv()
31  return stocks
32
33  == Main Loop ==
34
35  alt#Gold #LightBlue while True
36
37  main -> libtrading: trading.get_epoch()
38  activate libtrading
39  return epoch
```

```
40
41  alt#Gold #LightGreen if new epoch
42  main -> libstock: stocks = libstock.Stocks()
43  activate libstock
44  return stocks
45  end
46
47  main -> libstock: stocks.get_ticker()
48  activate libstock
49  return ticker
50
51  main -> libdisplay: display.update(ticker)
52  activate libdisplay
53  libdisplay -> libdisplay: tkinter.write(stockTicker)
54  libdisplay -> libdisplay: tkinter.update()
55  return
56
57  main -> libcamera: camera.find_buddy()
58  activate libcamera
59  libcamera -> libcamera: coord = camera.detect.detectMultiScale(cap)
60  return coord
61
62  alt#Gold #LightGreen if buddy in view
63  main -> libtrade: trading.increment_counter()
64  activate libtrade
```

```
65 return
66 end
67
68 main -> libtrade: count = trading.get_counter()
69 activate libtrade
70 return count
71
72 alt#Gold #LightGreen if count > threshold
73 main -> libtrade: trading.trade(tricker, coord)
74 activate libtrade
75 libtrade -> libtrade: positions = rh.positions()
76
77 libtrade -> libtrade: open = trading.get_market_open()
78
79 alt#Gold #LightYellow if market not open
80 libtrade -> libtrade: trading.add_to_buffer(ticker)
81 else if market open
82 alt#Gold #LightBlue if need to sell
83 libtrade -> libtrade: rh.place_market_sell_order\n(positions.last())
84 end
85 libtrade -> libtrade: size = trading.buffer_size()
86
87 alt#Gold #LightBlue if size > 0
88 libtrade -> libtrade: trading.add_to_buffer(ticker)
89 libtrade -> libtrade: ticker = trading.get_first_in_buffer()
```

```
90  end
91
92  libtrade -> libtrade: results = rh.place_market_buy_order(ticker)
93
94  end
95
96  return results
97  end
98  end
99
100  @enduml
```