Interim Report

Radek Chramosil

WorldQuant University

E-mail: radek@keemail.me

This document is the Interim Report for the Capstone Project. It shows the basic structure for the database that holds data the data from crypto-currency exchange Binance and some initial data analysis.

***Keywords*—** High frequency trading, Market microstructure, Algorithmic trading

Introduction

The structure of the database is described below. Additionally, I provide more details about the data analyses so far carried out as well as some suggestion for trading possible trading strategies. Some plots for second data set are provided. The asset I worked with is **TUSDBTC** as quoted on *Binance Exchange*. The asset is Bitcoin quoted in USD. The access to the full exchange is free and level 2 data is available for every tick. The data was collected on second basis for one hour. The collection was done with 1 second frequency as opposed to per trade basis (I am still unsure how to handle data collection per trade basis). The data set is therefore some 3440 full market book hits; some 100 quotes for bid prices and some 100 quotes for ask prices.

Research Topics

This sections highlights some issues in the so called capital market micro-structure. I focus on technical aspects mostly. It is often difficult to identify or read about the specifics, say, of an API as used by some exchange. So, I use my work on Bitcoin and Binance exchange as an example. I mostly focus on discuss about US stock market as this is the most commonly discussed topic in the literature. This is continuous action order-driven market. I provide the time units definitions for convenience.

Second to lower time intervals	
1 second	1 000 milliseconds
1 second	1 000 000 microseconds
1 second	1 000 000 000 nanoseconds

Electronic Trading Chain

There are many aspects that can be discussed with regards to trading chain links between brokers, exchanges, investors (retail, institutional or algorithmic trading engines). However, Durbin (*5*) mentions that for for a high frequency firm access the exchange for two reasons only. One is to submit their trade flow (trade execution) and the other is the receipt of market data. This can be generalised to any market participant. Hence, I provide the Figure 1 that shows high level overview.

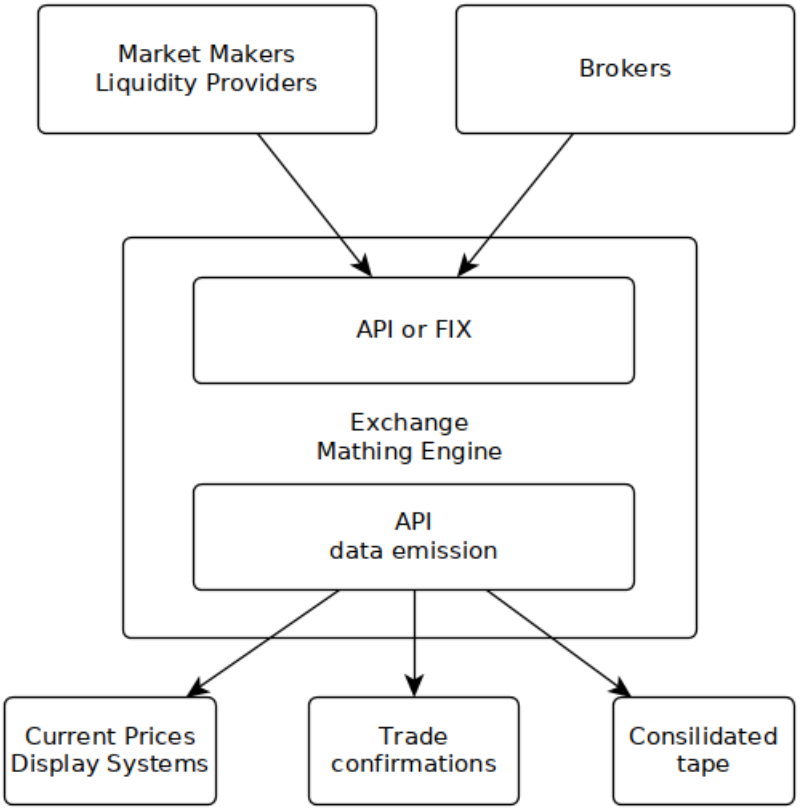

Figure 1: Market Structure Overview

Matching Engine The core about any trade linking the matching mechanism. Johnson (*9*) provides high level structure for the matching engine. Each order (that is price and quantity for bid or ask) is processed as it arrives to the matching engine. The below steps are taken each time there is a new order.

- The order instruction is added to internal order book
- Trade matching rules are then used to see if any matches are now possible
- The order book is updated to reflect any changes
- Execution notifications are sent to any result matches

These steps essentially are the core of any order driven market. Figure 2 shows how LSE displays their market order book to their clients (*14*).

In an attempt to assess how this matching algorithm might work, I used Binance exchange as an example. This exchange sees the order book at a new state each time there is an order and uses a data field 'UpdatedID' to mark this state. The exchange members are the able to request the order-book state stamped by this ID. Binance uses only REST API which we can query programmatically, say using Python's Requests package. On the contrary, on the NYSE's website we can read about different services this exchange offers to their clients (*15*). We see that they have an event-based depth of book feed available, which is not something Binance offers as yet. Programmatically, event driven API would be a better option for us, however we can expect that NYSE charges their fee for this service.

To understand how to classify the different levels of service an exchange might provide, it is useful to consider the latency differences. O'Hara (*17*) provides a good summary of services for Tokyo exchange. This exchange offers Arrownet, the standard service, which has the latency of several milliseconds. Their priority service allows users to place devices at data center entry points for their network, lowering the latency to 260 microseconds. Finally, the co-location at their primary side offers 15.7 nanoseconds service. We can learn from this that the simplest and most straight forwards way to classify the services the exchange provide is based around the time latency. Unfortunately, the exchanges are not transparent around the latencies. NYSE does not talk about latencies at all on their website (*15*). Binance simply leaves this issue to the market participants as any one can access their matching engine over the Internet.

Further, the important aspect of co-location is that once co-located the market participants have the cable of the same length, see the interview with Manoj Narang (*20*). Once, we pay for the co-location service, we should have the same delay as anybody else who pays for the service.

Financial Information Exchange protocol (FIX protocol, see Johnson (*9*) for details, it is just a version of XML) and exchange application programming interface (API) provide a way to submit orders. The broker orders as as well as market makers therefore must reach the matching engine the same way even though that the difference is that some people press buttons on their screen and some people monitor the market programmatically and their programme will submit the order. It only that seems to matter is the latency. FIX should be slower as it needs to interpreted as opposed to and API but it is used as it is common to the whole market. Johnson also (*9*) mentions exchanges have separate servers for processing orders and emitting data. We would have to investigate what a specific exchanges does before we start trading.

After a trade is submitted to the machining engine, the machining engine must make its state available to the market, specifically market maker, broker, data vendors or authorities. This includes trade confirmation but also open orders as well as the matched trades. In the US an exchange may submit the confirmation firstly to Consolidated Tape Authority (CTA) - see Durbin (*5*). CTA then in turns should have all market data and some hedge funds use it as a data source. While Europe is missing the consolidated type all together as suggested by O'Hara (*17*). Perhaps, the rest of the world faces this issues too.

SETS screen

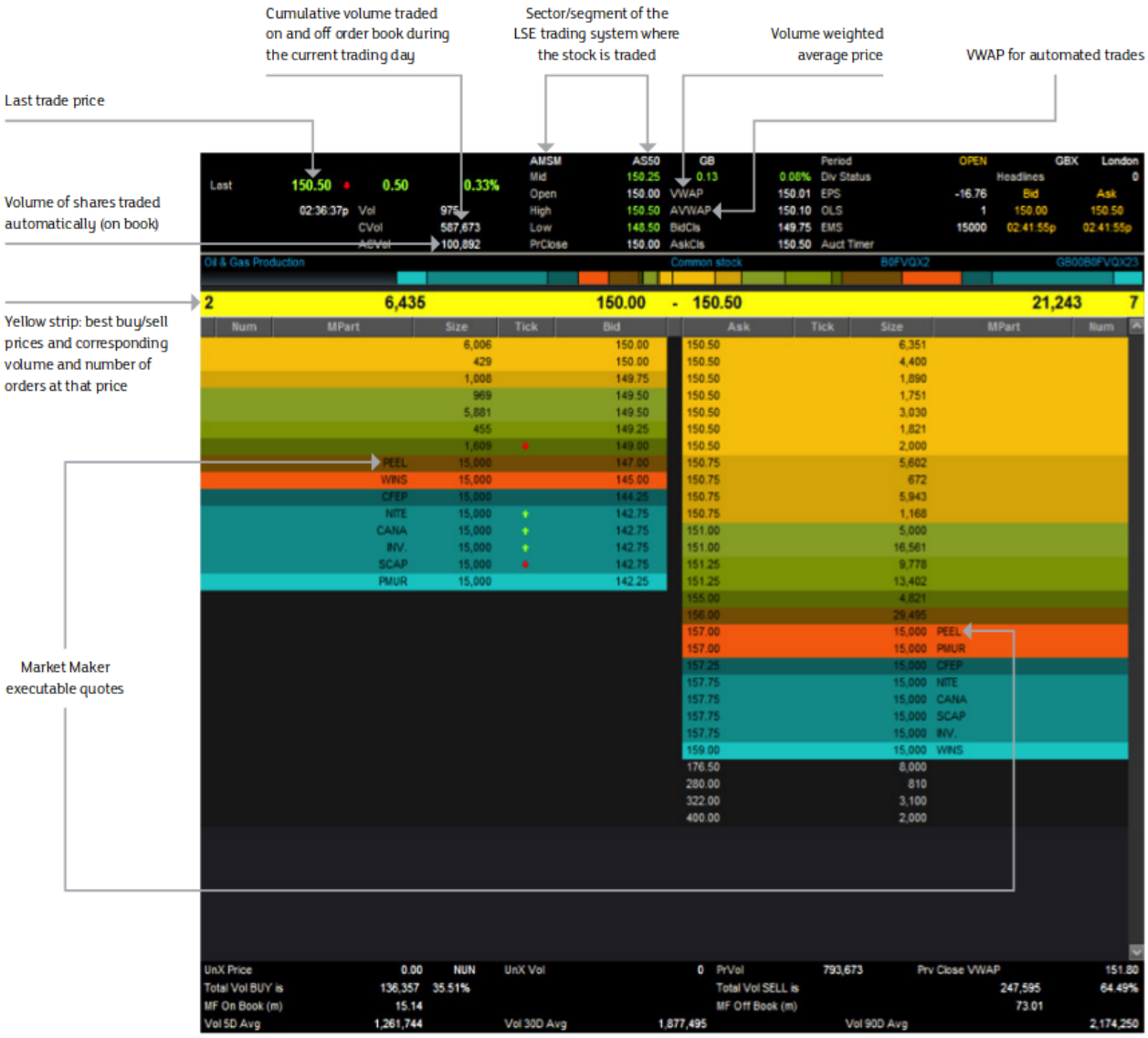

Figure 2: LSE Market Order Book screen

Brokers The traditional way for people to access the market is over the phone via a broker; refer to Harris (*6*) to see what specific applications or systems are involved. I just mention that in the principle for the order-driven market the order must end up in the same order book. However, brokers actually root the market order to the venue that pays them the most, see Harris (*6*) or O'Hara (*17*). O'Hara mentions that there are some 15 different stock exchanges and 50 alternative venues. In order for the broker to give their clients competitive edge over poor algorithmic trades, brokers now offer algorithmic execution for trades

as well. As en example, I look into Investment Technology Group's (ITG) executions strategies (*8*). Their clients can benefit from a large variety of different execution algorithms or build their own. ITG also DMA in some forms. So, we see that the playing field is levelling up even for retail traders.

Algorithmic Trading Engines The structure for an algorithmic trading engine is reasonably well described by Durbin (*5*). He provides summarises his ideas in the below flow chart in Figure 3.

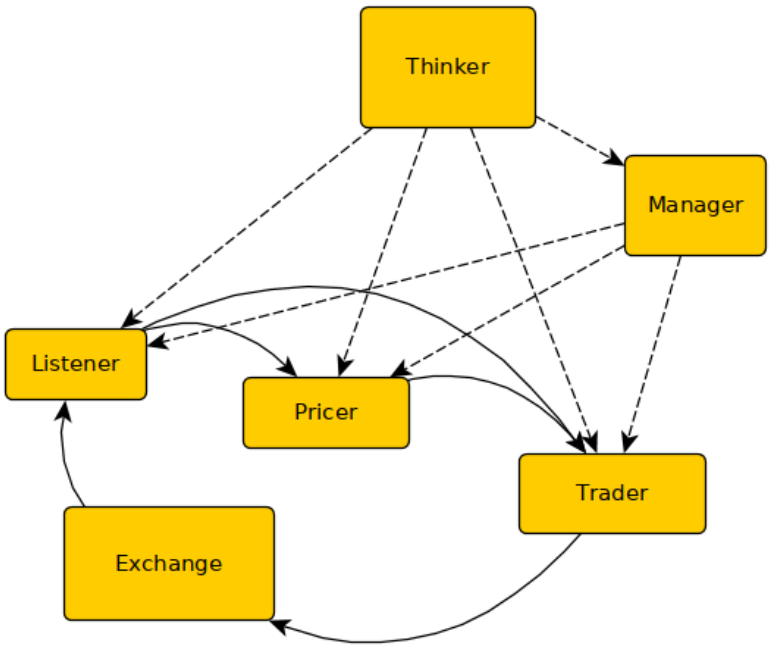

Figure 3: High Frequency Trader - Operational Structure

He defines the terms in the following. *Thinker* take direction from humans. This would GUI for the application. *Manager* controls on the other components. So, this would be the main server or the main program that oversees all the programs. Then again we have *exchange*. This is the matching engine as discussed above. *Listener* takes in the market data and make them available to other components. Durbin states that this should sit as close as possible the market data source (exchange or CTA) and take in every tick. Listener also does checks for errors or arbitrage if multiple sources disagree. *Pricer* Durbin states that this component calculates the price of assets in the real-time however we know that we could use any strategy or use any mathematical model here, not just pricing. This application or a function receives the inventory information from *Manager*. *Trader* submits/cancels the orders in the exchange. Finally, *Manager* is a program that controls all the other components.

We can see how a general algorithmic trading operation may run from Durbin's discuss, however we know that the reality of algorithmic trading is more complicated than that. We also see that this structure fits well into our discussion well, see Figure 1. Further, Johnson (*9*) provides code C++ snippets that can be established in individual components.

DMA Services Johnson (*9*) provides a good definition for Direct Market Access (DMA), DMA means that a computerised system is reponsible for executing the orders to buy or sell a given asset, rather than being worked manually by a trader. It should be noted that the broker can give their client this access rather than the exchange itself, however many exchanges give DMA to their market makers. In all my discussion, I assumed DMA as the whole market shifts towards computerised systems anyway.

Benefits and Pitfalls of DMA technologies

DMA Types Johnson (*9*) provides a good overview of DMA services available. I just provide a brief definitions and suggest a way how we should study the pros and coins as market participants.

Direct Market Access In section, we define *Direct Market Access* (DMA) as the situation when broker's client makes use of the brokers infrastructure to send the orders to the exchange, like broker's own orders. *Sponsored Access* term is used when the broker's client is given broker's access credentials to execute the trades directly at the exchange, but not having to use the broker's infrastructure. This should be faster however there is some connection overhead for the broker to monitor the trading to prevent any excessive risk taking by the client. *Crossing* refers to the situation when brokers offers the access to the dark pool to their clients. This, Johnson claims, to be used when a client is looking to carry out a large trade and is trying to hide it from a many market participants that are able to respond to the orders visible in exchange order book. *Direct Liquidity Access* is then the term when the broker offers the access to dark pools and to the exchanges at the same time. In addition, some brokers started to use the term *Direct Strategy Access*. This means broker provides their client with execution algorithms.

Thus, we see that the way to compare different ways to access the markets can be classified by

- latency,
- venue availability,
- execution algorithm,
- price.

We would need to research what specific deals are available to us from different brokers and decided on the best deals based on the above criteria.

One final thought about DMA comparison is that in case of crypto-currency market, there are no brokers at all. One additional form of DMA is not have a broker.

As for the benefits, the obvious case for them that automated markets are more efficient so cheaper, that is the trading is cheaper, which was the original cause for automation, see Patterson (*18*). This authors talks about the fact that NASDAQ market maker kept their spreads artificially high in the past. The solution was to automate the trading. Also, one more obvious benefits is that direct market access provides greater transparency with regards to the price data and well as market order book. The exchange members can request all the data their desire, however in case of standard capital markets this subscription is paid. DMA technologies may allows us to bypass brokers, who often have a lot conflicts of interest with respect to their clients (broker can sell their client order flow or they execute market orders on behalf of their clients when they deem that the position is too risky and collect the trade fee).

This is the case in the crypto-currency space. In addition, we test trading strategies under live access to the market.

On the other hand, the market automation created problems at the same as it had provided us with benefits. Exchanges now sell access to a public information, they sell access to the market order book with difference latencies. As a results, they turn public information into private information for time intervals measured in microseconds or nanoseconds (*17*). In fact Brad Katsuyama claimed in his interview with BBC that the exchanges make more money by selling high speed data and technology than they make by matching trades . It is clear that some market participants benefit from this latency advantage at expense of the market participants that did not pay for the low latency access, see Patterson (*18*). Another problem is that the matching engines are not publicly available in standard markets, maybe not in some obscure crypto-currency open-source space. What this means that the exchange can create or special hidden order types, which are again given just to some specific market participants and the other may not know this, see Bodek (*2*). This author and active market participant claims that his algorithmic trading operation went in the red as he did not have access to the hidden marker order types while his competitors had. He talks about hidden Alphas in different order types. So, we that the problem is actually the fact that the matching engine is not public information just the API is.

Therefore, we can see that algorithmic trading creates transparency as more data is available now while at the same time it does exactly the opposite in some other sense.

Risks in DMA

DMA does have obvious risks. Knight Capital lost $440 million in trading loss due to a bug in their systems (*16*). The company suffered power disruptions and it was not able execute orders on some other day. AXA Rosenberg lost $217 million in client money due to an error in their model for managing risk (*4*). So, we see that one of the big problems and risks with DMA is the coding errors.

Further, the all technical risks that linked to running a data centres such as power shortages, car or track accidents causing evacuation of the area, flights patterns, flat plains, nuclear plants, rail lines, crime level and even light night strikes. All of these events could cause an outage in a data facility (*22*). DMA is directly dependent on all of these as it is carried out in the data centres physically next to where the matching engine server is located. In case of computer networks, we need to add risk that the connections may fail for the same reasons and many other as in case of a data centre.

As previously mentioned, we do not have a full access to the matching engine, so we run a risk that the exchange gives better order types so users over as another users.

There are also topics such as slippage, front-running, quote-staffing (see, Durbin (*5*)), however there are more general to the market structure not to the DMA itself.

Latency Types

The basic ideas behind the latency issue are well summarised by Kirilenko (*10*). He points that a standard way to measure latency is by determining the time it takes a given message to travel from source to destination and back - *round-trip time* (RTT). he talks about

- *communication latency* - can be reduced by co-location
- *market feed latency* - can be reduced by buying proprietary data feed

- *trading platform latency* - cannot be reduced by a trader, taken as given

Kirilenko and Lamacie (*11*) also points that the latency is a random variable as in Figure 4 where we can guess what the distribution properties are.

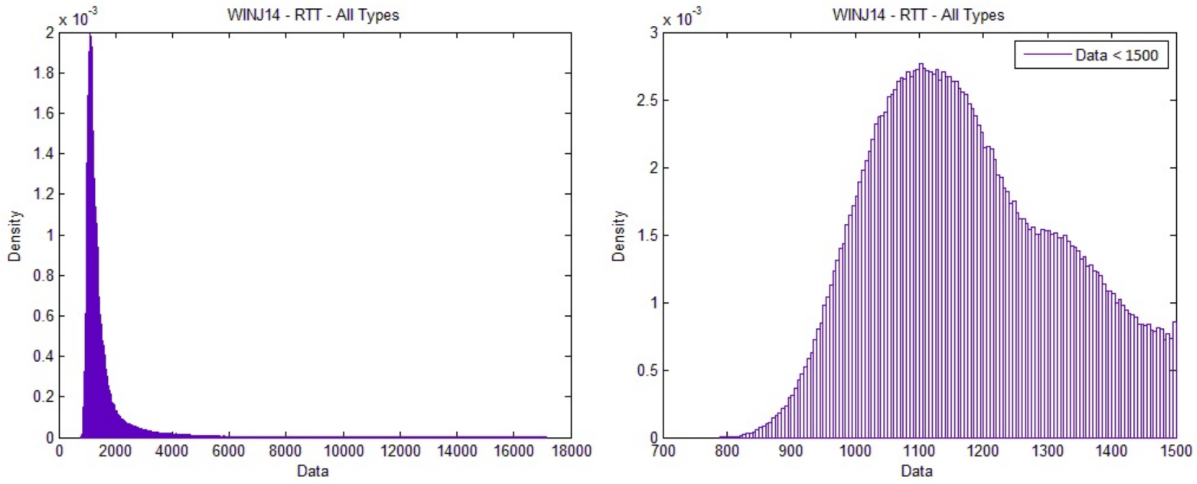

Figure 4: RTT - All Types: Submissions, Modify, Cancel - from (*11*)

Further, we can break down the latencies in a different way. Kumar (*12*) provides a useful overview of how trading system latencies can be defined and measured. He summaries the problem at hand as:

$$\begin{aligned}\text{Trading System Latency} = &(\text{All the processing node latencies})\\ &+ (\text{Operating system network stack latencies})\\ &+ (\text{The entire network device latencies})\\ &+ (\text{The entire wire time latencies})\end{aligned}$$

Kumar (*12*) then defines:

- *Processing node latency* is the time spent within the application business rules and logic.
- *Operating system network stack latency* is the time spent within operating system calls and hardware network interfaces.
- *Network device latencies* is the time spent in networking equipment (load balancers, firewalls and routers).
- *Wire time* is the time it takes for a message to transmitted on a physical medium. Specifically, a foot of a cable equates to a nanosecond (*1*). It is 50 to 55 micro seconds to data to travel one way on 10 km of fiber optic (*12*).

This discussion shows that to reduce the latency within a computer system that we need to drill into every step our computer does along the way between the the computer and the matching engine. That we would have to use the most efficient algorithms and data structures to handle the processing node latency. The industry uses real-time operating systems (*5*). Alternative way to use the just a chip set without the operating system. Wire lengths are reduced by co-location. The work documented by Algo-Logic may serve as a good example (*13*).

It is worth noting that a lot of low latency issues are linked to the US equity markets and the equity instruments that have multiple venues which then leads to the latency arbitrage, however one way to solve this is to just use instruments that have only one trading venue. We may stay from the race altogether. As a result, HFT firm may build its business around mathematical modelling, see how the company called RSJ does the business (*21*).

0.1 Typical Order Messaging System

I was not able to any specific information with regards to type order messaging process - Order Request, Data Conversion, Order Enrichment, Order Transmission and Exchange Acknowledgement for any standard exchange. The Investor Exchange provides REST API (*7*) for the data but not for orders. We check how to communicate to Binance exchange using a python package (*19*), however that is also REST API.

We are left with general suggestions about how to reduce latency. C++ is seen as the fastest language that gives control over the computer at very low level (memory pointers, memory management). Generally, it is good to keep the data in memory as well so many algorithmic trading operations use KDB, which is a in-memory relational time series database. The coding style should be modular and functional as opposed to object orientated as inheritance slows the program down. In addition, jobs like error handling comes with an overhead. So, more steps in the code, the better enough though we might be short on standard coding practices. As for operating system, Durbin (*5*) mentioned Linux (CentOS is mentioned by Lockwood (*13*)). However, we still should remove a lot of functionality, such as a graphical interface, for the server where would trade from. The point is that we wish to run our application instantaneously and at the constant time. We wish to remove as much randomness from our latency as possible. So, we can control the system. Obvious options to consider are parallel processing within a single server. This is both CPU multi-threading (Durbin (*5*) mentions Threading Building Blocks C++ runtime library by Intel) but also GPU processing as used in TensorFlow, however in our case we would not worry about neural networks but focused on, say, pricing 1000 options at the same time. The distributed computing is not used not used as there is an overhead to distributing the computation across multiple computers.

If we wish to go more experimental, we could start looking into new operating system or coding languages. Three-year-old Redox OS is a MIT licensed micro kernel with some 20 000 lines of code fully written in coding language called Rust. It is reasonable to expect this OS to be faster then Linux as Linux has millions of lines of code. Redox OS has modular structure for drivers, so we would be able to amend or change drivers as we like. Linux does not have this modularity. In addition, Rust is a new c-based language, it borrows from C++ and improves on the weakness of C++, for example it have native support for multi-threading. On the contrary, both Rust and Redox OS are not tested or industry proven as C++ and Linux is.

To sum up, the high frequency space is ever changing and all we can do is to keep as best as we can (or have the time to do so).

Strategy with High Frequency Data

I detail what I have worked on as an example usage of high frequency data.

Database Structure

I have included the below tables in this submission. The tables can be created by SQL commands in the "create_db.sql" file. The schema for the database is shows in Figure 5.

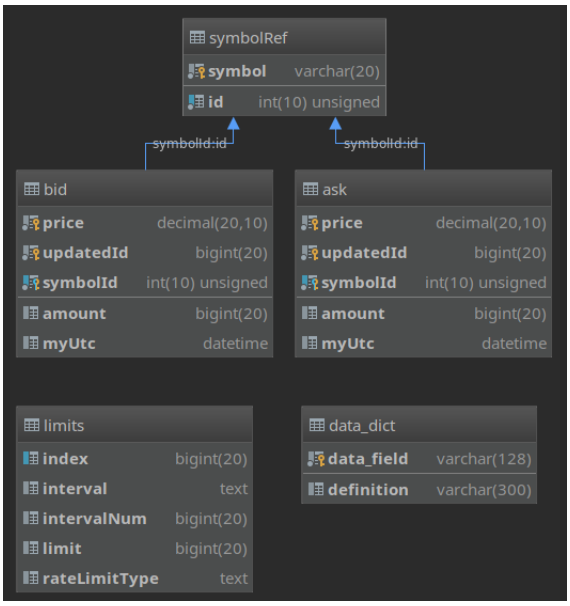

Figure 5: Schema

The tables can be dropped using the "rollback_db.sql" file. This file provides a logical inverse to the create operation. Additionally, I built functionality that can create table on fly using Pandas and SQL Alchemy however the default behaviours are not very good as it fails to create foreign keys and so on.

Tables

symbolRef. This is just a reference table for symbols. The symbols are strings, however it is more efficient to use integers for foreign keys and it is better to store integers than strings.

bid and ask. These to table are identical from data types point of view however the just store the bid and ask prices as well as my UTC time stamp and the server 'updatedId' flag. This is a large integer that gets changed each time there is a trade.

limits. . Basically, the exchange give users request or order limits in time intervals. We can make 1200 data request in a minute or 10 orders in a second and 100 000 orders in a day. If we are creating a real trading application we would have to consider these closely.

data_dict. This is manually created table and holds descriptions for the data fields in the database.

Bid and Ask

Bid and ask prices are split into two tables. The tables are identical. The tables from Figure 6 and Figure 7 carry the data for the symbol and specific Exchange ID as given by *updateId* column. My time stamps are in the *myUtc* column representing the time on my computer.

price	amount	updatedId	myUtc	symbol
0.00020497	1806	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020503	147	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020504	6	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020511	9	48208950	2018-11-19 21:03:04	TUSDBTC
0.0002052	4453	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020538	4453	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020546	18528	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020556	4453	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020558	1194	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020576	34	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020577	15	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020586	5	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020587	5	48208950	2018-11-19 21:03:04	TUSDBTC

Figure 6: Ask Price

price	amount	updatedId	myUtc	symbol
0.00020471	504	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020468	183	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020466	281728	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020465	10	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020463	49	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020461	20	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020456	20	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020455	10	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020454	5	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020453	10	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020452	10	48208950	2018-11-19 21:03:04	TUSDBTC
0.0002045	6	48208950	2018-11-19 21:03:04	TUSDBTC
0.00020449	30	48208950	2018-11-19 21:03:04	TUSDBTC

Figure 7: Bid Price

Exchange API

It is important to handle the communication to the Exchange well. The class *DataEndPoint* hold all methods that can collect the data from the exchange as opposed to submit trades. I leave it to the reader to read over the comments for these methods to see what can be easily pulled out from the exchange.

Data Analysis - First Look

I collected 1 hour of trading data on per second basis. Figure 8 shows cumulative bid and ask prices with quantity. The way how I think about this is it is supply and demand curves. We see that there is a lot of volatility over 1 minute interval. This is good as it means that we will be able to create strategies.

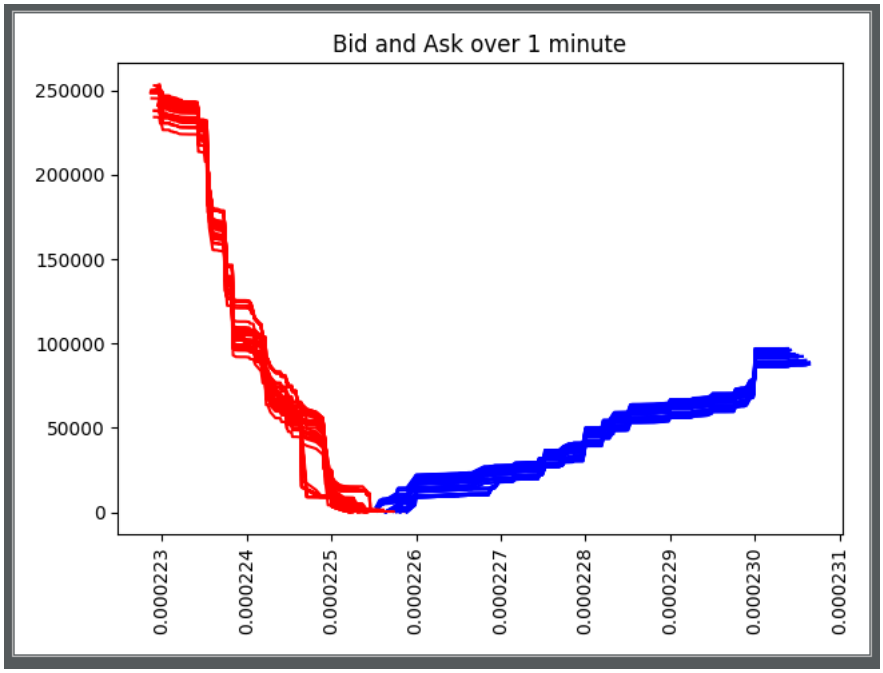

Figure 8: All bid and ask prices and quantities in 1 minute

The problem in the Figure 8 is that we do not see much details. So, I plotted the same time interval but with close look at what was happing around the spread in Figure 9. Again we see the volatility in the price within one minute.

We also have have look the maximum bid and minimum ask prices and plot them over the 1 hour interval. This is plotted in Figure 10. This shows that we do have trends and turning points happing over time again. We should use the trading strategies that we learning for this and we can starting simulating the trade at bid and ask prices.

Finally, we check the spread for the two currencies as in Figure 11. We immediately see that that we could create mean reversion strategy for the spread. It may not be viable on profit versus fee basis however we could tests this.

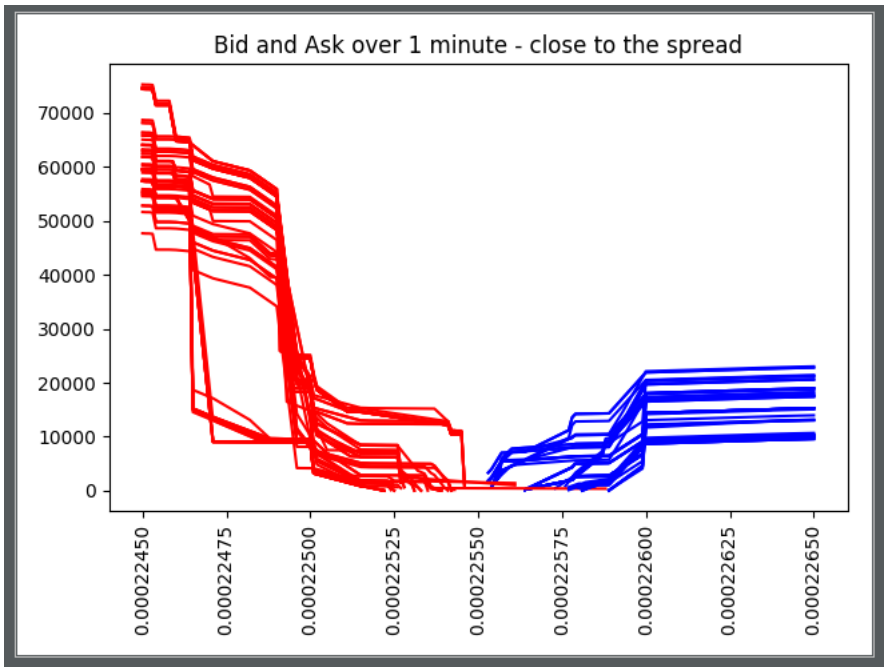

Figure 9: Detailed Price and quantity moments around the spread in 1 minute

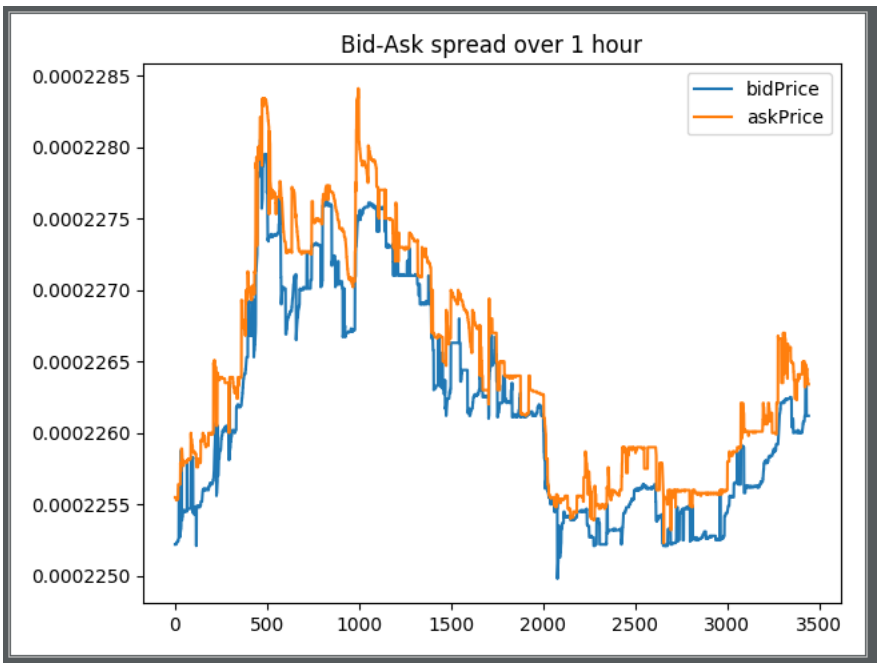

Figure 10: Max bid and min ask over 1 hour

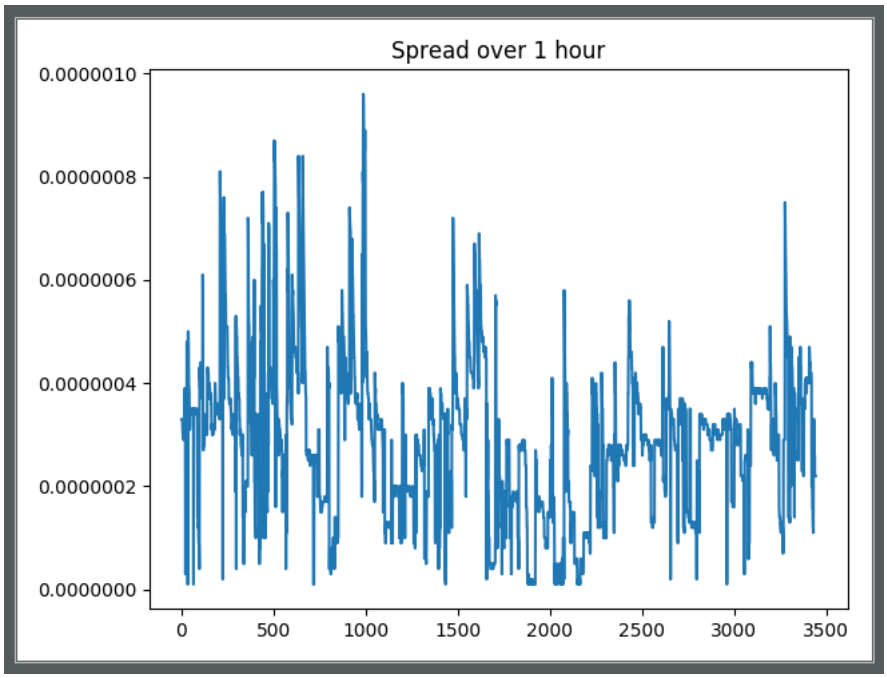

Figure 11: Max bid and min ask spread over 1 hour

Model

I tried to create my own supply and demand model as implied in the market order book. That is the model would suggest to us when there is a large deviation from market equilibrium. I thought about market middle price, P_{mid}, as

$$P_{mid} = \frac{Bid_{max} + Ask_{min}}{2}$$

where Bid_{max} is the maximum bid price for a given tick and Ask_{min} is the minimum ask price for the same tick.

The idea of my supply and demand estimator should oscillate around this middle price based on the current supply and demand. I attach a probability weights to each supply and demand amounts (see Figure 8 and 9 to get some feel for what the cumulative amounts look like). I assumed that the probability of arrivals of new orders follows log-normal probability where the orders arriving close to the maximum bid and minimum ask prices are most likely and as we go away from the this bid and ask spread, the arrival of new orders is less likely so they carry small probability weight.

More specifically, I used maximum likelihood estimator for mean, μ, and standard deviation, σ, for the log normal distribution.

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \ln(x_i), \quad \hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}[\ln(x_i) - \hat{\mu}]^2}$$

where x_i is the bid or ask price for the given state of the order book. I assumed that the bid prices and

ask prices are independent and they both come from a log normal distribution that is estimated using the current bid and ask prices separately.

We then use the relationship between log-normal and normal distribution to find the probabilities for each prices quoted in the order book. We can use the below to find the probability, P_x, of a given price level to occur.

$$P_x = F(x; \mu, \sigma) = \Phi\left(\frac{ln(x) - \mu}{\sigma}\right)$$

with Φ being the standard normal cumulative density function, see (*3*) for details. Considering the shape for the probability density, we know what it is suitable for the ask prices, however I simply reverse the order of the probability mass programmatically for the bid prices to achieve the notion that the prices close to bid are more likely to occur. I then find the expected bid and ask amounts, a, as

$$\mathbb{E}[a] = \sum_{i=1}^{n} P_{x_i} a_i$$

We then have a single number that is representing supply and demand that is probability weighted towards the current bid and asks. Further, I define total supply, S_T, and total demand, D_T as the sum of the amounts for bid and ask.

As suggested, I tried to define and oscillator around the middle price. I used the price adjustment, PA, as

$$PA_{ask} = \frac{\frac{S_T + P_T}{2} - (\mathbb{E}[a]_{Demand} - \mathbb{E}[a]_{Supply})}{S_T + P_T} \times \left(\frac{Ask_{min}}{Bid_{max}} - 1\right)$$

and

$$PA_{bid} = \frac{\frac{S_T + P_T}{2} - (\mathbb{E}[a]_{Supply} - \mathbb{E}[a]_{Demand})}{S_T + P_T} \times \left(\frac{Ask_{min}}{Bid_{max}} - 1\right)$$

When $\mathbb{E}[a]_{Demand}$ is greater than $\mathbb{E}[a]_{Supply}$ I compute

$$\text{adjusted price} = P_{mid}/(1 + PA_{bid})$$

and otherwise,

$$\text{adjusted price} = P_{mid} * (1 + PA_{ask})$$

The results of the computation are presented in Figure 12. We see that in each case my adjusted price is above or below the actual quoted bid or ask. This then suggests that we do not really have an oscillator, we rather indicate where there is a demand stronger and supply. My methods needs further work, however I decided to use the above calculation to design a simple trend following strategy as given in the next section.

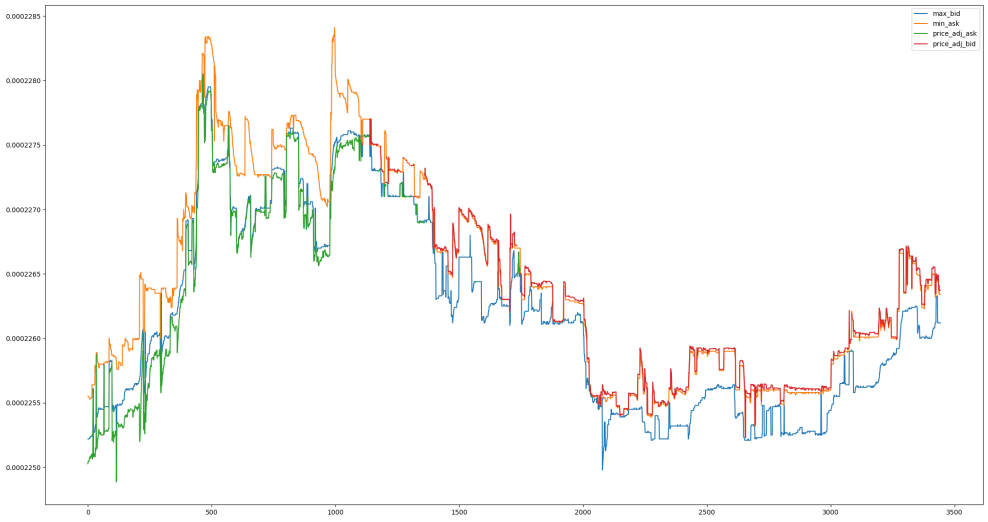

Figure 12: Supply and demand indicator

Strategy

As shown in the Figure 12 shows that we could use my model to simple follow the strong demand as an up-trend and strong supply as a down-trend. Figure 13 shows frequent switches between the strong supply and strong demand. Perhaps it indicates a small turning point in the market. Then, we do not wish to follow any trend. Figure 15 then show how we can combine the regime switches to a strategy. I used one last minute as an indication as to whether we should take a long a short position.

Figure 16 then show the final position. We take a long position, when the market in clearly in the up-trend or wise-versa. We take no position when supply are demand are balanced. The logic for that is

- Go long when demand is winning over last minute, that at least in last 60 observations, demand was the driver for price at least 55.
- Go short when in the last 60 observations, at least 57 observations were won by supply.
- We take no position otherwise.

The cumulative returns for this strategy are plotted in Figure 17. The loop for this strategy is shown below.

```
df['position'] = 0
for n in range(len(df)):
    if n > 60:
        view = df.loc[n-60: n, 'long_signal'].sum()
```

```
            if view > 55:
                df.loc[n, 'position'] = 1
            elif view < 3:
                df.loc[n, 'position'] = -1
            else:
                df.loc[n, 'position'] = 0
    # close the trade at the end of the time period
    df.loc[len(df)-1, 'position'] = 0
```

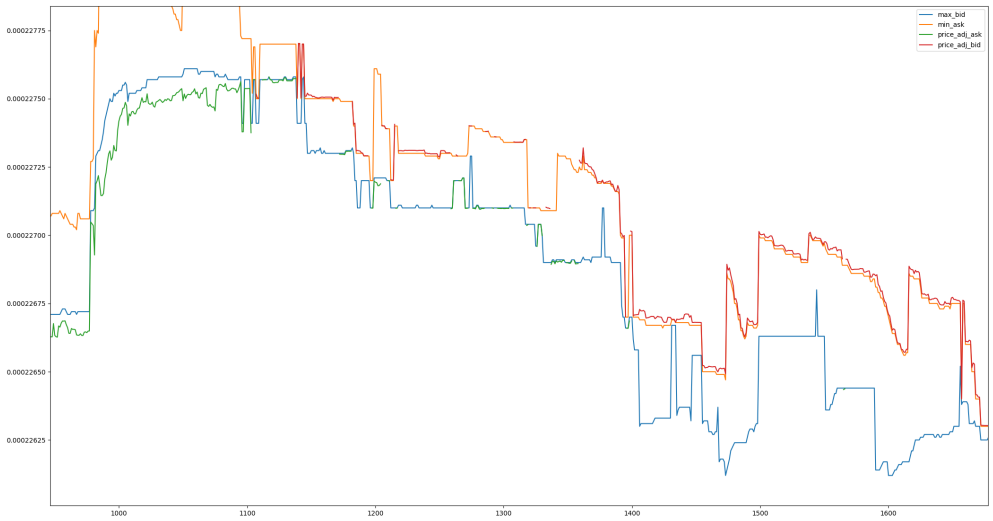

Figure 13: Possible turning point

Figures 14 and 15 show the mirrored regime switches. We see that they are identical so we can use just one of them.

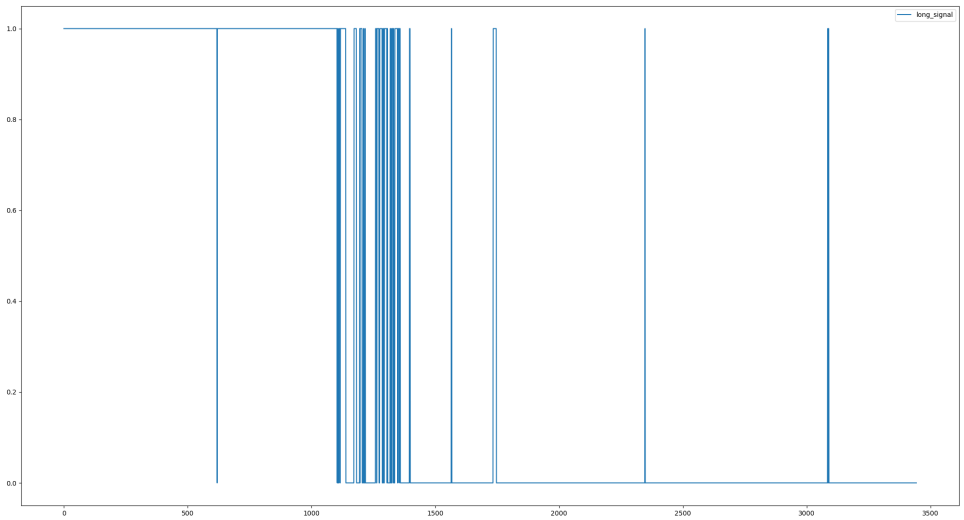

Figure 14: Short position indicator

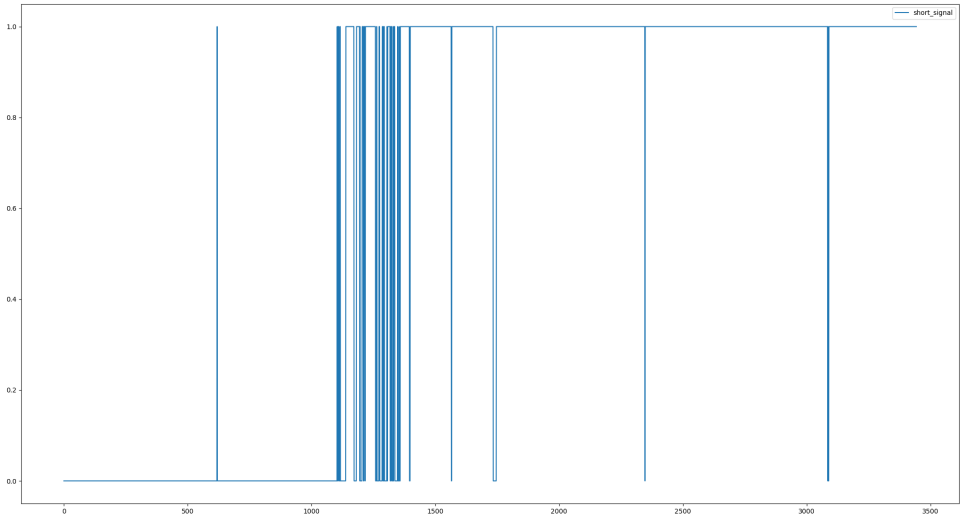

Figure 15: Long Position indicator

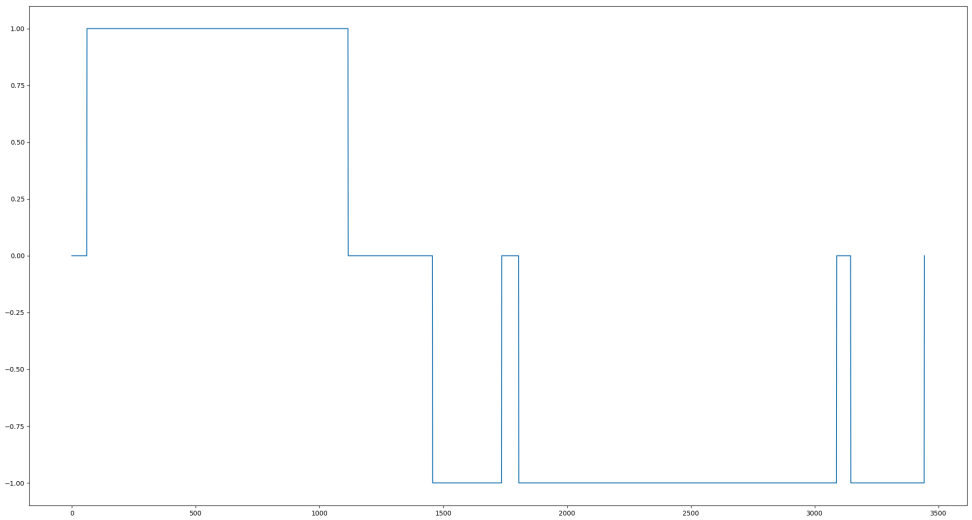

Figure 16: Position - Strategy

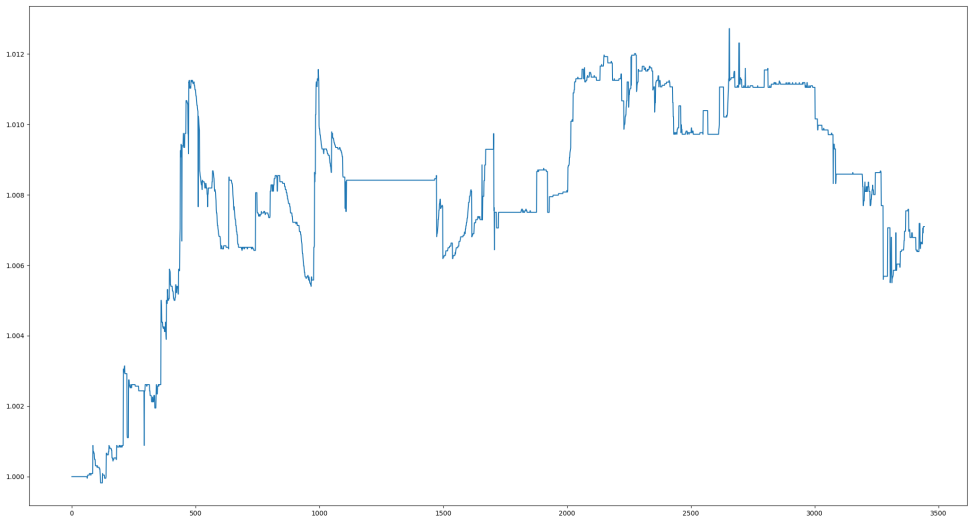

Figure 17: Position - Valuation

Strategy Summary

The KPIs are presented below. We have do a few trades within an hour of trading. KPIs are based on the middle price so it is just an approximation to the performance.

	KPIs
annualised_std	0.1124887732
avg_loss_return	-0.0022179186
avg_trade_return	0.0004751018
avg_win_return	0.0031681222
cagr	1.0285530229
lake_ratio	0.0025649288
max_consecutive_losers	3.0000000000
max_consecutive_winners	2.0000000000
max_drawdown	0.0071235493
mean_annualised_return	0.0836999763
number_of_trades	8.0000000000
sharp_ratio	0.7440740435
win_loss_ratio	-1.4284213327
win_rate	0.5000000000

Figure 18: KPI - sample

References and Notes

1. BBC *Slowing down the predatory high speed traders*. BBC, UK, 2017
2. Bodek, Haim *The Problem of HFT*. CreateSpace Independent Publishing Platform, US, 2013
3. Bury, Karl *Statistical Distributions in Engineering*. Cambridge University Press, UK, 1999
4. CBS News *AXA Rosenberg Paying $242M To Settle Case On Code*. CBS News, US, 2011
5. Durbin, Michael *All about high-frequency trading*. McGraw-Hill, New York, NY, 2010
6. Harris, Larry *Trading and Exchanges: Market Microstructure for Practitioners*. Oxford University Press, Oxford, 2003
7. IEX `https://iextrading.com/developer/docs/https://iextrading.com/developer/docs/`, last visited December 10, 2018
8. Investment Technology Group *Global Execution Strategies*. Dublin, Ireland, 2013
9. Johnson, Barry *Algorithmic trading & DMA: An introduction to direct access trading strategies*. 4Myeloma Press, London, UK, 2010
10. Kirilenko, Andrei *HFT Latency Regulation*. London, UK, 2016
11. Kirilenko, Andrei; Lamacie, Guilherme *Latency and Asset Prices*. London, UK, 2015
12. Kumar, Ganesh *High Frequency Trading systems and Latency measurements*. London, UK
13. Lockwood, John W., Madhu Monga *Implementing Ultra-Low-Latency Datacenter Services with Programmable Logic*. IEEE Computer Society, US, 2016
14. London Stock Exchange *A guide to London Stock Exchange trading services for equity securities*. London, UK, 2015
15. NYSE `https://www.nyse.com/market-data/real-time`, last visited December 10, 2018
16. NY Times *Knight Capital Suffers Power Failure*. NY Times, US, 2012
17. O'Hara, Maureen *High frequency market microstructure*. Juournal of Financial Economics, NY, 2014
18. Patterson, Scott *Dark Pools: The Rise of the Machine Traders and the Rigging of the U.S. Stock Market*. Crown Business, London, UK, 2013
19. Python-Binance `https://python-binance.readthedocs.io/en/latest/index.html`, last visited December 10, 2018
20. Reuters *HFT Debate With Haim Bodek and Manoj Narang*. Reuters, US, 2014
21. RSJ `https://www.rsj.com/en/securities/about-us`, last visited December 10, 2018
22. VPRO *Flash Crash 2010 - VPRO documentary*. VPRO, US, 2012