Part 3: Data Preprocessing, EDA, and Inference

April 13, 2017

1 Data Overview

The data consists of raw output collected by an MTConnect adapter connected to a GF AgieCharmilles or 'GF Agie' machine, manufactured by GF Machining Solutions. The GF Agie is a 5-axis machining center, meaning it is capable of moving in the X,Y,Z directions as well as rotating on the A and B axes (yaw and pitch), resulting in the capability to machine a complex and diverse array of parts.

The output from the machine covers many aspects of the part creation process and consists of over 100 data fields, some of which are referenced in a key data dictionary in the appendix. The raw data dump was originally organized into 57 text files, each representing a day's worth of machining activity. Over the 57 days of data we received, we found that 16 discrete parts were manufactured.

We first collated these 57 text documents into one table, resulting in a row count of about 3.1 million. The rationale behind this was because it is more meaningful for certain applications to analyze data by each manufactured part, rather than analyzing data by day. Thus we needed to join all the text files because some parts are machined over or across several days.

Each row in the joined table represents a single observation in time, and records such key features as X,Y,Z,A,B positions, various key happenings during that time period, and machine status. The next section demonstrates the process of collecting and aggregating the data for further pre-processing.

2 Data Loading

After receiving the data in txt files, we begin by aggregating and loading the data into R.

```
# Point R at the input files

file_list <- list.files("C:\\Users\\Viu52790\\Desktop\\GFAgie\\txts")
file_path_xml = "C:\\Users\\Viu52790\\Desktop\\GFAgie\\Devices.xml"
device_name = "GFAgie01"

# Read in the data and create the MTConnect Device in R

allPosition <- as.data.frame(matrix(0, nrow = 1, ncol = 7))

for(i in file_list){

name <- gsub("-",".",i)
name <- gsub(".csv","",name)
name <- gsub("GFAgie","Position_GF", name)

name2 <- gsub("-",".",i)
name2 <- gsub(".csv","",name)
name2 <- gsub("GFAgie","All_GF", name2)


file_path_dmtcd = i
temp <- assign(name, create_mtc_device_from_dmtcd(file_path_dmtcd,
    file_path_xml, device_name))
assign(name, merge(temp, "POSIT"))
assign(name2, merge(temp))

}
```

	timestamp	ApositionANGLEACTUAL	availAVAILABILITY	cnctempTEMPERATURENANANANACONDITION	commsCOMMUNICATIONSNANANANACONDITION
1	2016-05-07 05:00:00.000000	NA	UNAVAILABLE	UNAVAILABLE	UNAVAILABLE
2	2016-05-07 05:00:00.000000	NA	UNAVAILABLE	UNAVAILABLE	UNAVAILABLE
3	2016-05-07 05:00:00.000000	NA	UNAVAILABLE	UNAVAILABLE	UNAVAILABLE
4	2016-05-07 05:00:00.000000	NA	UNAVAILABLE	UNAVAILABLE	UNAVAILABLE
5	2016-06-15 05:00:00.000000	NA	UNAVAILABLE	UNAVAILABLE	UNAVAILABLE
6	2016-06-15 08:10:08.940684	NA	UNAVAILABLE	UNAVAILABLE	UNAVAILABLE
7	2016-06-15 08:10:08.940839	NA	UNAVAILABLE	UNAVAILABLE	UNAVAILABLE
8	2016-06-15 08:10:08.940886	NA	AVAILABLE	UNAVAILABLE	UNAVAILABLE
9	2016-06-15 08:10:08.941014	NA	AVAILABLE	UNAVAILABLE	UNAVAILABLE
10	2016-06-15 08:10:08.941030	NA	AVAILABLE	UNAVAILABLE	UNAVAILABLE
11	2016-06-15 08:10:08.946005	NA	AVAILABLE	UNAVAILABLE	Normal
12	2016-06-15 08:10:08.946052	NA	AVAILABLE	UNAVAILABLE	Normal
13	2016-06-15 08:10:08.946089	NA	AVAILABLE	UNAVAILABLE	Normal
14	2016-06-15 08:10:08.946131	NA	AVAILABLE	UNAVAILABLE	Normal
15	2016-06-15 08:10:08.946155	NA	AVAILABLE	Normal	Normal
16	2016-06-15 08:10:08.946183	NA	AVAILABLE	Normal	Normal
17	2016-06-15 08:10:08.946213	-118.6477	AVAILABLE	Normal	Normal
18	2016-06-15 08:10:09.084774	-118.6477	AVAILABLE	Normal	Normal
19	2016-06-15 08:10:09.372931	-118.6477	AVAILABLE	Normal	Normal
20	2016-06-15 08:10:09.522245	-118.6477	AVAILABLE	Normal	Normal
21	2016-06-15 08:10:09.948790	-118.6477	AVAILABLE	Normal	Normal
22	2016-06-15 08:10:10.092773	-118.6477	AVAILABLE	Normal	Normal

Looking at a snapshot of the initial table, it is clear we need to do some more investigation and maybe some data cleaning before we can begin any modeling. We constructed a data dictionary of available fields, which can be referenced in the appendix.

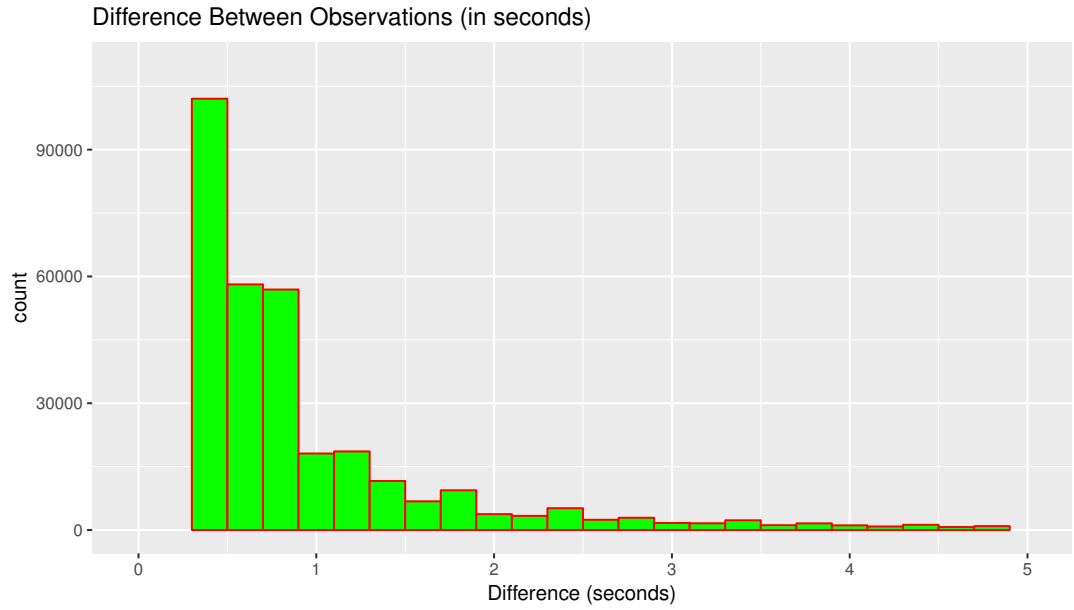
3 Exploratory Data Analysis and Data Verification

3.1 Time Series Exploration

We continue by exploring the various key features presented in the data and potential value it may have in modeling complexity. We decide to conduct a couple 'gut checks' to see if the data is actually usable, in addition to deriving some insights that can be used in modeling later. We determine that there is great value in looking at the distribution of timestamps for this, so we can get an idea of when data is recorded and if it is being recorded at a regular interval.

```
#distribution of timestamp differences - demonstrate rapid recording
    of times

timediffs <- as.data.frame(1:nrow(allPosition))

for(i in 1:(nrow(allPosition)-1)){
timediffs[i,] <- allPosition$timestamp[i+1] -
    allPosition$timestamp[i]
```

```
}

timediffs[nrow(timediffs),] <- 0

colnames(timediffs) <- 'Difference (s)'

ggplot(timediffs, aes(`Difference (s)`)) + geom_histogram(binwidth =
    .2, col = 'red', fill = 'green') +
ggtitle('Difference Between Observations (in seconds)') +
    xlim(-.01,5) + ylim(0, 110000)
```

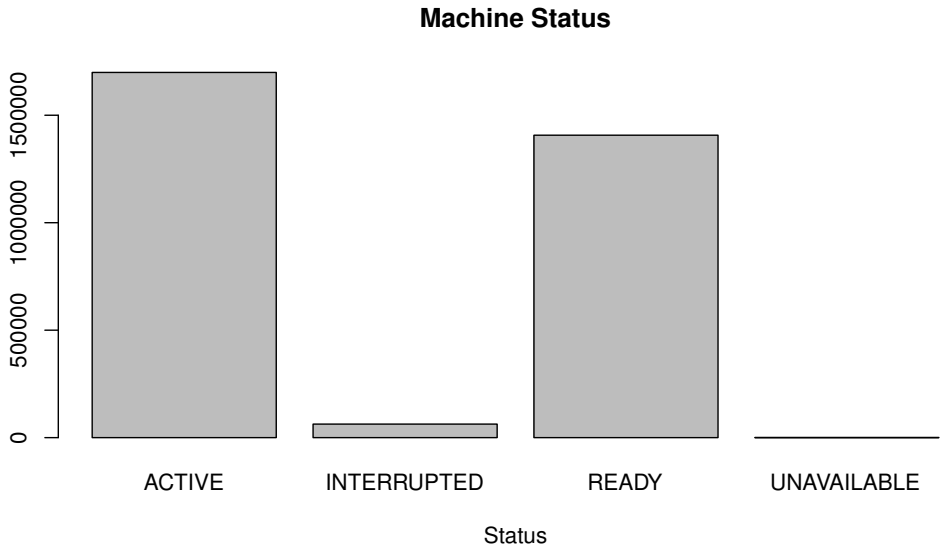


After removing outliers ($< 1\%$), we find that the lags between timestamps are largely under 1 second, with over 99% of observations having less than a 5 second lag from the immediately prior observation. From our domain knowledge of manufacturing, this is an acceptable tolerance and indicates the events are recorded on a semi-regular basis, with minimal inconsistency. It also indicates that there are no major gaps in data, with respect to time, in these 57 days of data. We conclude that the machine was continuously recording data for the entire 57 days and no serious omissions are present. Thus, the data is robust on the time level and no other processing needs to be done to regularize time periods.

3.2 Capacity Utilization

Next, we observe the status of the machine over these 57 days to get an idea of the capacity utilization of the machine. The purpose of this is to determine if there is a realistic representation of machine usage during the data's time frame and ascertain that the observations are not 'fishy' in the sense that they over or under-represent certain machine states. Because we found that the data was robust on the time level, this test's purpose is to further confirm the robustness of the data and give insight into how the machine was used.

```
#plot of machine status

counts <- table(allPosition[9])
barplot(counts, main="Machine Status",
xlab="Status")
```

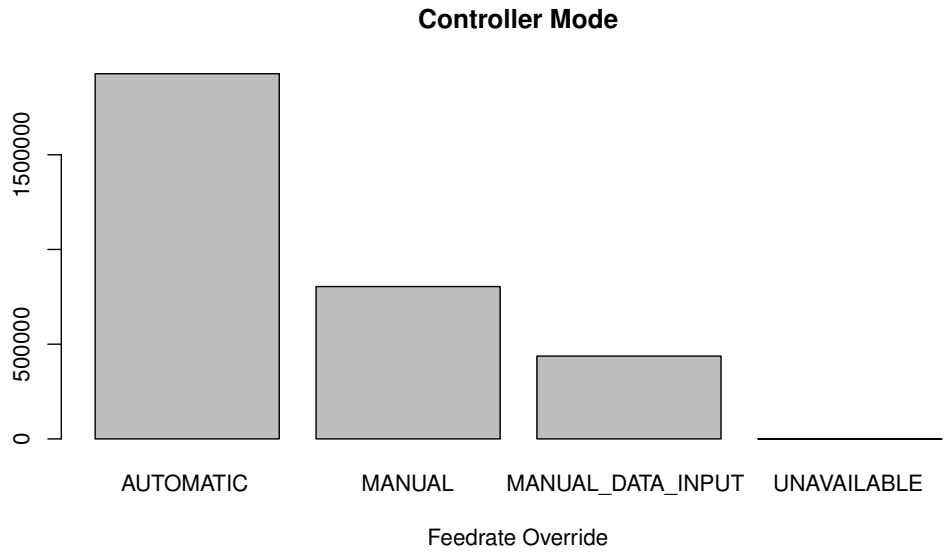


Based on our domain knowledge, we conclude that this is a realistic representation, over 57 days, of machine status. ACTIVE indicates the machine is being utilized, INTERRUPTED means there is an error of some sort, READY means the machine is not in error but not being utilized, and UNAVAILABLE means the machine is down for maintenance. With an approximate 50% utilization rate, the data represents realistic levels and patterns of machine usage.

3.3 Automatic or Manual?

We next investigate another key attribute, controller mode. Controller mode indicates if the machine is automatically machining based off a previous program or if it's being controlled by a skilled (or unskilled, if it unbeknownstly falls into the factory owner's child's hands) operator.

```
#plot controller mode

counts <- table(allPosition[14])
barplot(counts, main="Controller Mode",
xlab="Feedrate Override")
```

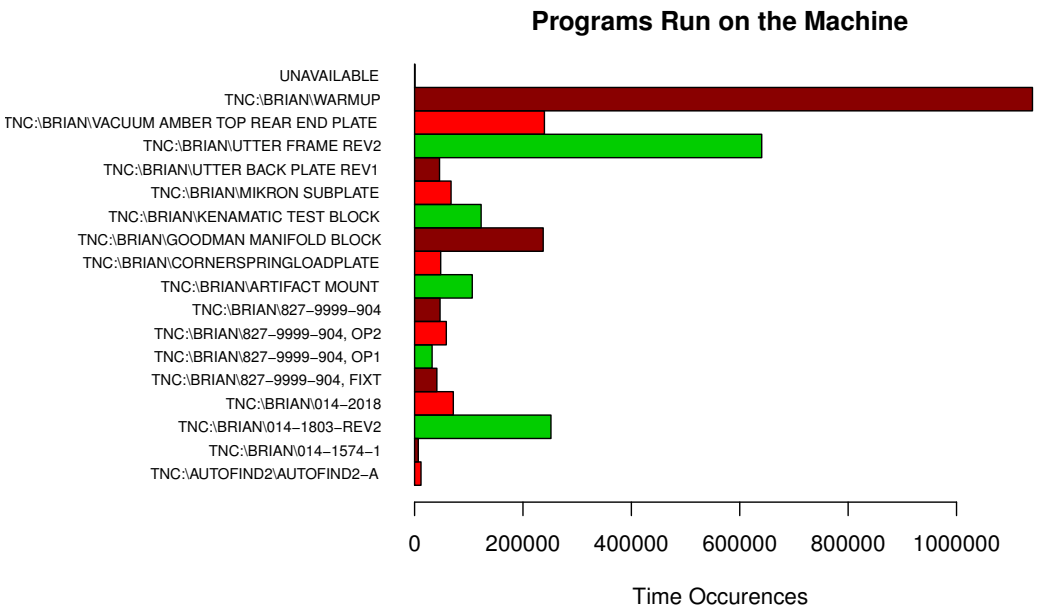


AUTOMATIC indicates the machine is running a pre-programmed automated routine, MANUAL indicates a manual takeover by a human, and MANUAL DATA INPUT means the program was altered by manual entered parameters. We see that the majority of the time, the machine is running an automated program. This distribution is not only realistic, it appears to be favorable to our particular goal of determining complexity. This is because it allows for a good mix between automated programs, which are often more complex, and manual input, which is often or necessarily less complex as it requires a high degree of attention and dexterity, both of which are constrained by the limits of human performance. This gives us a diverse set of data to analyze.

3.4 Machine Procedure

The most important plot generated during EDA is a comparison of which parts are being machined. This allows us to get an idea of part complexity based purely off the amount of time (approximated by observations) it took to machine the part, and allows us to see clearly the various routines that are run.

```
#plot type of machine procedure

counts <- table(allPosition[21])

colnamesbarplot <- names(counts)
colnamesbarplot <- gsub(".H","",colnamesbarplot)

par(las=1, mar=c(4,15,2, 2))

barplot(counts,space=c(0,5),legend.text=FALSE,beside=TRUE,horiz=TRUE,
density=NA,
col=c("red1","red4","green3"),
xlab="Time Occurences",
axes=TRUE, names.arg=colnamesbarplot, cex.names=.7, las=1, main =
    'Programs Run on the Machine')
```

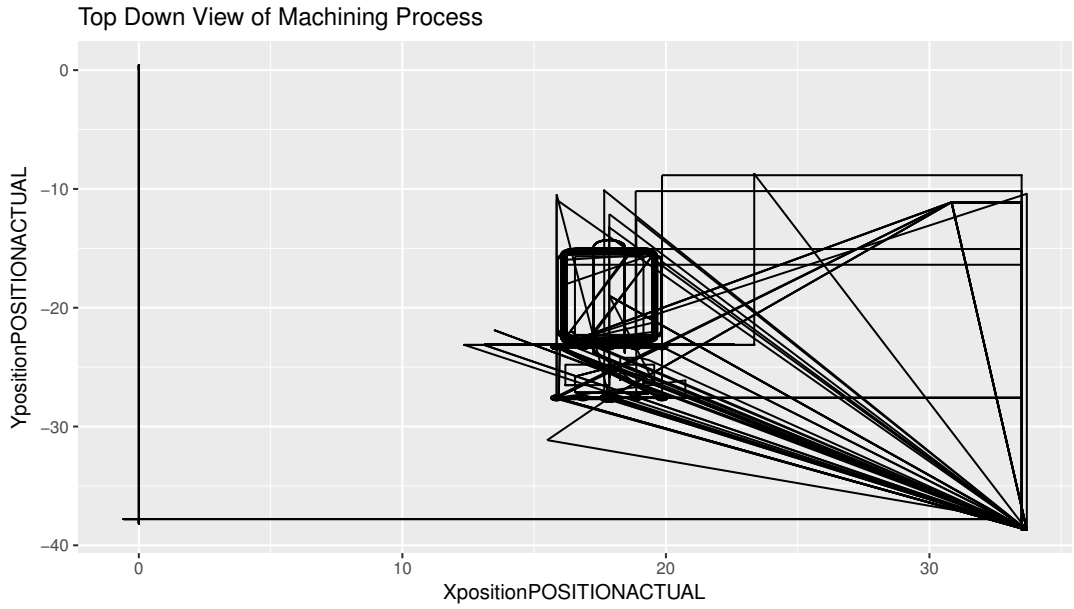


Based off our domain knowledge of manufacturing, the time it takes to machine a part is a decent though very broad approximation of how complex the part is. We plan on supplementing this with additional information of course, including but not limited to X,Y,Z,A,B position and path feedrates.

3.5 Part Visualizations

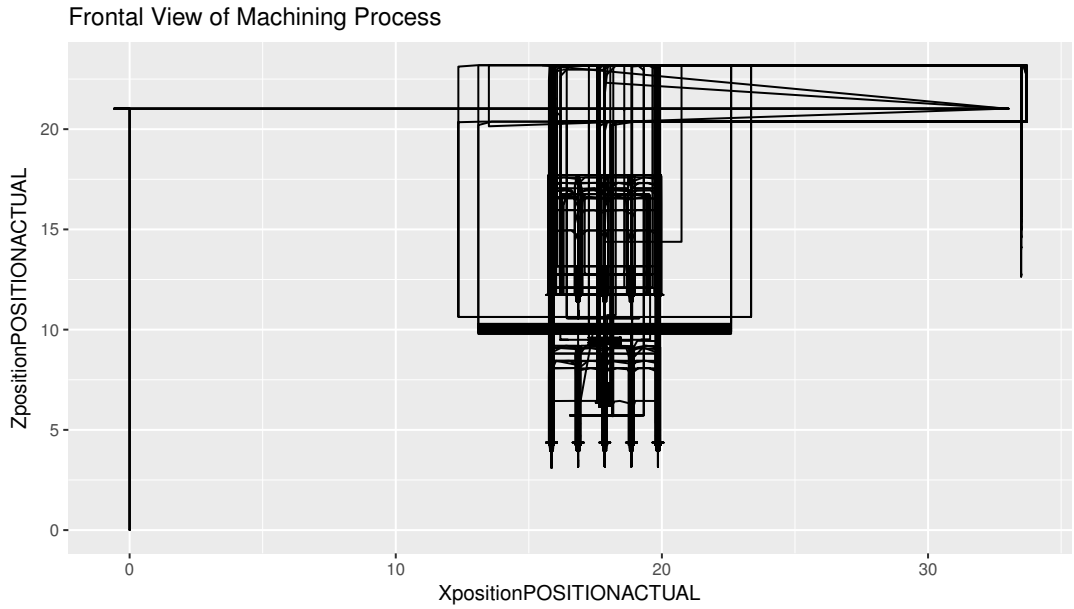
Our next step is to visualize a couple of these parts to get an idea of what they look like, and consequently how intricate or 'complex' they are. This will serve as guidance to come up with our final complexity quotient. We first choose to visualize and discuss the Goodman Manifold Block.

```
# Plotting Y position vs X position to get an idea of GOODMAN
    MANIFOLD BLOCK

ManifoldBlock <- allPosition[allPosition[21] ==
    'TNC:\\BRIAN\\GOODMAN MANIFOLD BLOCK.H',]

pos_data_mtc = select_(ManifoldBlock, colnames(allPosition[25]),
    colnames(allPosition[26]), colnames(allPosition[27]))

ggplot(pos_data_mtc) + geom_path(aes(x = XpositionPOSITIONACTUAL, y
    = YpositionPOSITIONACTUAL)) + ggtitle('Top Down View of
    Machining Process')
```




```
# Plotting X position vs Z position to get an idea of GOODMAN
    MANIFOLD BLOCK

ggplot(pos_data_mtc) + geom_path(aes(x = XpositionPOSITIONACTUAL, y
    = ZpositionPOSITIONACTUAL)) + ggtitle('Frontal View of Machining
```

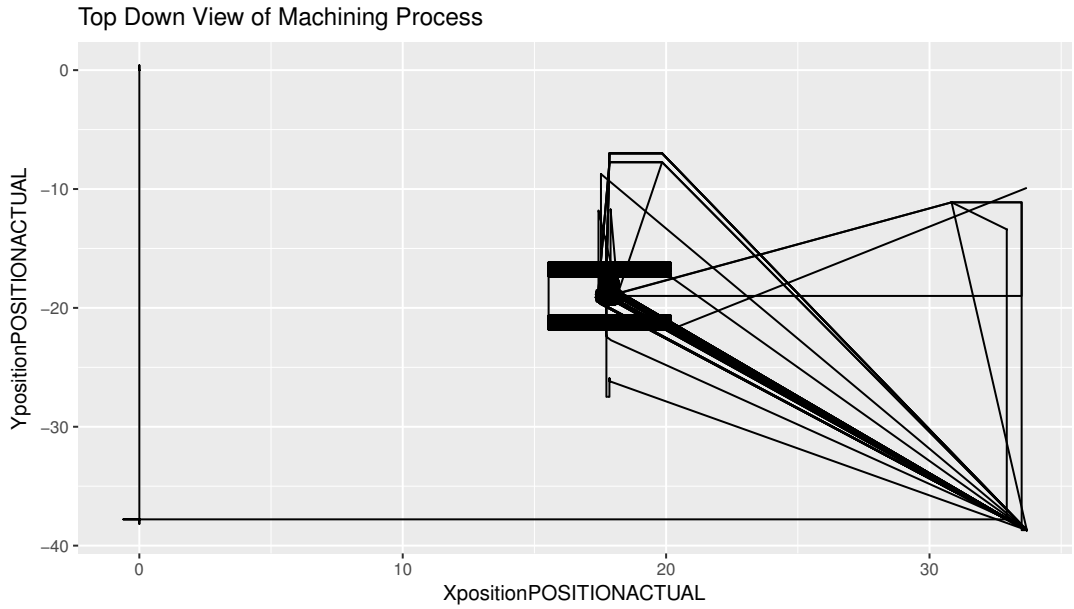
```
Process')
```



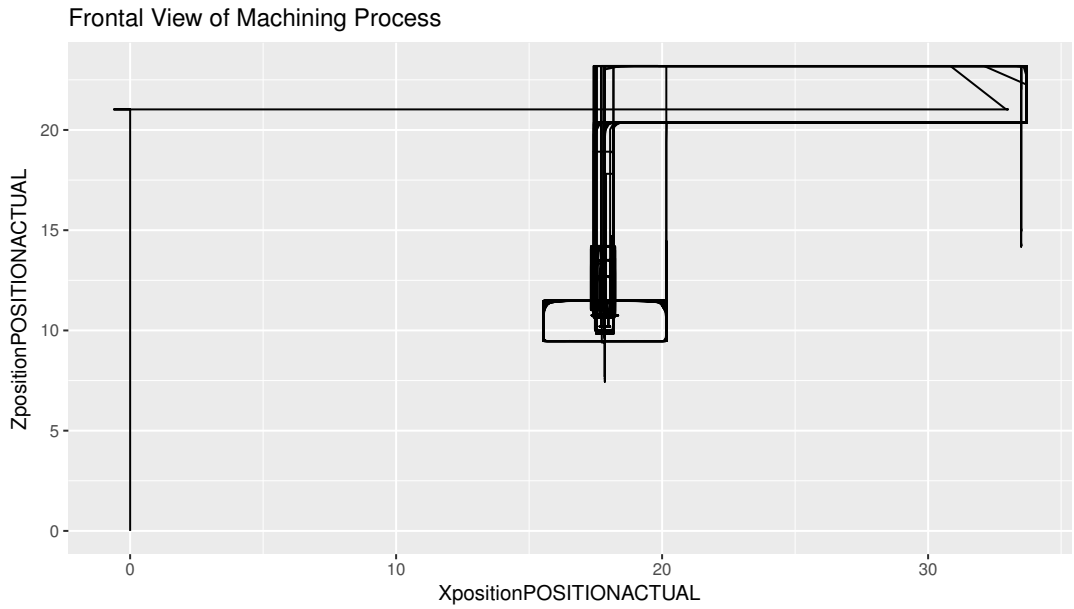
We compare this to an actual manifold block, and can see the machining patterns leading to the final product, verifying the robustness of the data and this method of visualization.

We do another 'gut check' comparison of actual object vs. machining position visualization just to make sure the data is robust, this time taking the corner springload plate as an example.

```
# Plotting Y position vs X position to get an idea of CORNER
    SPRINGLOAD PLATE

SpringPlate <- allPosition[allPosition[21] ==
    'TNC:\\BRIAN\\CORNERSPRINGLOADPLATE.H',]

pos_data_mtc = select_(SpringPlate , colnames(allPosition[25]),
    colnames(allPosition[26]), colnames(allPosition[27]))

ggplot(pos_data_mtc) + geom_path(aes(x = XpositionPOSITIONACTUAL, y
    = YpositionPOSITIONACTUAL)) + ggtitle('Top Down View of
    Machining Process')
```




```
# Plotting X position vs Z position to get an idea of CORNER
    SPRINGLOAD PLATE

ggplot(pos_data_mtc) + geom_path(aes(x = XpositionPOSITIONACTUAL, y
    = ZpositionPOSITIONACTUAL)) + ggtitle('Frontal View of Machining
    Process')
```



We compare this to an actual corner springload plate (only the middle, tubular portion of the part shown below), and can see the machining patterns leading to the final product, further verifying the robustness of the data and this method of visualization.

4 Data Inference

4.1 Baseline Complexity Computation

After verifying the robustness of the position data (A and B axes do not need to be verified visually; there is no meaning to plotting yaw/pitch), we next come up with an initial baseline to determine complexity - a simple sum of the percent of unique values in each machining sequence on each

axis, multiplied by the inverse of the ratio of the most frequently occurring observation to the second most frequently occurring observation (per axis). We then multiply that sum by the log of the number of time observations for that part as a whole. An equation and example table is provided below demonstrating the baseline complexity product:

$$C_{baseline} = \sum_{1}^{n} (\frac{percentUnique}{freqRatio}) * log(nObs)$$

Where

- $C_{baseline}$ = baseline complexity
- n = the number of axes in the dataset
- percentUnique = percent of unique values in each axis vector
- freqRatio = ratio of the count of the most occurring value to the count of the second most occurring value in each axis vector
- nObs = total number of observations in the entire dataset

```
#determine complexity

zerovarA <-
    nearZeroVar(ManifoldBlock$ApositionANGLEACTUAL,saveMetrics =
    TRUE,names = TRUE,allowParallel = TRUE)
zerovarC <-
    nearZeroVar(ManifoldBlock$CpositionANGLEACTUAL,saveMetrics =
    TRUE,names = TRUE,allowParallel = TRUE)
zerovarX <-
    nearZeroVar(ManifoldBlock$XpositionPOSITIONACTUAL,saveMetrics =
    TRUE,names = TRUE,allowParallel = TRUE)
zerovarY <-
    nearZeroVar(ManifoldBlock$YpositionPOSITIONACTUAL,saveMetrics =
    TRUE,names = TRUE,allowParallel = TRUE)
zerovarZ <-
    nearZeroVar(ManifoldBlock$ZpositionPOSITIONACTUAL,saveMetrics =
    TRUE,names = TRUE,allowParallel = TRUE)
```

```
zerovarManifoldBlock <- bind_rows(zerovarA, zerovarC, zerovarX,
    zerovarY, zerovarZ)

zerovarManifoldBlock$ID <- c('A','C','X','Y','Z')

zerovarManifoldBlock$complexity <- 1/zerovarManifoldBlock$freqRatio
    * zerovarManifoldBlock$percentUnique
```

	freqRatio	percentUnique	zeroVar	nzv	ID	complexity
1	1.655806621	0.5249498643	FALSE	FALSE	A	0.3170357321
2	1.539469367	0.2561553110	FALSE	FALSE	C	0.1663919507
3	2.061508787	15.6195756585	FALSE	FALSE	X	7.5767688972
4	1.130119658	14.3004600684	FALSE	FALSE	Y	12.6539344446
5	1.021576433	18.8788149446	FALSE	FALSE	Z	18.4800807189

By axis, freqRatio signifies the ratio of the count of the most common observation to the second most frequent one. percentUnique quantifies the percent of unique observations on each axis. Zerovar and nzv indicate if the vector displays zero variance (all values are in the vector are the same, indicating inactivity or something suspicious).

4.2 Baseline Complexity Logic

Our logic behind this calculation was as follows - the more unique values a part has on each axis, the more detail or 'intricacy' the part has. We need to juxtapose that with the frequency ratio: if the most frequently occurring position observation is many times more than the next occurring observation, then it deflates the impact of unique values. Thus, we must account for this and multiply the inverse of this value by the unique value proportion. Next, we take the log of the number of observations and not the raw value because time contributes logarithmically to complexity, not linearly, as economies of scale can be realized with larger parts. The final product of all three represents a baseline complexity, which is shown below.

```
ManifoldBlockComplexity <- sum(zerovarManifoldBlock$complexity) *
    log(nrow(ManifoldBlock))
```

Result: 485.119

We compare this to a less complex part, the mikron subplate, to see if our methodology checks out on a macro-level and passes the 'sniff test'.

A Mikron Subplate

```
MikronSubplateComplexity <- sum(zerovarMikronSubplate$complexity) *
    log(nrow(MikronSubplate))
```

Result: 115.457

We also constructed a full table of complexities using all the parts in the dataset for further analysis.

5