

# New Insights into Process Deviations Using Multivariate Control Charts

## Abstract

In this paper we capture multivariate batch data in the form of letters of the alphabet, using a LEGO® Mindstorms® kit. With a known training letter, unknown letters can be identified based on multivariate properties. An application has been built in JSL to reformat the output of the multivariate control chart platform into simple graphs with descriptive text of the principal components and interactive filters. The key steps in preparing, analysing and visualising the data will be demonstrated.

## Introduction

The manufacture of chemical active ingredients is a multivariate batch process. It can take experienced scientists years to understand how the various inputs to the process interact.

Persistent problems are often multivariate in nature (such as an interaction between temperature, pressure and an impurity), which can make them difficult to solve. While the problem remains, significant losses in productivity can occur.

By utilising domain experts in conjunction with the multivariate control charts in JMP®, it is often possible to troubleshoot the process deviation. Unfortunately, the output is encoded in eigenvalues and eigenvectors, which can be non-trivial to understand.

## Background

Batch processing follows predictable patterns and it is the deviations from these patterns that need to be highlighted for investigation. Whilst time based data is readily available thanks to the low cost of sensors and data storage systems, it is often ignored in favour of point (summary) data which has been historically more accessible.

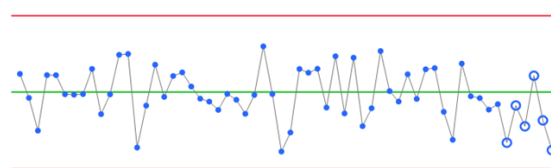
Some of the reasons for this are the time required to train scientists to deal with manipulation of comparatively large data sets and the relatively recent development of accessible tools to do so.

Process problems involving a single variable are usually spotted quickly and the issue rectified. Interactions between multiple variables can prove more challenging to identify and solve.

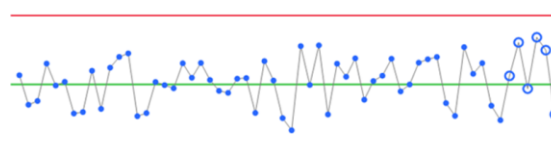
Control charts can be effective in monitoring key process parameters and identifying unusual patterns of behaviour. They are pragmatic tools which try to balance timely alerts to potential issues against the frequency of false out-of-control signals.

Even if the number of control charts to monitor a process can be kept to a manageable number there is a real risk of missing process changes. Consider the following example:

The levels of two components in an active chemical ingredient have an influence on how well it can be formulated into a final product. As a result the amounts of each are monitored by a control chart. A change was made to the process to improve the throughput and batches after the change are shown as hollow circles.



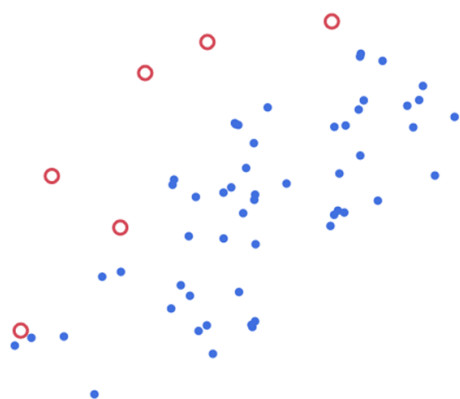
*Control Chart of X*



*Control Chart of Y*

Based on the individual control charts there is no strong indication that this process change has had any impact on the levels of these key components. However a plot of X versus Y (recent batches shown

as hollow circles) suggests that the levels of these two components relative to each other have changed.



*Plot of X versus Y*

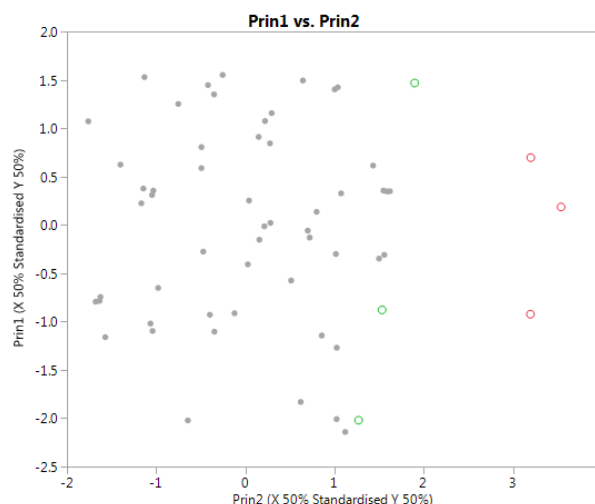
It may be that sometimes the new mixture formulates less well into final product. Based on the individual control charts there would be no reason to link this issue to the process change.

One way to address this problem would be to plot all two way plots as well as each control chart to guard against this behaviour. This rapidly becomes too many charts to reliably monitor and still doesn't cover three-way and more complex interactions.

## Multivariate Analysis

Well established techniques exist to deal with such problems such as Principle Component Analysis (PCA).<sup>1</sup> However it can be difficult for inexperienced practitioners to identify the key information amongst the separate elements of the output. Inspection of the scatter plot matrices of principle components can help to identify batches which have different multidimensional properties. Use of colour and/or icons is required to display the time element and it is subjective how distinct a point needs to be before it is declared to be different.

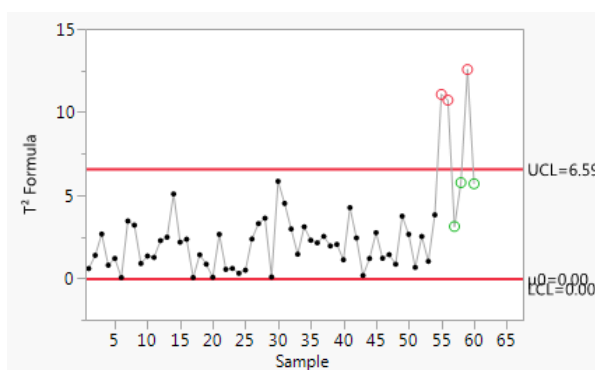
<sup>1</sup> For an introductory explanation to PCA see: <https://georgemdallas.wordpress.com/2013/10/30/principal-component-analysis-4-dummies-eigenvectors-eigenvalues-and-dimension-reduction/>



*Plot of X and Y in principle component space*

This subjectivity can be addressed using Hotelling's  $T^2$  statistic. The location in n-dimensional space (n = number of variables to describe a batch) of each batch can be captured as a single number. Using a training set of data the boundaries of the model which describes normal behaviour can be defined mathematically.

When the overall  $T^2$  statistic for each batch is plotted in time order along with the boundary value, the result is a multivariate control chart.



*Multivariate control chart of X and Y*

It is a simple and reproducible way of identifying batches which are different from the target. The challenge is now to identify why. This is where JMP® can help by allowing the data to be rapidly viewed in many different ways.

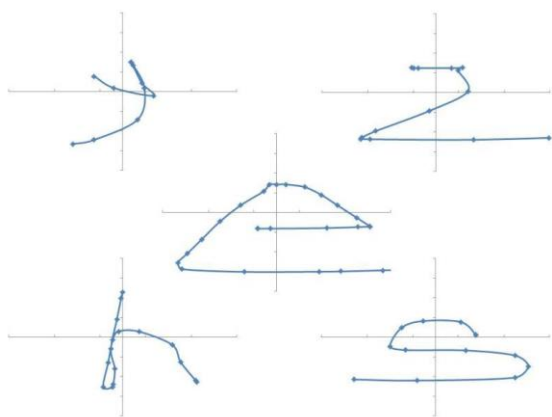
Whilst it is possible to visualise the problem when there are only 2 or 3 dimensions, it becomes difficult to grasp with 5 or more variables. A certain degree of trust is required by the user that the maths does

indeed hold at the higher dimensions. This lack of direct visualisation can be a second barrier to use of multivariate analysis.

## A Practical Example

In order to make the problem relatable to a wider audience the example of analysis of letter forms will be used. Each letter is analogous to a batch and can be described using summary statistics. The question to be answered: "Is the new unknown letter like my known training letter or not?"

The X and Y position data of a stylus was captured whilst individual letters were written using a LEGO® Mindstorms® robotics kit (see appendix 2 and 3 for details).

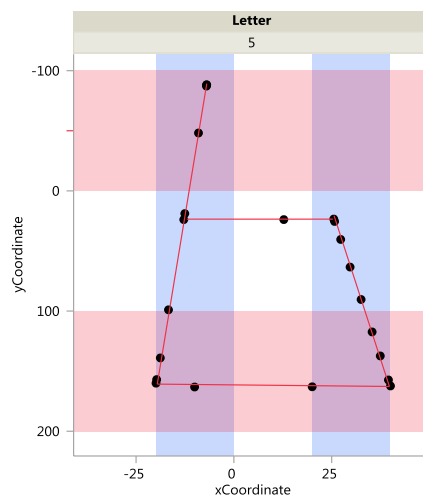


Example captured letter forms

The minimum number of batches required as a training set is equal to the number of variables used to describe a batch (this is a limitation of the mathematics). The ideal training set has ten times the number of variables used to describe a batch (in order to capture sufficient variability to get a good description of the model space).

Since control charts are a pragmatic indicator rather than a precise science somewhere between these two extremes is sufficient.

In optical character recognition (OCR) the major challenge is identifying where each letter form starts and ends in the presence of noise. The rules used to identify individual letters once they have been isolated are relatively simple and the principles have been borrowed for this example.



Parameterisation of the letter form

The letters can be summarised using some of the following parameters:

1. width
2. height
3. path length (total ink used)
4. number of times the letter intersects the horizontal top third line
5. number of times the letter intersects the horizontal bottom third line
6. number of times the letter intersects the vertical left third line
7. number of times the letter intersects the vertical right third line
8. time taken to draw the letter
9. segment containing the start point
10. segment containing the end point

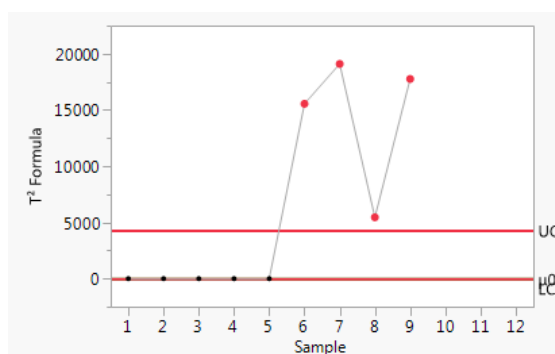
The resultant data is equivalent to batch summary properties such as processing behaviour and quality data.

Letter	Height	Width	Path Length	Lower Third	Upper Third	Group
1	72	17	126	2	0	Training
2	70	17	129	2	2	Training
3	67	15	120	2	2	Training
4	74	14	126	2	2	Training
5	78	14	134	2	2	Training
9	49	6	142	5	2	Test
10	54	8	164	4	2	Test
12	49	12	135	4	4	Test
13	61	8	171	4	2	Test

## Analysis

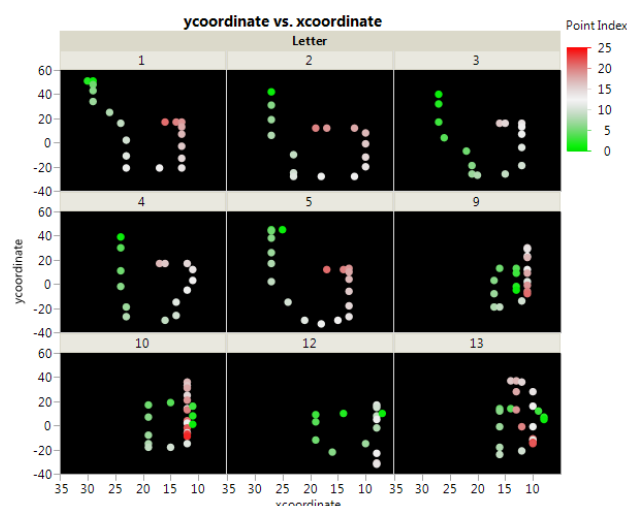
The analysis steps carried out are detailed in Appendix 1 and outlined below:

1. Define the training batches. They should exhibit typical variability so the multivariate model is neither too strict nor loose.
2. Standardise the data (subtract the mean of the training data and divide by the standard deviation of the training data).  
This step is vital if you want to separate the magnitude of each variable from its importance in influencing the multivariate control chart.
3. Set up the multivariate control chart using only the training data.
4. Save the  $T^2$  formula and principle components. This is done from the hotspot menus on the multivariate control chart and principle component platforms respectively.
5. Evaluate the  $T^2$  boundary formula (given in the JMP® help files) to get the model boundary. Save this value as a control limit to the  $T^2$  formula column.
6. Set up the control charts for each principle component using only the training data. This is to ensure that the control limits are set appropriately.



7. Plot the results for the training and test data as a multivariate control chart.
8. Do any of the batches have a  $T^2$  score higher than the boundary value?
9. Check the control charts of the principle components (or use the partition  $T^2$  score option in the multivariate control chart platform) to find out if the variation is in something that normally varies or not.
10. Use other JMP® platforms (parallel plots, cluster, 2-way cluster, multivariate matrix with change point detection) to identify the variables/mechanism for investigation.

An example set of 5 training letters (b) and two pairs of test letters (d & a).



## Conclusion

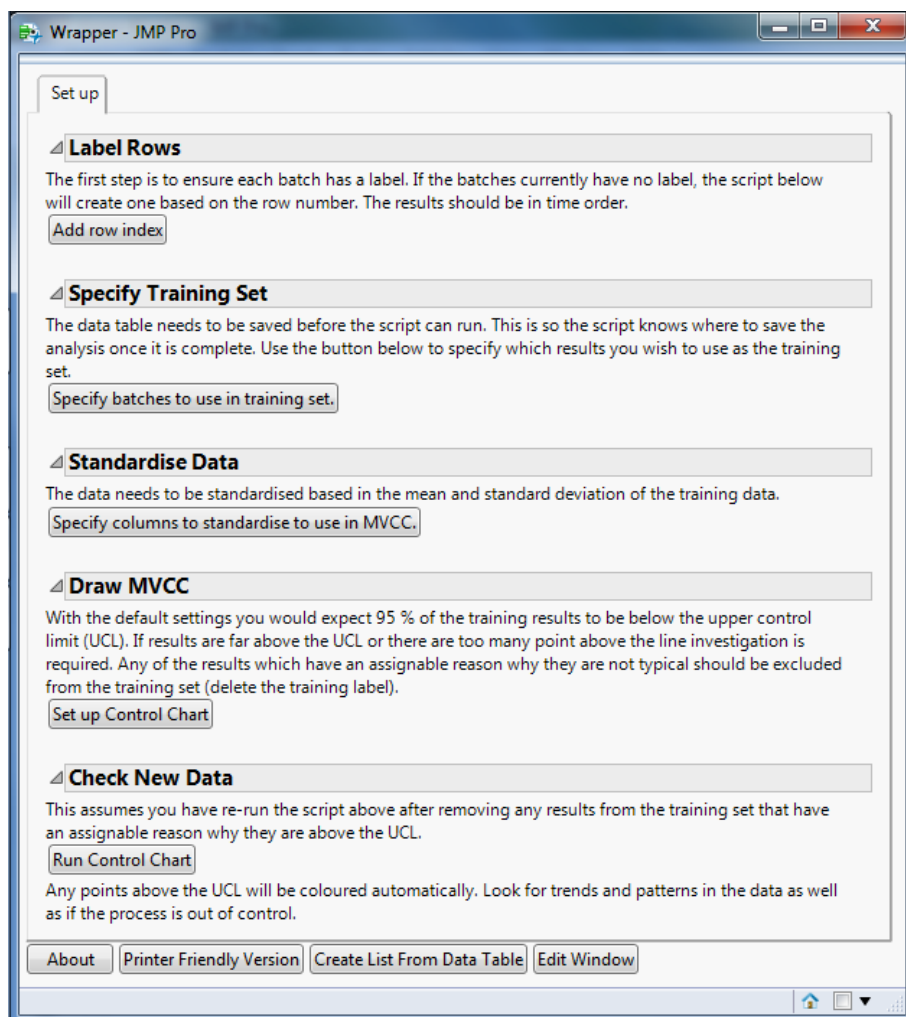
The multivariate control chart provides a consistent way to compare batches to an ideal group. Based on the relationships between variables in the ideal group a score can be assigned which highlights if the new batch is consistent with the ideal. Other platforms in JMP® can then be used to investigate how they are different.

## Stephen Pearson, PhD, Chemical Process Statistician, Syngenta

Stephen Pearson provides statistical and data science support to scientists at both the Grangemouth and Huddersfield manufacturing sites. He is experienced in building systems to automate the collection, processing, analysis and reporting of experimental and manufacturing data. Pearson has a PhD in chemistry from The University of Edinburgh.

## Appendix 1: Screenshots of the JMP® script

A script was written to automate the steps of the analysis. Whilst it is possible to obtain some user input and then carryout the rest of the analysis automatically, the script was split into sections to retain user engagement.

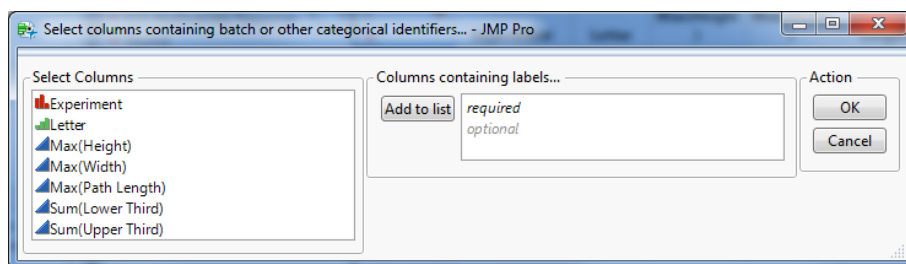


### Label Rows

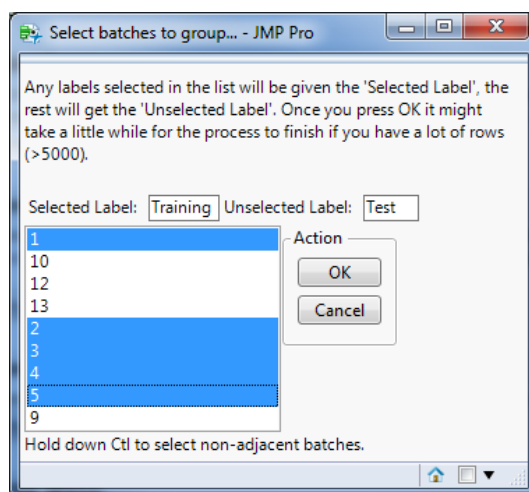
The data points require a label so they can be split into training and test groups. If one is not already present then a script is provided to add a row number column (as character data).

### Specify Training Set

The user is asked to specify the column (character data) containing the labels. The label column could be unique identifiers for each row or an existing set of group labels.



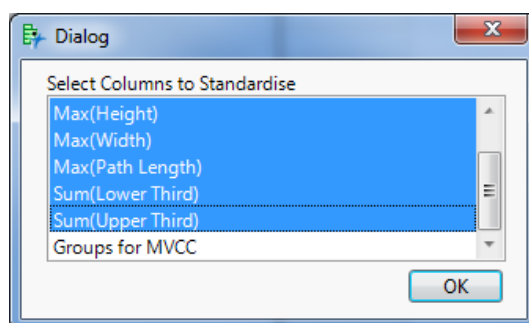
The user is then presented with a list of distinct labels from that column from which to select the batches to be labelled as Training and Test.



## Standardise Data

When carrying out PCA analysis the default option is to standardise the data so that large numbers and small numbers are given equal weighting in the analysis. This is done by subtracting the mean of the column and dividing by the column standard deviation. The default option for the multivariate control chart is to use the unstandardized numbers. This is undesirable in many cases as it is the changes in small impurities that are often of the most interest.

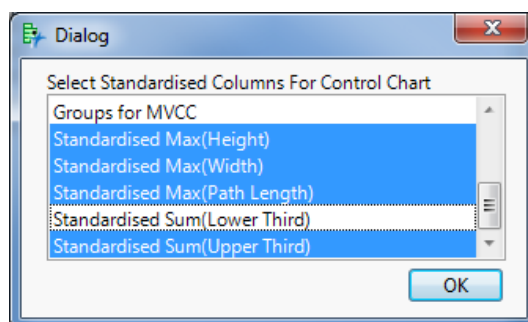
The user selects the columns in the analysis they wish to standardise.



A new column is created for each variable in which the mean of the *training* group is subtracted from each value before dividing by the standard deviation of the *training* group (if the standard deviation is zero then that parameter cannot be used in setting up the control chart).

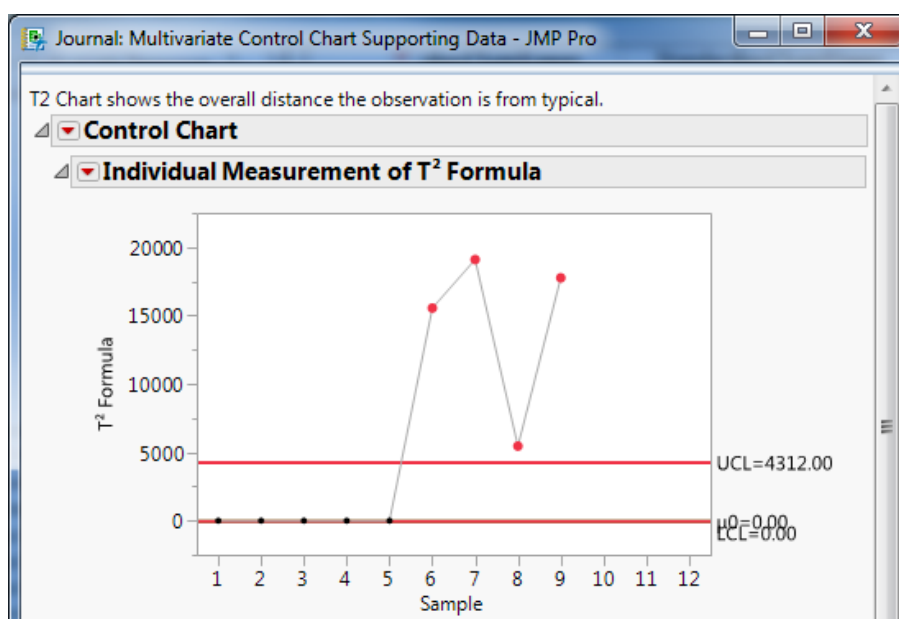
## Draw MVCC

The user is asked to specify which standardised columns they wish to base the multivariate control chart on. The number of selected columns must be less than or equal to the number of rows in the training group.



The user is then presented with a wait message whilst a series of data tables and platforms in JMP® are opened and closed. The windows are shown to the user to reassure them that something is happening. The multivariate control chart platform, PCA and run chart platforms are run using only data from the training batches.

The  $T^2$  formula is saved to a column and the upper limit is saved as a column property. Each of the principle components are also saved with control limits. The PCA platform could be used for the entire analysis once you understand how the  $T^2$  formula is created. Partitioning of the  $T^2$  score into major and minor parts is currently only possible from the multivariate control chart platform without writing new code.



Training batches are coloured black, new batches that are in control are green and out of control batches are coloured red. As far as making a decision about if there is a potential issue that needs investigation this graph is simple to interpret.

At this point a user experienced with PCA/parallel plots would need to sit down with the domain expert to discuss potential lines of investigation. This has the additional benefit of getting someone unfamiliar with the process involved in the investigation (they tend to ask obvious questions the domain expert wouldn't consider).



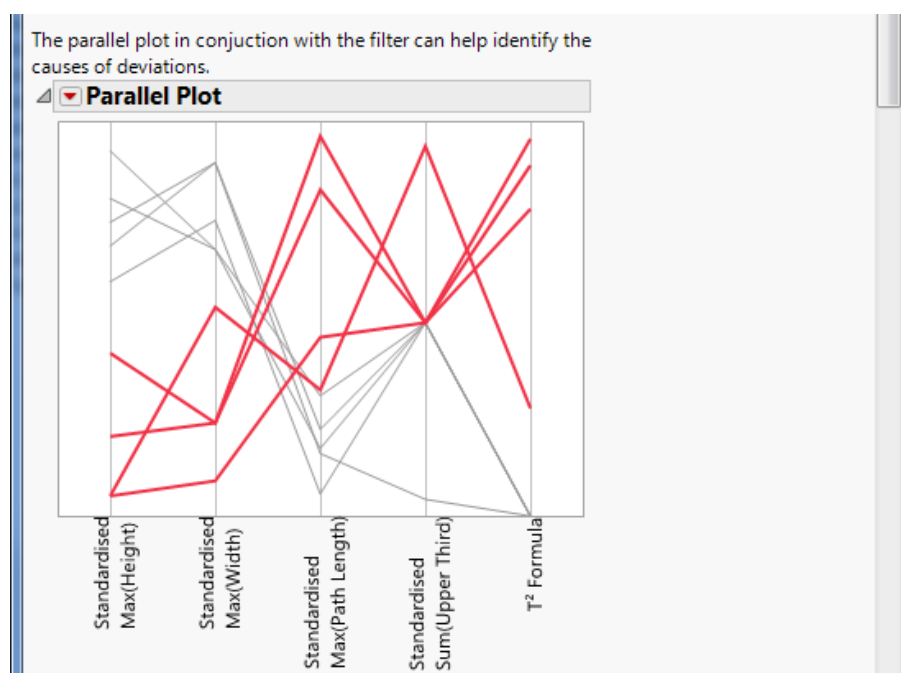
## Check New Data

The multivariate control chart platform saves the settings from the training data as a separate table which can then be loaded when new data is to be analysed. In reality it is more likely a user would append new data to an existing table or already have a table of historic data to investigate.

This script uses formula and limits saved to the data table and creates a new journal including the new data. It can be helpful to run change point detection on the  $T^2$  statistic. It may be that whilst all the batches are in control a shift in the process has occurred.

## Getting to the Cause of the Deviation

The parallel plot is an alternative method to PCA to display multivariate data. For inexperienced users it tends to be easier to interpret than PCA scores and loadings plots. It is displayed in the journal after the MVCC and the out of control batches are highlighted.



It can be beneficial to carry out cluster analysis next. The number of clusters should be increased until the training group are in a single cluster by themselves. Selecting the colour cluster option from the hot spot menu will highlight the groups on the MVCC and parallel plot. This will help indicate how similar the out of control batches are to each other.

Next in the journal are control charts of each principle component. They are labelled to indicate how much of the normal variation is explained by that component and the main variables which contribute to it. It is often here the domain user will spot a group of variables which they attribute to a particular part of the process such as impurities derived from a particular starting material. The principle components can be relabelled by altering the units in the column properties.

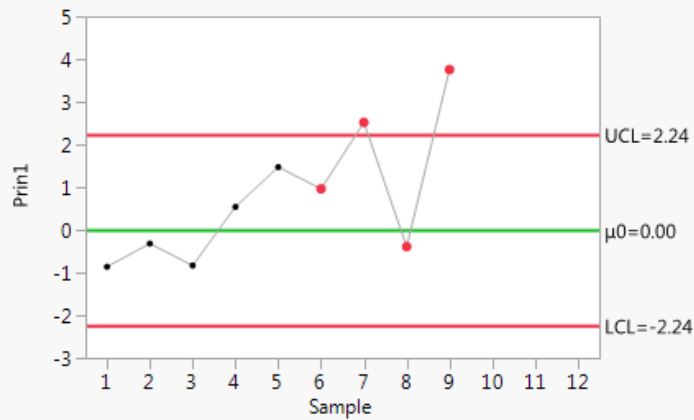
It is also here where it can be found if the out of control signal is coming from a component which normally varies or one which doesn't explain much of the variability in the training groups.



Prin1(54.24 % of typical variation) is comprised of: Max(Height)  
31% Standardised Max(Width) 25% Standardised Max(Path Length)  
28% Standardised Sum(Upper Third) 17%

Control Chart

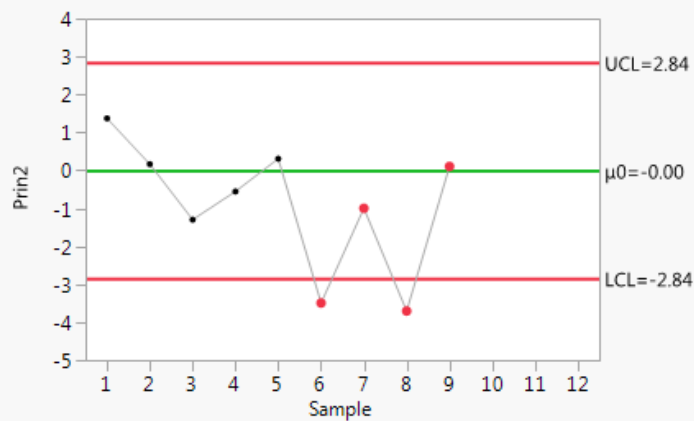
Individual Measurement of Prin1



Prin2(32.81 % of typical variation) is comprised of: Max(Height)  
19% Standardised Max(Width) 25% Standardised Max(Path Length)  
22% Standardised Sum(Upper Third) 34%

Control Chart

Individual Measurement of Prin2



Prin3(12.86 % of typical variation) is comprised of: Max(Height)  
14% Standardised Max(Width) 31% Standardised Max(Path Length)  
25% Standardised Sum(Upper Third) 30%

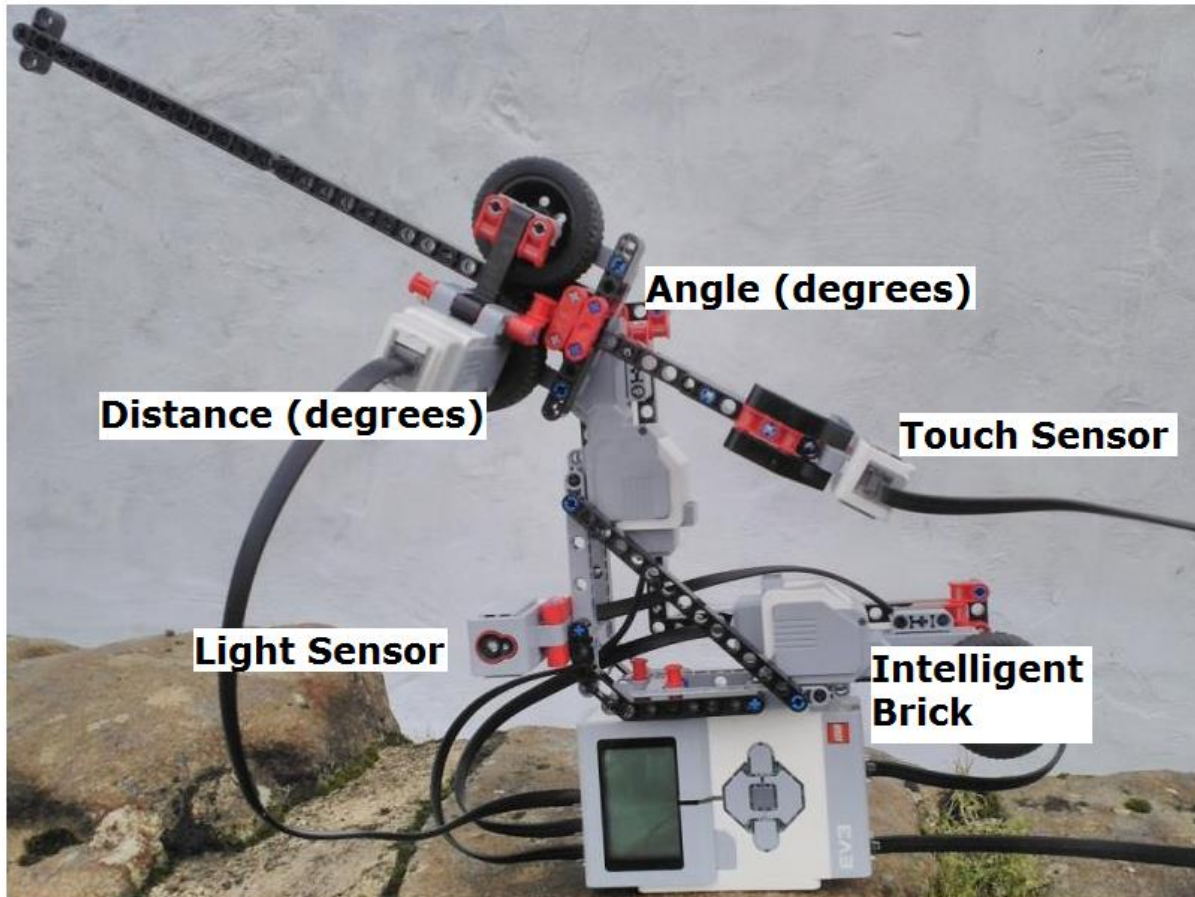
Control Chart

Individual Measurement of Prin3



## Appendix 2: The LEGO® MINDSTORMS® robot

The robot is constructed from a single LEGO® EV3 Home edition kit (Set: 31313). The servo motors contain rotation sensors allowing them to act as inputs alongside the light and touch sensor. This setup is effectively a digital stylus system which includes the ability to assign each letter that is drawn a unique identifier.



The brick is polled on a 50 millisecond basis for the sensor data which is recorded to a csv file. The pen coordinates are captured in Polar format and then converted mathematically to Cartesian format for display and analysis.

## Appendix 3: The data logging software

The LEGO® EV3 kit contains a Linux based computer which you can communicate with via USB, Bluetooth and WiFi. This functionality allows the robot to be programmed via almost any computer language including Java (<http://www.lejos.org/>), C++ (<http://www.monobrick.dk/>) and Linux (<http://www.ev3dev.org/>).

There are also a number of different application programming interfaces (API) available including for the Windows® operating system (<https://github.com/BrianPeek/legoev3>) which is used in this project. The code in this project was written in C# and is based on the tutorial application. Using a PC equipped with a software development environment such as Microsoft® Visual Studio® or Xamarin® in conjunction with the Lego.Ev3.Desktop.dll is all that is required (besides an EV3 kit).

The application consists of an interactive window. The C# code required for the application is shown under MainWindow.xaml and MainWindow.xaml.cs. An outline of the functionality follows.

The window has the following elements:

- A cursor showing the current X and Y position of the stylus.
- A button to log the sensor data.
- A button to save the logged data.
- A button to clear the logged data.
- A counter indicating the index assigned to the letter being drawn currently.
- An indicator of if the stylus is pressed against the page.

The background C# code:

- Connect to the Brick via Bluetooth and COM5 (requires the EV3 brick and PC to have been paired previously)
- Play a tone to indicate the connection was successful
- Create a list to hold the logged data
- Using the BrickChangedEvent get the values from the EV3 brick every time a sensor value changes.
  - Convert the rotation readings from the motors into X and Y coordinates using Polar to Cartesian coordinate conversion.
  - Scale and offset the values so they fit in the display window.
  - If the colour sensor detects a colour increment a counter. This is stored along with the coordinates to differentiate between letters.
  - If the request to log has been made save the transformed sensor values to the list.
  - Update the text and cursor position in the Window
- If the write log button is pressed, store the values of the list to C:\EV3\Log.csv.

**MainWindow.xaml**

```

<Window x:Class="LegoLogger.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:LegoLogger"
    mc:Ignorable="d"
    Title="MainWindow" Height="631.824" Width="550">
<Grid Loaded="Grid_Loaded">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="183*" />
        <ColumnDefinition Width="184*" />
        <ColumnDefinition Width="175*" />
    </Grid.ColumnDefinitions>
    <Button x:Name="StartLoggingButton" Content="Start Logging" HorizontalAlignment="Center"
Margin="14,123,11,442" VerticalAlignment="Center" Width="150" Click="StartLoggingButton_Click" Height="40"
FontSize="24" Grid.Column="2" />
    <Button x:Name="StopLoggingButton" Content="Stop Logging" HorizontalAlignment="Center"
Margin="14,168,11,397" VerticalAlignment="Center" Width="150" Click="StopLoggingButton_Click" Height="40"
FontSize="24" Grid.Column="2" />
    <Button x:Name="WriteLogButton" Content="Write Log" HorizontalAlignment="Center" Margin="14,213,11,352"
VerticalAlignment="Center" Width="150" Click="WriteLogButton_Click" Height="40" Grid.Column="2" FontSize="24" />
    <Button x:Name="ClearLogButton" Content="Clear Log" HorizontalAlignment="Center" Margin="14,258,11,307"
VerticalAlignment="Center" Width="150" Click="ClearLogButton_Click" Height="40" Grid.Column="2" FontSize="24" />
    <TextBlock x:Name="textBlock1" HorizontalAlignment="Center" Margin="12,10,12,0" TextWrapping="Wrap"
VerticalAlignment="Top" Grid.ColumnSpan="3" Height="56" Width="518" FontSize="21.333"><Run Text="Th"/><Run
Text="is"/><Run Text=" program collect"/><Run Text="s the"/><Run Text=" values of "/><Run Text="the "/><Run
Text="sensors. "/><Run Text="The logged data is written"/><Run Text=" to C:\EV3\Log.csv"/></TextBlock>
    <TextBlock x:Name="textBlock" HorizontalAlignment="Left" Margin="10,570,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Grid.ColumnSpan="3" Height="25" Width="497" FontSize="16"><Run Text="LEGO EV3 Data
Logger V"/><Run Text="3"/><Run Text=".0"/><LineBreak/></TextBlock>
    <Label x:Name="GUILetterCount" Content="0" HorizontalAlignment="Left" Margin="20,82,0,0"
VerticalAlignment="Top" FontSize="18.667" Width="50" Height="33" Grid.Column="1" />
    <Label x:Name="label" Content="Current Letter Count :" HorizontalAlignment="Left" Height="33"
Margin="10,82,0,0" VerticalAlignment="Top" Width="193" FontSize="18.667" Grid.ColumnSpan="2" />
    <Label x:Name="label_Copy" Content="X :" HorizontalAlignment="Left" Height="33" Margin="15,308,0,0"
VerticalAlignment="Top" Width="30" FontSize="18.667" Grid.Column="2" />
    <Label x:Name="label_Copy1" Content="Y :" HorizontalAlignment="Left" Height="33" Margin="101,308,0,0"
VerticalAlignment="Top" Width="30" FontSize="18.667" Grid.Column="2" />
    <Label x:Name="label_Copy2" Content="Pen Pressed :" HorizontalAlignment="Left" Height="33"
Margin="13,341,0,0" VerticalAlignment="Top" Width="123" FontSize="18.667" Grid.Column="2" />
    <Label x:Name="GUIxCoordinate" Content="0" HorizontalAlignment="Left" Margin="50,308,0,0"
VerticalAlignment="Top" Grid.Column="2" FontSize="18.667" Width="50" Height="33" />
    <Label x:Name="GUIyCoordinate" Content="0" HorizontalAlignment="Left" Margin="136,308,-11,0"
VerticalAlignment="Top" Grid.Column="2" FontSize="18.667" Width="50" Height="33" />
    <Label x:Name="GUIIsWriting" Content="0" HorizontalAlignment="Left" Margin="136,341,0,0"
VerticalAlignment="Top" FontSize="18.667" Width="31" Height="33" Grid.Column="2" />
    <Rectangle Grid.ColumnSpan="2" Fill="#FFF4F4F5" HorizontalAlignment="Left" Height="443" Margin="10,122,0,0"
Stroke="Black" VerticalAlignment="Top" Width="357" />
    <Line x:Name="GUICursorV" HorizontalAlignment="Left" VerticalAlignment="Top" Stroke="Blue"
X1="0" X2="20" Y1="10" Y2="10" Height="441" Margin="12,124,0,0" Width="355" Grid.ColumnSpan="2" />
    <Line x:Name="GUICursorH" HorizontalAlignment="Right" VerticalAlignment="Top" Stroke="Blue"
X1="10" X2="10" Y1="0" Y2="20" Height="441" Margin="0,124,0,0" Width="355" Grid.ColumnSpan="2" />
</Grid>
</Window>

```

**MainWindow.xaml.cs**

```
using System;
using System.Collections.Generic;
using System.Windows;
using Lego.Ev3.Core;
using Lego.Ev3.Desktop;

namespace LegoLogger
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        Brick _brick;

        public MainWindow()
        {
            InitializeComponent();

            // Create a list to hold sensor data
            public List<string> LogList = new List<string>();
            public bool IsLogging = new bool();
            public int LetterCount = new int();
            public float startDistance = new float();

            // Stuff that happens when the program is loaded
            private async void Grid_Loaded(object sender, RoutedEventArgs e)
            {
                // Connect to brick via bluetooth
                _brick = new Brick(new BluetoothCommunication("COM5"));
                _brick.BrickChanged += _brick_BrickChanged;
                await _brick.ConnectAsync(TimeSpan.FromMilliseconds(50));
                await _brick.DirectCommand.PlayToneAsync(50, 1000, 300);
                // Add comma seperated titles to list
                LogList.Add("IsWriting,NewLetter,LetterCount,xCoordinate,yCoordinate,Size");
                // Initialise variables
                IsLogging = false;
                LetterCount = 0;
                // Set Sensor to Color Mode
                _brick.Ports[InputPort.Three].SetMode(ColorMode.Color);
                startDistance = -444;
            }

            //Stuff that happens every time the sensor values change
            private void _brick_BrickChanged(object sender, BrickChangedEventArgs e)
            {
                var IsWriting = _brick.Ports[InputPort.One].SValue;
                var NewLetter = _brick.Ports[InputPort.Three].SValue;
                var Angle = _brick.Ports[InputPort.A].SValue;
```

```

var Distance = _brick.Ports[InputPort.C].SValue;
Distance = (Distance - startDistance) * -1;
var Size = _brick.Ports[InputPort.B].SValue;
var xActual = Distance * Math.Sin(Angle * (Math.PI/180));
var yActual = Distance * Math.Cos(Angle * (Math.PI / 180));
xActual = xActual * 1.5;
yActual = yActual -221 ;

// Logic to Control if to save sensor data
if (IsLogging)
{
    //create comma seperated list of data
    string LogValues = IsWriting.ToString() + "," + NewLetter.ToString() + "," + LetterCount.ToString() +
    "," + xActual.ToString() + "," + yActual.ToString() + "," + Size.ToString();
    //add to list
    LogList.Add(LogValues);
}
// Logic to count colour sensor interactions
if (NewLetter>=1)
{
    LetterCount = LetterCount + 1;
}

// Update GUI
GUILetterCount.Content = LetterCount.ToString();
GUIxCoordinate.Content = xActual.ToString();
GUIyCoordinate.Content = yActual.ToString();
GUIIsWriting.Content = IsWriting.ToString();

// Cursor formatting
GUICursorV.X1 = 178 + xActual - 10;
GUICursorV.X2 = 178 + xActual + 10;
GUICursorV.Y1 = 221 + yActual;
GUICursorV.Y2 = 221 + yActual;
GUICursorH.X1 = 178 + xActual;
GUICursorH.X2 = 178 + xActual;
GUICursorH.Y1 = 221 + yActual - 10;
GUICursorH.Y2 = 221 + yActual + 10;
}
//action to take on UI button presses
private void StartLoggingButton_Click(object sender, RoutedEventArgs e)
{
    IsLogging = true;
}

private void StopLoggingButton_Click(object sender, RoutedEventArgs e)
{
    IsLogging = false;
}

```

```
private void WriteLogButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        //method 1 to write log
        System.IO.File.WriteAllLines(@"C:\EV3\Log.csv", LogList);
    }
    catch (Exception)
    {
        System.Windows.MessageBox.Show("Unable to write to file. Check it is not in use");
        //throw;
    }
}

private void ClearLogButton_Click(object sender, RoutedEventArgs e)
{
    LogList.Clear();
    LogList.Add("IsWriting,NewLetter,LetterCount,xCoordinate,yCoordinate,Size");
    LetterCount = 0;
}

private class GUIText
{
}
}
```