FLAME TREATMENT OF POLYPROPYLENE PLASTICS WITHIN AK STONE GUARDS

Q. IMMELMAN

21459282

Dissertation submitted in partial fulfilment of the requirements for the degree Master of Engineering at the Potchefstroom Campus of the North-West University, South Africa

Promoter: Prof. J. H. Wichers

November 2008

DEDICATION

This dissertation is dedicated to my parents who motivated me to continue with my studies, especially when it seemed a daunting unending challenge. I also acknowledge Professor J. H. Wichers for his assistance and support in transforming my ideas into a dissertation.

This dissertation is in loving memory of Dr. Leslie Immelman (1933 – 1994), my grandfather and role model. Dr. Leslie Immelman was a truly inspirational individual who taught me to think as an individual, with morality and has inspired me throughout my life.

To my grandparents on my mothers side, Agnes and Ivan Barnard I dedicate this to them for the knowledge's gained through the experiences they afforded me in my childhood. The macaroni and cheeses used to fuel my efforts during the long nights conducting my research.

In conclusion I wish to thank AK Stone Guards for presenting me with this challenge, for the use of their equipment and for funding my research. Without this support I would not have been able to complete my research project.

FLAME TREATMENT OF POLYPROPYLENE PLASTICS WITHIN AK STONE GUARDS

M(ENG) 2008
QUINTIN IMMELMAN
CRCED (Vaal)
NORTH-WEST UNIVERSITY, POTCHEFSTROOM

ABSTRACT

Flame treatment of polypropylene is a polarization process which improves the adhesion qualities of the substrate (Kostyk, 2000). Flame treatment is known as a "Pre-Treatment" process and is widely used in industry to promote adhesion before the application of a wide range of processes. Processes such as gluing, painting, lamination and printing are all preceded by a flame treatment process (Eckert, 2004).

The flame treatment process is achieved using a gas burner, normally burning liquid propane gas, which is fired directly onto the surface to be treated (Sabreen, 2005). The flame is passed over the surface at a distance and speed so as to treat the entire surface and not burn the substrate. Chemical changes occur in the substrate however these changes do not penetrate very deep into the substrate (Cain, 2000).

It is important to remember this is a scientific process and the variables speed, distance and temperature of the flame must be controlled with a certain amount of accuracy. The purpose of this research is to determine the ideal conditions which result in the optimum adhesion between the substrate and the proceeding processes. Once the ideal conditions have been determined the research will continue to find the lower limits to gain adequate adhesion thereby tailoring the process for high volume production.

ACKNOWLEDGEMENT

I wish to thank Professor J. H Wichers for his assistance and direction in obtaining my Masters degree in Engineering. Professor Wichers has spent much time reviewing this dissertation and thereby ensuring its accuracy and professional acceptance.

To my parents who pushed me to continue my studies and always allowing me the opportunity to explore the world even if it meant dismantling my toys to discover how they worked or the use of video cameras and electronic devices. My parents always tried to stimulate my mental abilities and never denied me the chance to experience new technologies.

I further wish to extend my gratitude to AK Stone Guards who funded my research and presented me with the unique challenge to flame treat polypropylene plastics. AK Stone Guards sacrificed a tremendous amount of man-hours in anticipation of this research completion as they required it to introduce a significant cost saving and process improvement into the company.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF GRAPHS

LIST OF ACRONYMS

ABB – Asea Brown Boveri Ltd

AKS – AK Stone Guards

°C – Degrees Celsius

C – Carbon

H - Hydrogen

H_2O - Water

LPG – Liquid Propane Gas

OH^- - Hydroxyl

PP – Polypropylene

WBS – Work Breakdown Structure

1.0 CHAPTER 1 - INTRODUCTION

AK Stone Guards is a leading manufacturer of plastic components for the major motor manufacturers in South Africa. They manufacture amongst other things plastic bumpers which are painted within AK Stone Guards to match the colour of the intended vehicle. (AK Stone Guards, 2008)

AK Stone Guards was experiencing high costs associated with the fluorination pre-treatment process. Costs were incurred through the cost to treat the substrate, damages from the process and transportation costs. Fluorination is an external supplier process which AK Stone Guards has no control over thereby eliminating the possibility of modifying the system.

Flame treatment of polypropylene plastics is accomplished utilising the principle of bombarding the plastic substrate using an oxidising flame (Australian Combustion Services , 2004). The addition of heat and oxygen causes a chemical reaction to take place at a depth of between 2 and 10 nanometres into the substrate (Sabreen, 2005). This causes an increase in surface energy and the formation of hydroxyl (OH^-) groups. The higher surface energy and hydroxyl groups of the substrate results in an increase in the adhesion qualities of the substrate and allow for bonding of substances such as paint or glue (Kostyk, 2000).

The purpose of this research is to determine the factors required in chemically altering the plastic substrate to enable adhesion of both paints and glues. The main areas of interest within this research are the speed of the flame passing over the surface, temperature of the flame and the distance of the flame from the substrate.

1.1 WHAT IS FLAME TREATMENT

Polyolefin's such as polypropylene are very attractive to industry and have a host of applications ranging from bumpers to chemical containers (Lenntech, 2007). The disadvantages of polyolefin's are the poor bonding ability due to the low surface energy of the material (Cain, 2000). To increase the bonding ability of polyolefin's the surface is pre-treated with an oxygen rich flame. The process chemically changes the substrates surface molecular structure in a controlled manner (Cain, 2007). Changing the molecular structure

increases the surface energy and "wet ability" of the substrate making it compatible with coatings and materials (Kostyk, 2000).

1.2 APPLICATIONS OF FLAME TREATMENT

Flame treatment is used as a pre-treatment process for a large number of applications. These include but are not limited to painting, gluing, lamination and applying of specialized coatings such as adhesive backings (Eckert, 2004). Flame treatment is not limited to the use on polyolefin's and can be applied to fields as diverse as the textile industry for the application of backings, the carpet industry applying backings to the carpet, or to the manufacture of carton boards (Grant, 2004). Flame treatment does not necessarily have to be used to increase adhesion qualities it may also be used in degreasing aluminium foils (KEIKO, 1997).

1.3 PROBLEM STATEMENT

AK Stone guards paint a range of polypropylene products for the motor industry of South Africa. The painting of polypropylene has to be preceded by a treatment process to promote the adhesion of the paint to the surface. Large amounts of money was spent fluorinating the products as a treatment for paint adhesion. This money was spent on an external company and costs were incurred through a 266km round trip product transportation, damages to the plastic products by the treatment supplier and the cost of treatment. To save money and process time the decision was made to treat the plastics in-house with a flame treatment process. Early attempts at flame treatment were unsuccessful and the paint was failing the industry tests for adhesion. It was determined that the variables of velocity and distance of the flame from the substrate were incorrect causing the flame treatment process to be less than optimal.

The purpose of this research is therefore to determine the ideal conditions for the velocity of the flame passing over the substrate and the distance of the flame from the substrate. Under these ideal conditions the surface should be at maximum treatment yielding the greatest adhesion properties. The researcher acknowledges that many other variables may

exist which could affect the change in the adhesion properties of the polypropylene, however these are outside the scope of this research. Once the optimum conditions have been determined the research will continue to determine the maximum conditions of velocity and distance to yield the greatest production capacity at acceptable adhesion levels. The temperature of the flame is an important factor and will influence the results obtained from both velocity and distance, however this variable will be a fixed constant within a range obtained from research. In addition the stoichiometry of the reacting gasses forming the flame will not be investigated and is a topic for further research.

1.3.1 SUB PROBLEMS

Sub Problem 1 – Flame Velocity
The ideal speed of the flame travelling over the product must be determined.

Sub Problem 2 – Flame Distance
The ideal distance of the flame from the product must be determined.

Sub Problem 3 – Adhesion Production Characteristics
The greatest velocity and distance must be determined to yield an adequate level of adhesion reducing cycle times and increasing output for production.

1.3.2 THE DEPENDENT VARIABLE

The dependent variable in this research project is the strength of adhesion between the substrate and any bonding agent. All of the independent variables have an influence on the adhesion strength. Adhesion strength will be measured using an adhesion strength testing apparatus and results will be in the unit Newton (N). For use of this data results will be converted to Pascal's.

1.3.3 THE INDEPENDENT VARIABLES

The researcher acknowledges that many other independent variables may exist which could influence the change in adhesion from flame treatment of polypropylene. These

independent variables have been reduced to the three sub problems for the purpose of this research.

1.4 RESEARCH HYPOTHESIS

H1. Flame treatment of polypropylene will increase the adhesion properties of the material.

H2. If the velocity of the flame traversing the surface of the substrate increases beyond the upper limit of the optimum setting the adhesion strength will decrease. Subsequently should the velocity decrease sufficiently, substrate damage will occur rendering the product unusable.

H3. Should the distance of the flame and the substrate increases beyond the upper limit of the optimum setting the adhesion strength will decrease. If the distance is decreased below the lower limits, substrate damage may occur and the process repeatability may be compromised due to greater levels of accuracy required to prevent the gas head from colliding with the substrate. In both cases the hypothesis only holds true if there is no compensation in velocity to counter this effect.

H4. The system will be able to accurately repeat the process in a production environment.

H5. There exists a narrow band of values in which a change in velocity and or distance from the substrate surface will not have any significant decrease in adhesion strength.

1.5 AIMS AND OBJECTIVES

The aim of this research is to first establish if flame treatment of polypropylene has any influence on the adhesion properties of the material. If flame treatment influences adhesion the velocity and distance of the flame from the substrate yielding the greatest increase in adhesion will be determined. The research will then further analyse the data from the experiment to determine the velocity range and distance best suited to a **mass production environment**. The researcher will then attempt to derive a mathematical formula to best describe the experimental data for use in the calculation of adhesion at various velocities and distances.

1.6 DELIMITATIONS OF THE STUDY

The research will be conducted at AK Stone Guards. One of AK Stone Guard's ABB Robots will be used to that ensure all experiments are conducted accurately. The robot's velocity can be set programmatically and has internal controls to ensure the speed is accurate. ABB Robots have a tolerance of ±0.4mm, so small it may be ignored resulting in a repeatable and accurate distance control between the flame and the substrate. Using the ABB Robot makes it possible to alter a single variable per experiment. All testing pertinent to adhesion will be conducted at the AK Stone Guards testing laboratories.

The research will not be investigating the affects of the temperature of the flame on adhesion. The flame temperature will be set to 938°C. Any decrease in adhesion over time between flame treating and bonding to the substrate will not be investigated; panels will be painted immediately after treatment. Flame treatment of other materials will not be investigated in this research as the researcher is only interested in polypropylene.

1.7 ASSUMPTIONS

- The paints and glues used in the experiments will be consistent with respect to adhesion.
- The painting and glue application processes will be consistent between experiments.
- The time between flaming and painting or gluing will be constant.

1.8 IMPORTANCE OF THE STUDY

The research should develop a process to provide high adhesion properties in polypropylene plastics for use in gluing and painting operations at AK Stone Guards. The process will theoretically have a higher output when compared to current processes used for surface treatment resulting in an increase in the company's efficiency and contributing to the profitability of the organization. Pre-treating the substrate in-house allows for greater control and flexibility to AK Stone Guards. The process can be tailored to reduce the amount of damages in comparison with the current process which is outsourced allowing for no process modifications by AK Stone Guards. This also gives AK Stone Guards better accountability in the event of customer field complaints with respect to adhesion problems and corrective actions can be executed.

1.9 PROJECT PLANNING

1.9.1 BUDGET FOR FLAMING RESEARCH PROJECT

	Costs Incurred By:	Quantity		Costs	
		Minimum	Maximum	Minimum	Maximum
1	Polypropylene Panels for use as test pieces	60	70	R 55.50	R 315.00
2	White Paint (Liters)	3	5	R 975.00	R 1 625.00
3	Adhesion Testing Machine Design @R720/day	3	5	R 2 160.00	R 3 600.00
4	Adhesion Testing Machine Materials	1	1	R 2 655.00	R 2 655.00
5	Construction of Adhesion Testing Machine	1	2	R 1 000.00	R 2 000.00
6	Software for adhesion machine @R720/day	14	28	R 10 080	R 20 160
7	Labour 8 Hours / Day @R90/Hour	5	10	R 3 600.00	R 7 200.00
8	Use of Robot @ R450 / Hour	2	4	R 900.00	R 1 800.00
9	Documentation	1	1	R 1 200.00	R 2 300.00
10	Total			R 22 625.50	R 41 655.00

Table 1. 1 Budget For Flaming Research Project

1.9.2 TIME FRAME FOR FLAMING RESEARCH PROJECT

Below is a table graphically representing the project time plan.

Task	Year 1: 2008						
	May	June	July	August	September	October	November
Research Flame Bonding	X	X	X				
Complete Chapter 1 and 2 of Dissertation				X			
Present to Professor J. H. Wichers For Initial Evaluation				X			
Design Experiments				X			
Complete Chapter 3 of Dissertation				X	X		
Design Adhesion Testing Machine				X			
Build Adhesion Testing Machine				X	X	X	
Conduct Experiments and Evaluate Results						X	
Complete Dissertation						X	X
Submit Dissertation for Review							X
Alter Dissertation According To criticism from Review							X
Submit Final Draft of Dissertation							X

Table 1. 2 Timing Plan For Flaming Research Project

2.0 CHAPTER 2 - LITERATURE REVIEW

2.1 INTRODUCTION

The intention of this research is to firstly determine the ideal conditions under which flame treatment of polypropylene plastics will yield the highest adhesion properties. The second part of this research is to find the range of values to increase adhesion to an acceptable level while achieving the greatest production capacity. The two main aspects affecting the independent variable will be under investigation, namely the velocity and the distance. For the purpose of this research the temperature of the flame will be a constant and not investigated.

The adhesion between the polypropylene substrate and the bonding agent to be applied is achieved through the increase in surface energy, wet-ability of the plastic and the formation of hydroxyl groups in the chemical structure (Sabreen, 2005). This change in molecular structure is achieved through the application of an oxygen rich flame passing over the surface of the substrate (Sabreen, 2005). Polypropylene has a very low initial surface energy measuring 29 Dynes/cm in comparison with ABS plastic at 42 Dynes/cm (Kostyk, 2000). The adhesion strength is dependent on the amount of increased energy, wet-ability and the quantity of hydroxyl groups formed (Sabreen, 2005). These factors are influenced by the velocity of the flame passing over the substrate, the distance of the flame from the substrate and temperature of the flame (Grant, 2004).

The temperature of the flame provides the additional energy required for the formation of chemical bonds (hydroxyl groups) as well as increasing the surface energy of the substrate (a, a, a, & Heath, 1994). Thermal activated atoms and molecules are produced in the hot flame and this activated species can then be deposited into the surface of the substrate increasing adhesion properties (Eckert, 2004).

In addition to the formation of hydroxyl groups other higher surface energy groups such as carbonyl, carboxyl and amide groups are formed. These groups are formed through the oxidation of the surface of the plastic by the flame through a free radical mechanism. The increase in adhesion can also be linked to the scission and cross linking of the molecular chain (Petrie, 2006).

The distance of the flame in relation to the substrate influences the adhesion. If the distance is too great the activated species is unable to adequately reach the substrate and form chemical changes. The temperature of the substrate is not raised sufficiently thus decreasing the adhesion strength. There is an optimum distance which is known as the active zone of the flame (Grant, 2004). This is the distance between the head and the substrate yielding maximum adhesion and is to be determined in the research.

Velocity, the speed at which the flame traverses the substrate has a double affect. It determines the cycle time of the operation as well as influencing the adhesion strength between the substrate and any bonding agents (Eckert, 2004). If the velocity is set too slow substrate damage may occur rendering the product unusable, the gas consumption is higher than expected and the surface may become over treated (Grant, 2004, p. 2). However if the velocity is too fast the surface is not adequately exposed to the active species and the temperature of the substrate is not raised sufficiently to form chemical changes (Eckert, 2004). The ideal band is to be determined in this study.

2.2 CLARIFICATION OF CONCEPTS

2.2.1 VELOCITY OF THE FLAME

The velocity at which the flame traverses over the surface of the substrate has a direct impact on the adhesion strength gained by the process (Eckert, 2004, p. 4). The velocity controls the amount of heat the substrate will get and the quantity of activated atoms reaching the surface (Eckert, 2004, p. 4). These factors have a direct impact on adhesion. The melting point of polypropylene is around 160°C (Wikipedia, 2008). The velocity must be of adequate speed to ensure the substrate does not get hotter than 160°C, melting the substrate, and slow enough to adequately treat the surface.

2.2.2 DISTANCE FROM THE SUBSTRATE

The distance from the substrate is measured between the bottom of the gas head and the top of the product. The distance influences adhesion by regulating the temperature of the substrate and the amount of activated atoms reaching the surface (Eckert, 2004, p. 4). The

greater the distance the lower the temperature and the less activated atoms will reach the substrate. When the distance is too low the temperature may increase above 160°C melting the product and the control of the gas head must be more precise so the head will not foul with the product (Eckert, 2004, p. 4) (Wikipedia, 2008).

2.2.3 TEMPERATURE OF THE FLAME

The flame must have enough energy to cause chemical bonds to form and enough free oxygen molecules as a building block for the chemical reactions (Eckert, 2004, p. 3). The flame should be blue in colour indicating an oxygen rich flame (Eckert, 2004, p. 3). If the flame is yellow in colour carbonisation will occur on the substrate and the bonding agent will not adhere on the surface (Scribd, 2008, p. 9). The intensity of the flame determines the velocity and distance which the gas head must travel at (Eckert, 2004, p. 4).

2.2.4 ADHESION STRENGTH

Adhesion strength is the force at which the bonding agent adheres to the substrate (DeFelsko, 2004). It can therefore be redefined as a measure of the force required to strip the bonding agent from the substrate. Traditionally the unit for adhesion strength is in dynes per centimeter (Wolf, 2004, p. 2) but recently units such as Pascal's and Newtons are used (DeFelsko, 2004). For the purpose of this study adhesion will be measured in Newtons.

2.2.5 ROBOTIC FLAMING

To keep the dependent variables constant during experimentation an ABB Robotic arm will be used to conduct the experiments. The robot has a tolerance of ±0.4mm so small it may be ignored therefore the distance of the flame to the bumper will accurately be repeated (RobotWorx). The Robot has control units allowing the user to set the velocity of the arm and to easily make changes to this velocity. In addition to the advantages for experimentation the arm is ideal for high volume production once the research has been completed. It offers a degree of safety for the operator in a production environment and accurate repeatability of the ideal conditions.

2.2.6 CLEANING

All surfaces to be further treated must be adequately cleaned to remove any grease, oils and impurities on the substrate (Fisher, 2006). Grease, oils and impurities are transferred via contact between the surface and human operators handling the product. Although many of these contaminates may be burned off (Eckert, 2004, p. 4) a thin residual layer may remain behind lowering the adhesion properties and giving false values in the experimentation (Master Bond Inc, 2007). All surfaces will be cleaned with isopropyl alcohol to ensure they are properly cleaned for further processes.

2.2.7 CONTROL

One piece of polypropylene will be cleaned and painted without any pre-treatment process. This will be used to determine whether or not the pre-treatment process using an oxygen rich flame has an operational advantage for further processes. This will determine if the polypropylene is left untreated would it have sufficient adhesion characteristics for further processes.

2.3 FLAME TREATMENT OF POLYPROPYLENE PLASTICS

The dependent variables of velocity, temperature and distance have an impact on the strength of adhesion between the substrate and any bonding agents applied (Wolf, Corona Treatment: A Process Overview). The polypropylene substrate to be treated is comprised of long carbon and hydrogen chains of molecules as seen in figure 1 and figure 2 below (American Chemistry Council, 2007).

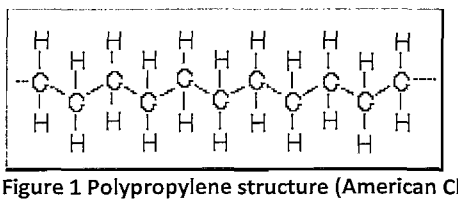


Figure 1 Polypropylene structure (American Chemistry Council, 2007)

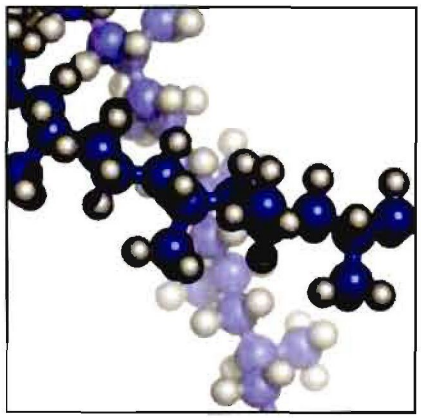


Figure 2 Ball-and-Stick Model of Polypropylene (Wikipedia)

The atoms of carbon are bonded together with single non-polar covalent bonds (Garver, 2006). These carbon chains form the backbone of the compound (American Chemistry Council, 2007). Attached to each carbon atom is a hydrogen atom experiencing Van der Wal forces between the bonds (Garver, 2006). Polypropylene is a stable inert compound with uses ranging from food packaging to bumpers for automobiles (Wikipedia, 2008). Unfortunately polypropylene is difficult to bond to due to the low surface energy and un-reactive surface chemistry (M.D. Green, 2000).

To increase the adhesion properties of polypropylene various surface treatment processes have been developed (Fisher, 2006). Each method has its own set of pros and cons. For this study Flame treatment will be the only method under investigation. Flame treatment uses an oxygen rich flame to bombard the surface of the polypropylene forming hydroxyl (OH^-) groups, carboxyl groups, esters and ethers at a depth up to 10nm into the surface. These groups which form onto the molecular structure, up to 10nm (Wolf, 2004), convert the substrate from a non-polar entity to a polar entity enhancing the adhesion qualities of the plastic (Eckert, 2004, p. 3).

In order to achieve these bonds energy has to be applied to the surface, this is induced in the form of the flame (Eckert, 2004, p. 3). It is critical that the flame be oxygen rich (Eckert, 2004, p. 3). The free oxygen molecules bond with the hydrogen found in the long hydrocarbon chains of the plastic forming the hydroxyl groups. The free oxygen "activates" the substrates surface (Grant, 2004, p. 2). In addition to forming new bonds, the flame breaks up some of the long carbon chains found in the surface making chemical linking to

these chains easier (Eckert, 2004, p. 3). An indication of a flame which is properly set is a blue colour as seen in figure 3 below (Eckert, 2004, p. 3).

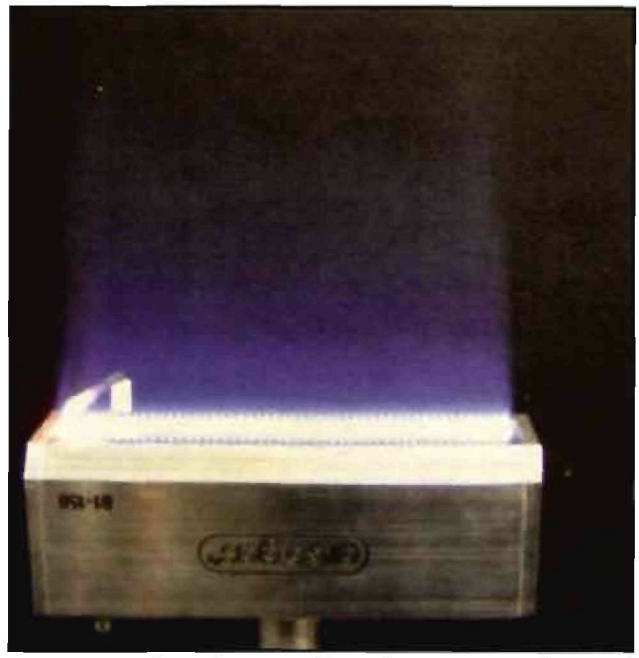


Figure 3 Properly set flame used in surface treatment. (Arcogas)

Forming the bonds requires energy however too much energy can have undesirable effects (Eckert, 2004, p. 4). The breaking of the carbon chains continues beyond the desired point and the substrate melts and may even burn. To prevent damage to the substrate the dependent variables velocity and distance are used to control the amount of energy the surface receives (Eckert, 2004, p. 4). Flame treatment operates within a narrow band of velocity and distance providing the correct amount of energy and free oxygen molecules to the surface (Grant, 2004, p. 2).

The velocity is the speed at which the flame traverses over the surface of the substrate. Velocity is influenced by the temperature of the flame; the hotter the flame the faster the velocity must be to keep the amount of energy reaching the surface within tolerance (Eckert, 2004, p. 4). If the velocity is too fast then the flame passes over the surface without forming sufficient chemical reactions (Eckert, 2004, p. 4). The number of hydroxyl groups formed will be too few to have an adequate impact on adhesion.

When the velocity is under the lower limit substrate damage may occur (Eckert, 2004). Damages from a velocity which is too slow can occur in the form of distortion, melting, splitting too many carbon hydrogen molecular bonds (Eckert, 2004) *and even* the burning of

the substrate. The most optimum condition for velocity will be on the upper limit of the ideal band, this is to maximise the amount of product which can be treated per working day minimising the cycle time.

The last dependent variable under investigation is the distance of the flame from the substrate. This gap must be close enough for the product to pass through the "active" oxidising zone of the flame and far enough to be out of the reducing zone (Grant, 2004, p. 3). The reducing zone is found from the base of the burner to the tips of the light blue cones of the flame (The Columbia Encyclopedia, Sixth Edition, 2001). The active oxidising zone is between the tips of the cones and the end of the flame (The Columbia Encyclopedia, Sixth Edition, 2001).

The length of the cones may be set by varying the pressure of the gas entering into the head. The longer the cones are the more beneficial to production as this increases the distance and allows for a greater tolerance when passing over the product. By increasing the tolerance the chances of fouling with the bumper and damaging the substrate diminish. The diagram figure 5 shows the two zones of the flame. Figure 4 below is the original picture modified by the author to create figure 5.

Figure 4 Original picture of the flame (Eckert, 2004)

From the text above and the picture obtained in the documentation from Eckert the picture of the flame below was drawn by the researcher. This picture shows the various zones of the flame.

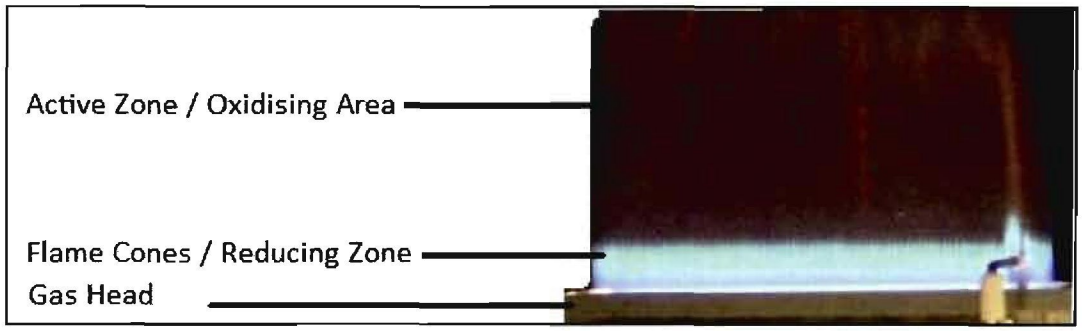


Figure 5 Zones of an oxygen rich flame.

The maximum temperature of the flame is located on the tip of the active zone just before the flame disappears into the atmosphere.

Further bonding agents such as paints and glues form chemical bonds with the newly formed hydroxyl groups (Jing Songa, 2007). The strength of the adhesion between the polypropylene and the bonding agents is dependent on the number of hydroxyl groups formed in the flaming process (Jing Songa, 2007). The bonding agents adhere to the hydroxyl groups due to their polar nature unlike the non-polar polypropylene substrate (Jing Songa, 2007) (Garver, 2006). Figure 6 below shows the basic formation of these hydroxyl groups.

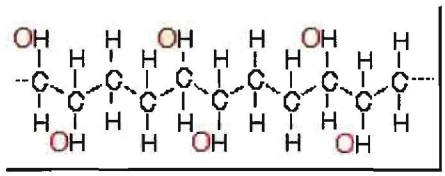


Figure 6 Simplified diagram of Basic Hydroxyl Groups Formed in Substrate.

Flame treatment increases the "surface wet out" which determines how well a liquid flows over the surface and intimately covers the substrate (Kostyk, 2000). Maximum adhesion occurs when the bonding agent thoroughly wets out the surface (Kostyk, 2000). This increases the contact surface area and allows the bonding agent to have maximum exposure to chemical linkages in the substrate. Figure 7 below highlights the increased contact area and shows the difference between non-treated and treated surfaces.

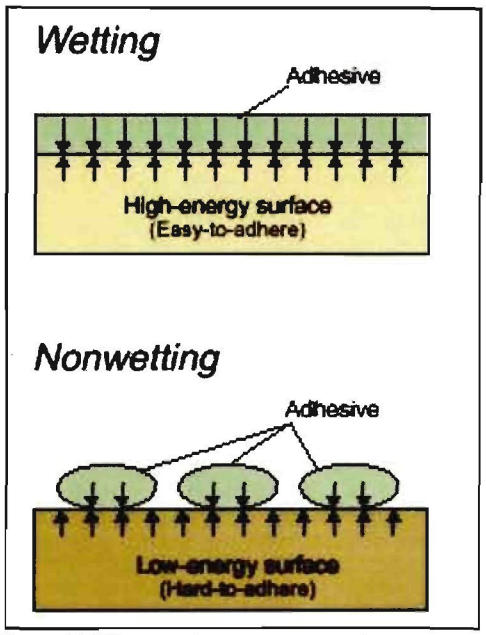


Figure 7 Difference between wet and non wet surfaces (Kostyk, 2000).

In conclusion flame treatment increases the adhesion characteristics of the substrate by forming polar hydroxyl groups between 2 and 10nm into the surface. This property can be further exploited by other processes allowing bonding to the substrate. It is therefore beneficial to determine the ideal conditions for velocity and distance from the substrate to enable maximum adhesion. This process can be used in industry to enable successful applications of glues and paints for a wide range of applications.

3.0 CHAPTER 3 - EMPIRICAL INVESTIGATION

In order to determine the ideal values for velocity and the distance of the flame from the substrate an experimental process will be used. The velocity and distance from the substrate is dependant on the temperature of the flame therefore the temperature will be set at a constant. One panel will be painted, untreated by the flaming process, and will serve as a control to which all measurements can be compared against. The velocity and distance of the flame from the substrate will be varied thereby comprising the dataset.

3.1 PRIMARY DATA

The primary data gathered from the experiment will be used for the analysis. The two dependent variables, velocity and distance of the flame, will be the only values influenced by the outcome of the experiment. All data is to be collected and tabularised for later analysis with the goal of finding a correlation between the velocity and distance of the flame on paint adhesion.

3.2 CRITERIA FOR THE ADMISSIBILITY OF THE DATA

Data in the final conclusion will only be considered valid if the panel passes a paint adhesion test. For production purposes the greatest velocity and distance from the substrate will be considered most economical. This will yield the fastest cycle time and greater tolerance for the application of the flame. All results from the experiment will be considered and evaluated for the derivation of an adhesion formula. The adhesion formula will be used to calculate the adhesion of a bonding agent to a substrate at a given velocity and distance. Conversely the distance or velocity can be obtained for a required adhesion.

3.3 MEASURING INSTRUMENTS

All test pieces will be numbered and the characteristics logged in a table. An ABB Robot IRB-6000 M93 will be used to perform the flaming operation. The ABB Robot allows the user to specify the speed of the robotic arm in mm/s without modifying any of the other

characteristics within the program. A calibrated Mitutoyo vernier will aid in setting the distance of the flame from the substrate.

The temperature of the flame is measured using a HI 9053 K-Type Thermocouple with portable microcomputer. The unit has been calibrated and has an accuracy of ±0.5°C.

The distance is determined between the end of the gas head and the top of the substrate being treated. The adhesion strength between the bonding agent and the substrate will be measured with the aid of a specialised adhesion testing machine designed and built for this research. Details of this machine can be found in Appendix B. A software program was developed by the researcher to capture the data from the machine for analysis. The software was written in Visual C++ and is tailored to the research with graphing capabilities to view the forces relayed from the machine.

Due to budget constraints the researcher had to design and build a machine to test the adhesion between the polypropylene and the paint. The machine consists of three parts the mechanical component representing the physical machine, the electronic component giving the machine a degree of intelligence and the software component which is embedded onto the microcontroller in the electronic component.

The machine uses these basic elements to perform the experiments. The basic concept of the machine is to convert angular movement into lateral movement. The lateral movement applies force between the polypropylene panel and the dumbbell glued onto the surface of the panel. As the force increases it is electronically measured using a load cell. The load cell converts force both compressive and tensile into an electrical voltage.

It is the voltage that the electronic component of the machine measures. The load cell has a very close to linear increase in voltage in relation to the force applied to it. The voltage is directly proportional to the forces acting on the load cell. The machine records the changes in voltage and converts the voltage into force using the software component of the machine.

The change in force continues until the dumbbell has been pulled off of the panel and the force then rapidly falls to zero. At the instant before the dumbbell has been stripped from the panel the force is at a maximum and this value is recorded for the experimental data.

The mechanical part of the machine was designed on CAD by the researcher. The thickness of the support plate was chosen against standard sizes available to the researcher and then evaluated using the bending equations in Appendix B. The support plate had to be thick enough to resist excessive flexing as this would compromise the data from the experiment. At the time the machine was designed it was not known to the researcher how much force would be required to pull the paint from the polypropylene so the machine was developed to tolerate one metric ton of force.

With the support plate calculated the handle used to convert angular rotation into lateral force had to be calculated. The calculation of the handle length can be found in Appendix B. The handle must provide enough leverage to allow the machine to pull one metric ton of force using the operator's strength. The force generated by the handle is the force used to pull the dumbbell from the panel.

The electronic component of the machine was designed and built by the researcher. The schematic diagram was drawn on CAD followed by the printed circuit board layout. The CAD for the printed circuit board was emailed to a company specialising in the etching of these boards. When the bare circuit board arrived the researcher populated the board by soldering the components on to it. Although the machine does not take full advantage of all the functions on the board it was designed for future use in other applications.

With the printed circuit board populated and the machine built the embedded software on the microcontroller had to be developed. The researcher developed this software using Keil C++ for the ST Arm microcontroller. The software measures the voltage generated by the load cell through an analog to digital converter. This voltage is then transformed into a numeric value representing the force on the load cell.

The software had to be calibrated to determine the correct multiplier to be used in converting the voltage into a force. The researcher used a digital scale to measure a series of weights up to 10kg. These weights were then applied to the load cell and the voltage read off. The value of the weight was then divided by the value for the voltage to get the multiplier. The software handles the relaying of the force to a personal computer via the USB port. Device enumeration, driver allocation, device requirements had to be sent to the computer when the unit is connected. The software must establish two data pipes for

communication one IN pipe and one OUT pipe is used. The unit communicates continuously with computer sending the force as it is converted.

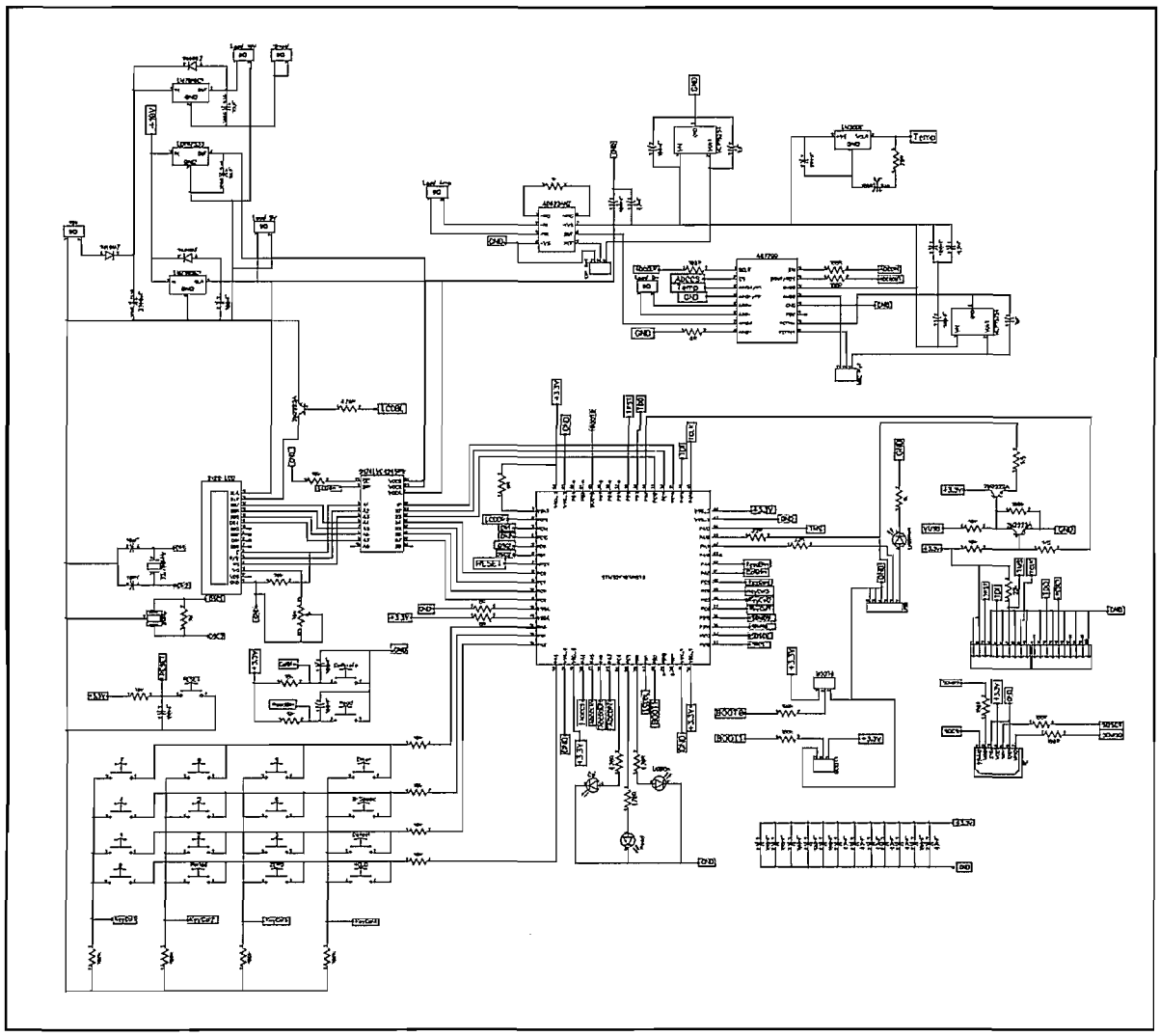


Figure 8 Adhesion Testing Machine Schematic Design

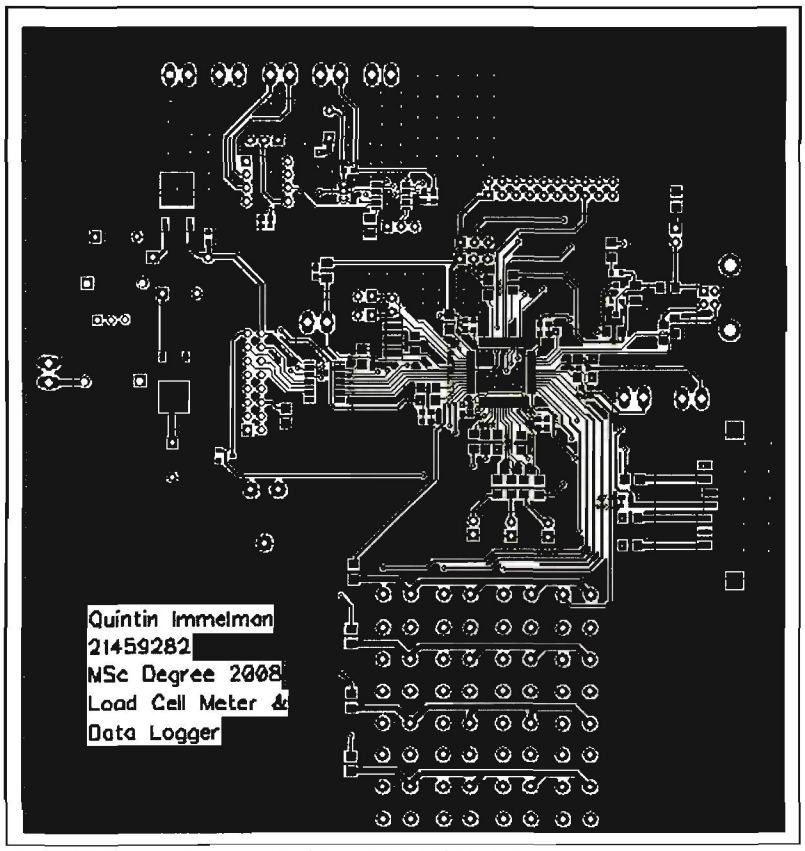


Figure 9 Adhesion Testing Machine PC-Board Layout

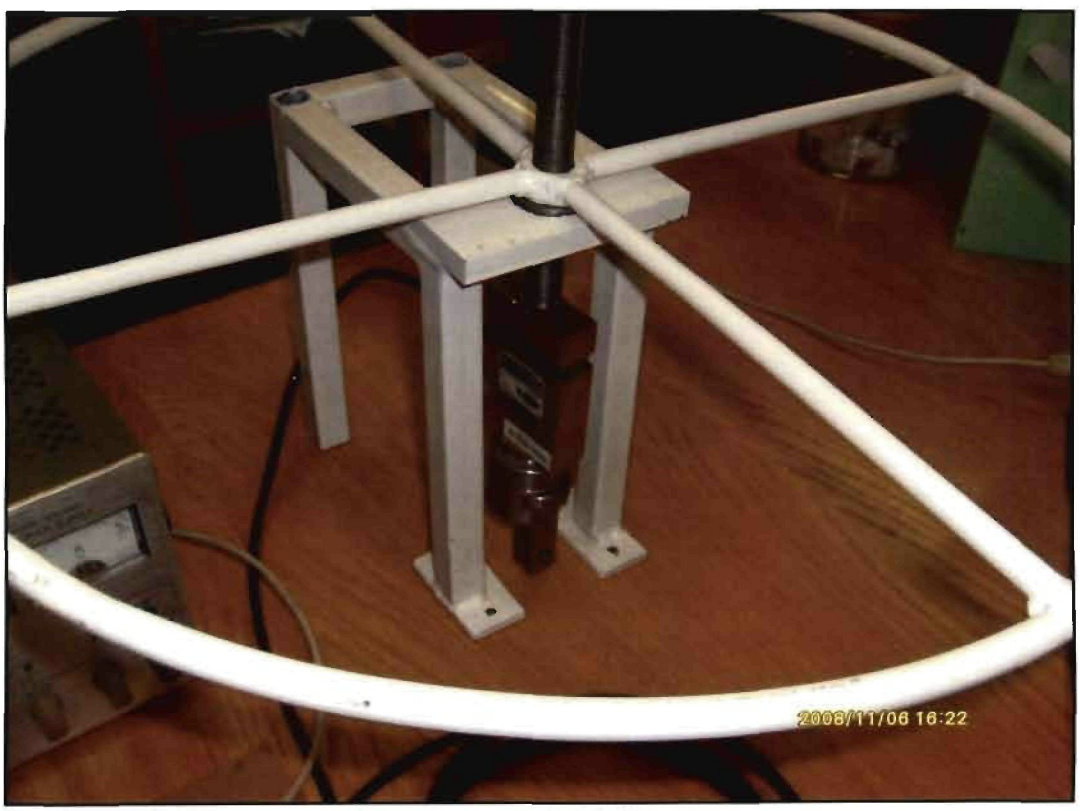


Figure 10 Mechanical component of the Adhesion Testing Machine

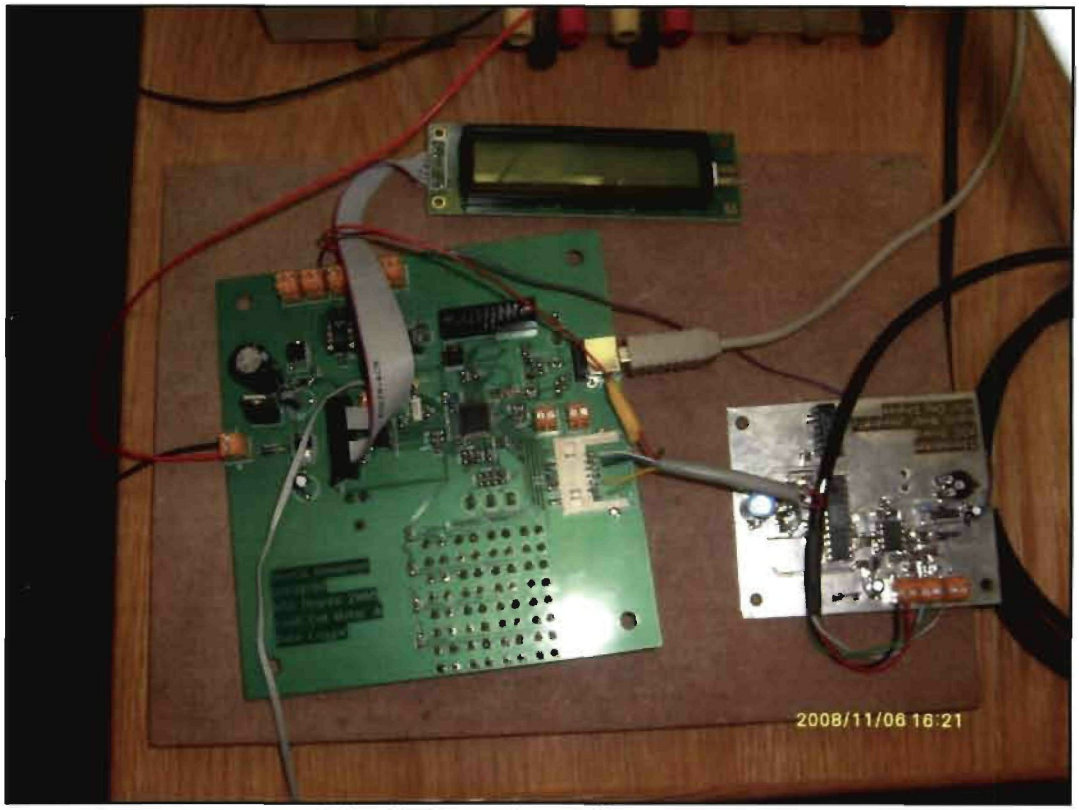


Figure 11 Electronic component of the Adhesion Testing Machine

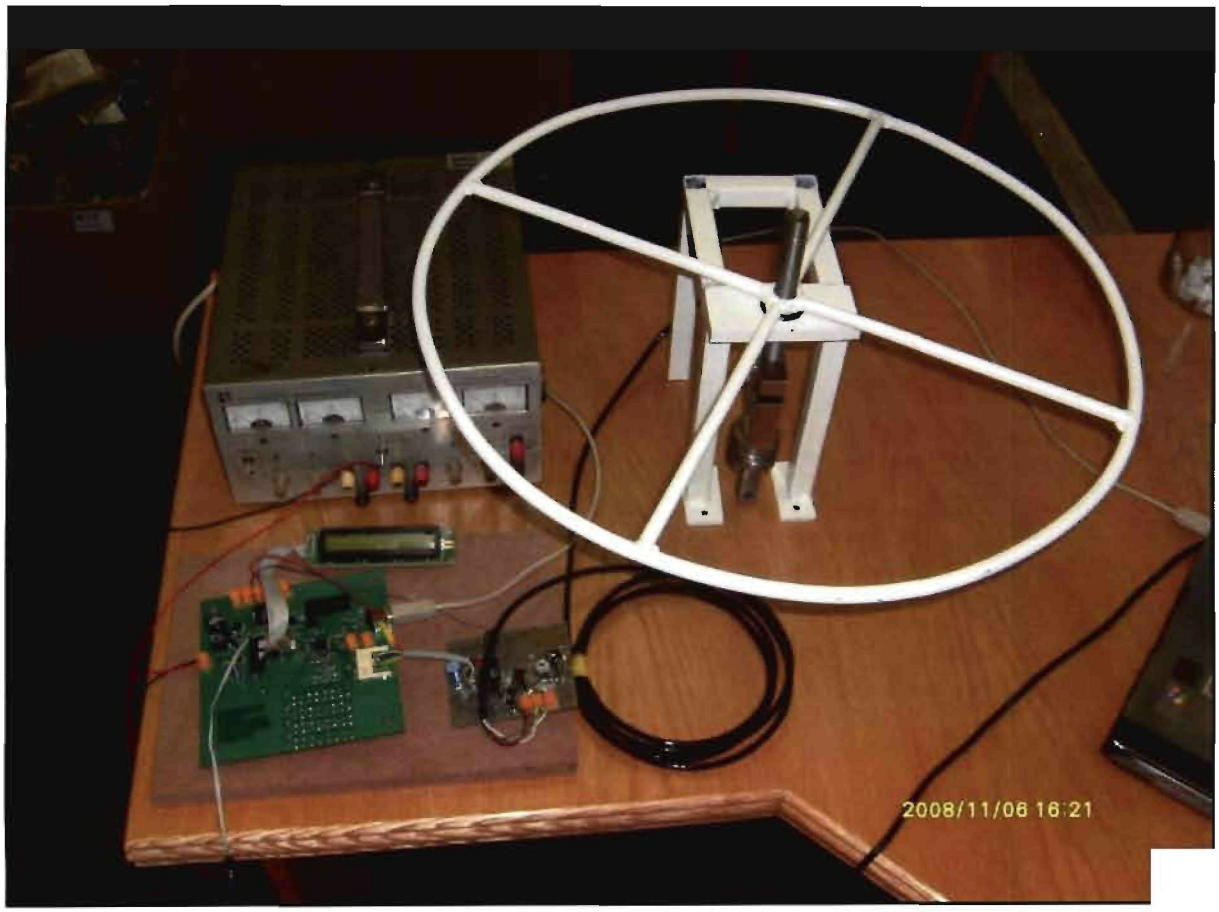


Figure 12 The Complete Adhesion Testing Machine

The final component of the testing unit was to develop the software to be run on a personal computer. The researcher wrote this software in Microsoft Visual C++. The software will run on any Windows environment from Windows XP upwards. The software receives the force from the machine through the data pipes established in the embedded software.

The user of the program connects to the machine by clicking on a button placed in a ribbon. When the user connects to the machine the program will record the data from the machine. The user can change the way the data is received, data can be received continuously or only when there has been a change in the force. Data is displayed in a tabular form and a graph is dynamically drawn as the data is received. The units of the data can be changed at any time selecting between Grams, Kilograms or Newtons.

Many tests can be opened at the same time allowing the data to be compared between experiments. The program plots the graph of the data as well as the graph of any file chosen by the user to be compared against. This graph can be drawn on its own or superimposed onto the data from the current experiment. The user of the software can hold in the control key on the keyboard and pass the mouse over the graph. This will draw a line on the graph to represent the current position of the pointer and then display the value of the force at that point. This is useful for analysing the data as the maximum force can be easily found and read off. When the experiment has concluded the data can be saved for later analysis.

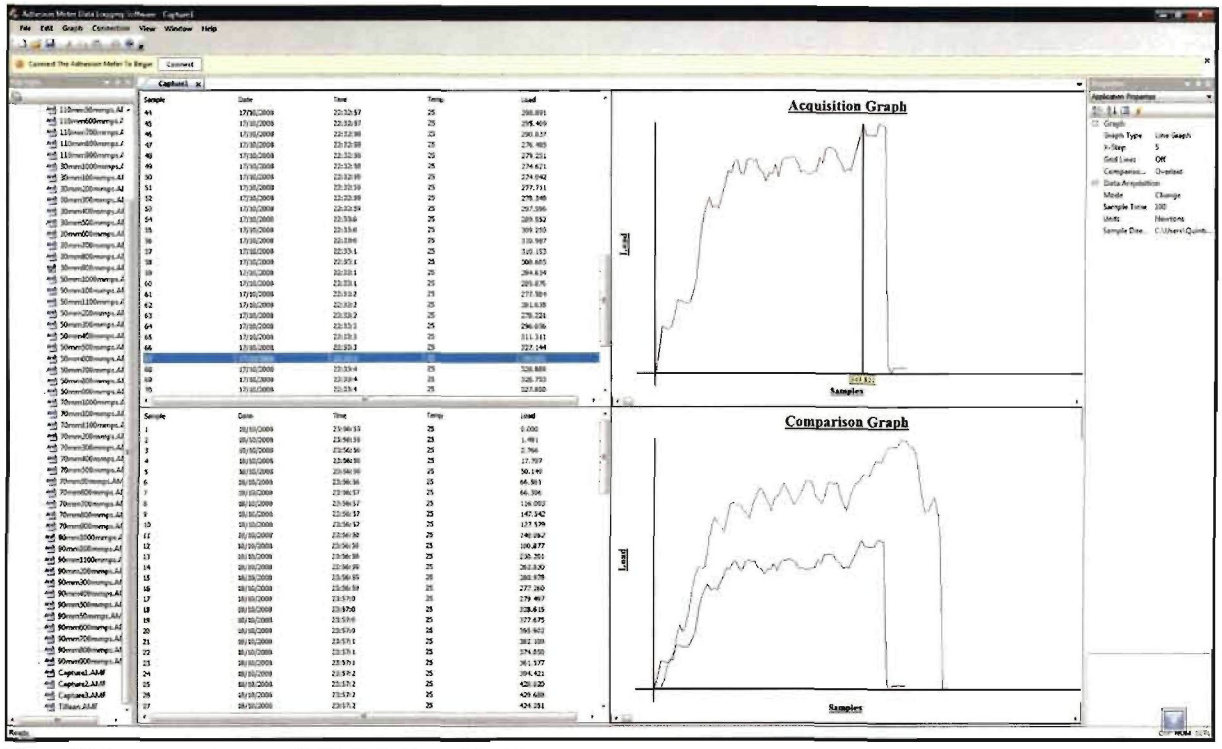


Figure 13 Computer Program With Data From Machine

A small piece of the software can be found in Appendix C and the full source code is on the CD accompanying the research. The compiled program packaged as a windows installation along with the data from the tests in also on the CD.

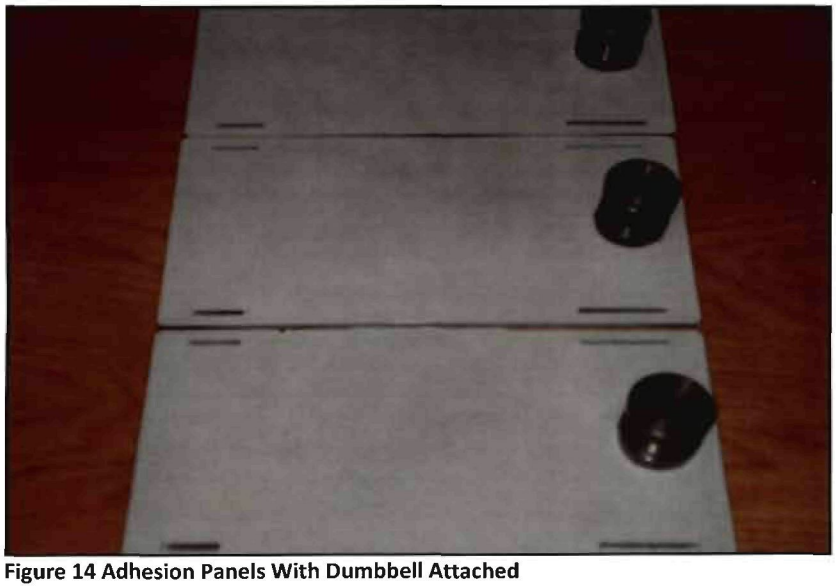


Figure 14 Adhesion Panels With Dumbbell Attached

3.4 THE RESEARCH PROCESS

The research experiment will be conducted in the following logical order:

Step 1 Design a table to store results and determine the number of experiments to be performed.

Step 2 Obtain polypropylene panels moulded from the same material used in the manufacture of bumpers.

Step 3 Number each panel and record the characteristics of the test to be performed on the panel.

Step 4 Set the temperature of the flame to 938°C.

Step 5 Measure the length of the flame from the base of the gas burner to the tip of the flame.

Step 6 Program the ABB Robot to travel over the panel at the desired distance from the substrate.

Step 7 Set the speed of the robot in mm/s

Step 8 Clean the panel with Standox 11100 thinners to remove dirt and grease.

Step 9 Flame-treat the panel with the programmed characteristics.

Step 10 Iterate steps 6, 7, 8 and 9 until all values under investigation have been modelled.

Step 11 Paint all the panels in a white solid motor paint.

Step 12 Paint one control panel, untreated by the flaming process, in the same white solid motor paint.

Step 13 Wait three days for the paint on the panels to fully cure.

Step 14 Test the Adhesion of the paint with the adhesion testing machine.

Step 15 Record all test values in a table for later analysis.

Step 16 Analyse the results and derive the adhesion formula.

Step 17 Write the conclusion to the experiment.

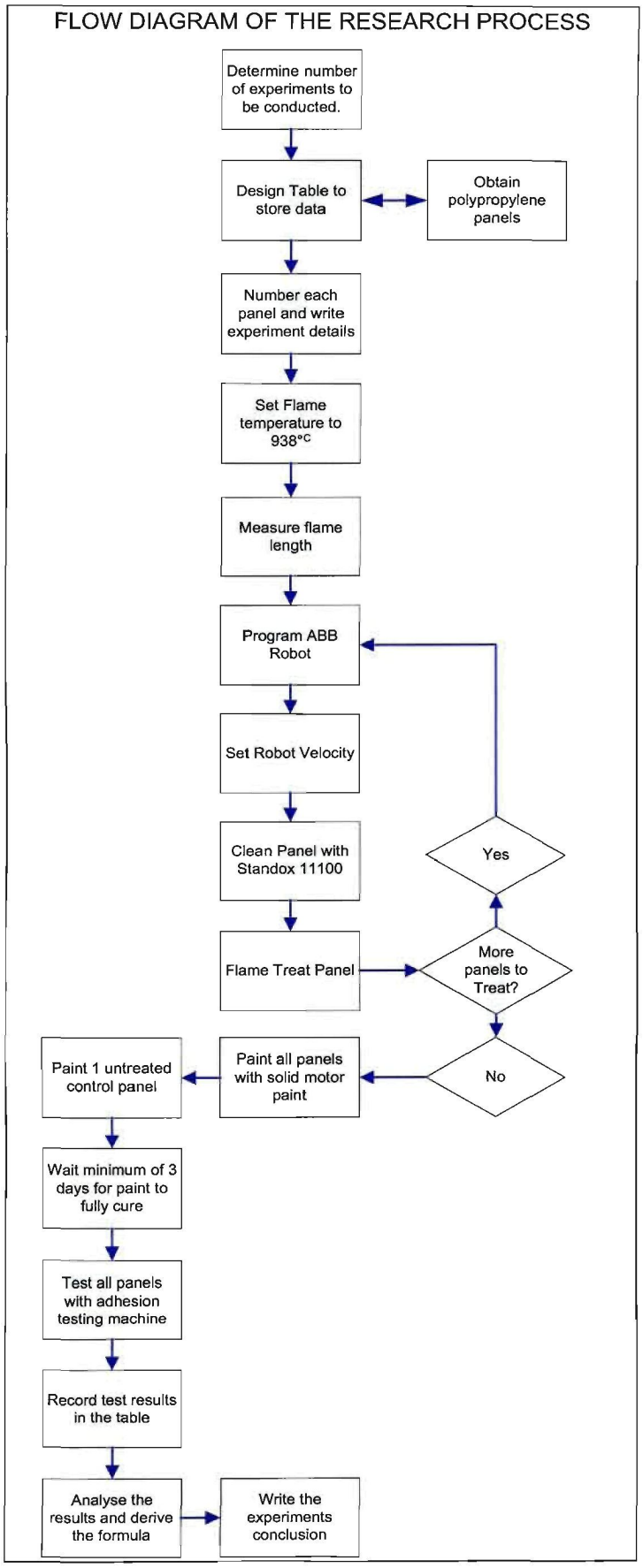


Figure 15 Research process flow diagram.

3.4.1 ADHESION TEST PROCEDURE

The adhesion tests will be conducted in the following logical order:

Step 1 Lightly scour the surface of the paint panel to be tested with P220 sandpaper.

Step 2 Clean both the panel and the test dumbbell with mentholated spirits.

Step 3 Glue the dumbbell onto the painted surface with SM 20 Methacrylate glue.

Step 4 Clean area around dumbbell before glue fully sets.

Step 5 Wait 20 minutes for the glue to fully cure.

Step 6 Cut the paint around the dumbbell to limit the area of the adhesion test.

Step 7 Load Adhesion Testing Program.

Step 8 Connect the Adhesion Testing Machine to the computers USB port.

Step 9 Slide the panel and dumbbell into the quick coupler on the testing machine.

Step 10 Click on the Connect button in the program.

Step 11 Turn the handle to pull the dumbbell off of the panel.

Step 12 Record the results from the program into an Excel spreadsheet.

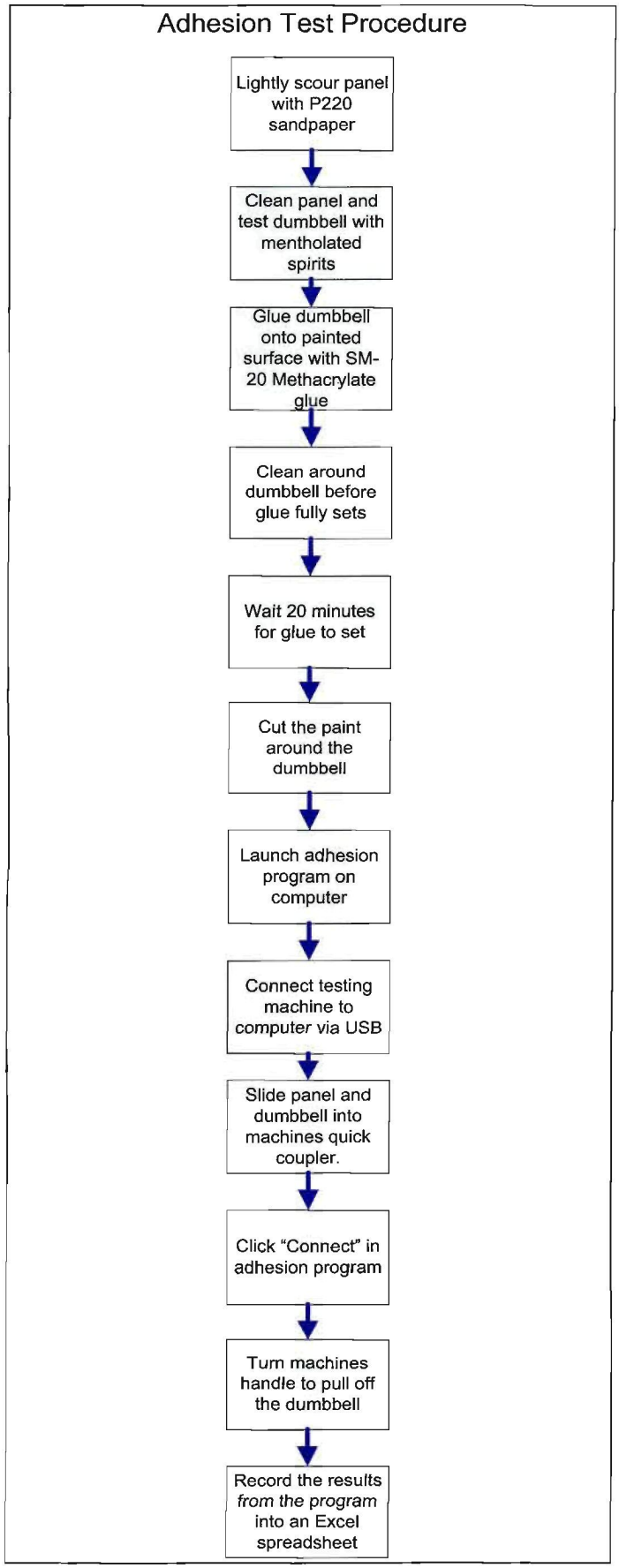


Figure 16 Adhesion Test Procedure Flow Diagram

3.5 TREATMENT OF THE DATA

3.5.1 Sub Problem 1 – Flame Velocity

The ideal speed of the flame travelling over the product must be determined.

3.5.1.1 NATURE OF THE DATA

The data required to evaluate the velocity of the flame as it passes over the substrate will be measured in mm/s. The ABB Robot uses mm/s as an input function for controlling the velocity of the robot. This value will be read directly from the control panel.

3.5.1.2 LOCATION OF THE DATA

All velocity readings will be obtained directly from the control panel of the robot. Data is displayed on the LCD Screen when viewing the program being executed.

3.5.1.3 MEANS TO OBTAIN THE DATA

The ABB Robot will be used to measure the velocity. The Robot has been calibrated and therefore the value entered into the program for the velocity will become the data required for the experiment.

3.5.1.4 TREATMENT OF THE DATA

The velocity will be varied, increasing with each new panel. Once all variations have been completed for the distance the process will repeat for the next distance. The panels will be painted in a solid white motor paint and left for three days to complete the cross linking process curing the paint fully. The panels will be tested using the adhesion testing machine to measure the adhesion force between the paint and the substrate. These results will be tabularised for later analysis.

3.5.1.5 REPORTING OF THE DATA

The results will be tabulated and then graphed to give a visual representation of the information gained from the experiment.

3.5.2 Sub Problem 2 – Flame Distance

The ideal distance of the flame from the product must be determined.

3.5.2.1 NATURE OF THE DATA

The distance of the flame and the substrate will be measured in millimeters. This measurement will be from the base of the gas burner to the top surface of the substrate.

3.5.2.2 LOCATION OF THE DATA

A predefined range of distances will be determined before the experiment begins. The distance will be confirmed during the experiment at the ABB Robot.

3.5.2.3 MEANS TO OBTAIN THE DATA

A calibrated Mitutoyo vernier will be used to set the distance of the gas burner from the substrate.

3.5.2.4 TREATMENT OF THE DATA

The distance will remain constant during the variations on the velocity. When the range of velocity has been modelled the distance will be increased to the next value and the process repeated. Once the entire range has been modelled the panels will be painted in a solid white motor paint and left for three days to fully cure the paint. The panels will then be tested to measure the adhesion force between the paint and the substrate. Results will be tabulated for later analysis.

3.5.2.5 REPORTING OF THE DATA

The results of the adhesion test will be tabulated and graphed giving a visual representation of the information gained from the experiment.

3.5.3 Sub Problem 3 – Adhesion Production Characteristics

The greatest velocity and distance must be determined to yield an adequate level of adhesion reducing cycle times and increasing output for production.

3.5.3.1 NATURE OF THE DATA

The velocity is measured in mm/s and the distance of the flame from the substrate in mm.

3.5.3.2 LOCATION OF THE DATA

The data will be found in the results of the experiment.

3.5.3.3 MEANS TO OBTAIN THE DATA

The graphs drawn will be used to find the values for the greatest velocity and distance still within an adequate adhesion strength range.

3.5.3.4 TREATMENT OF THE DATA

Not applicable

3.5.3.5 REPORTING OF THE DATA

The results will feature in the conclusion of the experiment.

4.0 CHAPTER 4 - RESULTS

4.1 RESULTS WITH RESPECT TO HYPOTHESIS 1

Flame treatment of polypropylene will increase the adhesion properties of the material.

Velocity	Adhesion At 30mm (kPa)	Adhesion At 50mm (kPa)	Adhesion At 70mm (kPa)	Adhesion At 90mm (kPa)	Adhesion At 110mm (kPa)
50 $mm.s^{-1}$	**Damage**	**Damage**	509.54	373.37	417.74
100 $mm.s^{-1}$	528.45	379.98	393.60	415.39	436.47
200 $mm.s^{-1}$	408.53	300.15	194.15	347.06	411.19
300 $mm.s^{-1}$	280.92	454.43	533.86	301.35	350.27
400 $mm.s^{-1}$	394.34	304.35	413.82	297.90	293.71
500 $mm.s^{-1}$	580.84	349.62	675.82	319.92	231.72
600 $mm.s^{-1}$	568.42	310.78	488.75	354.06	284.29
700 $mm.s^{-1}$	726.64	352.97	528.66	283.92	289.05
800 $mm.s^{-1}$	668.08	321.22	554.26	265.05	194.47
900 $mm.s^{-1}$	815.81	423.79	584.76	367.87	180.22
1000 $mm.s^{-1}$	639.52	361.22	357.34	243.37	181.78
1100 $mm.s^{-1}$	529.03	370.39	300.53	285.22	130.86

Table 2. 1 Data from adhesion tests on flame treated polypropylene panels.

Panel	Adhesion (kPa)
Untreated Panel	**0**

Table 2. 2 Data from untreated polypropylene panel.

The results were obtained by performing the experiment as set out in chapter 3. From tables 2.1 and 2.2 flame treatment of polypropylene has an affect on the adhesion properties of the material. The untreated panel had zero adhesion with the paint peeling off before the panel could be tested in the machine. Hypothesis 1 is therefore proven true.

4.2 RESULTS WITH RESPECT TO HYPOTHESIS 2

If the velocity of the flame traversing the surface of the substrate increase beyond the upper limit of the optimum setting the adhesion strength will decrease. Subsequently should the velocity decrease sufficiently substrate damage will occur rendering the product unusable.

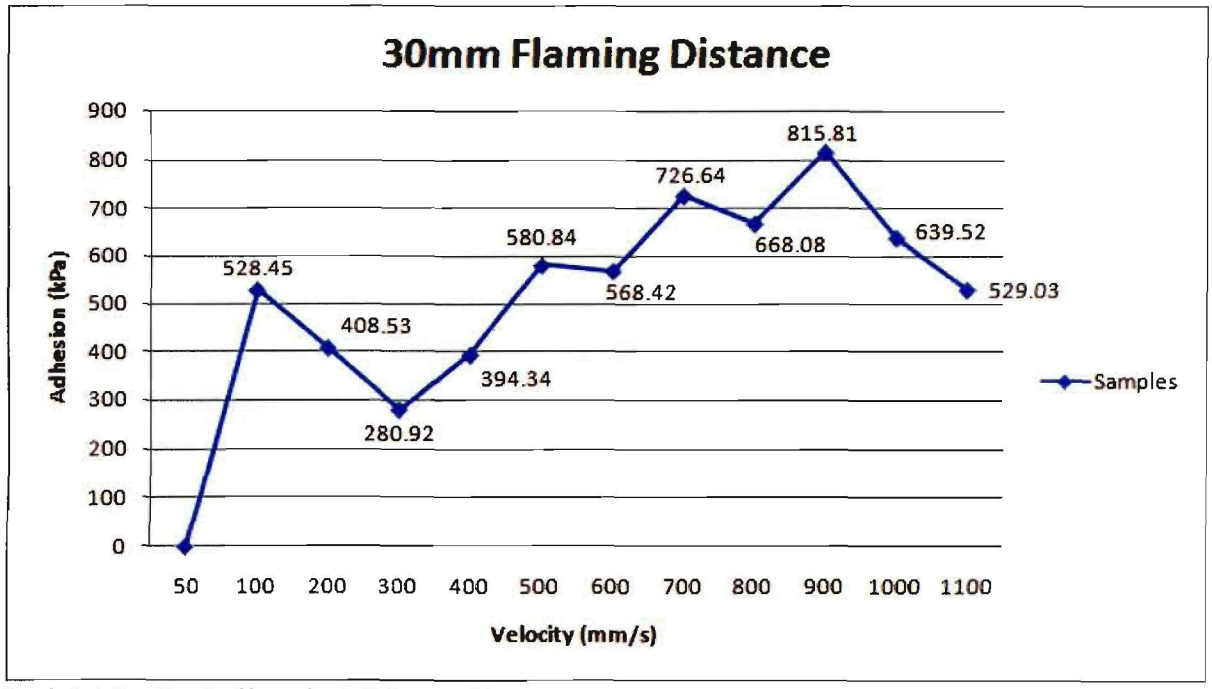
For reader convenience the data relating to this hypothesis has been repeated below.

Velocity	Adhesion (kPa) At 30mm
50 $mm.s^{-1}$	**Damage**
100 $mm.s^{-1}$	528.45
200 $mm.s^{-1}$	408.53
300 $mm.s^{-1}$	280.92
400 $mm.s^{-1}$	394.34
500 $mm.s^{-1}$	580.84
600 $mm.s^{-1}$	568.42
700 $mm.s^{-1}$	726.64
800 $mm.s^{-1}$	668.08
900 $mm.s^{-1}$	815.81
1000 $mm.s^{-1}$	639.52
1100 $mm.s^{-1}$	529.03

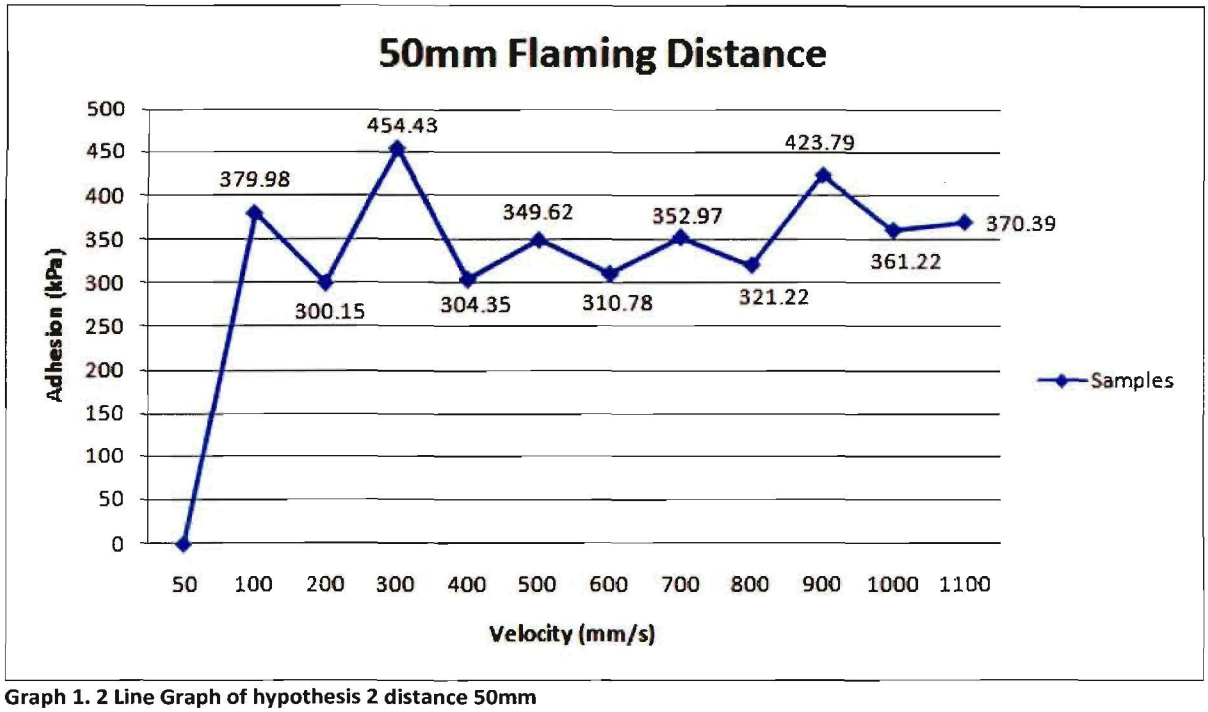
Table 3. 1 Adhesion at distance of 30mm proof of Hypothesis 2

Velocity	Adhesion (kPa) At 50mm
50 $mm.s^{-1}$	**Damage**
100 $mm.s^{-1}$	379.98
200 $mm.s^{-1}$	300.15
300 $mm.s^{-1}$	454.43
400 $mm.s^{-1}$	304.35
500 $mm.s^{-1}$	349.62
600 $mm.s^{-1}$	310.78
700 $mm.s^{-1}$	352.97
800 $mm.s^{-1}$	321.22
900 $mm.s^{-1}$	423.79
1000 $mm.s^{-1}$	361.22
1100 $mm.s^{-1}$	370.39

Table 3. 2 Adhesion at distance of 50mm proof of Hypothesis 2



Graph 1. 1 Line Graph of hypothesis 2 distance 30mm



Graph 1. 2 Line Graph of hypothesis 2 distance 50mm

The results were obtained by performing the experiment as set out in chapter 3. From the tables and the graphs the first hypothesis is proven true. When the velocity is too slow substrate damage occurs. This damage is in the form of warping and or burning of the product. As the velocity increases the adhesion increases. This increase in adhesion continues until a maximum point is reached. Further increase in the velocity results in a

decrease in adhesion until the flaming no longer has any effect. The decrease in adhesion occurs as a result of the flame passing over the substrate too fast for any chemical reaction to take place.

4.3 RESULTS WITH RESPECT TO HYPOTHESIS 3

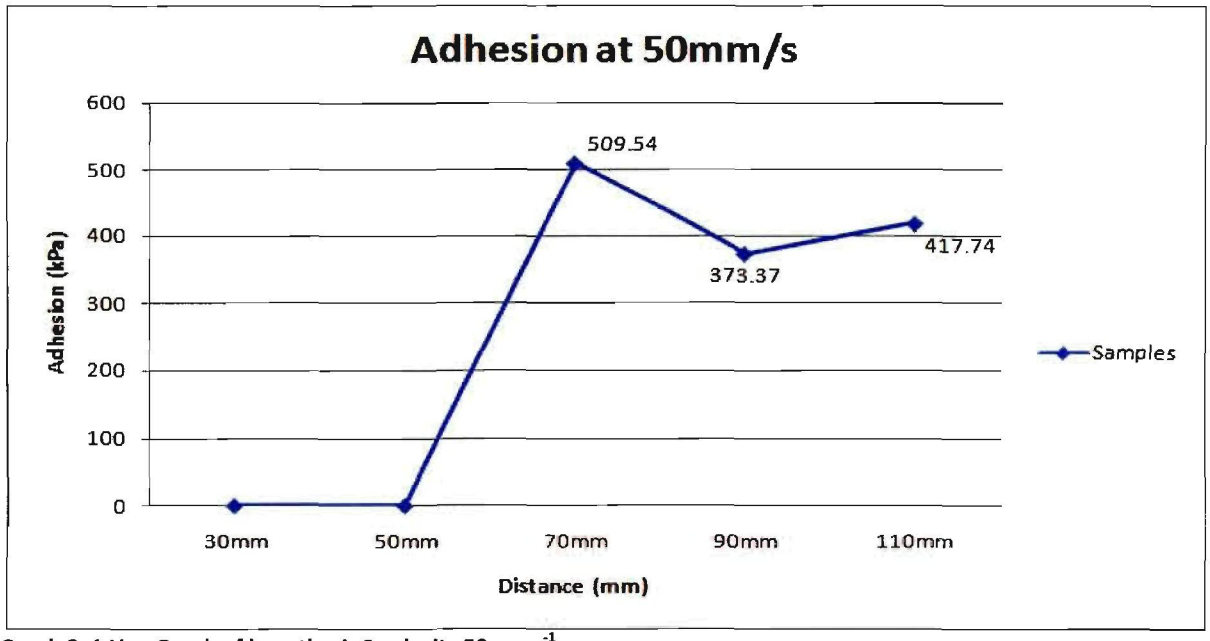
Should the distance of the flame and the substrate increases beyond the upper limit of the optimum setting the adhesion strength will decrease. If the distance is decreased below the lower limits, substrate damage may occur and the process repeatability may be compromised due to greater levels of accuracy required to prevent the gas head from colliding with the substrate. Both cases the hypothesis only holds true if there is no compensation in velocity to counter this effect.

Distance	Adhesion (kPa) at 50 $mm.s^{-1}$
30mm	0 Damage
50mm	0 Damage
70mm	509.54
90mm	373.37
110mm	417.74

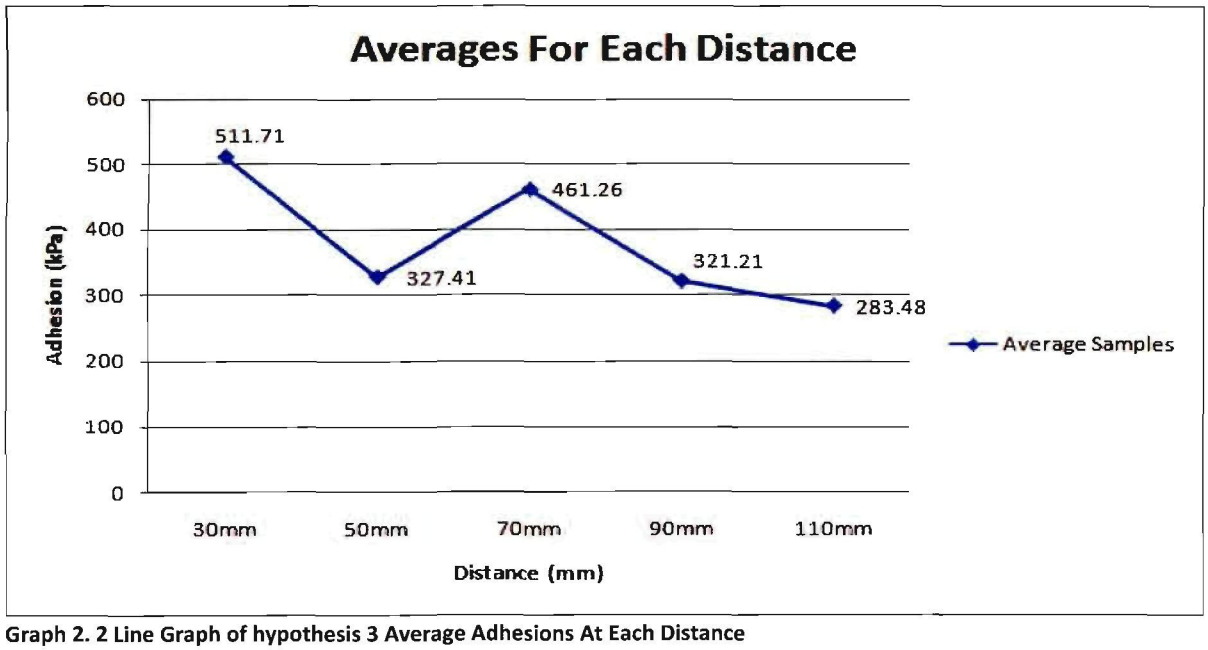
Table 4. 1 Adhesion at velocity of 50mm.s-1 proof of Hypothesis 3

Distance	Average Adhesion (kPa)
30mm	511.71
50mm	327.41
70mm	461.26
90mm	321.21
110mm	283.48

Table 4. 2 Average Adhesion velocity 50 mm.s-1 - 1100mm.s-1 proof Hypothesis 3



Graph 2. 1 Line Graph of hypothesis 3 velocity 50mm.s^{-1}



Graph 2. 2 Line Graph of hypothesis 3 Average Adhesions At Each Distance

Results were obtained by performing the experiment as detailed in chapter 3. The data for velocities of 50mm.s^{-1} and the average velocity over all the distances were chosen for analysis to highlight both extremes relating to the hypothesis. Data in table 4.1 with velocity 50mm.s^{-1} shows substrate damage at lower distances, 30mm and 50mm respectively. The damage was in the form of distortion. The damage is a result of overexposure to the high temperatures of the flame and the increased exposure time from the slow velocity.

Data in table 4.2 with the average velocity indicates a reduction in adhesion as the distance of the flame from the substrate increases. The substrate did not receive sufficient treatment to increase the adhesion properties to the levels achieved in lower distances.

4.4 RESULTS WITH RESPECT TO HYPOTHESIS 4

The system will be able to accurately repeat the process in a production environment.

Hypothesis 1, 2 and 3 have been proven true. Flame treatment has an affect on the adhesion properties of polypropylene. The velocity and the distance of the flame from the substrate control the amount of adhesion gained through flame treatment. From the literature survey the ABB robot has an accuracy of ±0.4mm. These robots are used in production system worldwide for tasks as complex as welding and assemblies. The ABB Robot can accurately control both the distance and the velocity of the flame proving hypothesis 4. With the use of a Robot the system is able to be accurately repeated in a production environment. The program loaded on the ABB robot for flame treating bumpers is in a format that can not be read by normal computers. This software can only be printed with the use of a specialised printer available for purchase from ABB Germany. The program is therefore not included in the dissertation but can be viewed on the Robot with prior arrangement with AK Stone Guards.

4.5 RESULTS WITH RESPECT TO HYPOTHESIS 5

There exists a narrow band of values in which a change in velocity and or distance from the substrate surface will not have any significant decrease in adhesion strength.

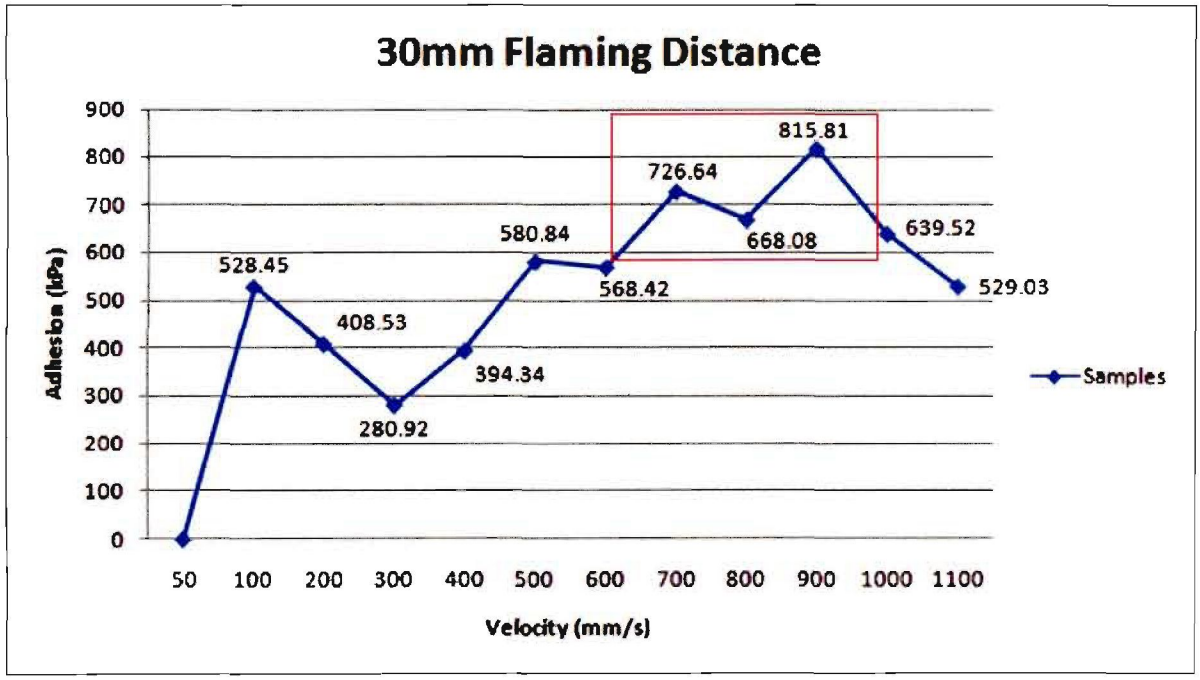
Velocity	Adhesion (kPa) at 30mm
50 $mm.s^{-1}$	Damage
100 $mm.s^{-1}$	528.45
200 $mm.s^{-1}$	408.53
300 $mm.s^{-1}$	280.92
400 $mm.s^{-1}$	394.34
500 $mm.s^{-1}$	580.84
600 $mm.s^{-1}$	568.42
700 $mm.s^{-1}$	726.64
800 $mm.s^{-1}$	668.08
900 $mm.s^{-1}$	815.81
1000 $mm.s^{-1}$	639.52
1100 $mm.s^{-1}$	529.03

Table 5. 1 Adhesion at distance of 30mm proof of hypothesis 5

Velocity	Adhesion (kPa) at 70mm
50 $mm.s^{-1}$	509.54
100 $mm.s^{-1}$	393.60
200 $mm.s^{-1}$	194.15
300 $mm.s^{-1}$	533.86
400 $mm.s^{-1}$	413.82
500 $mm.s^{-1}$	675.82
600 $mm.s^{-1}$	488.75
700 $mm.s^{-1}$	528.66
800 $mm.s^{-1}$	554.26
900 $mm.s^{-1}$	584.76
1000 $mm.s^{-1}$	357.34
1100 $mm.s^{-1}$	300.53

Table 5. 2 Adhesion at distance of 70mm proof of hypothesis 5

For reader convenience the data relating to this hypothesis has been repeated above.



Graph 3. 1 Line Graph of hypothesis 5 at distance of 30mm

Graph 3. 2 Line Graph of hypothesis 5 at distance of 70mm

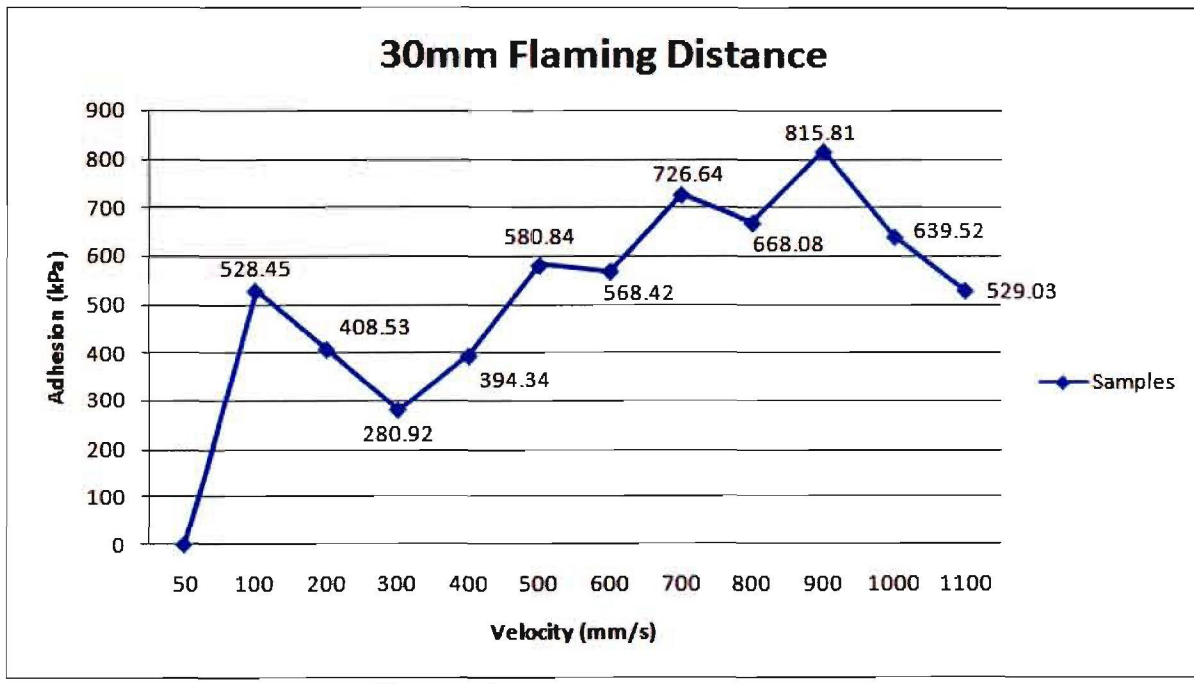
Results were obtained by performing the experiment as detailed in chapter 3. Both graphs 3.1 and 3.2 highlight a band of velocity where no substantial change in adhesion occurred. At 30mm the range of velocity includes 700mm.s^{-1} to 900mm.s^{-1}. When the distance is set to 70mm the band is increased from 500mm.s^{-1} to 900mm.s^{-1}. This confirms hypothesis 5 showing a range of velocity exists where any change will not significantly reduce the adhesion.

5.0 CHAPTER 5 - INTERPRETATION

From the data presented in Chapter 4 the velocity of the flame and the distance of the flame from the substrate has an affect on the adhesion properties of the polypropylene. Un-flamed panels showed zero adhesion properties proving that flame treatment has an influence on the adhesion of polypropylene. The data shows how the adhesion changes with the increase or decrease in the velocity or distance and these changes are not linear. The nonlinearity of the data is due in part to the slight changes in the plastic panel's chemical characteristics and the properties of the flame at various zones.

The flame produced from a gas head can be divided into 2 zones. An oxidising zone and a reducing zone, depending on the zone the panel was passing through during the experiment the data reflected the change in adhesion. The change was not linear as a result of the properties of the zone and the amount of treatment received.

The experiment showed that the oxidising zone of the flame has an influence on the adhesion properties and that a panel can be both over and under-treated. Over and under-treating of a panel will result in the same decrease in adhesion. This is supported by the following graph, graph 4.1:

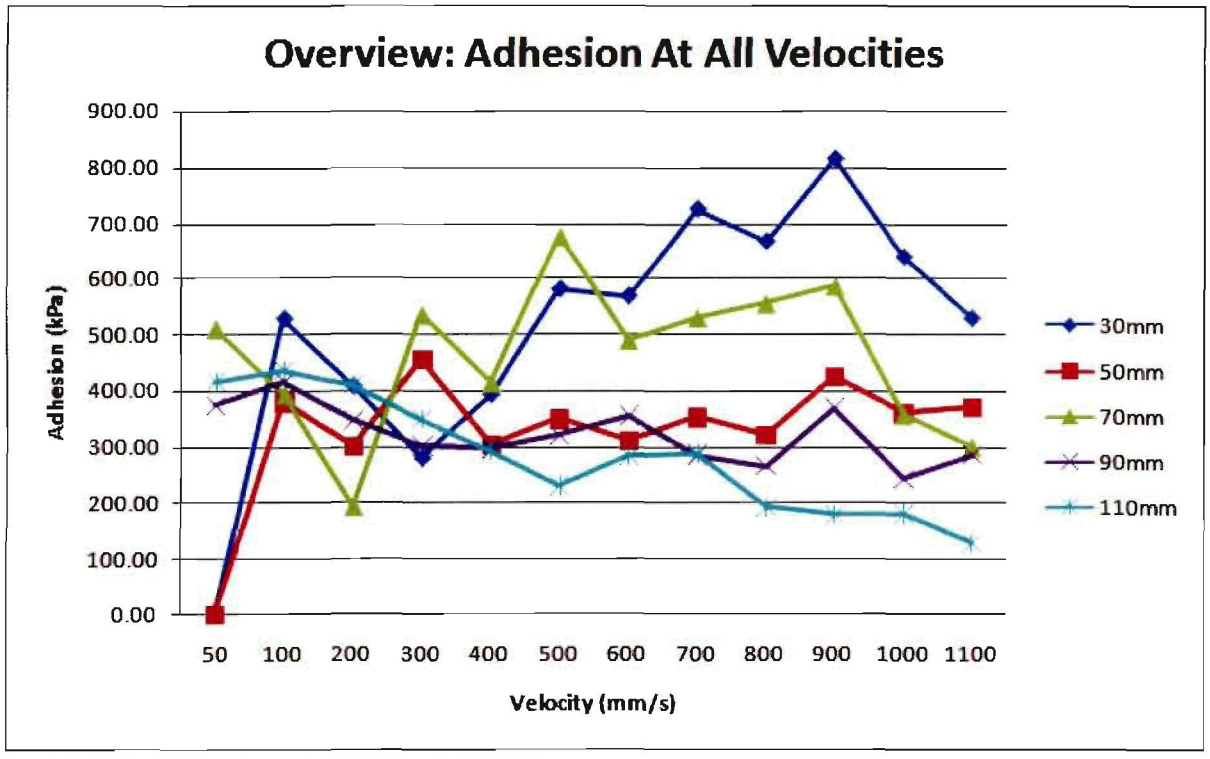


Graph 4. 1 Under and over-treating effects on adhesion

At a distance of 30mm and velocity of 100mm.s^{-1} the adhesion was not at an optimum level, 528.45kPa, when the velocity increased to 1100mm.s^{-1} the adhesion fell to about the same

point 529.03kPa. The panel was over-treated at the lowest velocity reducing the affect of flaming and when under-treated the same result was obtained.

As the flame was moved further from the panel the adhesion at certain velocities began to increase. The increase in adhesion can be attributed to the characteristics of the treatment process approaching optimum levels for velocity and distance. The adhesion became stable over a larger range of velocities reducing the affect of the variations in velocity. Graph 4.2 shows the adhesion at all distances through the range of velocities.



Graph 4. 2 All velocities through the range of distances

A distance of 70mm from the panel yielded the best results for a production environment. The adhesion remained more stable over a wider range of velocities. As the distance increased from 70mm the adhesion began to decrease with an increase in velocity and the range of optimum values reduced in size. At distances of 110mm or greater the adhesion begins to fall to zero for all increases in velocity.

The reduction in adhesion from this point forward can be attributed to under treating of the panels. The panel is not in the ideal point within the flames zone and the flame is passing too quickly over the panel. The increase in velocity reduced the amount of chemical reaction

taking place resulting in lower adhesions. As the speed increases the adhesion starts declining. At this point the adhesion is indirectly proportional to the velocity of the flame.

From the data presented above it can be stated that further increase in the distance of the flame will result in the adhesion falling to zero for all velocities. The panel will be outside of the flame and no treatment will occur. The relationship between adhesion and distance will change to indirectly proportional and no change in velocity will affect the adhesion properties.

The data from the experiment has yielded three formulas to calculate the adhesion at various distances and velocities. If the distance is between 30mm and 50mm formula 1 can be applied. At 70mm formula 2 is applied and greater then 80mm formula 3 holds true. Formula 3 has one exception to the rule, at a distance of 110mm the calculation for the adhesion changes from A=f(x) + f(x-1) x 1.41471 to A=f(x) + f(x+1) x 1.41471. The formula was broken into three to handle the dynamic changing and nonlinearity of the data.

Formula 1:
A = Adhesion
x = velocity
d = distance
f(x) = Change in Adhesion
$30 \leq d \leq 50$

$$f(x) = \left((Ex^2 + Fx + G) + \left((Hx^3 + Ix^2 + Jx + K) \sin\left(\left(\frac{2\pi}{210} \right) (1.05x - 52.5) \right) \right) \right)$$

$$E = 9.5 \times 10^{-6} d - 485 \times 10^{-6}$$

$$F = -18.775 \times 10^{-3} d + 0.96535$$

$$G = 3.6601d - 68.715$$

$$H = 1.952 \times 10^{-6} d - 60.13 \times 10^{-6}$$

$$I = -5.08 \times 10^{-3} d + 0.1557$$

$$J = 137.475 \times 10^{-3} d - 5.869$$

$$K = -20.725d + 1027.765$$

$$x \in (100,200,300,400,500,600,700,800,900,1000,1100)$$
$$A = f(x) + f(x-1) \times 1414.71$$

Formula 2:

A = Adhesion
x = velocity
d = distance
f(x) = Change in Adhesion
d = 70

$$f(x) = \left(B + (C \times \sin\left(\left(\frac{2\pi}{210}\right)(1.05x - 52.5)\right)\right)$$

$$B = -5.5 \times 10^{-4}x^2 + 654.6 \times 10^{-3}x + 26.091$$

$$C = 9.44 \times 10^{-8}x^3 - 7.71 \times 10^{-4}x^2 + 1.3013x - 21.509$$

$x \in (100,200,300,400,500,600,700,800,900,1000,1100)$
$A = f(x) + f(x-1) \times 1414.71$

Formula 3:

A = Adhesion
x = velocity
d = distance
f(x) = Change in Adhesion
$90 \leq d \leq 110$

$$f(x) = \left((Bx + C) + \left((Dx + E) \times \left(\sin\left(\frac{2\pi}{210}\right)(1.05x - 52.5)\right)\right)\right)$$

Where:

$$B = -2.855 \times 10^{-3}d + 0.21065$$

$$C = 1.3855d + 11.295$$

$$D = 1.285 \times 10^{-3}d + 0.12615$$

$$E = -0.563d + 220.92$$

$x \in (100,200,300,400,500,600,700,800,900,1000,1100)$

When d < 100

$$A = f(x) + f(x-1) \times 1414.71$$

When 100 >= d <= 110

$$A = f(x) + f(x+1) \times 1414.71$$

The formulas listed above approximate the data from the experiment and gives the change in velocity between two elements in the velocity data set.

Example, if the adhesion is required at a velocity of 700mm.s^{-1} at a distance of 30mm.

x = 700
d = 30.
d ≥ 30
Applying Formula 1 between data set 600 and 700 mm.s^{-1}:

$$f(x) = \left((Ex^2 + Fx + G) + \left((Hx^3 + Ix^2 + Jx + K) \sin\left(\left(\frac{2\pi}{210} \right) (1.05x - 52.5) \right) \right) \right)$$

$$E = 9.5 \times 10^{-6} d - 485 \times 10^{-6}$$

$$F = -18.775 \times 10^{-3} d + 0.96535$$

$$G = 3.6601d - 68.715$$

$$H = 1.952 \times 10^{-6} d - 60.13 \times 10^{-6}$$

$$I = -5.08 \times 10^{-3} d + 0.1557$$

$$J = 137.475 \times 10^{-3} d - 5.869$$

$$K = -20.725d + 1027.765$$

$$x \in (100,200,300,400,500,600,700,800,900,1000,1100)$$

$$A = f(x) + f(x-1) \times 1.41471$$

$$E = 9.5 \times 10^{-6}(30) - 485 \times 10^{-6}$$

$$\underline{\boldsymbol{E = -0.2 \times 10^{-3}}}$$

$$F = -18.755 \times 10^{-3}(30) + 0.96535$$

$$\underline{\boldsymbol{F = 0.4027}}$$

$$G = 3.6601(30) - 68.715$$

$$\underline{\boldsymbol{G = 41.088}}$$

$$H = 1.952 \times 10^{-6}(30) - 60.13 \times 10^{-6}$$

$$\underline{\boldsymbol{H = -1.57 \times 10^{-6}}}$$

$$I = -5.08 \times 10^{-3}(30) + 0.1557$$

$$\underline{\boldsymbol{I = 0.0033}}$$

$$J = 137.475 \times 10^{-3}(30) - 5.869$$

$$\underline{\boldsymbol{J = -1.74475}}$$

$$K = -20.725(30) + 1027.765$$

$$\underline{\boldsymbol{K = 406.015}}$$

$$f(700) = \Bigg((-0.2 \times 10^{-3}(700)^2 + 0.4027(700) + 41.088) + \big((-1.57 \times 10^{-6}(700)^3) + 0.0033(700)^2 - 1.74475(700) + 406.015\big)\left(\sin\frac{2\pi}{210}(1.05(700) - 52.5)\right)\Bigg)$$

$$\underline{\boldsymbol{f(700) = 488.158}}$$

At f(x) = 600
F(600) = 3.172

$$A = f(x) + f(x-1) \times 1414.71$$

A = (488.158 + 2.663) x 1414.71

$\underline{A_{(700)} = 694.37\text{kPa}}$

From the data the adhesion at 700mm.s^{-1} is 694.37kPa. We have a less than 10% error between the data and the formula at 4.34%. This error is due to the nonlinearity of the data.

6.0 CHAPTER 6 - CONCLUSION

Flame treatment of polypropylene plastics has an affect on the adhesion properties of the material. Without flame treatment the plastic showed no adhesion between the substrate and the paint applied to the surface. Both the velocity of the flame travelling over the plastic and the distance of the flame from the surface had an affect on the adhesion properties. It was shown that over-treating the surface by flaming too close to the plastic or at speeds too slow had a negative impact on the adhesion properties.

It was also shown that if the surface was under-treated by either moving too fast over the surface or at great distances the adhesion was less than optimal. From the research it can be concluded that flaming is dependant on the velocity and the distance of the flame to yield the greatest adhesion.

The research has shown that the flaming process is not very sensitive as a region of velocity and distances exists where small changes will not significantly reduce the adhesion properties of the material. A formula was derived by the researcher to calculate the adhesion at a given velocity and distance.

The most optimum distance for production purposes was at 70mm from the substrate. The adhesion remains constant over a larger set of velocities making this more stable in a production environment. A Distance of 30mm yielded the greatest adhesion strength at $900mm.s^{-1}$ with an adhesion of 815.81kPa.

There is no established standard for the value of adhesion strength for paint and must be determined by the user of the various painting systems (Phil, 2003). Different paint systems sold will result in different adhesion values on products that have been flamed. The maximum adhesion at 30mm, $900mm.s^{-1}$ is with respect motor industry paint on bumper grade polypropylene. This value has importance to AK Stone Guards as this is the system they are currently using. The derived formula for adhesion strength should hold true for similar systems.

6.1 FURTHER RESEARCH AREAS

The study of flame treatment of polypropylene can be taken further to investigate the effects of temperature of the flame on the adhesion properties. The stoichiometry of the

flame and the effects on adhesion is a topic for further research. The time between flaming and painting can be studied to determine if the adhesion strength deteriorates over time. The chemistry of the flame treatment process can yield a study on it own. This study will investigate the reasons for the increase in adhesion and the chemical process resulting in the surface modification.

The experiment should be repeated several times with the parameters used in this study to improve on the results. The average values of the experimental data may then be applied to further refine the formulas and reduce intrinsic variation.

Other factors influencing the changes in adhesion can be investigated. This research will first aim to discover if any other factors exist influencing the adhesion and secondly what is the value of the influence caused by these factors.

This research study can itself be expanded further. Time restraints and the scope of the Masters degree prevented the further study of this subject. The data gained through the experiments have suggested that a better mathematical explanation can be concluded. The adhesion is a result of the temperature of the flame, velocity and the distance of the flame from the polypropylene. All three areas contribute to the amount of energy transferred to the polypropylene panel. The energy is required to form the bonds causing the hydroxyl, carboxyl and carbonyl groups as well as the molecular chain scission and cross-linking. The amount of energy the panel receives is therefore proportional to the increase in adhesion of the panel from the treatment process.

Relating the three variables to the energy change on the panel at the time of flaming can result in a second order differential equation. This will better suit the theory of the flaming process where at lower speeds and distances the panel undergoes over treatment resulting in lower adhesion levels. This adhesion will then steadily increase to a maximum point before falling to zero again. The fall in adhesion is now a result of under treatment from excessive speeds and distances.

The hypothesis indicates that the system will have a steady state at zero. Any increase in either the velocity or the distance will yield no change in the adhesion making it steady. The gradual increase from zero adhesion strength to the maximum value will make up the

transient portion of the hypothesis. A second order differential equation can be used to calculate the transient state equation Y_{CF} and the particular integral Y_{PI} for use in adhesion calculations.

Second order differential equations are used to model systems with transient and steady states, the voltage in an electronic circuit comprised of a capacitor, resistor and an inductor is an example of this use. The voltage on the capacitor for example will steadily increase until the capacitor is fully charged, where the capacitor is now at its steady state.

To form the second order differential equation a relationship of the order below is required.

$$\pm w\frac{d^2e}{dt^2} \pm x\frac{de}{dt} \pm y(e) = z$$

w, x, y and z are constants.
e = energy
t = time.

Time is used in the equation as the amount of energy received is dependant on the time of exposure to the flame.

Some relationship must be found between either the temperature of the flame, velocity or distance where the second derivative will result in the energy received by the panel. Another relationship must be found between these variables where the first derivative will result in the energy. The last relationship will be where the energy is directly related to the one of the variables.

The result of this should be an empirical formula describing the adhesion at any distance, velocity and temperature and the change in the adhesion between two intervals.

In conclusion this topic has many avenues yet to be researched on levels for both Masters and Doctorial candidates. Flame treatment of polypropylene is a very research rich and interesting field of study. Results from any further research will benefit industry into the future.

7.0 CHAPTER 7 – REFERENCES

BIBLIOGRAPHY

a, S., a, E. S., a, D. M., & Heath, R. J. (1994, October 2). *The Journal of Adhesion* . Retrieved August 12, 2008, from informaworld: http://www.informaworld.com/smpp/content~content=a757237638~db=all

AK Stone Guards. (2008). *AK Stone Guards*. Retrieved August 5, 2008, from AK Sport: http://www.aksport.co.za/ak_stoneguards.php

American Chemistry Council. (2007). *Plastics: The Basics of Polymer Chemistry*. Retrieved April 29, 2008, from AmericanChemistry: http://www.americanchemistry.com/s_plastics/doc.asp?CID=1102&DID=4664

Arizona State University. (n.d.). *The A-B-C's of the Generic Sine Curve y = Asin(Bx+c)*. Retrieved October 31, 2008, from Arizona State University: http://math.la.asu.edu/~walker/mat170/homework/sine_curves.htm

Australian Combustion Services . (2004). *PRETREATMENT OF POLYOLEFINS*. Retrieved April 29, 2008, from Australian Combustion Services : http://www.combust.com.au/plastic/polyol.htm

Bourne, M. (2008, June 30). *1. Graphs of y = a sin x and y = a cos x*. Retrieved October 31, 2008, from Iinteractive Mathematics: http://www.intmath.com/Trigonometric-graphs/1_Graphs-sine-cosine-amplitude.php

Bourne, M. (2008, February 10). *Composite Trigonimetric Curves*. Retrieved October 31, 2008, from Interactive Mathematics: http://www.intmath.com/Trigonometric-graphs/6_Composite-trigonometric-graphs.php

Cain, R. (2000, April). *Status Report on Polypropylene Homopolymer* . Retrieved April 28, 2008, from Warwick Manufacturing: http://www.warwick.ac.uk/atc/materials/PAAS/PPreport/PPreport.html

Cain, R. (2007). *The Lectro – Heat Blown Air Plasma System for Polypropylene*. Retrieved April 22, 2008, from Azom.com: http://www.azom.com/details.asp?ArticleID=1523

Chemical of the Week. (n.d.). Retrieved April 30, 2008, from http://scifun.chem.wisc.edu/CHEMWEEK/POLYMERS/Polymers.html

Columbia University Press. (2001, July). *The Columbia Encyclopedia, Sixth Edition*. Retrieved August 18, 2008, from Bartleby.com: http://www.bartleby.com/65/fl/flame.html

DeFelsko. (2004). *Measurement of Adhesion Strength*. Retrieved April 20, 2008, from DeFelsko: http://www.defelsko.com/applications/adhesion_strength/adhesion-testing.htm

Drotsky, J. (2005). *Strength of Materials for Technicians Third Edition.* Sandton: Heinemann Publishers (Pty) Ltd.

Ebberts, D. (2006). *Simulation*. Retrieved October 31, 2008, from MotionScript.com: http://www.motionscript.com/mastering-expressions/simulation-basics-1.html

Eckert, W. (2004). *Improvement of adhesion on polymer film, foil and paperboard by flame treatment.* Retrieved April 27, 2008, from Arcogas: www.tappi.org/content/enewsletters/eplace/2004/7-5arcogas.pdf

Fisher, L. W. (2006, December 1). *The secret to strong, reliable adhesive bonding is to start with good clean surfaces*. Retrieved April 28, 2008, from Better bonds : http://machinedesign.com/ContentItem/59182/Betterbonds.aspx

Garver, L. (2006). *Lindsay Garver-Couch Potato*. Retrieved April 29, 2008, from Materials Science & Chemistry: http://www.ccmr.cornell.edu/education/ret/jims/LindsG.html

Grant, E. (2004). *Polymers Laminates Adhesives Coatings & Extrusions. All Of These Can Benefit In Some Way From Flame Treatment*. Retrieved April 20, 2008, from http://www.tappi.org/content/enewsletters/eplace/2004/16-2Grant.pdf

Jing Songa, U. G. (2007, June 8). *Flame treatment of low-density polyethylene: Surface chemistry across the length scales*. Retrieved August 18, 2008, from ScienceDirect: http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6THY-4P0X56G-3&_user=10&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_version=1&_urlVersion=0&_userid=10&md5=858d66fdc4f9b912e2b7d5a8a77c2d86

Kaplan, E. F. (2007). *Surface Treatment*. Retrieved April 29, 2008, from Plasma Technology Systems: http://www.plasmatechsystems.com/pubs/surfacetreatment.asp

KEIKO(, W. (1997). *Application of flame treatment for degreasing aluminum foil.* Retrieved April 22, 2008, from Science Links Japan: http://sciencelinks.jp/j-east/article/200003/000020000399A1043023.php

Kostyk, D. B. (2000, June 15). *Bonding low surface energy plastics* . Retrieved April 30, 2008, from Machine Design: http://machinedesign.com/ContentItem/70562/Bondinglowsurfaceenergyplastics.aspx

Lenntech. (2007). *What is Polypropylene?* Retrieved April 29, 2008, from Lenntech: http://www.lenntech.com/polypropylene.htm

M.D. Green, F. G. (2000, February 23). *Characterisation & Comparison of Surface Modification on Homopolymer Polypropylene*. Retrieved April 22, 2008, from Department of Mechanical Engineering, University of Bristol: http://www.warwick.ac.uk/atc/materials/PAAS/USAconference/conference.html

Master Bond Inc. (2007). *BONDING POLYETHYLENE, POLYPROPYLENE, THEIR COPOLYMERS & ALLOYS* . Retrieved April 29, 2008, from Master Bond Inc: http://www.masterbond.com/bssp/bspolye.html

Nolan, M. (2000, February 29). *Flame Treatment – Corona's Poor Cousin?* Retrieved April 30, 2008, from Sherman Treaters North America: http://www.shermantreaters.co.uk/acrobat/flame.pdf

Petrie, E. M. (2006). *Handbook of Adhesives and Sealants 2nd Edition*. Retrieved August 13, 2008, from Google Books: http://books.google.com/books?id=5Tv_MX9OLyMC&pg=PA262&lpg=PA262&dq=Chemistry+of+Polypropylene+Flame+Treatment&source=web&ots=X5OgQ2XzK3&sig=YRzGqovc7X2vv3K0rW9kwNWMiUk&hl=en&sa=X&oi=book_result&resnum=1&ct=result#PPA262,M1

Phil. (2003, December). *Testing adhesion properties of a finish coating will ensure a successful performance from the product.* Retrieved October 31, 2008, from Finish coatings system adhesion and test methods:

http://www.defelsko.com/applications/adhesion_strength/WoodDigestsFinishing-Adhesion1203.pdf

RobotWorx. (n.d.). *ABB IRB 6000 Robot*. Retrieved August 18, 2008, from RobotWorx: http://www.robots.com/abb.php?robot=irb+6000

RoyMech. (2008, April 30). *Friction Factors*. Retrieved August 19, 2008, from RoyMech: http://www.roymech.co.uk/Useful_Tables/Tribology/co_of_frict.htm#Power_Screws

Roymech. (2008, May 29). *Power Screws*. Retrieved August 19, 2008, from RoyMech: http://www.roymech.co.uk/Useful_Tables/Cams_Springs/Power_Screws_1.html

Ryton. (2004, July 16). Retrieved April 30, 2008, from Ryton Polyphenylene Sulfide Resins: http://www.cpchem.com/enu/docs_ryton/TSM283.pdf

Sabreen, J. D. (2005, October). *Flame Plasma Surface Treatment Improves Adhesion of Polymers*. Retrieved April 28, 2008, from Plastics Decorating Magazine: http://plasticsdecorating.com/articlesdisplay.asp?ID=53

Scribd. (2008). *Carbon Compound I* . Retrieved August 18, 2008, from Scribd: http://www.scribd.com/doc/2479450/Carbon-Compound-I

Statistics New Zealand. (2008, October 14). *Residuals and R squared - teachers page*. Retrieved October 31, 2008, from Statistics New Zealand: http://www.stats.govt.nz/schools-corner/secondary/teachers/residuals_and_rsquared_for_teachers.htm

Technologies, f. (2008). *Flame Treatment*. Retrieved April 22, 2008, from fts Technologies: http://www.ftstechnologies.com/flame.htm

Widener University. (n.d.). *Linear Regression*. Retrieved October 31, 2008, from Widener University: http://science.widener.edu/svb/stats/regress.html

Wikipedia. (2008, April 30). *Polypropylene*. Retrieved April 30, 2008, from Wikipedia: http://en.wikipedia.org/wiki/Polypropylene

Wolf, R. A. (n.d.). *Corona Treatment: A Process Overview*. Retrieved August 18, 2008, from IDS Packaging :

http://www.idspackaging.com/Common/Paper/Paper_177/Corona%20Treatment-%20A%20Process%20Overview.htm

Wolf, R. A. (2004, June 9). *SURFACE ACTIVATION SYSTEMS FOR OPTIMIZING ADHESION TO POLYMERS*. Retrieved April 30, 2008, from Enercon Industries Corporation: http://www.enerconind.com/files/7f/7fec8d62-6caa-47c0-882f-800158cddf7f.pdf

APPENDIX-A DERIVATION OF ADHESION FORMULA

A.1. DETERMINING THE TYPE OF GRAPH

To calculate the formulas, the raw data was used from the testing machine in Newtons. The formula was derived and a multiplicand added to convert the results of the formula from Newtons to Pascals.

Unconverted Data taken at 110mm From Testing Machine:

Velocity	Distance at 110mm (N)
100 $mm.s^{-1}$	308.52
200 $mm.s^{-1}$	290.65
300 $mm.s^{-1}$	247.59
400 $mm.s^{-1}$	207.61
500 $mm.s^{-1}$	163.79
600 $mm.s^{-1}$	200.95
700 $mm.s^{-1}$	204.32
800 $mm.s^{-1}$	137.46
900 $mm.s^{-1}$	127.39
1000 $mm.s^{-1}$	128.49
1100 $mm.s^{-1}$	92.50

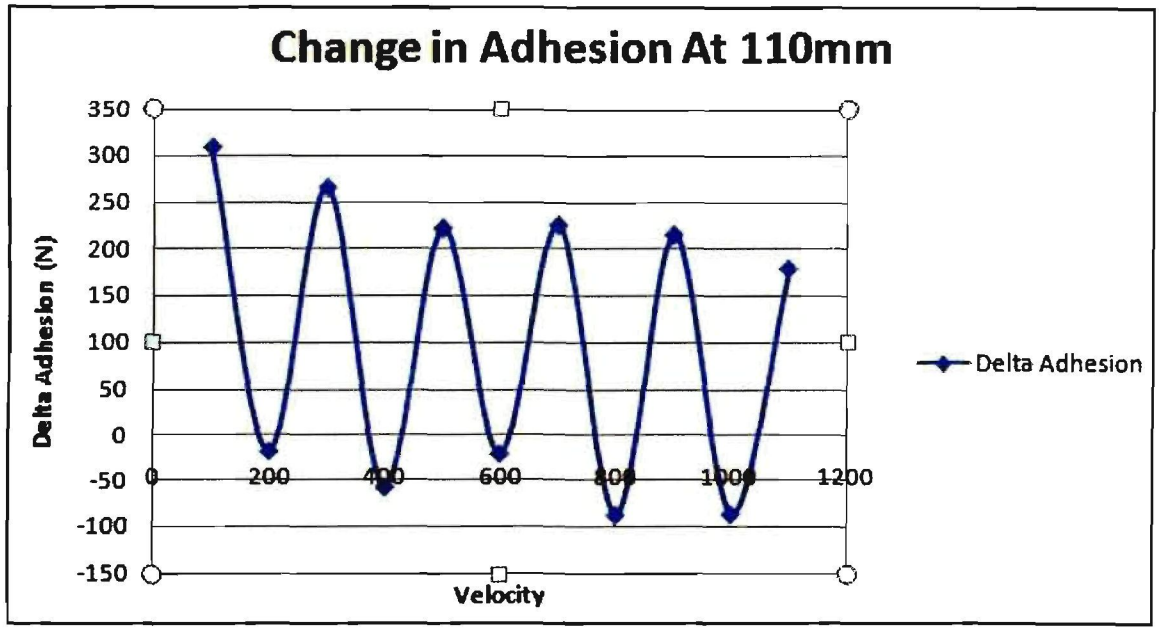
Table 6. 1 Original experimental data at 110mm

The change in adhesion was calculated by subtracting the adhesion at x from the difference of adhesion at x-1.

Velocity	Change in adhesion at 110mm (N)
100 $mm.s^{-1}$	308.52
200 $mm.s^{-1}$	-17.86
300 $mm.s^{-1}$	265.46
400 $mm.s^{-1}$	-57.85
500 $mm.s^{-1}$	221.64
600 $mm.s^{-1}$	-20.69
700 $mm.s^{-1}$	225.01
800 $mm.s^{-1}$	-87.55
900 $mm.s^{-1}$	214.94
1000 $mm.s^{-1}$	-86.45
1100 $mm.s^{-1}$	178.96

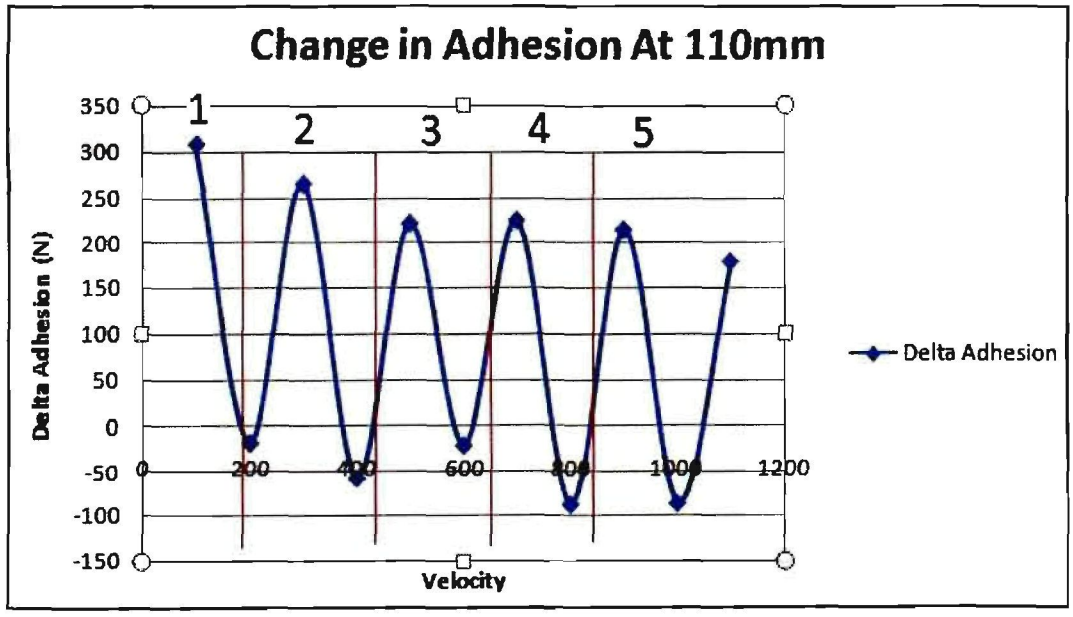
Table 6. 2 Change in adhesion at 110mm

The change in adhesion was graphed to see the resulting graphic. This is important in defining a formula to describe the data.



Graph 5. 1 Delta adhesion at 110mm

From graph 5.1 showing the change in adhesion the data resembles a sinusoidal graph with growth and a positive phase shift. Closer examination shows the graph makes 5 cycles over the velocity range from 50mm.s^{-1} to 1100mm.s^{-1}.



Graph 5. 2 Delta adhesion sinusoidal graph sine cycles.

The amplitude of the sin graph is the energy of the graph (Bourne, 1. Graphs of y = a sin x and y = a cos x, 2008). The amplitude is the height of the graph from 0 to $\pi/2$. The amplitude for the one half of the cycle is equal to that of the other half of the cycle. The amplitude is always positive.

To calculate the amplitude of the graph in 5.1 add the absolute values of the peaks and the troughs of each cycle 1 to 5. Table 6.3 shows the peaks and troughs of each cycle. Table 6.4 represents the sum of each period and the amplitude of the periods. The amplitude is half the sum of the absolute values of the peaks and troughs.

A.2. AMPLITUDE OF THE GRAPH

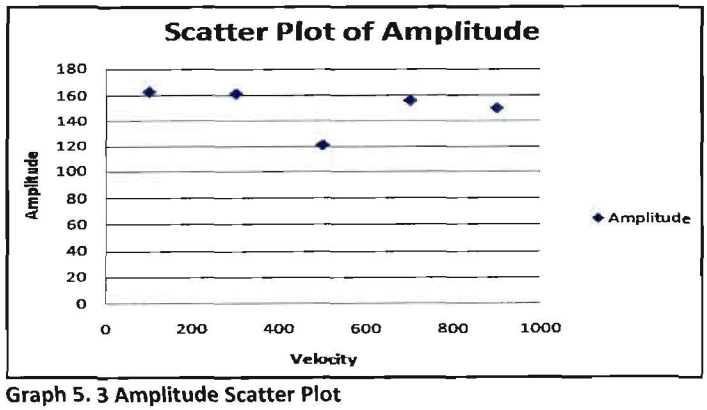
Velocity mm.s^{-1}	Peak	Velocity mm.s^{-1}	Trough
100	308.52	200	-17.86
300	265.46	400	-57.85
500	221.64	600	-20.69
700	225.01	800	-87.55
900	214.94	1000	-86.45
1100	178.96		

Table 6. 3 Peaks and troughs of the cycles in the delta sin graph

Cycle Number	Peak Trough Sum	Amplitude
1	326.38	163.19
2	323.32	161.66
3	242.34	121.17
4	312.57	156.28
5	301.40	150.70

Table 6. 4 Peak and trough sum and amplitude for each cycle

Drawing a scatter chart, graph 5.3, of the sum of the peaks and troughs show the data has an upwards trend. The data has to be regressed to be used.



Graph 5. 3 Amplitude Scatter Plot

The regression technique was based on the scatter plot of the amplitude. In this case a linear regression was used, for others a cubic regression was more appropriate. The formulas to calculate linear regression are taken from Widener University (Widener University):

$$s_{xy}=ss_{xy}=\Sigma\left(x-x_{ave}\right)\cdot\left(y-y_{ave}\right)$$

$$s_{yy}=ss_{yy}=\Sigma\left(y-y_{ave}\right)^2$$

$$s_{xx}=ss_{xx}=\Sigma\left(x-x_{ave}\right)^2$$

Peak at Velocity	$(x-x_{avg})*(y-y_{avg})$
100	-5036.52
300	-2211.76
500	0
700	1136.36
900	39.52
S_{xy}	**-6072.40**

Table 6. 5 Calculation of Sxy for linear regression

Peak at Velocity	$(y-y_{AVG})^2$
100	158.54
300	122.29
500	866.16
700	32.282
900	0.0097
S_{yy}	**1179.29**

Table 6. 6 Calculation of Syy for linear regression

Peak at Velocity	$(x-x_{AVG})^2$
100	160000
300	40000
500	0
700	40000
900	160000
S_{xx}	**400000**

Table 6. 7 Calculation of Sxx for linear regression

$$\text{Slope}=m=\frac{s_{xy}}{s_{xx}}$$

$$\text{Intercept}=b=y_{ave}-\left(m\cdot x_{ave}\right)$$

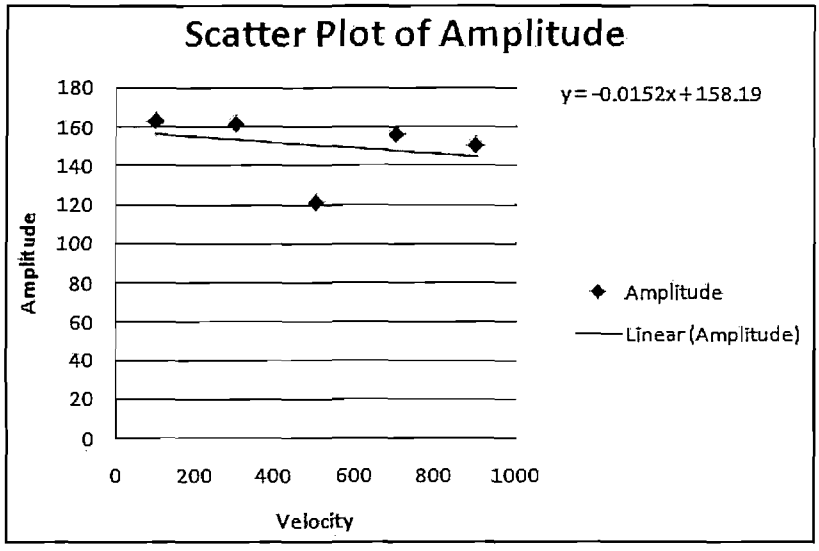
The formula used to calculate the slope and intercept of the graph (Widener University).

Slope = -6072.4 / 400000
Slope = -0.0152

Intercept = 150.6032 – (-0.0152*500)
Intercept = 158.19

The formula for the linear regressed data is therefore:

y = -0.0152x + 158.19



Graph 5. 4 Scatter plot with regression line

The new values for the sum of the peak trough calculations and the amplitude are now represented in Table 6.8 below:

Cycle Number	Old Peak Trough	New Peak Trough	New Amplitude
1	326.38	313.34	156.67
2	323.32	307.26	153.63
3	242.34	301.18	150.59
4	312.57	295.10	147.55
5	301.40	289.02	144.51

Table 6. 8 New regressed peak trough values and new amplitude

A.3. FREQUENCY OF THE SIN GRAPH

The type of graph and amplitude has now been calculated the next step is to calculate the frequency of the sin function. The general equation for a sin graph is y = A sin(Bx + C). B controls the length of a cycle and can stretch or shrink the function in along the x-axis. The general equation for the period of a sine graph is $2\pi/B$. If B is larger than 1 the period shrinks and if B is greater than 1 the period stretches (Arizona State University).

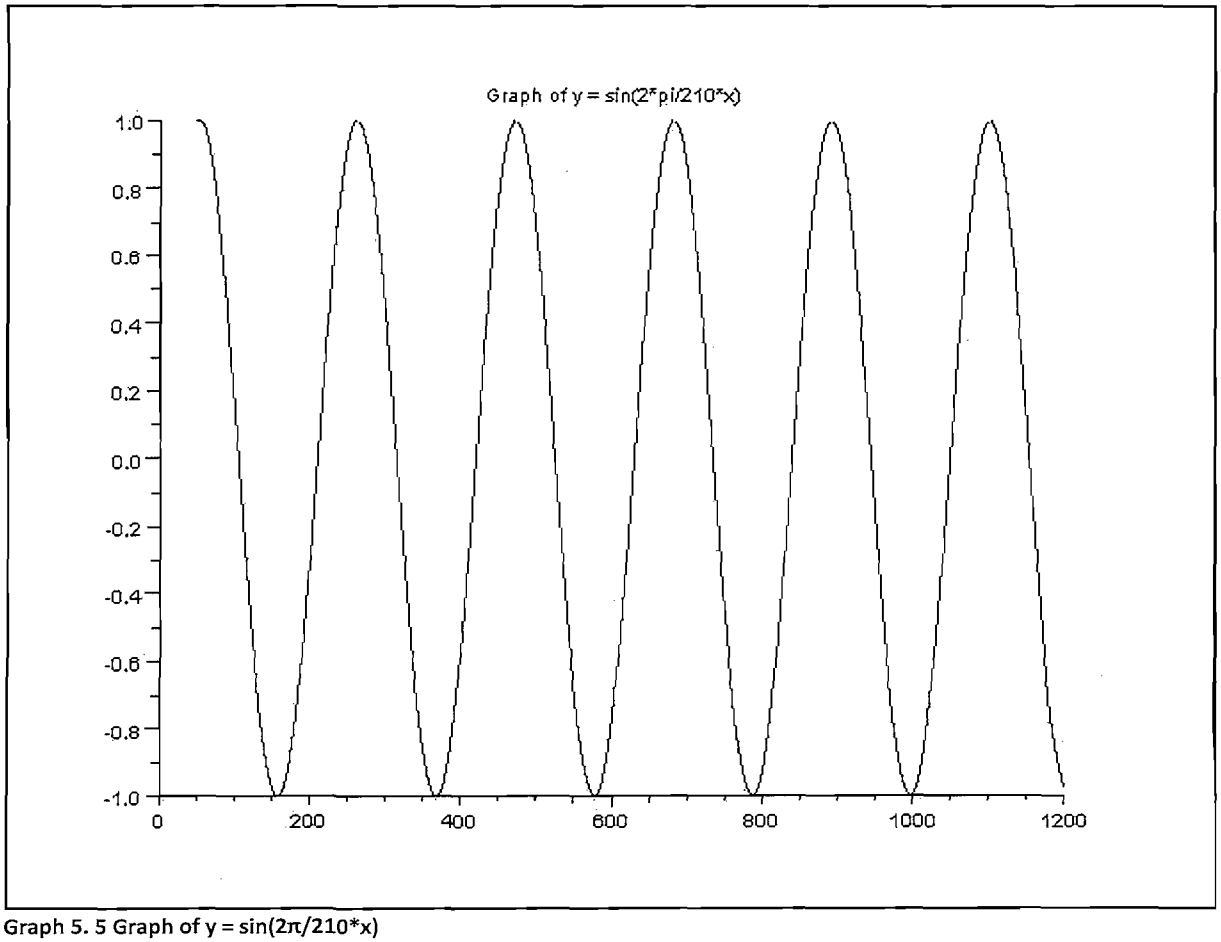
Calculating the period:

$$\frac{2\pi}{B} = \frac{(1100-50)}{5}$$

$$B = \frac{2\pi}{210}$$

Graphing the function it is visible that to bring the graph in line with the data the graph must also change dynamically as the velocity increases. This growth is slightly negative as can be seen from graph 5.1.

From graph 5.5 at point x = 200 the function should be at its trough. At point x = 1000 the graph is very close to being on its trough. The function must increase in the frequency as the velocity increases.



Graph 5. 5 Graph of y = sin(2π/210*x)

The first peak on the graph is at the point where y = 1. This is the first quarter of the sine wave. The trough is in the third quarter of the sine wave and is therefore the x-value for the peak multiplied by 3. To calculate the peak:

$$1 = \sin\left(\frac{2\pi}{210}x\right)$$

$$\sin^{-1}(1) = \frac{2\pi}{210}x$$

$$x = \frac{\pi}{2} \times \frac{210}{2\pi}$$

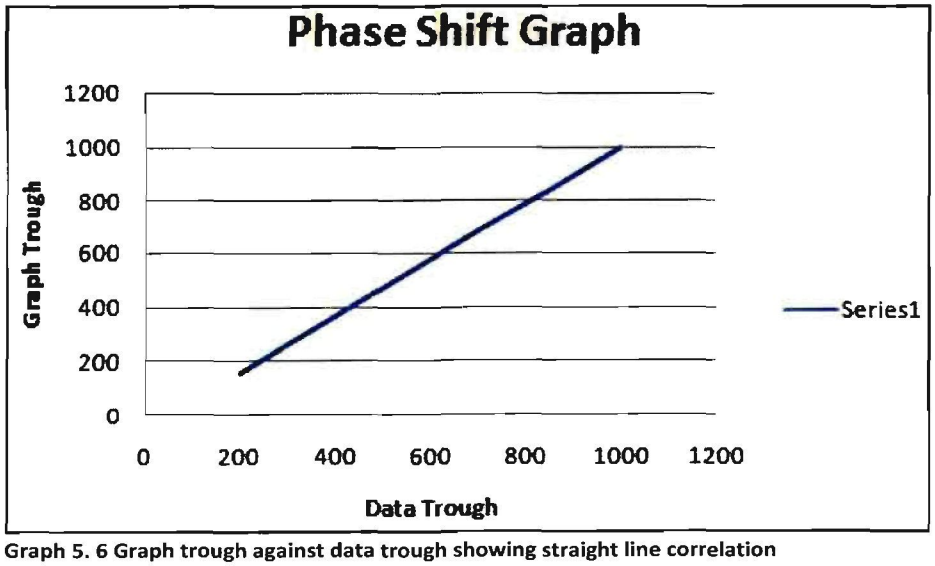
$$x = 52.5$$

The first trough is therefore 52.5 x 3 = 157.5.From this point on each trough will occur one full cycle from each successive trough.

Trough Number	Data Value	Graph Value	Delta Trough
1	200	157.50	42.50
2	400	367.50	32.50
3	600	577.50	22.50
4	800	787.50	12.50
5	1000	997.50	2.50

Table 6. 9 Troughs of the data and the graph

Plotting the graph 5.6 trough and the required troughs shows a straight line correlation:



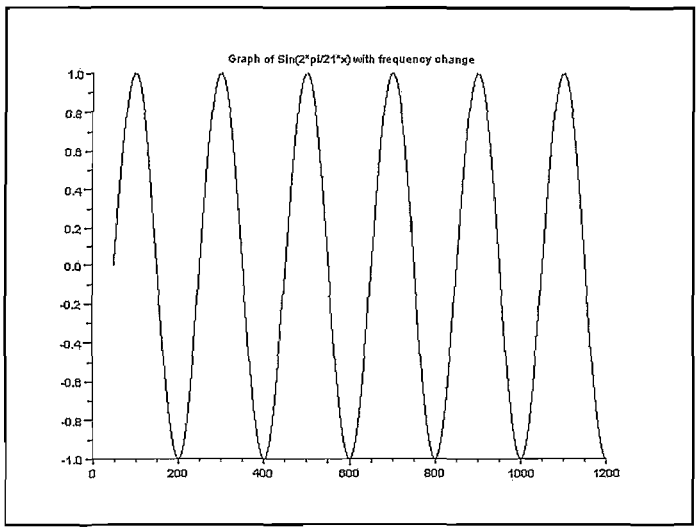
Graph 5. 6 Graph trough against data trough showing straight line correlation

The equation for the phase shift is therefore:

$$y = 1.05x - 52.5$$

The formula for the sine graph changes to:

$$y = \sin\left(\left(\frac{2\pi}{210}\right)(1.05x - 52.5)\right)$$



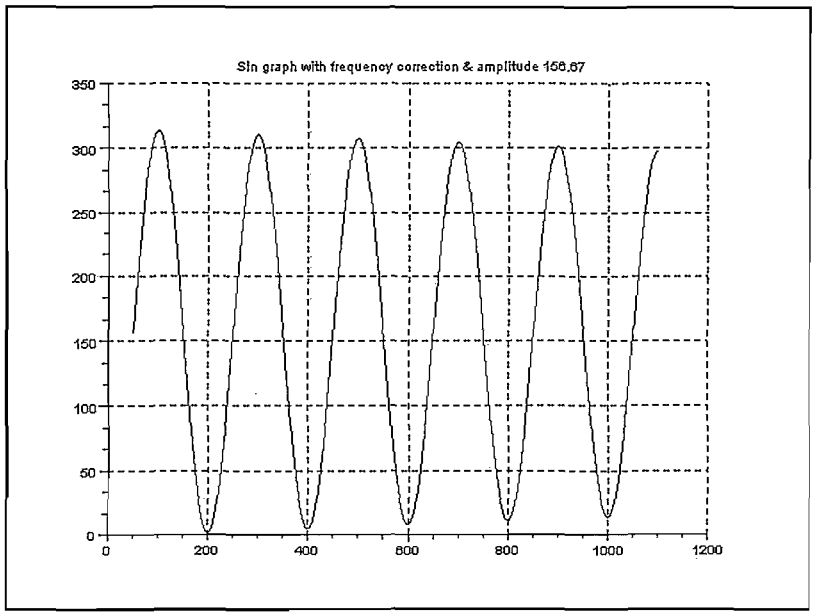
Graph 5. 7 New sine graph with frequency change

A.4. AMPLITUDE CHANGE CALCULATION FOR FORMULA

The amplitude has been previously calculated for each cycle but has to now be converted into a form to suit the general equation. For a sine graph with increasing amplitude the value of the velocity plotted on the x-axis is multiplied by the sine function (Bourne, Composite Trigonimetric Curves, 2008). The general equation is:

$$y = x \times \sin(x)$$

Applying the amplitude of cycle 1 of 156.67 we can see the graph must move up. To do this the graph must be plotted on a straight line graph. This superimposition will move the graph.

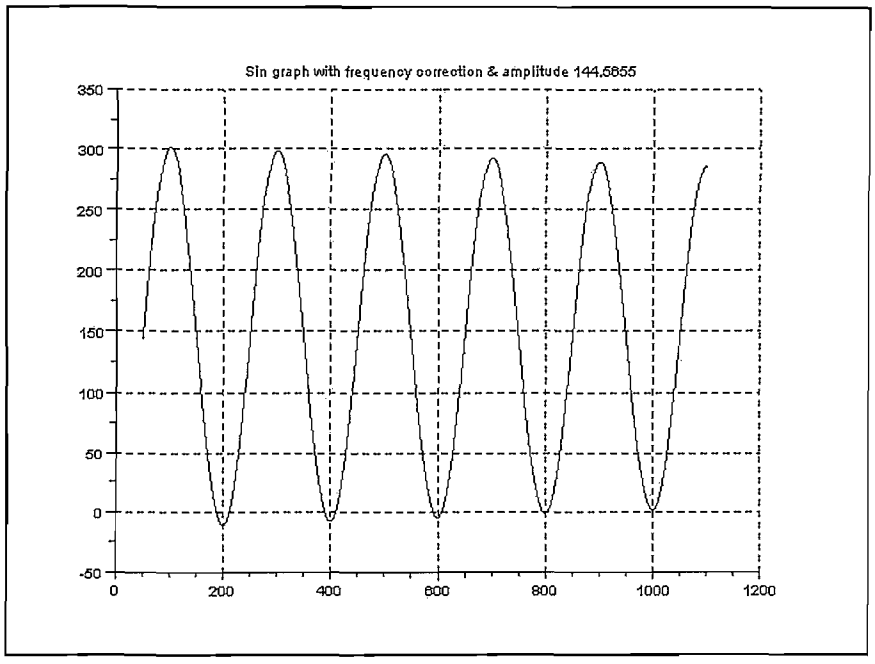


Graph 5. 8 Sine graph with amplitude (152.855)

To superimpose a straight line graph onto the sine graph the equation for the graph is added to the function of the sine graph (Bourne, Composite Trigonimetric Curves, 2008). The general formula becomes:

$$y = (mx + c) + \big(x \times sin(x)\big)$$

The data for the formula has been regressed so the best fit between the peak and trough of cycles must be used. For cycle 1 the function at the trough gives a y value of -17.869. The best way to fit the data is to move the graph to the trough by a value x. Subtract the difference between the graphs function at the peak and the data value for the peak. Divide the difference by 2 and shift the graph up by this value. This makes the error equal between the values of the peak and the trough. For cycle 1 the point at 200mm.s^{-1} must move up 137.281 units. The difference between the y value of the graph and the data is 308.52 – (f(100) = 293.951) is 14.569. This means the graph must move upwards a further 7.2845 to 144.5655.

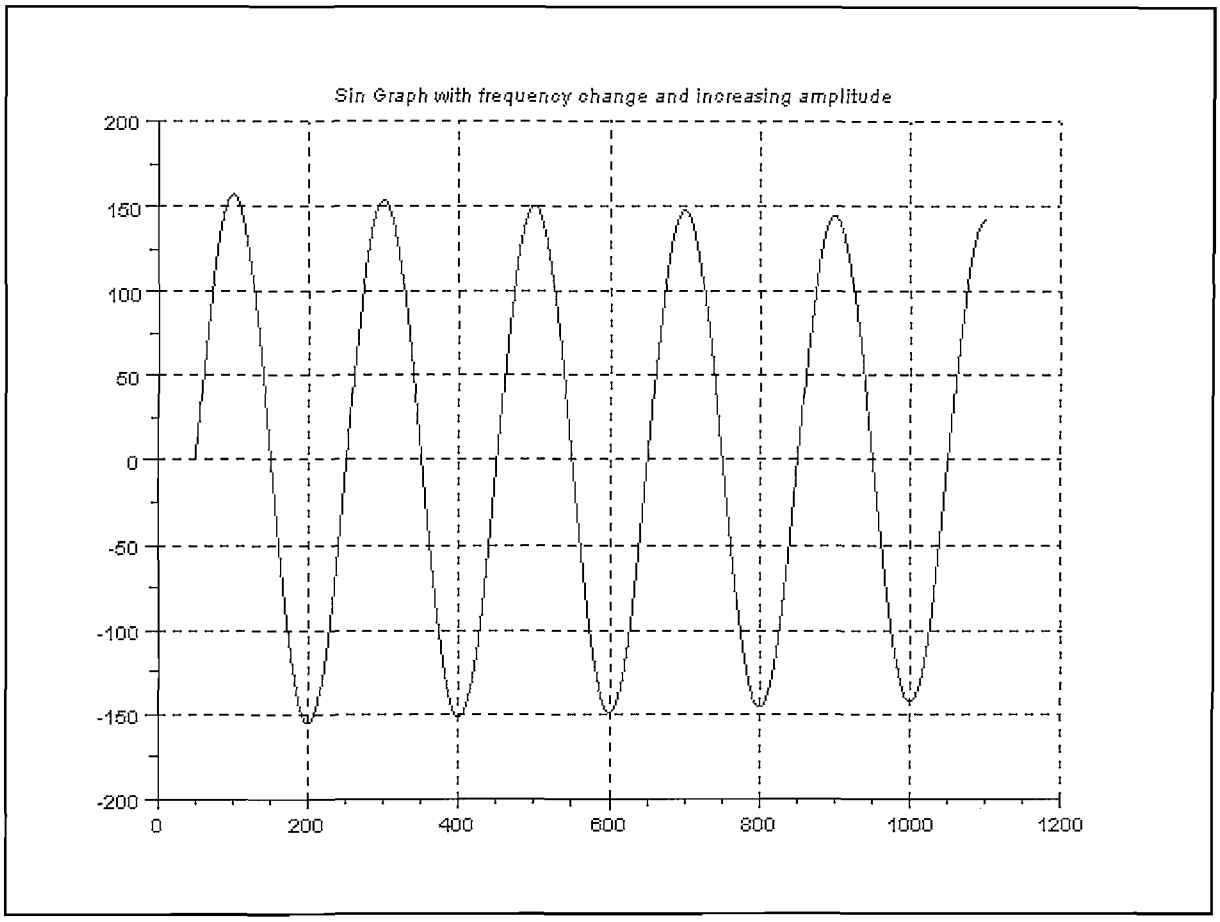


Graph 5. 9 Sine graph superimposed on the streight line

The start point of the graph is set however the amplitude remains fixed over the range of the velocity. This is not true, to correct this problem the equation derived previously and used to calculate the values in Table 6.8 is added to the formula. The formula with amplitude only changes to the following:

$$f(x) = (-0.0152x + 158.19) \times \left(\sin\left(\left(\frac{2\pi}{210}\right)(1.05x - 52.5)\right)\right)$$

This formula now plots the following graph:

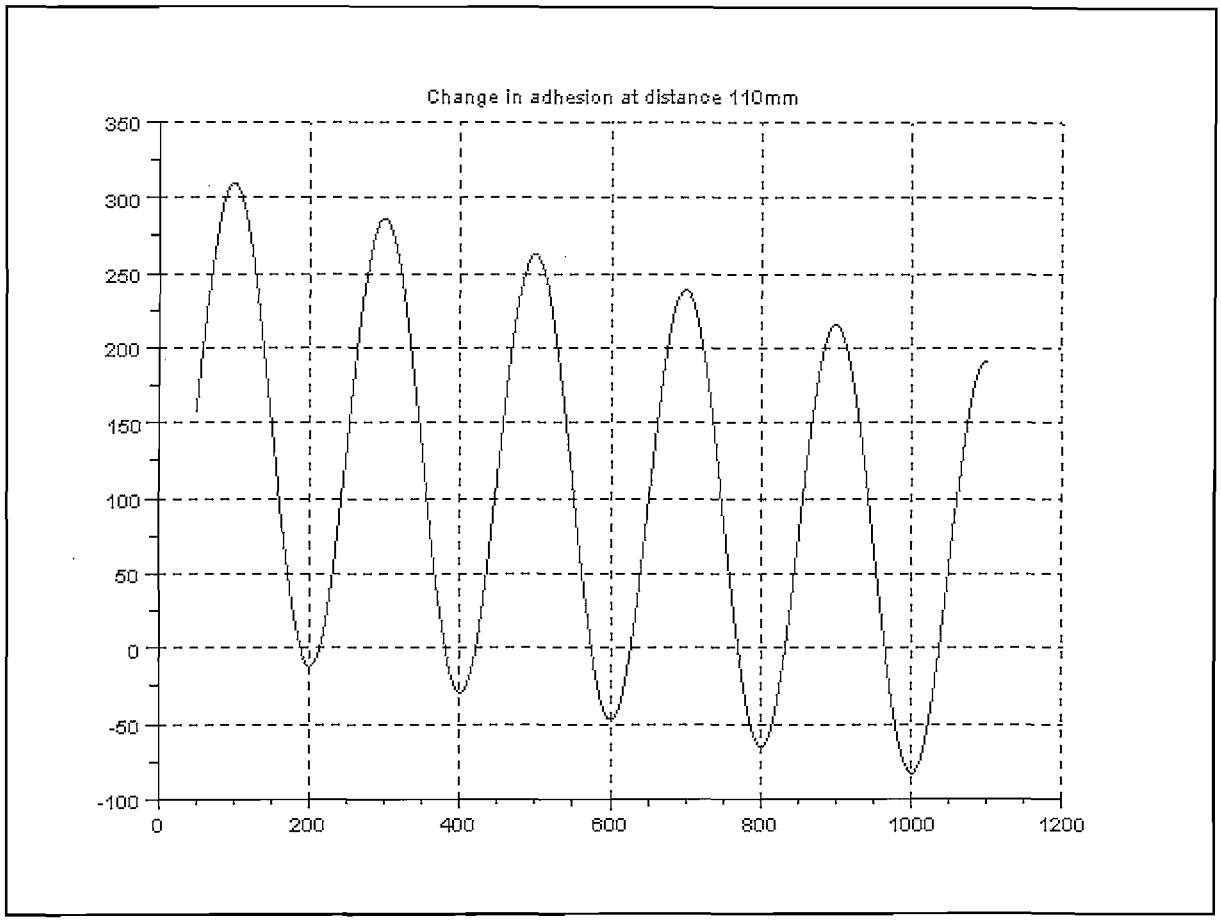


Graph 5. 10 Sine function with increasing amplitude

From the graph it is clear that the graph does not fit the data. The first visible problem is that the first period is too low and that at velocity of 1000 the value is well below the data. The graph must move upwards for this we superimpose a straight line graph using the techniques calculated earlier. This graph will lie at a gradient to better fit the data. The final formula for the change in adhesion at 110mm with the conversion to Pascals is now:

$$f(x) = \left((-0.1034x + 167.7) + \left((-0.0152x + 158.19) \sin\left(\left(\frac{2\pi}{210} \right) (1.05x - 52.5) \right) \right) \right)$$

The Graph plotted for this function now looks as follows (Graph 5.11):



Graph 5. 11 Final Graph for change in adhesion at 110mm

The formula represents the change in adhesion. To calculate the adhesion at a distance of 110mm apply the formula:

$$A = f(x) + f(x+1) \times 1.41471$$

Table 6.10 below is proof for the formula. Adhesion from the experiment was compared to the adhesion calculated from the formula. The difference between the two was computed and the absolute error percentage calculated. The error percentages were averaged to see the average total error of the formula. The formula is accepted if the average total error is fewer than 10%.

Velocity	Experiment Adhesion (kPa)	Formula Adhesion (kPa)	Delta Adhesion (kPa)	Error Percentage
100 mm.s^{-1}	436.47	421.44	15.03	3.44
200 mm.s^{-1}	411.19	387.89	23.30	5.66
300 mm.s^{-1}	350.27	362.93	-12.66	3.61
400 mm.s^{-1}	293.71	329.37	-35.66	12.14
500 mm.s^{-1}	231.72	304.42	-72.70	31.37
600 mm.s^{-1}	284.29	270.86	13.43	4.72
700 mm.s^{-1}	289.05	245.91	43.14	14.92
800 mm.s^{-1}	194.47	212.35	-17.88	9.19
900 mm.s^{-1}	180.22	187.39	-7.17	3.97
1000 mm.s^{-1}	181.78	153.84	27.94	15.37
1100 mm.s^{-1}	130.86	128.88	1.98	1.52
Average Error Percentage				9.63

Table 6. 10 Table of values to confirm formula at 110mm

Table 6.10 shows the formula works at 110mm. The slight variations are due to the regression of the data and the nonlinearity of the experimental data. Other factors such as the panel plastic and mixing of the batches of paint can be the cause of the slight inaccuracies. The total average error is under the 10% band and therefore the formula can be accepted.

To derive the formulas at 30mm, 50mm, 70mm and 90mm the same recipe was followed.

Formula at 90mm with average error of 9.836%:

$$f(x) = \left((-0.0463x + 135.99) + \left((0.0105x + 170.25) \sin\left(\left(\frac{2\pi}{210} \right) (1.05x - 52.5) \right) \right) \right)$$

To convert the two equations into a general solution a straight line property between the two equations was assumed. Therefore using the distance from the flame as the value for

change between the two equations the general formula for a distance between 90mm and 110mm is:

$$f(x) = \left((Bx + C) + \left((Dx + E) \times \left(\sin\left(\frac{2\pi}{210}\right)(1.05x - 52.5)\right)\right)\right)$$

Where:

$$B = -2.855 \times 10^{-3}d + 0.21065$$

$$C = 1.3855d + 11.295$$

$$D = 1.285 \times 10^{-3}d + 0.12615$$

$$E = -0.563d + 220.92$$

A = Adhesion
x = velocity
d = distance
f(x) = Change in Adhesion

$$A = f(x) + f(x + 1) \times 1.41471$$

The Remaining formulas were calculated in the same way.

Formula 70mm (9.141%)

$$f(x) = (-5.5 \times 10^{-4}x^2 + 654.6 \times 10^{-3} + 26.091 + (9.44 \times 10^{-8}x^3 - 7.71 \times 10^{-4}x^2 + 1.3013x - 21.509)\sin\left(\left(\frac{2\pi}{210}\right)(1.05x - 52.5)\right)$$

Formula 50mm (9.802%)

$$f(x) = (-1 \times 10^{-5}x^2 + 26.6 \times 10^{-3}x + 114.29) + \left((37.47 \times 10^{-6}x^3 - 98.31 \times 10^{-3}x^2 + 1.0047x - 8.4766)\sin\left(\left(\frac{2\pi}{210}\right)(1.05x - 52.5)\right)\right)$$

Formula 30mm (9.946%)

$$f(x) = (-0.2 \times 10^{-3}x^2 + 0.4021x + 41.088) + \left((-1.57 \times 10^{-6}x^3 + 3.3 \times 10^{-3}x^2 - 1.7448x + 406.02)\sin\left(\left(\frac{2\pi}{210}\right)(1.05x - 52.5)\right)\right)$$

Final Formula at d = 70mm

$$f(x) = \left(B + (C \times \sin\left(\left(\frac{2\pi}{210}\right)(1.05x - 52.5)\right)\right)$$

$$B = -5.5 \times 10^{-4}x^2 + 654.6 \times 10^{-3}x + 26.091$$

$$C = 9.44 \times 10^{-8}x^3 - 7.71 \times 10^{-4}x^2 + 1.3013x - 21.509$$

$$A = f(x) + f(x-1) \times 1.41471$$

Final Formula 30 – 50mm

$$f(x) = \left((Ex^2 + Fx + G) + \left((Hx^3 + Ix^2 + Jx + K)\sin\left(\left(\frac{2\pi}{210}\right)(1.05x - 52.5)\right)\right)\right)$$

$$E = 9.5 \times 10^{-6}d - 485 \times 10^{-6}$$

$$F = -18.775 \times 10^{-3}d + 0.96535$$

$$G = 3.6601d - 68.715$$

$$H = 1.952 \times 10^{-6}d - 60.13 \times 10^{-6}$$

$$I = -5.08 \times 10^{-3}d + 0.1557$$

$$J = 137.475 \times 10^{-3}d - 5.869$$

$$K = -20.725d + 1027.765$$

$x \in (100,200,300,400,500,600,700,800,900,1000,1100)$

$$A = f(x) + f(x-1) \times 1.41471$$

APPENDIX B - ADHESION TESTING MACHINE

B.1 - HANDLE LENGTH CALCULATION

Symbol	Description	Units
Θ	Thread Angle	Radians
α	Helix / Lead Angle	Radians
μ_s	Screw Friction Coefficient	
μ_c	Nut Collar Friction Coefficient	
d_m	Mean Screw Diameter	Meters
d_{mc}	Mean Collar Diameter	Meters
W	Load To Be Lifted	Newtons
T_R	Torque Required To Raise Load	Newton Meters

Raising Torque Formula:

$$T_R = \frac{d_m W}{2}\left[\frac{\mu_s + \cos\theta_n \tan\alpha}{\cos\theta_n - \mu_s \tan\alpha}\right] + \frac{d_{mc}\mu_c W}{2}$$

Values for M16 Screw Thread

Symbol	Value
Θ	1.0472 Rad
α	2.9833×10^{-2}
μ_s	0.25
μ_c	0.25
d_m	0.016m
d_{mc}	0.024m
W	9810N

Torque Calculation:

$$T_R = \frac{(0.016)(9810)}{2}\left[\frac{0.25 + \cos(1.0472)\tan(2.9833\times10^{-2})}{\cos(1.0472) - 0.25\tan(2.9833\times10^{-2})}\right] + \frac{(0.024)(0.24)(9810)}{2}$$

$$T_R = 78.48[0.5378695] + 28.2528$$

$$\therefore \boldsymbol{T_R = 70.4648Nm}$$

Handle Calculation

$$T = F.d$$

$$70.4648 = 287.61d$$

$$\therefore \boldsymbol{d = 0.245m}$$ **The handle must be 245mm long to raise a maximum force of 1 ton.**

B.2 - SUPPORT PLATE SECOND MOMENT OF AREA

Symbol	Description	Unit
I_x	Second Moment of Area	mm^4
b	Breadth	mm
d	Depth	mm

SECOND MOMENT OF AREA FOR RECTANGLE ABOUT X-AXIS THROUGH THE CENTROID DERIVATION

$$I_x = \int_{\frac{-d}{2}}^{\frac{d}{2}} y^2 \, dA$$

$$I_x = \int_{\frac{-d}{2}}^{\frac{d}{2}} y^2 b \, dy$$

$$I_x = \left[\frac{y^3 b}{3}\right]_{\frac{-d}{2}}^{\frac{d}{2}}$$

$$I_x = \frac{(\frac{d}{2})^3 b}{3}$$

$$I_x = \frac{d^3 b}{24} - \left[\frac{-d^3 b}{24}\right]$$

$$\therefore I_x = \frac{d^3 b}{12}$$

Plate Characteristics

Symbol	Value
b	100
d	12

B.3 - SECOND MOMENT OF AREA FOR SUPPORT PLATE

$$I_x = \frac{d^3 b}{12}$$

$$I_x = \frac{12^3 \times 100}{12}$$

$$\therefore I_x = 1.4400 \times 10^4 mm^4$$

B.4 - SUPPORT PLATE DEFLECTION

Symbol	Description	Unit
I	Second Moment of Area	mm^4
E	Modulus of Elasticity	Pa
x	Distance	mm
w	load per unit length	N/mm
M	Moment	
L	Length of Beam	mm

B.5 - SUPPORT PLATE DIAGRAM

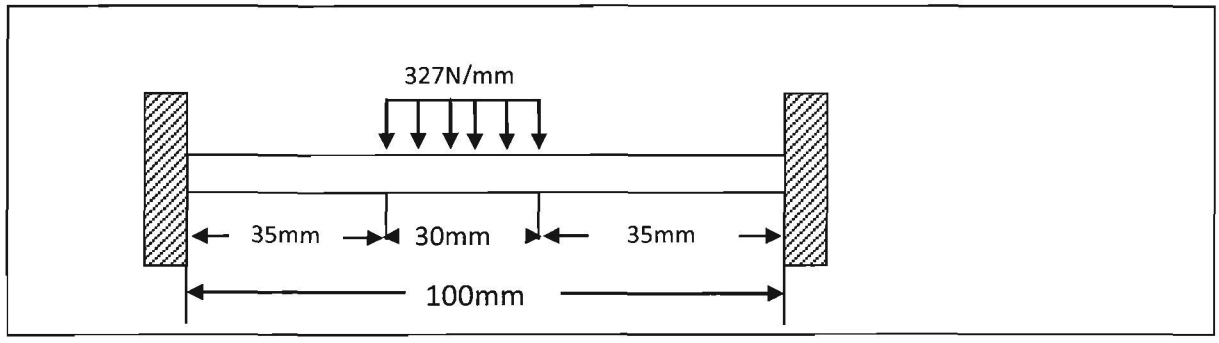


Plate is supported on both ends Therefore:

Symbol	Value
I	$1.4400x10^4mm^4$
E	$210kPa(N/mm^2)$
W	327N/mm
L	100mm

$$R = \frac{327 \times 30}{2}$$

$$\therefore R = 4.905kN$$

$$EI\frac{d^2y}{dx^2} = M$$

$$\therefore EI\frac{d^2y}{dx^2} = 4905x - M - \frac{327}{2}(x-35)^2 + \frac{327}{2}(x-65)^2$$

$$EI\int \frac{4905x^2}{2} - Mx - \frac{109}{2}(x-35)^3 + \frac{109}{2}(x-65)^3 + A\,dx$$

$$The\ slope = 0\ where\ x = 0$$

$$\therefore A = 0$$

$And\ the\ Slope = 0\ where\ x = 50$

$$0 = \frac{(4905)50^2}{2} - 50M - \frac{109}{2}(15)^3$$

$$\therefore M = 118.9463\text{kN}$$

$$EI \int \frac{4905x^2}{2} - 118946.25x - \frac{109}{2}(x-35)^3 + \frac{109}{2}(x-65)^3\, dx$$

$$EIy = \frac{4905x^3}{6} - \frac{118946.25x^2}{2} - \frac{109}{6}(x-35)^4 + \frac{109}{6}(x-65)^4 + B$$

$when\ x = 0, deflection = 0;\ \therefore B = 0$

$$\therefore EIy = 817.5x^3 - 59473.125x^2 - 13.625(x-35)^4 + 13.625(x-65)^4$$

$y_{max}\ is\ where\ x = 50$

$$\therefore y_{max} = \frac{1}{EI}[(817.5)(50)^3 - (59473.125)(50)^2 - (13.625)(15)^4]$$

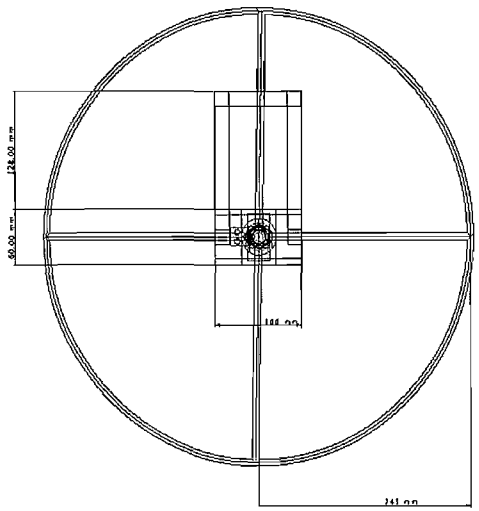
$$\therefore y_{max} = \frac{1}{(210\times10^3)(1.4400\times10^4)}[(817.5)(50)^3 - (59473.125)(50)^2 - (13.625)(15)^4]$$

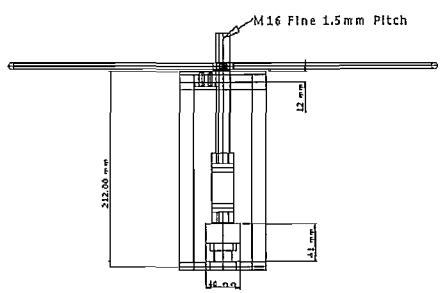
$$y_{max} = -1.57622 \times 10^{-2} mm$$

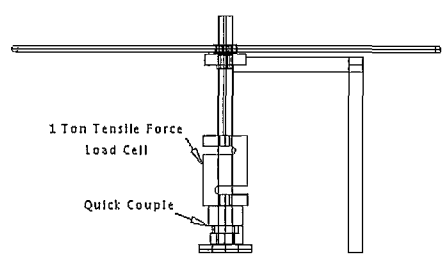
$$y_{max} = 1.57622 \times 10^{-2} mm\ downwards$$

B.6 - CAD RENDERING OF ADHESION TESTING MACHINE

B.7 - TECHNICAL DRAWING OF ADHESION TESTING MACHINE







APPENDIX C –SOURCE CODE FOR COMPUTER SOFTWARE 1 FILE ONLY

```
// AdhesionMeter.cpp : implementation of the CAdhesionMeterView class
// Quintin Immelman 2008
// Masters Degree Engineering
// North West University

#include "stdafx.h"
#include "AdhesionMeter.h"
#include "MainFrm.h"
#include "FileView.h"
#include "OutputWnd.h"

#include <stdio.h>
#include <stdlib.h>

#include <initguid.h>
#include <wtypes.h>
#include <conio.h>
#include "AdhesionMeterDoc.h"
#include "AdhesionMeterView.h"
#include <time.h>
#include <sys/timeb.h>

#define MAX_LOADSTRING 256

extern "C"
{
	#include "hidsdi.h"
	#include <setupapi.h>
}
extern CString CompareFileName, OpenDataFileName;
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
//////////////////Define Events//////////////////////////////
CEvent g_startTimer;
CEvent g_endTimer;
//CEvent g_endRead;
//////////////////HID Variables/////////////////////////
	DWORD			ActualBytesRead;
	bool		ApplicationActive;
	DWORD			BytesRead;
	HIDP_CAPS		Capabilities;
	DWORD			cbBytesRead;
	PSP_DEVICE_INTERFACE_DETAIL_DATA detailData;
	bool			DeviceDetected;
	HANDLE			DeviceHandle;
	DWORD			dwError;
	HANDLE			hEventObject;
	HANDLE			hDevInfo;
	HANDLE			ThreadHandle;
	DWORD			ThreadID;
	GUID			HidGuid;
	OVERLAPPED		HIDOverlapped;
	char			InputReport[3];
	ULONG			Length;
	LPOVERLAPPED lpReadOverLapped;
	DWORD			NumberOfBytesRead;
	HANDLE			ReadHandle;
	HANDLE			WriteHandle;
	ULONG			Required;
	CString				ValueToDisplay;
	bool FirstRunHID = TRUE;
```

```
// CAdhesionMeterView

IMPLEMENT_DYNCREATE(CAdhesionMeterView, CFormView)

BEGIN_MESSAGE_MAP(CAdhesionMeterView, CFormView)
	ON_COMMAND(ID_CONNECTION_CONNECT, &CAdhesionMeterView::OnConnectionConnect)
	ON_COMMAND(ID_CONNECTION_DISCONNECT,
&CAdhesionMeterView::OnConnectionDisconnect)
	ON_WM_SIZE()
	ON_WM_PAINT()
	ON_WM_HSCROLL()
	ON_WM_TIMER()
	ON_WM_ERASEBKGND()
	ON_COMMAND(ID_GRAPH_GRID, &CAdhesionMeterView::OnGraphGrid)
	ON_COMMAND(ID_MESSAGE_CONNECT, &CAdhesionMeterView::CaptionButton)
	ON_COMMAND(ID_MESSAGE_DISCONNECT,
&CAdhesionMeterView::DisconnectCaptionButton)
	ON_COMMAND(ID_FILE_SAVE, &CAdhesionMeterView::SaveData)
	ON_COMMAND(ID_FILE_OPEN, &CAdhesionMeterView::OpenData)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CFormView::OnFilePrintPreview)
	ON_MESSAGE(ON_OPEN_COMPFILE, OnOpenCompFile)
	ON_MESSAGE(ON_OPEN_DATAFILE, OnOpenDataFile)
	ON_MESSAGE(ON_PROPERTIES_GRID_ON, OnPropertiesGridOn)
	ON_MESSAGE(ON_PROPERTIES_GRID_OFF, OnPropertiesGridOff)
	ON_MESSAGE(ON_PROPERTIES_GRAPHTYPE, OnPropertiesGraphType)
	ON_MESSAGE(ON_PROPERTIES_XSTEP, OnPropertiesGraphXStep)
	ON_MESSAGE(ON_PROPERTIES_UNITS, OnPropertiesUnits)
	ON_MESSAGE(ON_PROPERTIES_CAPTURE_MODE, OnPropertiesModeChange)
	ON_MESSAGE(ON_PROPERTIES_TIME_CHANGED, OnPropertiesTimeChange)
	ON_MESSAGE(ON_PROPERTIES_COMPGRAPH_OVER, OnPropertiesCompGraphOver)
	ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()

// CAdhesionMeterView construction/destruction

CAdhesionMeterView::CAdhesionMeterView()
	: CFormView(CAdhesionMeterView::IDD)
{
	//Printer Variable Setup
	m_pMemDC = new CDC ;
	m_pBm = new CBitmap;

}

CAdhesionMeterView::~CAdhesionMeterView()
{
	//Clean Up Printer Variables
	delete m_pMemDC;
	delete m_pBm;
}

void CAdhesionMeterView::DoDataExchange(CDataExchange* pDX)
{
	CFormView::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_DataList, DataListCtrl);
	DDX_Control(pDX, IDC_GraphFrame, GFrameCtrl);
	DDX_Control(pDX, IDC_GraphScroll, GScrollCtrl);
	DDX_Control(pDX, IDC_DataList2, CompListCtrl);
	DDX_Control(pDX, IDC_GraphScroll2, CScrollCtrl);
	DDX_Control(pDX, IDC_GraphFrame2, CBoxCtrl);
}

BOOL CAdhesionMeterView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CFormView::PreCreateWindow(cs);
```

```
}

void CAdhesionMeterView::OnInitialUpdate()
{
	CFormView::OnInitialUpdate();
	CWinApp* pApp = AfxGetApp();
	xFormGraph1.eDx  = (FLOAT) 0.0;
	xFormGraph2.eDx  = (FLOAT) 0.0;
	ResizeParentToFit();
	Graph1Pointer = 0;
	//Get Information From The Registry
	CString RegGrid, RegGraphType, RegUnits, RegMode, RegCompGraphOverlaid;
	int RegStep = 0;
	int RegTimer = 0;
	RegGrid = pApp->GetProfileStringW(_T("Settings"), _T("Grid"), _T("On"));
	RegGraphType = pApp->GetProfileStringW(_T("Settings"), _T("Graph"), _T("Line
Graph"));
	RegStep = pApp->GetProfileIntW(_T("Settings"), _T("XStep"), 5);
	RegUnits = pApp->GetProfileStringW(_T("Settings"), _T("Units"),
_T("Grams"));
	RegMode = pApp->GetProfileStringW(_T("Settings"), _T("Mode"),
_T("Continuous"));
	RegTimer = pApp->GetProfileIntW(_T("Settings"), _T("Sample Time"), 100);
	RegCompGraphOverlaid = pApp->GetProfileStringW(_T("Settings"), _T("Comp
Graph"), _T("Single"));
	CaptureTime = RegTimer;
	GraphXStep = RegStep;
	if(RegGrid == _T("On"))
		GridLines = TRUE;
	else if(RegGrid == _T("Off"))
		GridLines = FALSE;
	if(RegGraphType == _T("Line Graph"))
		GraphType = 1;
	else if(RegGraphType == _T("Poly Bezer Graph"))
		GraphType = 2;
	else if(RegGraphType == _T("Bar Graph"))
		GraphType = 3;
	if(RegUnits == _T("Grams"))
		UnitsType = 1;
	else if(RegUnits == _T("Kilograms"))
		UnitsType = 2;
	else if(RegUnits == _T("Newtons"))
		UnitsType = 3;
	if(RegMode == _T("Continuous"))
		CaptureMode = 1;
	else if(RegMode == _T("Change"))
		CaptureMode = 2;
	if(RegCompGraphOverlaid == _T("Single"))
		CompGraphOverlaid = FALSE;
	else if(RegCompGraphOverlaid == _T("Overlaid"))
		CompGraphOverlaid = TRUE;
	//subMenu->CheckMenuItem(ID_GRAPH_GRID, MF_CHECKED);
	//Graph List Control
	DataListCtrl.SetExtendedStyle(LVS_REPORT|LVS_EX_GRIDLINES|LVS_EX_DOUBLEBUFFE
R|LVS_EX_FULLROWSELECT);
	DataListCtrl.InsertColumn(0, _T("Sample"), LVCFMT_LEFT, 50, 1);
	DataListCtrl.InsertColumn(1, _T("Date"), LVCFMT_LEFT, 50, 1);
	DataListCtrl.InsertColumn(2, _T("Time"), LVCFMT_LEFT, 50, 1);
	DataListCtrl.InsertColumn(3, _T("Temp"), LVCFMT_LEFT, 50, 1);
	DataListCtrl.InsertColumn(4, _T("Load"), LVCFMT_LEFT, 50, 1);
	//Compare List Control
	CompListCtrl.SetExtendedStyle(LVS_REPORT|LVS_EX_GRIDLINES|LVS_EX_DOUBLEBUFFE
R|LVS_EX_FULLROWSELECT);
	CompListCtrl.InsertColumn(0, _T("Sample"), LVCFMT_LEFT, 50, 1);
	CompListCtrl.InsertColumn(1, _T("Date"), LVCFMT_LEFT, 50, 1);
	CompListCtrl.InsertColumn(2, _T("Time"), LVCFMT_LEFT, 50, 1);
	CompListCtrl.InsertColumn(3, _T("Temp"), LVCFMT_LEFT, 50, 1);
	CompListCtrl.InsertColumn(4, _T("Load"), LVCFMT_LEFT, 50, 1);
```

```
	GScrollCtrl.SetScrollRange(0, 2103);
	CScrollCtrl.SetScrollRange(0, 2103);
	GScrollCtrl.SetScrollPos(0);
	CScrollCtrl.SetScrollPos(0);
	//GScrollCtrl.EnableScrollBar(ESB_ENABLE_BOTH);
}

void CAdhesionMeterView::OnRButtonUp(UINT nFlags, CPoint point)
{
	ClientToScreen(&point);
	OnContextMenu(this, point);
}

void CAdhesionMeterView::OnContextMenu(CWnd* pWnd, CPoint point)
{
	theApp.GetContextMenuManager()->ShowPopupMenu(IDR_POPUP_EDIT, point.x,
point.y, this, TRUE);
}


// CAdhesionMeterView diagnostics

#ifdef _DEBUG
void CAdhesionMeterView::AssertValid() const
{
	CFormView::AssertValid();
}

void CAdhesionMeterView::Dump(CDumpContext& dc) const
{
	CFormView::Dump(dc);
}

CAdhesionMeterDoc* CAdhesionMeterView::GetDocument() const // non-debug version is
inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CAdhesionMeterDoc)));
	return (CAdhesionMeterDoc*)m_pDocument;
}
#endif //_DEBUG


bool CAdhesionMeterView::FindLoadCell(void)
{
	HIDD_ATTRIBUTES                                    Attributes;
	SP_DEVICE_INTERFACE_DATA                devInfoData;
	bool                                               LastDevice = FALSE;
	int                                                       MemberIndex = 0;
	bool                                               MyDeviceDetected =
FALSE;
	LONG                                               Result;
	const unsigned int VendorID = 0x0925;
	const unsigned int ProductID = 0x1299;
	Length = 0;
	detailData = NULL;
	DeviceHandle=NULL;
	HidD_GetHidGuid(&HidGuid);
	hDevInfo=SetupDiGetClassDevs(&HidGuid, NULL, NULL,
DIGCF_PRESENT|DIGCF_INTERFACEDEVICE);
	devInfoData.cbSize = sizeof(devInfoData);
	MemberIndex = 0;
	LastDevice = FALSE;
	do
	{
		MyDeviceDetected=FALSE;

		Result=SetupDiEnumDeviceInterfaces
			(hDevInfo,
```

```
                        0,
                        &HidGuid,
                        MemberIndex,
                        &devInfoData);

                if (Result != 0)
                {
                        Result = SetupDiGetDeviceInterfaceDetail
                                (hDevInfo,
                                &devInfoData,
                                NULL,
                                0,
                                &Length,
                                NULL);
                        detailData = (PSP_DEVICE_INTERFACE_DETAIL_DATA)malloc(Length);
                        detailData -> cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);
                        Result = SetupDiGetDeviceInterfaceDetail
                                (hDevInfo,
                                &devInfoData,
                                detailData,
                                Length,
                                &Required,
                                NULL);
                        DeviceHandle=CreateFile
                                (detailData->DevicePath,
                                GENERIC_READ|GENERIC_WRITE,
                                FILE_SHARE_READ|FILE_SHARE_WRITE,
                                (LPSECURITY_ATTRIBUTES)NULL,
                                OPEN_EXISTING,
                                0,
                                NULL);
                        //Set the Size to the number of bytes in the structure.
                        Attributes.Size = sizeof(Attributes);
                        Result = HidD_GetAttributes
                                (DeviceHandle,
                                &Attributes);
                        MyDeviceDetected = FALSE;
                        if (Attributes.VendorID == VendorID)
                        {
                                if (Attributes.ProductID == ProductID)
                                {
                                        MyDeviceDetected = TRUE;
                                        GetDeviceCapabilities();
                                        WriteHandle=DeviceHandle;
                                        PrepareForOverlappedTransfer();
                                }
                                else
                                        CloseHandle(DeviceHandle);
                        }
                        else
                                CloseHandle(DeviceHandle);
                free(detailData);
                } //if (Result != 0)
                else
                        LastDevice=TRUE;
                MemberIndex = MemberIndex + 1;
        } //do
        while ((LastDevice == FALSE) && (MyDeviceDetected == FALSE));
        if (MyDeviceDetected == FALSE)
        {
                MessageBox(_T("Failed to detect Adhesion Meter unit.\nPlease ensure it
is correctly connected."),
                        _T("Adhesion Meter USB Connection Error"), MB_ICONERROR|MB_OK);
        }
        else
                SetupDiDestroyDeviceInfoList(hDevInfo);
        return MyDeviceDetected;
}
```

```
void CAdhesionMeterView::GetDeviceCapabilities(void)
{
	PHIDP_PREPARSED_DATA	PreparsedData;
	HidD_GetPreparsedData (DeviceHandle, &PreparsedData);
	HidP_GetCaps (PreparsedData, &Capabilities);
	HidD_FreePreparsedData(PreparsedData);
}

DWORD CAdhesionMeterView::StaticIO_Thread(LPVOID Param)
{
	if (Param != NULL)
		return ((CAdhesionMeterView*)Param)->ReadReport();
	else
		return -1;
}

DWORD WINAPI CAdhesionMeterView::ReadReport()
{
	CString	ByteToDisplay = _T("");
	ULONG	InputReportLength = 0;
	CString	MessageToDisplay = _T("");
	ULONG	Result;
	//Read a report from the device.


	InputReportLength=Capabilities.InputReportByteLength;

	do
	{
		Result = HidD_FlushQueue(DeviceHandle);
		FlushFileBuffers(ReadHandle);
		Result = ReadFile (ReadHandle, InputReport,
			Capabilities.InputReportByteLength, &BytesRead,
			NULL);

		//ActualBytesRead = Capabilities.FeatureReportByteLength;
		ActualBytesRead = BytesRead;
		Result = WaitForSingleObject(hEventObject, 300);
		DisplayInputReport();

	}

	while ((DeviceDetected == TRUE) && (ApplicationActive == TRUE));
    return 0;
}

void CAdhesionMeterView::PrepareForOverlappedTransfer(void)
{
	if(hEventObject == 0)
	{
		hEventObject = CreateEvent(NULL, TRUE, TRUE, _T(""));
		HIDOverlapped.hEvent = hEventObject;
		HIDOverlapped.Offset = 0;
		HIDOverlapped.OffsetHigh = 0;
	}
	ReadHandle=CreateFile
		(detailData->DevicePath,
		GENERIC_READ|GENERIC_WRITE,
		FILE_SHARE_READ|FILE_SHARE_WRITE,
		(LPSECURITY_ATTRIBUTES)NULL,
		OPEN_EXISTING,
		0,
		NULL);
	ThreadHandle = CreateThread(NULL, 0,
		(LPTHREAD_START_ROUTINE)StaticIO_Thread,
		this, 0, &ThreadID);
}
```

```
// CAdhesionMeterView message handlers

void CAdhesionMeterView::OnConnectionConnect()
{
	if (FindLoadCell() == TRUE)
	{
		Timer = SetTimer(1, CaptureTime, NULL);
		AfxGetMainWnd()->SendMessage(WM_CAPTION_BAR_CONNECTED, 1, NULL);
	}
}

void CAdhesionMeterView::OnConnectionDisconnect()
{
	KillTimer(Timer);
	AfxGetMainWnd()->SendMessage(WM_CAPTION_BAR_CONNECTED, 2, NULL);
	if(DeviceDetected == TRUE)
	{
		FlushFileBuffers(ReadHandle);
		WaitForSingleObject(ThreadHandle, INFINITE);
		CloseHandle(DeviceHandle);
		CloseHandle(ReadHandle);
		CloseHandle(ThreadHandle);
		WaitForSingleObject(hEventObject, 300);
		InvalidateRect(InvGraph1);
	}
}

void CAdhesionMeterView::OnSize(UINT nType, int cx, int cy)
{
	CFormView::OnSize(nType, cx, cy);

	if(DataListCtrl)
	{
		DataListCtrl.EnableScrollBar(SB_VERT, ESB_ENABLE_BOTH);
		DataListCtrl.ShowScrollBar(SB_VERT, TRUE);
		DataListCtrl.MoveWindow(0, 0, cx/2, cy/2, TRUE);
		CRect DataSize;
		DataListCtrl.GetWindowRect(&DataSize);
		DataListCtrl.SetColumnWidth(0, DataSize.Width() / 5);
		DataListCtrl.SetColumnWidth(1, DataSize.Width() / 5);
		DataListCtrl.SetColumnWidth(2, DataSize.Width() / 5);
		DataListCtrl.SetColumnWidth(3, DataSize.Width() / 5);
		DataListCtrl.SetColumnWidth(4, (DataSize.Width() / 5) - 2);
	}
	if(CompListCtrl)
	{
		CompListCtrl.MoveWindow(0, cy/2, cx/2, cy/2, TRUE);
		CRect CompSize;
		CompListCtrl.GetWindowRect(&CompSize);
		CompListCtrl.SetColumnWidth(0, CompSize.Width() / 5);
		CompListCtrl.SetColumnWidth(1, CompSize.Width() / 5);
		CompListCtrl.SetColumnWidth(2, CompSize.Width() / 5);
		CompListCtrl.SetColumnWidth(3, CompSize.Width() / 5);
		CompListCtrl.SetColumnWidth(4, (CompSize.Width() / 5) - 2);
	}
	if(GFrameCtrl)
		GFrameCtrl.MoveWindow(cx/2, 0, cx/2 + 16, cy/2, TRUE);
	if(CBoxCtrl)
		CBoxCtrl.MoveWindow(cx/2, cy/2, cx/2, cy/2, TRUE);
	if(CScrollCtrl)
		CScrollCtrl.MoveWindow((cx/2) + 2, cy - 16, (cx/2)-2, 15, TRUE);
	if(GScrollCtrl)
		GScrollCtrl.MoveWindow((cx/2) + 2, (cy/2) - 16, (cx/2)-2, 15, TRUE);
	// TODO: Add your message handler code here
}

void CAdhesionMeterView::OnPaint()
```

```
{
	CPaintDC Pdc(this); // device context for painting
	CDC dc;
	CRect Window;
	GetClientRect(Window);
	dc.CreateCompatibleDC(&Pdc);
	CBitmap bDraw;
	bDraw.CreateCompatibleBitmap(&Pdc, Window.Width(), Window.Height());
	CBitmap *pOldBitmap = dc.SelectObject(&bDraw);
	dc.BitBlt(0, 0, Window.Width()/2, Window.Height(), &Pdc, 0, 0, SRCCOPY);
	dc.SetGraphicsMode(GM_ADVANCED);
	CRect Graph1Rect, Graph2Rect;
	HRGN hrgn;
	CRgn hrgn2;
	HRGN None;
	CFont TitleFont, yFont, xFont, LabelFont;
	TitleFont.CreateFontW(28, 0, 0, 0, FW_BOLD , FALSE, TRUE, FALSE,
ANSI_CHARSET, OUT_DEFAULT_PRECIS,
		CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH | FF_SWISS,
_T("Times New Roman"));
	yFont.CreateFontW(18, 0, 900, 900, FW_BOLD , FALSE, TRUE, FALSE,
ANSI_CHARSET, OUT_DEFAULT_PRECIS,
		CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH | FF_SWISS,
_T("Times New Roman"));
	xFont.CreateFontW(18, 0, 0, 0, FW_BOLD , FALSE, TRUE, FALSE, ANSI_CHARSET,
OUT_DEFAULT_PRECIS,
		CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH | FF_SWISS,
_T("Times New Roman"));
	LabelFont.CreateFontW(14, 0, 0, 0, FW_NORMAL , FALSE, FALSE, FALSE,
ANSI_CHARSET, OUT_DEFAULT_PRECIS,
		CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH | FF_SWISS,
_T("Times New Roman"));
	CPen RedPen, BlackPen, GreenPen, CartPen, GridPen;
	CBrush WhiteBrush2, RedBrush, BlueBrush;
	WhiteBrush2.CreateSolidBrush(RGB(255,255,255));
	RedBrush.CreateSolidBrush(RGB(255,0,0));
	BlueBrush.CreateSolidBrush(RGB(0, 112, 223));
	RedPen.CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
	BlackPen.CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
	GridPen.CreatePen(PS_DOT, 1, RGB(0, 0, 0));
	CartPen.CreatePen(PS_SOLID, 2, RGB(0, 0, 0));
	GreenPen.CreatePen(PS_SOLID, 1, RGB(0, 112, 223));
	Graph1Rect.SetRect(Window.right/2, Window.top, (Window.right) ,
(Window.bottom/2) - 16);
	Graph2Rect.SetRect(Window.right/2, Window.bottom/2, Window.right,
(Window.bottom) - 16);
	//Set Invalidation Rectangles
	InvGraph1.SetRect(Graph1Rect.left + 71, Graph1Rect.top + 52,
Graph1Rect.right, Graph1Rect.bottom - 41);
	InvGraph2.SetRect(Graph2Rect.left + 71, Graph2Rect.top + 52,
Graph2Rect.right, Graph2Rect.bottom - 41);
	InvValue1.SetRect(Graph1Rect.left + 71, Graph1Rect.bottom - 41,
Graph1Rect.right, Graph1Rect.bottom - 20);
	InvValue2.SetRect(Graph2Rect.left + 71, Graph2Rect.bottom - 41,
Graph2Rect.right, Graph2Rect.bottom - 20);
	//Fill Graph Area With White Block
	dc.FillRect(&Graph1Rect, &WhiteBrush2);
	dc.FillRect(&Graph2Rect, &WhiteBrush2);

	//Place Gridlines in Graph Area 1
	SelectObject(dc, CartPen);
	dc.MoveTo(Graph1Rect.left + 70, Graph1Rect.bottom - 20);
	dc.LineTo(Graph1Rect.left + 70, Graph1Rect.top + 50);
	dc.MoveTo(Graph1Rect.left + 40, Graph1Rect.bottom - 40);
	dc.LineTo(Graph1Rect.right + 600, Graph1Rect.bottom - 40);
	//Place Gridlines in Graph Area 2
	SelectObject(dc, CartPen);
	dc.MoveTo(Graph2Rect.left + 70, Graph2Rect.bottom - 20);
```

```
        dc.LineTo(Graph2Rect.left + 70, Graph2Rect.top + 50);
        dc.MoveTo(Graph2Rect.left + 40, Graph2Rect.bottom - 40);
        dc.LineTo(Graph2Rect.right + 600, Graph2Rect.bottom - 40);
        //Place Chart Title For Graph 1
        CFont* def_font = dc.SelectObject(&TitleFont);
        dc.SetBkColor(RGB(255, 255, 255));
        CSize Title1Length, Title2Length;
        Title1Length = dc.GetTextExtent(_T("Acquisition Graph"), 17);
        Title2Length = dc.GetTextExtent(_T("Comparison Graph"), 16);
        dc.TextOutW(((Graph1Rect.CenterPoint().x) - Title1Length.cx / 2),
Graph1Rect.top + 10, _T("Acquisition Graph"), 17);
        // Place Chart Title For Graph 2
        dc.TextOutW(((Graph2Rect.CenterPoint().x) - Title2Length.cx / 2),
Graph2Rect.top + 10, _T("Comparison Graph"), 16);
        //Place Chart Y-Axis For Graph 1
        dc.SelectObject(&yFont);
        dc.SetBkColor(RGB(255, 255, 255));
        CSize y1Length;
        y1Length = dc.GetTextExtent(_T("Load"), 4);
        dc.TextOutW(Graph1Rect.left + 10, (Graph1Rect.CenterPoint().y) + y1Length.cx
/ 2, _T("Load"), 4);
        // Place Chart Y-Axis For Graph 2
        dc.TextOutW(Graph2Rect.left + 10, (Graph2Rect.CenterPoint().y) + y1Length.cx
/ 2, _T("Load"), 4);
        //Place Chart X-Axis For Graph 1
        dc.SelectObject(&xFont);
        dc.SetBkColor(RGB(255, 255, 255));
        CSize x1Length;
        x1Length = dc.GetTextExtent(_T("Samples"), 7);
        dc.TextOutW(((Graph1Rect.CenterPoint().x) -x1Length.cx / 2),
Graph1Rect.bottom - 20, _T("Samples"), 7);
        // Place Chart X-Axis For Graph 2
        dc.TextOutW(((Graph2Rect.CenterPoint().x) - x1Length.cx / 2),
Graph2Rect.bottom - 20, _T("Samples"), 7);
        dc.SelectObject(def_font);
        //Set the Clipping Region and Limit it to the Graphs Only
        CRect BoundsRect1, BoundsRect2;
        BoundsRect1.CopyRect(&InvGraph1);
        BoundsRect2.CopyRect(&InvGraph2);
        hrgn = CreateRectRgn(BoundsRect1.left,BoundsRect1.top, BoundsRect1.right,
BoundsRect1.bottom + InvValue1.Height());
        hrgn2.CreateRectRgn(BoundsRect2.left,BoundsRect2.top, BoundsRect2.right,
BoundsRect2.bottom + InvValue2.Height());
        None = CreateRectRgn(0,0, 5, 5);
        SelectClipRgn(dc, hrgn);
        //Select Default Font
        dc.SelectObject(def_font);
        ////////////////////////////////
        // Graph 1 Plot Region
        ////////////////////////////////
        xFormGraph1.eM11 = (FLOAT) 1.0;
    xFormGraph1.eM12 = (FLOAT) 0.0;
    xFormGraph1.eM21 = (FLOAT) 0.0;
    xFormGraph1.eM22 = (FLOAT) 1.0;
        //xFormGraph1.eDx  = (FLOAT) 0.0; //Move Left Right
    xFormGraph1.eDy  = (FLOAT) 0.0;
    dc.SetWorldTransform(&xFormGraph1);
        ////////////////////////////////////////////////////////////////////////////
////////////////////////
        // Graph 1 Plot Data
        ////////////////////////////////////////////////////////////////////////////
////////////////////
        dc.SelectObject(RedPen);
        int PlotsA = 0;
        PlotsA = DataListCtrl.GetItemCount();
        double MaxNum = 0.00;
        for(int GetMaxNum = 0; GetMaxNum < PlotsA; GetMaxNum++)
        {
```

```
		size_t i;
		CString Data;
		char DataCh[8];
		double TestNum = 0.00;
		Data = DataListCtrl.GetItemText(GetMaxNum, 4);
		wcstombs_s(&i, DataCh, 7, Data, _TRUNCATE);
		TestNum = atof(DataCh);
		if(TestNum > MaxNum)
			MaxNum = TestNum;
}
double ScaleA = 0.00;
ScaleA = (InvGraph1.Height()) / (MaxNum);
POINT G1P[2000];
//G1P[0].x = InvGraph1.left;
//G1P[0].y = InvGraph1.bottom;
int Step = 0;
for(int GetPoints = 0; GetPoints < PlotsA; GetPoints++)
{
		size_t i;
		CString Data;
		char DataCh[8];
		double TempNum = 0.00;
		Data = DataListCtrl.GetItemText(GetPoints, 4);
		wcstombs_s(&i, DataCh, 7, Data, _TRUNCATE);
		TempNum = atof(DataCh);
		TempNum = TempNum * ScaleA;
		G1P[GetPoints].y = InvGraph1.bottom - TempNum;
		G1P[GetPoints].x = InvGraph1.left + Step;
		Step += GraphXStep;
}
//Full Data
if(PlotsA < 1997)
{
		for(int Full = 0; Full < 3; Full++)
		{
			size_t i;
			CString Data;
			char DataCh[8];
			double TempNum = 0.00;
			Data = DataListCtrl.GetItemText(PlotsA - 1, 4);
			wcstombs_s(&i, DataCh, 7, Data, _TRUNCATE);
			TempNum = atof(DataCh);
			TempNum = TempNum * ScaleA;
			G1P[PlotsA + Full].y = InvGraph1.bottom - TempNum;
			G1P[PlotsA + Full].x = InvGraph1.left + Step + Full;
		}
}
if(GraphType == 1)
{
		dc.MoveTo(G1P[0]);
		for(int PG = 0; PG < PlotsA; PG++)
			dc.LineTo(G1P[PG]);
}
else if(GraphType == 2)
{
		div_t Result;
		Result = div(PlotsA, 3);
		if(Result.rem == 0)
			dc.PolyBezier(G1P, PlotsA + 1);
		else if(Result.rem == 1)
			dc.PolyBezier(G1P, PlotsA +3);
		else if(Result.rem == 2)
			dc.PolyBezier(G1P, PlotsA + 2);
}
else if(GraphType == 3)
{
		for(int FR = 0; FR < PlotsA; FR++)
		{
```

```
                        CRect lpRect;
                        lpRect.bottom = InvGraph1.bottom;
                        lpRect.left = G1P[FR].x;
                        if(GraphXStep > 1)
                                lpRect.right = G1P[FR].x + GraphXStep - 1;
                        else if(GraphXStep == 1)
                                lpRect.right = G1P[FR].x + GraphXStep;
                        lpRect.top = G1P[FR].y;
                        dc.FillRect(&lpRect, &RedBrush);
                }
        }
        //Grid Lines
        if(GridLines == TRUE)
        {
                if(GraphXStep >= 5)
                {
                        int StepGrid = GraphXStep;
                        dc.SelectObject(GridPen);
                        for(int Grid = 0; Grid < PlotsA; Grid++)
                        {
                                if(Grid > 0)
                                {
                                        dc.MoveTo(InvGraph1.left + StepGrid,
InvGraph1.bottom);
                                        dc.LineTo(InvGraph1.left + StepGrid,
InvGraph1.top);
                                        StepGrid+= GraphXStep;
                                }
                        }
                }
        }
        //Moving Line
        if(DataListCtrl.GetItemCount() > 0)
        {
                dc.SelectObject(CartPen);
                dc.MoveTo(Graph1Pointer, InvGraph1.bottom);
                dc.LineTo(Graph1Pointer, InvGraph1.top);
                ////////////////////////////////////
                //Graph 1 Place Moving Line Label
                ////////////////////////////////////
                dc.SelectObject(BlackPen);

                dc.SelectObject(&LabelFont);
                dc.SetBkColor(RGB(255, 255, 204));
                CSize LabelLength;
                CString LabelValue;
                int Position = 0;
                for(int Clear = 0; Clear < DataListCtrl.GetItemCount(); Clear++)
                        DataListCtrl.SetItemState(Clear, FALSE, LVIS_SELECTED);
                //DataListCtrl
                for(int Get = 0; Get < DataListCtrl.GetItemCount(); Get++)
                {
                        int PointTest = 0;
                        PointTest = G1P[Get].x;
                        if(Graph1Pointer == PointTest)
                        {
                                LabelValue = DataListCtrl.GetItemText(Get, 4);
                                DataListCtrl.SetItemState(Get,
LVIS_SELECTED|LVIS_FOCUSED , LVIS_SELECTED|LVIS_FOCUSED);
                                DataListCtrl.EnsureVisible(Get, FALSE);
                                Position = Get;
                                break;
                        }
                        else
                        {
                                DataListCtrl.SetSelectionMark(-1);
                                LabelValue = _T("0");
                        }
```

```
                }
                if(GraphXStep == 1)
                {
                        if (Position < 20)
                        {
                                LabelLength = dc.GetTextExtent(LabelValue,
LabelValue.GetLength());
                                dc.Rectangle(Graph1Pointer + 2,
InvValue1.CenterPoint().y - 7, Graph1Pointer + (LabelLength.cx)+5,
InvValue1.CenterPoint().y + 8);
                                dc.TextOutW(Graph1Pointer + 4,
(InvValue1.CenterPoint().y - 6), LabelValue, LabelValue.GetLength());
                        }
                        else if(Position >= 20)
                        {
                                LabelLength = dc.GetTextExtent(LabelValue,
LabelValue.GetLength());
                                dc.Rectangle(Graph1Pointer - (LabelLength.cx /2)-2,
InvValue1.CenterPoint().y - 7, Graph1Pointer + (LabelLength.cx /2)+1,
InvValue1.CenterPoint().y + 8);
                                dc.TextOutW(Graph1Pointer - (LabelLength.cx / 2),
(InvValue1.CenterPoint().y - 6), LabelValue, LabelValue.GetLength());
                        }
                }
                else if(GraphXStep == 2)
                {
                        if (Position < 11)
                        {
                                LabelLength = dc.GetTextExtent(LabelValue,
LabelValue.GetLength());
                                dc.Rectangle(Graph1Pointer + 2,
InvValue1.CenterPoint().y - 7, Graph1Pointer + (LabelLength.cx)+5,
InvValue1.CenterPoint().y + 8);
                                dc.TextOutW(Graph1Pointer + 4,
(InvValue1.CenterPoint().y - 6), LabelValue, LabelValue.GetLength());
                        }
                        else if(Position >= 11)
                        {
                                LabelLength = dc.GetTextExtent(LabelValue,
LabelValue.GetLength());
                                dc.Rectangle(Graph1Pointer - (LabelLength.cx /2)-2,
InvValue1.CenterPoint().y - 7, Graph1Pointer + (LabelLength.cx /2)+1,
InvValue1.CenterPoint().y + 8);
                                dc.TextOutW(Graph1Pointer - (LabelLength.cx / 2),
(InvValue1.CenterPoint().y - 6), LabelValue, LabelValue.GetLength());
                        }
                }
                else if(GraphXStep == 3)
                {
                        if (Position < 7)
                        {
                                LabelLength = dc.GetTextExtent(LabelValue,
LabelValue.GetLength());
                                dc.Rectangle(Graph1Pointer + 2,
InvValue1.CenterPoint().y - 7, Graph1Pointer + (LabelLength.cx)+5,
InvValue1.CenterPoint().y + 8);
                                dc.TextOutW(Graph1Pointer + 4,
(InvValue1.CenterPoint().y - 6), LabelValue, LabelValue.GetLength());
                        }
                        else if(Position >= 7)
                        {
                                LabelLength = dc.GetTextExtent(LabelValue,
LabelValue.GetLength());
                                dc.Rectangle(Graph1Pointer - (LabelLength.cx /2)-2,
InvValue1.CenterPoint().y - 7, Graph1Pointer + (LabelLength.cx /2)+1,
InvValue1.CenterPoint().y + 8);
                                dc.TextOutW(Graph1Pointer - (LabelLength.cx / 2),
(InvValue1.CenterPoint().y - 6), LabelValue, LabelValue.GetLength());
```

```
                }
            }
            else if(GraphXStep >= 4)
            {
                if (Position < 4)
                {
                    LabelLength = dc.GetTextExtent(LabelValue,
LabelValue.GetLength());
                    dc.Rectangle(Graph1Pointer + 2,
InvValue1.CenterPoint().y - 7, Graph1Pointer + (LabelLength.cx)+5,
InvValue1.CenterPoint().y + 8);
                    dc.TextOutW(Graph1Pointer + 4,
(InvValue1.CenterPoint().y - 6), LabelValue, LabelValue.GetLength());
                }
                else if(Position >= 4)
                {
                    LabelLength = dc.GetTextExtent(LabelValue,
LabelValue.GetLength());
                    dc.Rectangle(Graph1Pointer - (LabelLength.cx /2)-2,
InvValue1.CenterPoint().y - 7, Graph1Pointer + (LabelLength.cx /2)+1,
InvValue1.CenterPoint().y + 8);
                    dc.TextOutW(Graph1Pointer - (LabelLength.cx / 2),
(InvValue1.CenterPoint().y - 6), LabelValue, LabelValue.GetLength());
                }
            }
            dc.SelectObject(def_font);
        }
        ////////////////////////////////
        // Graph 2 Plot Region
        ////////////////////////////////
        dc.SelectClipRgn(&hrgn2);
        xFormGraph2.eM11 = (FLOAT) 1.0;
    xFormGraph2.eM12 = (FLOAT) 0.0;
    xFormGraph2.eM21 = (FLOAT) 0.0;
    xFormGraph2.eM22 = (FLOAT) 1.0;
    xFormGraph2.eDy  = (FLOAT) 0.0;
    dc.SetWorldTransform(&xFormGraph2);
        ////////////////////////////////////////////////////////////////////////////
//////////////////////
        // Graph 2 Plot Data
        ////////////////////////////////////////////////////////////////////////////
//////////////////////
        dc.SelectObject(GreenPen);
        int PlotsB = 0;
        int PlotsOverlaid = 0;
        PlotsB = CompListCtrl.GetItemCount();
        PlotsOverlaid = DataListCtrl.GetItemCount();
        double CompMaxOverlaid = 0.00;
        double CompMaxNum = 0.00;
        double ScaleB = 0.00;
        ////////////////////////Compute Maximum Value For Graph
2//////////////////////
        for(int GetMaxNum = 0; GetMaxNum < PlotsB; GetMaxNum++)
        {
            size_t i;
            CString Data;
            char DataCh[8];
            double TestNum = 0.00;
            Data = CompListCtrl.GetItemText(GetMaxNum, 4);
            wcstombs_s(&i, DataCh, 7, Data, _TRUNCATE);
            TestNum = atof(DataCh);
            if(TestNum > CompMaxNum)
                CompMaxNum = TestNum;
        }
        ////////////////////////Compute Maximum Value For Graph 1
Overlaid//////////////////////
        for(int GetMaxNum = 0; GetMaxNum < PlotsOverlaid; GetMaxNum++)
        {
```

```
			size_t i;
			CString Data;
			char DataCh[8];
			double TestNum = 0.00;
			Data = DataListCtrl.GetItemText(GetMaxNum, 4);
			wcstombs_s(&i, DataCh, 7, Data, _TRUNCATE);
			TestNum = atof(DataCh);
			if(TestNum > CompMaxOverlaid)
				CompMaxOverlaid = TestNum;
		}

		if(CompMaxNum >= CompMaxOverlaid)
		{
			ScaleB = (InvGraph2.Height()) / (CompMaxNum);
		}
		else if(CompMaxNum < CompMaxOverlaid)
		{
			ScaleB = (InvGraph2.Height()) / (CompMaxOverlaid);
		}
		//////////////////////////End Compute Max Value For
Scale////////////////////////////////////
		POINT G2P[2000];
		POINT GOP[2000];
		int CompStep = 0;
		for(int GetPoints = 0; GetPoints < PlotsB; GetPoints++)
		{
			size_t i;
			CString Data;
			char DataCh[8];
			double TempNum = 0.00;
			Data = CompListCtrl.GetItemText(GetPoints, 4);
			wcstombs_s(&i, DataCh, 7, Data, _TRUNCATE);
			TempNum = atof(DataCh);
			TempNum = TempNum * ScaleB;
			G2P[GetPoints].y = InvGraph2.bottom - TempNum;
			G2P[GetPoints].x = InvGraph2.left + CompStep;
			CompStep += GraphXStep;
		}
		//Full Data
		if(PlotsB < 1997)
		{
			for(int Full = 0; Full < 3; Full++)
			{
				size_t i;
				CString Data;
				char DataCh[8];
				double TempNum = 0.00;
				Data = CompListCtrl.GetItemText(PlotsB - 1, 4);
				wcstombs_s(&i, DataCh, 7, Data, _TRUNCATE);
				TempNum = atof(DataCh);
				TempNum = TempNum * ScaleB;
				G2P[PlotsB + Full].y = InvGraph2.bottom - TempNum;
				G2P[PlotsB + Full].x = InvGraph2.left + CompStep + Full;
			}
		}
		////////////////////////////////////////////////////////////////////
		// Overlaid Graph 1 on Graph2 is TRUE Calculate Points
		////////////////////////////////////////////////////////////////////
		if(CompGraphOverlaid == TRUE)
		{
			int OverlaidStep = 0;
			for(int GetPoints = 0; GetPoints < PlotsOverlaid; GetPoints++)
			{
				size_t i;
				CString Data;
				char DataCh[8];
				double TempNum = 0.00;
				Data = DataListCtrl.GetItemText(GetPoints, 4);
```

```
                        wcstombs_s(&i, DataCh, 7, Data, _TRUNCATE);
                        TempNum = atof(DataCh);
                        TempNum = TempNum * ScaleB;
                        GOP[GetPoints].y = InvGraph2.bottom - TempNum;
                        GOP[GetPoints].x = InvGraph2.left + OverlaidStep;
                        OverlaidStep += GraphXStep;
                }
                //Full Data
                if(PlotsOverlaid < 1997)
                {
                        for(int Full = 0; Full < 3; Full++)
                        {
                                size_t i;
                                CString Data;
                                char DataCh[8];
                                double TempNum = 0.00;
                                Data = DataListCtrl.GetItemText(PlotsOverlaid - 1, 4);
                                wcstombs_s(&i, DataCh, 7, Data, _TRUNCATE);
                                TempNum = atof(DataCh);
                                TempNum = TempNum * ScaleB;
                                GOP[PlotsOverlaid + Full].y = InvGraph2.bottom -
TempNum;
                                GOP[PlotsOverlaid + Full].x = InvGraph2.left +
OverlaidStep + Full;
                        }
                }
        }
        ////////////////////////////////////////////////////////////////////////
        // END Overlaid Graph 1 on Graph2 is TRUE Calculate Points
        ////////////////////////////////////////////////////////////////////////
        if(GraphType == 1)
        {
                dc.MoveTo(G2P[0]);
                for(int PG = 0; PG < PlotsB; PG++)
                        dc.LineTo(G2P[PG]);
                if(CompGraphOverlaid == TRUE)
                {
                        dc.SelectObject(RedPen);
                        dc.MoveTo(GOP[0]);
                        for(int PO = 0; PO < PlotsOverlaid; PO++)
                                dc.LineTo(GOP[PO]);
                }
        }
        else if(GraphType == 2)
        {
                div_t ResultB;
                ResultB = div(PlotsB, 3);
                if(ResultB.rem == 0)
                        dc.PolyBezier(G2P, PlotsB + 1);
                else if(ResultB.rem == 1)
                        dc.PolyBezier(G2P, PlotsB +3);
                else if(ResultB.rem == 2)
                        dc.PolyBezier(G2P, PlotsB + 2);
                if(CompGraphOverlaid == TRUE)
                {
                        dc.SelectObject(RedPen);
                        div_t ResultO;
                        ResultO = div(PlotsOverlaid, 3);
                        if(ResultO.rem == 0)
                                dc.PolyBezier(GOP, PlotsOverlaid + 1);
                        else if(ResultO.rem == 1)
                                dc.PolyBezier(GOP, PlotsOverlaid +3);
                        else if(ResultO.rem == 2)
                                dc.PolyBezier(GOP, PlotsOverlaid + 2);
                }
        }
        else if(GraphType == 3)
        {
```

```
		if(CompGraphOverlaid == FALSE)
		{
			for(int FR = 0; FR < PlotsB; FR++)
			{
				CRect lpRect;
				lpRect.bottom = InvGraph2.bottom;
				lpRect.left = G2P[FR].x;
				if(GraphXStep > 1)
					lpRect.right = G2P[FR].x + GraphXStep - 1;
				else if(GraphXStep == 1)
					lpRect.right = G2P[FR].x + GraphXStep;
				lpRect.top = G2P[FR].y;
				dc.FillRect(&lpRect, &BlueBrush);
			}
		}
		else if(CompGraphOverlaid == TRUE)
		{
			dc.SelectObject(RedPen);
			int BarPoints = 0;
			if(PlotsB >= PlotsOverlaid)
				BarPoints = PlotsB;
			else if(PlotsB < PlotsOverlaid)
				BarPoints = PlotsOverlaid;
			for(int FR = 0; FR < BarPoints; FR++)
			{
				CRect lpOverRect, lpRect;
				lpOverRect.bottom = InvGraph2.bottom;
				lpOverRect.left = GOP[FR].x;
				lpRect.bottom = InvGraph2.bottom;
				lpRect.left = G2P[FR].x;
				if(GraphXStep > 1)
				{
					lpOverRect.right = GOP[FR].x + GraphXStep - 1;
					lpRect.right = G2P[FR].x + GraphXStep - 1;
				}
				else if(GraphXStep == 1)
				{
					lpOverRect.right = GOP[FR].x + GraphXStep;
					lpRect.right = G2P[FR].x + GraphXStep;
				}
				lpOverRect.top = GOP[FR].y;
				lpRect.top = G2P[FR].y;
				if(lpRect.top >= lpOverRect.top)
				{
					dc.FillRect(&lpOverRect, &RedBrush);
					dc.FillRect(&lpRect, &BlueBrush);
				}
				else if(lpRect.top < lpOverRect.top)
				{
					dc.FillRect(&lpRect, &BlueBrush);
					dc.FillRect(&lpOverRect, &RedBrush);
				}
			}
		}
	}
	//Grid Lines
	if(GridLines == TRUE)
	{
		if(GraphXStep >= 5)
		{
			int StepGridB = GraphXStep;
			dc.SelectObject(GridPen);
			for(int Grid = 0; Grid < PlotsB; Grid++)
			{
				if(Grid > 0)
				{
					dc.MoveTo(InvGraph2.left + StepGridB,
InvGraph2.bottom);
```

```
                                    dc.LineTo(InvGraph2.left + StepGridB,
InvGraph2.top);
                                    StepGridB+= GraphXStep;
                              }
                        }
                  }
            }
            ////////////////////////Moving Line/////////////////////////////////////
            if(CompListCtrl.GetItemCount() > 0)
            {
                  dc.SelectObject(CartPen);
                  dc.MoveTo(Graph2Pointer, InvGraph2.bottom);
                  dc.LineTo(Graph2Pointer, InvGraph2.top);
                  ////////////////////////////////////
                  //Graph 2 Place Moving Line Labe2
                  ////////////////////////////////////
                  dc.SelectObject(BlackPen);
                  CSize Label2Length;
                  CString Label2Value;
                  dc.SelectObject(&LabelFont);
                  dc.SetBkColor(RGB(255, 255, 204));
                  CSize LabelLength;
                  CString LabelValue;
                  int Position2 = 0;
                  for(int Clear = 0; Clear < CompListCtrl.GetItemCount(); Clear++)
                        CompListCtrl.SetItemState(Clear, FALSE, LVIS_SELECTED);
                  //DataListCtrl
                  for(int Get = 0; Get < CompListCtrl.GetItemCount(); Get++)
                  {
                        int PointTest = 0;
                        PointTest = G2P[Get].x;
                        if(Graph2Pointer == PointTest)
                        {
                              Label2Value = CompListCtrl.GetItemText(Get, 4);
                              CompListCtrl.SetItemState(Get,
LVIS_SELECTED|LVIS_FOCUSED , LVIS_SELECTED|LVIS_FOCUSED);
                              CompListCtrl.EnsureVisible(Get, FALSE);
                              Position2 = Get;
                              break;
                        }
                        else
                        {
                              CompListCtrl.SetSelectionMark(-1);
                              Label2Value = _T("0");
                        }
                  }
                  if (GraphXStep == 1)
                  {
                        if (Position2 < 20)
                        {
                              Label2Length = dc.GetTextExtent(Label2Value,
Label2Value.GetLength());
                              dc.Rectangle(Graph2Pointer + 2,
InvValue2.CenterPoint().y - 7, Graph2Pointer + (Label2Length.cx)+5,
InvValue2.CenterPoint().y + 8);
                              dc.TextOutW(Graph2Pointer + 4,
(InvValue2.CenterPoint().y - 6), Label2Value, Label2Value.GetLength());
                        }
                        else if(Position2 >= 20)
                        {
                              Label2Length = dc.GetTextExtent(Label2Value,
Label2Value.GetLength());
                              dc.Rectangle(Graph2Pointer - (Label2Length.cx /2)-2,
InvValue2.CenterPoint().y - 7, Graph2Pointer + (Label2Length.cx /2)+1,
InvValue2.CenterPoint().y + 8);
                              dc.TextOutW(Graph2Pointer - (Label2Length.cx / 2),
(InvValue2.CenterPoint().y - 6), Label2Value, Label2Value.GetLength());
                        }
```

```
		}
		else if (GraphXStep == 2)
		{
			if (Position2 < 11)
			{
				Label2Length = dc.GetTextExtent(Label2Value,
Label2Value.GetLength());
				dc.Rectangle(Graph2Pointer + 2,
InvValue2.CenterPoint().y - 7, Graph2Pointer + (Label2Length.cx)+5,
InvValue2.CenterPoint().y + 8);
				dc.TextOutW(Graph2Pointer + 4,
(InvValue2.CenterPoint().y - 6), Label2Value, Label2Value.GetLength());
			}
			else if(Position2 >= 11)
			{
				Label2Length = dc.GetTextExtent(Label2Value,
Label2Value.GetLength());
				dc.Rectangle(Graph2Pointer - (Label2Length.cx /2)-2,
InvValue2.CenterPoint().y - 7, Graph2Pointer + (Label2Length.cx /2)+1,
InvValue2.CenterPoint().y + 8);
				dc.TextOutW(Graph2Pointer - (Label2Length.cx / 2),
(InvValue2.CenterPoint().y - 6), Label2Value, Label2Value.GetLength());
			}
		}
		else if (GraphXStep == 3)
		{
			if (Position2 < 7)
			{
				Label2Length = dc.GetTextExtent(Label2Value,
Label2Value.GetLength());
				dc.Rectangle(Graph2Pointer + 2,
InvValue2.CenterPoint().y - 7, Graph2Pointer + (Label2Length.cx)+5,
InvValue2.CenterPoint().y + 8);
				dc.TextOutW(Graph2Pointer + 4,
(InvValue2.CenterPoint().y - 6), Label2Value, Label2Value.GetLength());
			}
			else if(Position2 >= 7)
			{
				Label2Length = dc.GetTextExtent(Label2Value,
Label2Value.GetLength());
				dc.Rectangle(Graph2Pointer - (Label2Length.cx /2)-2,
InvValue2.CenterPoint().y - 7, Graph2Pointer + (Label2Length.cx /2)+1,
InvValue2.CenterPoint().y + 8);
				dc.TextOutW(Graph2Pointer - (Label2Length.cx / 2),
(InvValue2.CenterPoint().y - 6), Label2Value, Label2Value.GetLength());
			}
		}

		else if(GraphXStep >= 4)
		{
			if (Position2 < 4)
			{
				Label2Length = dc.GetTextExtent(Label2Value,
Label2Value.GetLength());
				dc.Rectangle(Graph2Pointer + 2,
InvValue2.CenterPoint().y - 7, Graph2Pointer + (Label2Length.cx)+5,
InvValue2.CenterPoint().y + 8);
				dc.TextOutW(Graph2Pointer + 4,
(InvValue2.CenterPoint().y - 6), Label2Value, Label2Value.GetLength());
			}
			else if(Position2 >= 4)
			{
				Label2Length = dc.GetTextExtent(Label2Value,
Label2Value.GetLength());
				dc.Rectangle(Graph2Pointer - (Label2Length.cx /2)-2,
InvValue2.CenterPoint().y - 7, Graph2Pointer + (Label2Length.cx /2)+1,
InvValue2.CenterPoint().y + 8);
```

```
                        dc.TextOutW(Graph2Pointer - (Label2Length.cx / 2),
(InvValue2.CenterPoint().y - 6), Label2Value, Label2Value.GetLength());
                    }
            }
            dc.SelectObject(def_font);
      }
      //Delete Brushes
      SelectObject(dc, WhiteBrush2);
      DeleteObject(WhiteBrush2);
      SelectObject(dc, RedBrush);
      DeleteObject(RedBrush);
      //Delete Pens
      SelectObject(dc, RedPen);
      DeleteObject(RedPen);
      SelectObject(dc, BlackPen);
      DeleteObject(BlackPen);
      SelectObject(dc, GreenPen);
      DeleteObject(GreenPen);
      //Delete Font;
      TitleFont.DeleteObject();
      yFont.DeleteObject();
      LabelFont.DeleteObject();
      dc.FillRgn(&hrgn2, &RedBrush);
      XFORM Normal;
      Normal.eM11 = (FLOAT) 1.0;
   Normal.eM12 = (FLOAT) 0.0;
   Normal.eM21 = (FLOAT) 0.0;
   Normal.eM22 = (FLOAT) 1.0;
   Normal.eDy  = (FLOAT) 0.0;
      Normal.eDx  = (FLOAT) 0.0;
   dc.SetWorldTransform(&Normal);
      Pdc.BitBlt(0, 0, Window.Width(), Window.Height(), &dc, 0, 0, SRCCOPY);
      dc.SelectObject(pOldBitmap);
      ReleaseDC(&dc);
}

void CAdhesionMeterView::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
      // TODO: Add your message handler code here and/or call default
      int MinPos = 0;
      int MaxPos = 0;
      int CurPos = 0;
      if (pScrollBar->GetDlgCtrlID() == IDC_GraphScroll)
      {
            GScrollCtrl.GetScrollRange(&MinPos, &MaxPos);
            CurPos = GScrollCtrl.GetScrollPos();
            switch (nSBCode)
            {
                  case SB_LINELEFT:
                        {
                              if(CurPos > MinPos)
                                    CurPos--;
                        }
                  break;
                  case SB_LINERIGHT:
                        {
                              if(CurPos < MaxPos)
                                    CurPos++;
                        }
                  break;
                  case SB_PAGELEFT:
                        {
                              if (CurPos > MinPos)
                                    CurPos = max(MinPos, CurPos - 10);
                        }
                  break;
                  case SB_PAGERIGHT:
                        {
```

```
                                        if(CurPos < MaxPos)
                                                CurPos = min(MaxPos, CurPos + 10);
                                }
                        break;
                        case SB_THUMBTRACK:
                                CurPos = nPos;
                        break;
                }
                GScrollCtrl.SetScrollPos(CurPos);
                xFormGraph1.eDx = (FLOAT) -CurPos;
                InvalidateRect(InvGraph1);
                InvalidateRect(InvValue1);

        }
        else if(pScrollBar->GetDlgCtrlID() == IDC_GraphScroll2)
        {
                CScrollCtrl.GetScrollRange(&MinPos, &MaxPos);
                CurPos = CScrollCtrl.GetScrollPos();
                switch (nSBCode)
                {
                        case SB_LINELEFT:
                                {
                                        if(CurPos > MinPos)
                                                CurPos--;
                                }
                        break;
                        case SB_LINERIGHT:
                                {
                                        if(CurPos < MaxPos)
                                                CurPos++;
                                }
                        break;
                        case SB_PAGELEFT:
                                {
                                        if (CurPos > MinPos)
                                                CurPos = max(MinPos, CurPos - 10);
                                }
                        break;
                        case SB_PAGERIGHT:
                                {
                                        if(CurPos < MaxPos)
                                                CurPos = min(MaxPos, CurPos + 10);
                                }
                        break;
                        case SB_THUMBTRACK:
                                CurPos = nPos;
                        break;
                }
                CScrollCtrl.SetScrollPos(CurPos);
                xFormGraph2.eDx = (FLOAT) -CurPos;
                InvalidateRect(InvGraph2);
                InvalidateRect(InvValue2);
        }
        CFormView::OnHScroll(nSBCode, nPos, pScrollBar);
}

void CAdhesionMeterView::DisplayInputReport(void)
{
        CHAR   ReceivedByte1;
        CHAR   ReceivedByte2;
        CHAR   ReceivedByte3;
        CHAR   ReceivedByte4;
        CHAR   ReceivedByte5;
        CHAR   ReceivedByte6;
        CHAR   ReceivedByte7;
        CHAR   ReceivedByte8;
        CHAR   ReceivedByte9;
        CHAR   ReceivedByte10;
```

```
	CHAR	ReceivedByte11;
	CHAR	ReceivedByte12;
	CHAR	ReceivedByte13;

	ReceivedByte1 = InputReport[0];
	ReceivedByte2 = InputReport[1];
	ReceivedByte3 = InputReport[2];
	ReceivedByte4 = InputReport[3];
	ReceivedByte5 = InputReport[4];
	ReceivedByte6 = InputReport[5];
	ReceivedByte7 = InputReport[6];
	ReceivedByte8 = InputReport[7];
	ReceivedByte9 = InputReport[8];
	ReceivedByte10 = InputReport[9];
	ReceivedByte11 = InputReport[10];
	ReceivedByte12 = InputReport[11];
	ReceivedByte13 = InputReport[12];
	CString LoadWeight;
	int Items = 0;
	Items = DataListCtrl.GetItemCount();
	LoadWeight = ReceivedByte1;
	LoadWeight += ReceivedByte2;
	LoadWeight += ReceivedByte3;
	LoadWeight += ReceivedByte4;
	LoadWeight += ReceivedByte5;
	LoadWeight += ReceivedByte6;
	LoadWeight += ReceivedByte7;
	LoadWeight += ReceivedByte8;
	SYSTEMTIME SampleTime;
	GetLocalTime(&SampleTime);

	CString Number;
	CString Date;
	CString Time;
	Date.Format(_T("%d/%d/%d"), SampleTime.wDay, SampleTime.wMonth,
SampleTime.wYear);
	Time.Format(_T("%d:%d:%d"), SampleTime.wHour, SampleTime.wMinute,
SampleTime.wSecond);
	Number.Format(_T("%d"), Items +1);
	//Convert Units To The Required Format Before Displaying Then
	if(UnitsType != 1)
	{
		if(UnitsType == 2)
		{
			size_t DS;
			float Value = 0.00;
			char TempValue[8];
			wcstombs_s(&DS, TempValue, 7, LoadWeight, _TRUNCATE);
			Value = atof(TempValue);
			Value = Value / 1000;
			CString NewValue;
			NewValue.Format(_T("%2.3f"), Value);
			LoadWeight = NewValue;
		}
		else if(UnitsType == 3)
		{
			size_t DS;
			float Value = 0.00;
			char TempValue[8];
			wcstombs_s(&DS, TempValue, 7, LoadWeight, _TRUNCATE);
			Value = atof(TempValue);
			Value = Value / 1000 * 9.81;
			CString NewValue;
			NewValue.Format(_T("%2.3f"), Value);
			LoadWeight = NewValue;
		}
	}
	//End Convert Units To Required Format
```

```
	// Test Capture Mode 1=Continuous, 2=Change
		if (CaptureMode == 1)
		{
			DataListCtrl.InsertItem(Items, Number);
			DataListCtrl.SetItem(Items, 1, 1, Date, NULL, 1, 1, 1);
			DataListCtrl.SetItem(Items, 2, 1, Time, NULL, 1, 1, 1);
			DataListCtrl.SetItem(Items, 3, 1, _T("25"), NULL, 1, 1, 1);
			DataListCtrl.SetItem(Items, 4, 1, LoadWeight, NULL, 1, 1, 1);
			DataListCtrl.EnsureVisible(Items, FALSE);
			InvalidateRect(InvGraph1);
			if(CompGraphOverlaid == TRUE)
				InvalidateRect(InvGraph2);
		}
		else if(CaptureMode == 2)
		{
			if(OldCaptureValue != LoadWeight)
			{
				DataListCtrl.InsertItem(Items, Number);
				DataListCtrl.SetItem(Items, 1, 1, Date, NULL, 1, 1, 1);
				DataListCtrl.SetItem(Items, 2, 1, Time, NULL, 1, 1, 1);
				DataListCtrl.SetItem(Items, 3, 1, _T("25"), NULL, 1, 1,
1);
				DataListCtrl.SetItem(Items, 4, 1, LoadWeight, NULL, 1,
1, 1);
				DataListCtrl.EnsureVisible(Items, FALSE);
				InvalidateRect(InvGraph1);
				if(CompGraphOverlaid == TRUE)
					InvalidateRect(InvGraph2);
				OldCaptureValue = LoadWeight;
			}
		}
	// End Capture Mode

}
void CAdhesionMeterView::OnTimer(UINT_PTR nIDEvent)
{
	ReadReport();
	CFormView::OnTimer(nIDEvent);
}

BOOL CAdhesionMeterView::OnEraseBkgnd(CDC* pDC)
{
	// TODO: Add your message handler code here and/or call default

	return TRUE;
}

void CAdhesionMeterView::OnGraphGrid()
{
//	CMenu* subMenu = mmenu->GetSubMenu(2);
//	CString T;
	AfxGetMainWnd()->PostMessageW(WM_MENU_GRIDLINES, NULL, NULL);
	if(!GridLines)
	{
		GridLines = TRUE;
	}
	else
	{
		GridLines = FALSE;
	}
	InvalidateRect(InvGraph1);
	InvalidateRect(InvGraph2);
}
////////////////////////////////////////////////////////////////////////////
// User has turned the grid lines ON in the Properties Window
////////////////////////////////////////////////////////////////////////////
LRESULT CAdhesionMeterView::OnPropertiesGridOn(WPARAM wp, LPARAM lp)
{
```

```
	AfxGetMainWnd()->PostMessageW(WM_MENU_GRIDLINES, NULL, NULL);
	GridLines = TRUE;
	InvalidateRect(InvGraph1);
	InvalidateRect(InvGraph2);
	return 0L;
}
////////////////////////////////////////////////////////////////////////////
// User has turned the grid lines OFF in the properties window
////////////////////////////////////////////////////////////////////////////
LRESULT CAdhesionMeterView::OnPropertiesGridOff(WPARAM wp, LPARAM lp)
{
	AfxGetMainWnd()->PostMessageW(WM_MENU_GRIDLINES, NULL, NULL);
	GridLines = FALSE;
	InvalidateRect(InvGraph1);
	InvalidateRect(InvGraph2);
	return 0L;
}
////////////////////////////////////////////////////////////////////////////
// User has selected a different graph in the properties window
////////////////////////////////////////////////////////////////////////////
LRESULT CAdhesionMeterView::OnPropertiesGraphType(WPARAM wp, LPARAM lp)
{
	if(wp == 1)
		GraphType = 1;
	else if(wp == 2)
		GraphType = 2;
	else if(wp == 3)
		GraphType = 3;
	InvalidateRect(InvGraph1);
	InvalidateRect(InvGraph2);
	return 0L;
}
////////////////////////////////////////////////////////////////////////////
// User has changed the XStep in the Properties Window
////////////////////////////////////////////////////////////////////////////
LRESULT CAdhesionMeterView::OnPropertiesGraphXStep(WPARAM wp, LPARAM lp)
{
	GraphXStep = wp;
	InvalidateRect(InvGraph1);
	InvalidateRect(InvGraph2);
	return 0L;
}
////////////////////////////////////////////////////////////////////////////
// User has changed the Mode in the Properties Window
////////////////////////////////////////////////////////////////////////////
LRESULT CAdhesionMeterView::OnPropertiesModeChange(WPARAM wp, LPARAM lp)
{
	if(wp == 1)
		CaptureMode = 1;
	else if(wp == 2)
		CaptureMode = 2;
	return 0L;
}
////////////////////////////////////////////////////////////////////////////
// User has changed the Comparison Graph Type, Single or Overlaid
////////////////////////////////////////////////////////////////////////////
LRESULT CAdhesionMeterView::OnPropertiesCompGraphOver(WPARAM wp, LPARAM lp)
{
	if(wp == 1)
		CompGraphOverlaid = FALSE;
	else if(wp == 2)
		CompGraphOverlaid = TRUE;
	InvalidateRect(InvGraph1);
	InvalidateRect(InvGraph2);
	return 0L;
}
////////////////////////////////////////////////////////////////////////////
// User has changed the Capture Time in the Properties Window
```

```
////////////////////////////////////////////////////////////////////////////
LRESULT CAdhesionMeterView::OnPropertiesTimeChange(WPARAM wp, LPARAM lp)
{
	CaptureTime = wp;
	return 0L;
}
LRESULT CAdhesionMeterView::OnPropertiesUnits(WPARAM wp, LPARAM lp)
{
	int DataListCounter = 0;
	int CompListCounter = 0;
	DataListCounter = DataListCtrl.GetItemCount();
	CompListCounter = CompListCtrl.GetItemCount();
	if(wp == 1)
	{
		if(UnitsType != 1)
		{
			for(int Rep = 0; Rep < DataListCounter; Rep++)
			{
				size_t DS;
				char TempValue[8];
				float Value = 0;
				CString DataValue;
				DataValue = DataListCtrl.GetItemText(Rep, 4);
				wcstombs_s(&DS, TempValue, 7, DataValue, _TRUNCATE);
				Value = atof(TempValue);
				if(UnitsType == 2)
					Value = Value * 1000;
				else if(UnitsType == 3)
					Value = (Value / 9.81) * 1000;
				CString NewValue;
				NewValue.Format(_T("%2.0f"), Value);
				DataListCtrl.SetItemText(Rep, 4, NewValue);
			}
			for(int Rep = 0; Rep < CompListCounter; Rep++)
			{
				size_t DS;
				char TempValue[8];
				float Value = 0;
				CString DataValue;
				DataValue = CompListCtrl.GetItemText(Rep, 4);
				wcstombs_s(&DS, TempValue, 7, DataValue, _TRUNCATE);
				Value = atof(TempValue);
				if(UnitsType == 2)
					Value = Value * 1000;
				else if(UnitsType == 3)
					Value = (Value / 9.81) * 1000;
				CString NewValue;
				NewValue.Format(_T("%2.0f"), Value);
				CompListCtrl.SetItemText(Rep, 4, NewValue);
			}
		}
	}
	else if(wp == 2)
	{
		if(UnitsType != 2)
		{
			for(int Rep = 0; Rep < DataListCounter; Rep++)
			{
				size_t DS;
				char TempValue[8];
				float Value = 0;
				CString DataValue;
				DataValue = DataListCtrl.GetItemText(Rep, 4);
				wcstombs_s(&DS, TempValue, 7, DataValue, _TRUNCATE);
				Value = atof(TempValue);
				if(UnitsType == 1)
					Value = Value / 1000;
				else if(UnitsType == 3)
```

```
                        Value = Value / 9.81;
                    CString NewValue;
                    NewValue.Format(_T("%2.3f"), Value);
                    DataListCtrl.SetItemText(Rep, 4, NewValue);
                }
                for(int Rep = 0; Rep < CompListCounter; Rep++)
                {
                    size_t DS;
                    char TempValue[8];
                    float Value = 0;
                    CString DataValue;
                    DataValue = CompListCtrl.GetItemText(Rep, 4);
                    wcstombs_s(&DS, TempValue, 7, DataValue, _TRUNCATE);
                    Value = atof(TempValue);
                    if(UnitsType == 1)
                        Value = Value / 1000;
                    else if(UnitsType == 3)
                        Value = Value / 9.81;
                    CString NewValue;
                    NewValue.Format(_T("%2.3f"), Value);
                    CompListCtrl.SetItemText(Rep, 4, NewValue);
                }
            }
        }
        else if(wp == 3)
        {
            if(UnitsType != 3)
            {
                for(int Rep = 0; Rep < DataListCounter; Rep++)
                {
                    size_t DS;
                    char TempValue[8];
                    float Value = 0;
                    CString DataValue;
                    DataValue = DataListCtrl.GetItemText(Rep, 4);
                    wcstombs_s(&DS, TempValue, 7, DataValue, _TRUNCATE);
                    Value = atof(TempValue);
                    if(UnitsType == 1)
                        Value = Value / 1000 * 9.81;
                    else if(UnitsType == 2)
                        Value = Value * 9.81;
                    CString NewValue;
                    NewValue.Format(_T("%2.3f"), Value);
                    DataListCtrl.SetItemText(Rep, 4, NewValue);
                }
                for(int Rep = 0; Rep < CompListCounter; Rep++)
                {
                    size_t DS;
                    char TempValue[8];
                    float Value = 0;
                    CString DataValue;
                    DataValue = CompListCtrl.GetItemText(Rep, 4);
                    wcstombs_s(&DS, TempValue, 7, DataValue, _TRUNCATE);
                    Value = atof(TempValue);
                    if(UnitsType == 1)
                        Value = Value / 1000 * 9.81;
                    else if(UnitsType == 2)
                        Value = Value * 9.81;
                    CString NewValue;
                    NewValue.Format(_T("%2.3f"), Value);
                    CompListCtrl.SetItemText(Rep, 4, NewValue);
                }
            }
        }
        UnitsType = wp;
        return 0L;
}
////////////////////////////////////////////////////////////////////////////
```

```
// Mouse has Moved:- User for graph line poitner
////////////////////////////////////////////////////////////////////////////
void CAdhesionMeterView::OnMouseMove(UINT nFlags, CPoint point)
{
	if(InvGraph1.PtInRect(point) == TRUE && nFlags == MK_CONTROL)
	{
		if(DataListCtrl.GetItemCount() > 0)
		{
			int PX = 0;
			int PutPoint = 0;
			div_t PointDiv;
			PX = point.x - InvGraph1.left+ (xFormGraph1.eDx * -1);
			PointDiv = div(PX , GraphXStep);
			PutPoint = PointDiv.quot * GraphXStep + InvGraph1.left;
			Graph1Pointer = PutPoint ;
			DataListCtrl.SetFocus();
			InvalidateRect(InvGraph1);
			InvalidateRect(InvValue1);
		}
	}
	else if(InvGraph2.PtInRect(point) == TRUE && nFlags == MK_CONTROL)
	{
		if(CompListCtrl.GetItemCount() > 0)
		{
			int PX = 0;
			int PutPoint = 0;
			div_t PointDiv;
			PX = point.x - InvGraph2.left+ (xFormGraph2.eDx * -1);
			PointDiv = div(PX , GraphXStep);
			PutPoint = PointDiv.quot * GraphXStep + InvGraph2.left;
			Graph2Pointer = PutPoint ;
			CompListCtrl.SetFocus();
			InvalidateRect(InvGraph2);
			InvalidateRect(InvValue2);
		}
	}
	CFormView::OnMouseMove(nFlags, point);
}
////////////////////////////////////////////////////////////////////////////
// User has clicked on the Connect Button in the Caption Bar
////////////////////////////////////////////////////////////////////////////
void CAdhesionMeterView::CaptionButton(void)
{
	if (FindLoadCell() == TRUE)
	{
		AfxGetMainWnd()->SendMessage(WM_CAPTION_BAR_CONNECTED, 1, NULL);
		Timer = SetTimer(1, CaptureTime, NULL);
	}
}
////////////////////////////////////////////////////////////////////////////
// User has clicked on the Disconnect Button In the Caption Bar
////////////////////////////////////////////////////////////////////////////
void CAdhesionMeterView::DisconnectCaptionButton(void)
{
	AfxGetMainWnd()->SendMessage(WM_CAPTION_BAR_CONNECTED, 2, NULL);
	KillTimer(Timer);
	if(DeviceDetected == TRUE)
	{
		FlushFileBuffers(ReadHandle);
		WaitForSingleObject(ThreadHandle, INFINITE);
		CloseHandle(DeviceHandle);
		CloseHandle(ReadHandle);
		CloseHandle(ThreadHandle);
		WaitForSingleObject(hEventObject, 300);
		InvalidateRect(InvGraph1);
	}
}
////////////////////////////////////////////////////////////////////////////
```

```
// Save Data To File
////////////////////////////////////////////////////////////////////////////
void CAdhesionMeterView::SaveData()
{
	CFile SaveDataFile;
	CAdhesionMeterDoc *pDoc = GetDocument();
	CString FileName = pDoc->GetTitle();
	FileName += _T(".AMF");
	TCHAR strFilter[] = _T("Adhesion Meter Files (*.AMF)|*.AMF||");
	CFileDialog FileDialog(FALSE, _T("*.AMF"), FileName, OFN_OVERWRITEPROMPT,
strFilter);
	int Lines = 0;
	Lines = DataListCtrl.GetItemCount();
	if(FileDialog.DoModal() == IDOK)
	{
		SaveDataFile.Open(FileDialog.GetFileName(), CFile::modeCreate |
CFile::modeWrite);
		CArchive ar(&SaveDataFile, CArchive::store);
		for(int Save = 0; Save < Lines; Save++)
		{
			CString SampleNumber, Sample, Date,	Time, Temp,	Load;
			SampleNumber.Format(_T("%d"), Save+1);
			Sample = SampleNumber;
			Date = DataListCtrl.GetItemText(Save, 1);
			Time = DataListCtrl.GetItemText(Save, 2);
			Temp = DataListCtrl.GetItemText(Save, 3);
			Load = DataListCtrl.GetItemText(Save, 4);
			//Make Sure To Save In Grams ONLY!
			if(UnitsType != 1)
			{
				size_t DS;
				char TempValue[8];
				float Value = 0.00;
				wcstombs_s(&DS, TempValue, 7, Load, _TRUNCATE);
				Value = atof(TempValue);
				if(UnitsType == 2)
					Value = Value * 1000;
				else if(UnitsType == 3)
					Value = (Value / 9.81) * 1000;
				CString NewValue;
				NewValue.Format(_T("%2.0f"), Value);
				Load = NewValue;
			}
			//End Convert To Grams
			if(Save == 0)
				pDoc->Output = Sample + _T(",") + Date + _T(",") + Time
+ _T(",") + Temp + _T(",") + Load;
			else if(Save > 0)
				pDoc->Output = _T("\n") + Sample + _T(",") + Date +
_T(",") + Time + _T(",") + Temp + _T(",") + Load;

			pDoc->Serialize(ar);
		}
		ar.Close();
		SaveDataFile.Close();

		AfxGetMainWnd()->PostMessage(WM_UPDATE_FILE_VIEW, NULL, NULL);
	}
}
////////////////////////////////////////////////////////////////////////////
// Read Data From File
////////////////////////////////////////////////////////////////////////////

void CAdhesionMeterView::OpenData()
{
	CWinApp* pApp = AfxGetApp();
	CFile OpenDataFile;
	TCHAR strFilter[] = _T("Adhesion Meter Files (*.AMF)|*.AMF||");
```

```
	CFileDialog OpenDialog(TRUE, _T("*.AMF"), NULL, OFN_FILEMUSTEXIST |
OFN_HIDEREADONLY, strFilter);
	if(OpenDialog.DoModal() == IDOK)
	{

		CAdhesionMeterDoc *pDoc = GetDocument();

		int SampleNumber = 0;
		CString DataLine = _T("");
		if(OpenDataFile.Open(OpenDialog.GetFileName(), CFile::modeNoTruncate
| CFile::modeRead) == FALSE)
		{
			MessageBox(_T("Failed To Open File."), _T("File Open Error"),
MB_ICONERROR|MB_OK);
			return;
		}
		DataListCtrl.DeleteAllItems();
		CArchive ar(&OpenDataFile, CArchive::load);
		if(!ar.ReadString(DataLine))
			return;
		do
		{
			if(DataLine.GetLength() == 0)
				continue;
			CString LineText, PutSample, PutDate, PutTime, PutTemp,
PutLoad;
			LineText = DataLine;
			AfxExtractSubString(PutSample, LineText, 0, ',');
			AfxExtractSubString(PutDate, LineText, 1, ',');
			AfxExtractSubString(PutTime, LineText, 2, ',');
			AfxExtractSubString(PutTemp, LineText, 3, ',');
			AfxExtractSubString(PutLoad, LineText, 4, ',');
			//Convert Units To The Required Format Before Displaying Then
			if(UnitsType != 1)
			{
				if(UnitsType == 2)
				{
					size_t DS;
					float Value = 0.00;
					char TempValue[8];
					wcstombs_s(&DS, TempValue, 7, PutLoad, _TRUNCATE);
					Value = atof(TempValue);
					Value = Value / 1000;
					CString NewValue;
					NewValue.Format(_T("%2.3f"), Value);
					PutLoad = NewValue;
				}
				else if(UnitsType == 3)
				{
					size_t DS;
					float Value = 0.00;
					char TempValue[8];
					wcstombs_s(&DS, TempValue, 7, PutLoad, _TRUNCATE);
					Value = atof(TempValue);
					Value = Value / 1000 * 9.81;
					CString NewValue;
					NewValue.Format(_T("%2.3f"), Value);
					PutLoad = NewValue;
				}
			}
			//End Convert Units To Required Format
			DataListCtrl.InsertItem(SampleNumber, PutSample);
			DataListCtrl.SetItem(SampleNumber, 1, 1, PutDate, NULL, 1, 1,
1);
			DataListCtrl.SetItem(SampleNumber, 2, 1, PutTime, NULL, 1, 1,
1);
			DataListCtrl.SetItem(SampleNumber, 3, 1, PutTemp, NULL, 1, 1,
1);
```

```
                        DataListCtrl.SetItem(SampleNumber, 4, 1, PutLoad, NULL, 1, 1,
1);
                        SampleNumber++;
                } while (ar.ReadString(DataLine));
                ar.Close();
                OpenDataFile.Close();
                InvalidateRect(InvGraph1);
                InvalidateRect(InvValue1);
                if(CompGraphOverlaid == TRUE)
                {
                        InvalidateRect(InvGraph2);
                        InvalidateRect(InvValue2);
                }
        }
}
////////////////////////////////////////////////////////////////////////
// Open Comparison File
////////////////////////////////////////////////////////////////////////
LRESULT CAdhesionMeterView::OnOpenCompFile(WPARAM wp, LPARAM lp)
{
                CFile OpenDataFile;
                CAdhesionMeterDoc *pDoc = GetDocument();

                int SampleNumber = 0;
                CString DataLine = _T("");
                if(OpenDataFile.Open(CompareFileName, CFile::modeNoTruncate  |
CFile::modeRead) == FALSE)
                {
                        MessageBox(_T("Failed To Open File."), _T("File Open Error"),
MB_ICONERROR|MB_OK);
                        return 0L;
                }
                CompListCtrl.DeleteAllItems();
                CArchive ar(&OpenDataFile, CArchive::load);
                if(!ar.ReadString(DataLine))
                        return 0L;
                do
                {
                        if(DataLine.GetLength() == 0)
                                continue;
                        CString LineText, PutSample, PutDate, PutTime, PutTemp,
PutLoad;
                        LineText = DataLine;
                        AfxExtractSubString(PutSample, LineText, 0, ',');
                        AfxExtractSubString(PutDate, LineText, 1, ',');
                        AfxExtractSubString(PutTime, LineText, 2, ',');
                        AfxExtractSubString(PutTemp, LineText, 3, ',');
                        AfxExtractSubString(PutLoad, LineText, 4, ',');
                        //Convert Units To The Required Format Before Displaying Then
                        if(UnitsType != 1)
                        {
                                if(UnitsType == 2)
                                {
                                        size_t DS;
                                        float Value = 0.00;
                                        char TempValue[8];
                                        wcstombs_s(&DS, TempValue, 7, PutLoad, _TRUNCATE);
                                        Value = atof(TempValue);
                                        Value = Value / 1000;
                                        CString NewValue;
                                        NewValue.Format(_T("%2.3f"), Value);
                                        PutLoad = NewValue;
                                }
                                else if(UnitsType == 3)
                                {
                                        size_t DS;
                                        float Value = 0.00;
                                        char TempValue[8];
```

```
                                wcstombs_s(&DS, TempValue, 7, PutLoad, _TRUNCATE);
                                Value = atof(TempValue);
                                Value = Value / 1000 * 9.81;
                                CString NewValue;
                                NewValue.Format(_T("%2.3f"), Value);
                                PutLoad = NewValue;
                        }
                }
                //End Convert Units To Required Format
                CompListCtrl.InsertItem(SampleNumber, PutSample);
                CompListCtrl.SetItem(SampleNumber, 1, 1, PutDate, NULL, 1, 1,
1);
                CompListCtrl.SetItem(SampleNumber, 2, 1, PutTime, NULL, 1, 1,
1);
                CompListCtrl.SetItem(SampleNumber, 3, 1, PutTemp, NULL, 1, 1,
1);
                CompListCtrl.SetItem(SampleNumber, 4, 1, PutLoad, NULL, 1, 1,
1);
                SampleNumber++;
        } while (ar.ReadString(DataLine));
        ar.Close();
        OpenDataFile.Close();
        InvalidateRect(InvGraph2);
        InvalidateRect(InvValue2);
    return 0L;
}
////////////////////////////////////////////////////////////////////////
// Open Data File From CFileView
////////////////////////////////////////////////////////////////////////
LRESULT CAdhesionMeterView::OnOpenDataFile(WPARAM wp, LPARAM lp)
{
    CFile OpenDataFile;
        CAdhesionMeterDoc *pDoc = GetDocument();

        int SampleNumber = 0;
        CString DataLine = _T("");
        if(OpenDataFile.Open(OpenDataFileName, CFile::modeNoTruncate  |
CFile::modeRead) == FALSE)
        {
                MessageBox(_T("Failed To Open File."), _T("File Open Error"),
MB_ICONERROR|MB_OK);
                return 0L;
        }
        DataListCtrl.DeleteAllItems();
        CArchive ar(&OpenDataFile, CArchive::load);
        if(!ar.ReadString(DataLine))
                return 0L;
        do
        {
                if(DataLine.GetLength() == 0)
                        continue;
                CString LineText, PutSample, PutDate, PutTime, PutTemp,
PutLoad;
                LineText = DataLine;
                AfxExtractSubString(PutSample, LineText, 0, ',');
                AfxExtractSubString(PutDate, LineText, 1, ',');
                AfxExtractSubString(PutTime, LineText, 2, ',');
                AfxExtractSubString(PutTemp, LineText, 3, ',');
                AfxExtractSubString(PutLoad, LineText, 4, ',');
                //Convert Units To The Required Format Before Displaying Then
                if(UnitsType != 1)
                {
                        if(UnitsType == 2)
                        {
                                size_t DS;
                                float Value = 0.00;
                                char TempValue[8];
                                wcstombs_s(&DS, TempValue, 7, PutLoad, _TRUNCATE);
```

```
                                        Value = atof(TempValue);
                                        Value = Value / 1000;
                                        CString NewValue;
                                        NewValue.Format(_T("%2.3f"), Value);
                                        PutLoad = NewValue;
                                }
                                else if(UnitsType == 3)
                                {
                                        size_t DS;
                                        float Value = 0.00;
                                        char TempValue[8];
                                        wcstombs_s(&DS, TempValue, 7, PutLoad, _TRUNCATE);
                                        Value = atof(TempValue);
                                        Value = Value / 1000 * 9.81;
                                        CString NewValue;
                                        NewValue.Format(_T("%2.3f"), Value);
                                        PutLoad = NewValue;
                                }
                        }
                        //End Convert Units To Required Format
                        DataListCtrl.InsertItem(SampleNumber, PutSample);
                        DataListCtrl.SetItem(SampleNumber, 1, 1, PutDate, NULL, 1, 1,
1);
                        DataListCtrl.SetItem(SampleNumber, 2, 1, PutTime, NULL, 1, 1,
1);
                        DataListCtrl.SetItem(SampleNumber, 3, 1, PutTemp, NULL, 1, 1,
1);
                        DataListCtrl.SetItem(SampleNumber, 4, 1, PutLoad, NULL, 1, 1,
1);
                        SampleNumber++;
                } while (ar.ReadString(DataLine));
                ar.Close();
                OpenDataFile.Close();
                InvalidateRect(InvGraph1);
                InvalidateRect(InvValue1);
                if(CompGraphOverlaid == TRUE)
                {
                        InvalidateRect(InvGraph2);
                        InvalidateRect(InvValue2);
                }
        return 0L;
}
////////////////////////////////////////////////////////////////////////////
// Printing and Print Preview Functions
////////////////////////////////////////////////////////////////////////////

BOOL CAdhesionMeterView::OnPreparePrinting(CPrintInfo* pInfo)
{
        pInfo->SetMaxPage(1);

        pInfo->m_pPD->m_pd.Flags &= ~PD_NOSELECTION;
        pInfo->m_pPD->m_pd.hInstance = AfxGetInstanceHandle();
        // TODO:  call DoPreparePrinting to invoke the Print dialog box
        return DoPreparePrinting(pInfo);
        //return CFormView::OnPreparePrinting(pInfo);
}
```