ARMY RESEARCH LABORATORY

Digitizing Printed Graphs Using C++

by Robert J. Yager

ARL-TN-577 **September 2013**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TN-577 **September 2013**

Digitizing Printed Graphs Using C++

Robert J. Yager
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
September 2013	Final	1–30 June 2013

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Digitizing Printed Graphs Using C++	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)	5d. PROJECT NUMBER
Robert J. Yager	AH80
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
U.S. Army Research Laboratory ATTN: RDRL-WML-A Aberdeen Proving Ground, MD 21005-5066	ARL-TN-577

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Occasionally, tabulated data is required where only graphical data is available. In such cases, obtaining tabulated data by hand can be time consuming. The C++ code described in this report provides a method for converting from graphical to tabulated data. It records mouse clicks to capture axis and graph information and then performs a transformation from screen coordinates to graph coordinates. The origin of the graph need not be present, and the axes can either be linear or logarithmic. Axes that are rotated and/or skewed are accounted for.

15. SUBJECT TERMS

graph, digitize, linear, logarithm, logarithmic, C++

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Robert J. Yager
Unclassified	Unclassified	Unclassified	UU	24	19b. TELEPHONE NUMBER (*Include area code*) 410-278-6689

Contents

List of Figures

List of Tables

Acknowledgments

The author would like to thank Mr. Luke Strohm of the U.S. Army Research Laboratory's Weapons and Materials Research Directorate. Mr. Strohm provided technical and editorial recommendations that improved the quality of this report.

1. Introduction

Occasionally, tabulated data is required where only graphical data is available. In such cases, obtaining tabulated data by hand can be time consuming. The C++ code described in this report provides a method for converting from graphical to tabulated data. It records mouse clicks to capture axis and graph information and then performs a transformation from screen coordinates to graph coordinates. The origin of the graph need not be present, and the axes can either be linear or logarithmic. Axes that are rotated and/or skewed are accounted for.

The C++ programming language does not natively support capturing mouse clicks. Thus, the code that is presented in this report is necessarily platform specific. It has been written to be compatible with Microsoft's Visual C++ compiler. Converting the code to be compilable on a different platform should be relatively straightforward—all Microsoft-specific code is limited to the GetLClick() function, which is described in section 6.

The accuracy of the conversion from screen coordinates to graphical coordinates is dependent upon how closely the integer values that result from capturing screen coordinates match the real numbers represented by graphs. As a result, two pieces of hardware will help to maximize the accuracy of the conversion. The first is a high-resolution monitor (the higher the better). The second is a variable dot-per-inch (DPI) mouse. A high-resolution monitor is useful because the extra pixels allow integers to be chosen that are closer approximations to real numbers, when compared to lower-resolution monitors. A variable DPI mouse is useful because it allows for precise control of the mouse pointer.

2. Derivation of Transformation

Given a point, $\vec{p}$, and a pair of axes, $\hat{x}$ and $\hat{y}$, that are each defined by two points, ($\vec{x}_1, \vec{x}_2$) and ($\vec{y}_1, \vec{y}_2$), where the locations of $\vec{p}$, $\vec{x}_1$, $\vec{x}_2$, $\vec{y}_1$, and $\vec{y}_2$ are all known in screen coordinates, we wish to find $\vec{p}$ with respect to the $\hat{x}$ and $\hat{y}$ axes. Figure 1 presents a visual of the problem.

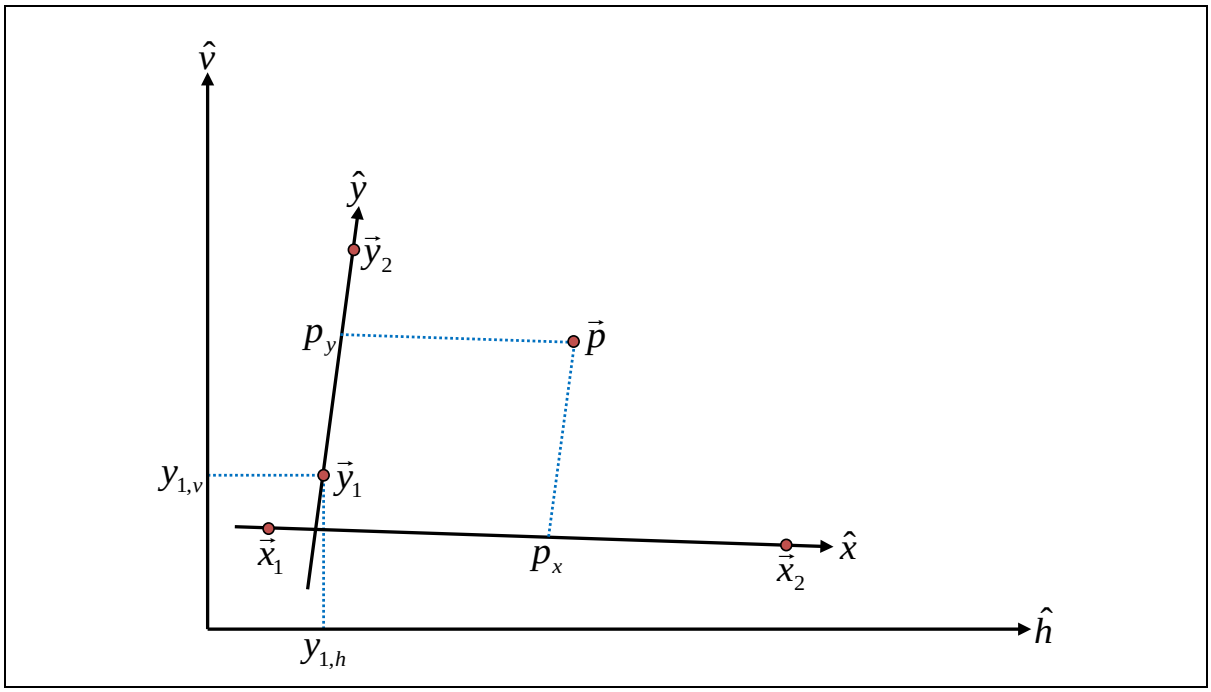

Figure 1. $\vec{p}$, $\vec{x}_1$, $\vec{x}_2$, $\vec{y}_1$, and $\vec{y}_2$ in screen coordinates ($\hat{h}$, $\hat{v}$) and graph coordinates ($\hat{x}$, $\hat{y}$).

Equations 1 through 5 provide algebraic definitions for the components of $\vec{p}$, $\vec{x}_1$, $\vec{x}_2$, $\vec{y}_1$, and $\vec{y}_2$:

$$\vec{p} = p_h \hat{h} + p_v \hat{v} = p_x \hat{x} + p_y \hat{y}\,, \tag{1}$$

$$\vec{x}_1 = x_{1,h} \hat{h} + x_{1,v} \hat{v} = x_{1,x} \hat{x} + x_{1,y} \hat{y}\,, \tag{2}$$

$$\vec{x}_2 = x_{2,h} \hat{h} + x_{2,v} \hat{v} = x_{2,x} \hat{x} + x_{2,y} \hat{y}\,, \tag{3}$$

$$\vec{y}_1 = y_{1,h} \hat{h} + y_{1,v} \hat{v} = y_{1,x} \hat{x} + y_{1,y} \hat{y}\,, \tag{4}$$

and

$$\vec{y}_2 = y_{2,h} \hat{h} + y_{2,v} \hat{v} = y_{2,x} \hat{x} + y_{2,y} \hat{y}\,. \tag{5}$$

It is assumed that p_h, p_v, $x_{1,h}$, $x_{1,v}$, $x_{1,x}$, $x_{2,h}$, $x_{2,v}$, $x_{2,x}$, $y_{1,h}$, $y_{1,v}$, $y_{1,y}$, $y_{2,h}$, $y_{2,v}$, and $y_{2,y}$ are all known. It is further assumed that

$$x_{1,y} = x_{2,y} \tag{6}$$

and

$$y_{1,x} = y_{2,x}\,. \tag{7}$$

Begin by finding expressions for $\hat{x}$ and $\hat{y}$ in terms of screen coordinates.

$$\hat{x} = \frac{\vec{x}_2 - \vec{x}_1}{\left|\vec{x}_2 - \vec{x}_1\right|} \tag{8}$$

$$\Rightarrow \hat{x} = \frac{\left(x_{2,h} - x_{1,h}\right)\hat{h} - \left(x_{2,v} - x_{1,v}\right)\hat{v}}{\sqrt{\left(x_{2,x} - x_{1,x}\right)^2 + \left(x_{2,y} - x_{1,y}\right)^2}}. \tag{9}$$

From equation 6,

$$\hat{x} = \frac{\left(x_{2,h} - x_{1,h}\right)\hat{h} - \left(x_{2,v} - x_{1,v}\right)\hat{v}}{x_{2,x} - x_{1,x}}. \tag{10}$$

Thus,

$$\boxed{x_h = \frac{x_{2,h} - x_{1,h}}{x_{2,x} - x_{1,x}}} \tag{11}$$

and

$$\boxed{x_v = \frac{x_{1,v} - x_{2,v}}{x_{2,x} - x_{1,x}}}. \tag{12}$$

Similarly,

$$\boxed{y_h = \frac{y_{2,h} - y_{1,h}}{y_{2,y} - y_{1,y}}} \tag{13}$$

and

$$\boxed{y_v = \frac{y_{2,v} - y_{1,v}}{y_{2,y} - y_{1,y}}}. \tag{14}$$

Boxed equations represent equations that are coded into the Convert() function, which is presented in section 8.

Next, let $\vec{O}$ be the location where the line defined by the points $\vec{x}_1$ and $\vec{x}_2$ intersects the line defined by the points $\vec{y}_1$ and $\vec{y}_2$. From Yager,[1] $\vec{O}$ is given by

$$\vec{O} = \vec{x}_1 + \left(\vec{x}_2 - \vec{x}_1\right)t_0 = \vec{y}_1 + \left(\vec{y}_2 - \vec{y}_1\right)t_1, \tag{15}$$

where

$$\begin{bmatrix} t_0 \\ t_1 \end{bmatrix} = \begin{bmatrix} x_{1,h} - x_{2,h} & y_{2,h} - y_{1,h} \\ x_{1,v} - x_{2,v} & y_{2,v} - y_{1,v} \end{bmatrix}^{-1} \begin{bmatrix} x_{1,h} - y_{1,h} \\ x_{1,v} - y_{1,v} \end{bmatrix}. \tag{16}$$

Solving for t_0 and t_1,

[1]Yager, R. J. *Two-Dimensional Translations, Rotations, and Intersections using C++*; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, 2013.

$$\boxed{t_0 = \frac{(y_{2,v} - y_{1,v})(x_{1,h} - y_{1,h}) - (y_{2,h} - y_{1,h})(x_{1,v} - y_{1,v})}{(x_{1,h} - x_{2,h})(y_{2,v} - y_{1,v}) - (y_{2,h} - y_{1,h})(x_{1,v} - x_{2,v})}} \tag{17}$$

and

$$\boxed{t_1 = \frac{(x_{1,h} - x_{2,h})(x_{1,v} - y_{1,v}) - (x_{1,v} - x_{2,v})(x_{1,h} - y_{1,h})}{(x_{1,h} - x_{2,h})(y_{2,v} - y_{1,v}) - (y_{2,h} - y_{1,h})(x_{1,v} - x_{2,v})}}. \tag{18}$$

From equation 15,

$$\boxed{O_h = x_{1,h} + (x_{2,h} - x_{1,h})t_0}, \tag{19}$$

$$\boxed{O_v = x_{1,v} + (x_{2,v} - x_{1,v})t_0}, \tag{20}$$

$$\boxed{O_x = x_{1,x} + (x_{2,x} - x_{1,x})t_0}, \tag{21}$$

and

$$\boxed{O_y = y_{1,y} + (y_{2,y} - y_{1,y})t_1}. \tag{22}$$

Finally, find p_x and p_y.

$$\vec{p} - \vec{O} = (p_x - O_x)\hat{x} + (p_y - O_y)\hat{y} = (p_h - O_h)\hat{h} + (p_v - O_v)\hat{v} \tag{23}$$

$$\Rightarrow (p_x - O_x)(x_h\hat{h} + x_v\hat{v}) + (p_y - O_y)(y_h\hat{h} + y_v\hat{v}) = (p_h - O_h)\hat{h} + (p_v - O_v)\hat{v}. \tag{24}$$

In matrix form, equation 24 becomes

$$\begin{bmatrix} x_h & y_h \\ x_v & y_v \end{bmatrix}\begin{bmatrix} p_x - O_x \\ p_y - O_y \end{bmatrix} = \begin{bmatrix} p_h - O_h \\ p_v - O_v \end{bmatrix}. \tag{25}$$

Solving equation 25 for the right-side column vector,

$$\begin{bmatrix} p_x - O_x \\ p_y - O_y \end{bmatrix} = \begin{bmatrix} x_h & y_h \\ x_v & y_v \end{bmatrix}^{-1}\begin{bmatrix} p_h - O_h \\ p_v - O_v \end{bmatrix} \tag{26}$$

$$\Rightarrow \begin{bmatrix} p_x - O_x \\ p_y - O_y \end{bmatrix} = \frac{1}{x_h y_v - x_v y_h}\begin{bmatrix} y_v & -y_h \\ -x_v & x_h \end{bmatrix}\begin{bmatrix} p_h - O_h \\ p_v - O_v \end{bmatrix}. \tag{27}$$

Thus,

$$\boxed{p_x = \frac{y_v(p_h - O_h) - y_h(p_v - O_v)}{x_h y_v - x_v y_h} + O_x} \tag{28}$$

and

$$\boxed{p_y = \frac{x_h(p_v - O_v) - x_v(p_h - O_h)}{x_h y_v - x_v y_h} + O_y}. \tag{29}$$

3. Working With Logarithmic Axes

If values for the $\hat{x}$ axis are scaled logarithmically, then the values for $x_{1,x}$, $x_{2,x}$, and p_x must be scaled. If a is assumed to be the base for the scaling, then $x_{1,x}$, $x_{2,x}$, and p_x can be scaled using equations 30 through 32.

$$\boxed{x'_{1,x} = \log_a(x_{1,x})}. \tag{30}$$

$$\boxed{x'_{2,x} = \log_a(x_{2,x})}. \tag{31}$$

$$\boxed{p'_x = a^{p_x}}. \tag{32}$$

Similarly, if values for the $\hat{y}$ axis are scaled logarithmically, then $y_{1,y}$, $y_{2,y}$, and p_y can be scaled using equations 33 through 35, where b is the base for the scaling.

$$\boxed{y'_{1,y} = \log_b(y_{1,y})}. \tag{33}$$

$$\boxed{y'_{2,y} = \log_b(y_{2,y})}. \tag{34}$$

$$\boxed{p'_y = b^{p_y}}. \tag{35}$$

4. Storing Screen Coordinates: COORD Structs

COORD structs can be used to store locations of points in screen coordinates.

4.1 COORD Code

```
struct COORD{//<================================STORAGE FOR SCREEN COORDINATES
  int h,v;//<-----------HORIZONTAL AND VERTICAL SCREEN COORDINATES (IN PIXELS)
};//~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~03JUL2013~~~~~~
```

4.2 COORD Parameters

h **h** specifies the horizontal position of a point (in pixels).
v **v** specifies the vertical position of a point (in pixels).

5. Storing Axes Information: AXES Structs

AXES structs are used to store all of the axes information required to convert from screen coordinates to graph coordinates.

5.1 AXES Code

```
struct AXES{//<==================================STORAGE FOR AXES INFORMATION
  COORD x1,x2,y1,y2;//<---------------COORDINATES FOR AXES REFERENCE LOCATIONS
  double x1x,x2x,y1y,y2y;//<---------------------------------ASSOCIATED VALUES
  bool xlog,ylog;//<------------ARE AXES LOGARITHMIC (TRUE) OR LINEAR (FALSE)?
  double xbase,ybase;//<---------------------------BASES FOR LOGARITHMIC AXES
};//~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~03JUL2013~~~~~~
```

5.2 AXES Parameters

x1 **x1** specifies $x_{1,h}$ and $x_{1,v}$ (see figure 1 and equation 2).

x2 **x2** specifies $x_{2,h}$ and $x_{2,v}$ (see figure 1 and equation 3).

y1 **y1** specifies $y_{1,h}$ and $y_{1,v}$ (see figure 1 and equation 4).

y2 **y2** specifies $y_{2,h}$ and $y_{2,v}$ (see figure 1 and equation 5).

x1x **x1x** specifies $x_{1,x}$ (see figure 1 and equation 2).

x2x **x2x** specifies $x_{2,x}$ (see figure 1 and equation 3).

y1y **y1y** specifies $y_{1,y}$ (see figure 1 and equation 4).

y2y **y2y** specifies $y_{2,y}$ (see figure 1 and equation 5).

6. Recording Mouse-Click Locations: The GetLClick() Function

The GetLClick() function can be used to record the position of a left mouse click in screen coordinates. The function contains a loop that requires either a left mouse click or escape-key press to exit. If a left mouse click is detected, then the position of the mouse click at press is recorded to the input parameter **C**, and the function returns true. If an escape-key press is detected, then the function returns false without modifying **C**. Note that this function uses Microsoft-specific code.

6.1 GetLClick() Code

```
inline bool GetLClick(//<=RECORD LEFT-MOUSE-CLICK POSITION(MICROSOFT SPECIFIC)
    COORD &C,//<-------------------------------STORAGE FOR OUTPUT COORDINATES
    const char* s){//<-------------TEXT TO DISPLAY, ALONG WITH CURSOR POSITION
  POINT P;
  while(!GetAsyncKeyState(VK_LBUTTON)){//..while left mouse button not pressed
    if(GetAsyncKeyState(VK_ESCAPE))return false;
    GetCursorPos(&P),printf("\r%s%5d,%5d",s,P.x,P.y);}
  C.h=P.x,C.v=P.y;
  while(GetAsyncKeyState(VK_LBUTTON));//....wait for left-mouse-button release
  printf("\n");
  return true;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~16JUL2013~~~~~~
```

6.2 GetLClick() Parameters

C **C** specifies the location of a left mouse click at press.

s **s** specifies text that will be displayed to the left of the cursor-position display.

6.3 GetLClick() Return Value

The GetLClick() function returns true if a left mouse click is detected (and recorded) and false if an escape-key press is detected.

---

7. Recording Axes Information: The GetAxes() Function

The GetAxes() function can be used to populate an AXES struct. The code leads the user through a series of typed inputs and mouse clicks. Before running the function, a graph should be displayed on the screen. Once the function is running, avoid clicking the mouse other than as directed. For best accuracy, the graph should be displayed as large as possible.

7.1 GetAxes() Code

```
inline AXES GetAxes(){//<============================RECORDS AXES INFORMATION
  AXES A;
  cout<<"Enter value for axis 1, reference 1: ",cin>>A.x1x;
  cout<<"Enter value for axis 1, reference 2: ",cin>>A.x2x;
  cout<<"Is axis 1 log based? (y or n):";
  char c;/*<-*/cin>>c;A.xlog=(c=='Y'||c=='y');
  if(A.xlog)std::cout<<"Enter base for axis 1: ",cin>>A.xbase;
  cout<<"\nEnter value for axis 2, reference 1: ",cin>>A.y1y;
  cout<<"Enter value for axis 2, reference 2: ",cin>>A.y2y;
  cout<<"Is axis 2 log based? (y or n):";
  cin>>c;A.ylog=(c=='Y'||c=='y');
  if(A.ylog)cout<<"Enter base for axis 2: ",cin>>A.ybase;
  printf("\n");
  GetLClick(A.x1,"Click at axis 1, reference 1:");
  GetLClick(A.x2,"Click at axis 1, reference 2:");
  GetLClick(A.y1,"Click at axis 2, reference 1:");
  GetLClick(A.y2,"Click at axis 2, reference 2:");
  return A;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~18AUG2013~~~~~~
```

7.2 GetAxes() Command-Line Queries

1. **Enter reference values for axis 1.** When queried, enter graph values for two locations on one of the graph's axes. Choosing points that are spaced as far apart as possible will maximize the accuracy of the conversion.

2. **Specify whether or not axis 1 is logarithmically scaled.** If axis 1 is logarithmically scaled, enter "y" when queried. If the axis is linearly scaled, enter "n."

3. **If axis 1 is logarithmically scaled, enter the base for the scaling.** Common bases are 10, *e*, and 2.

4. **Repeat steps 1 through 3, except for axis 2.** It doesn't matter which axis, 1 or 2, relates to the horizontal axis and which relates to the vertical axis.

5. **Click at axes reference locations.** Position the mouse cursor at each of the reference locations and click the left mouse button to record the position. Be careful to click at the locations for which graph values were previously entered.

7.3 GetAxes() Return Value

The GetAxes() function returns an AXES struct that contains the axes information gathered by the command-line queries and mouse clicks.

8. Converting From Screen Coordinates: The Convert() Function

The Convert() function uses the boxed equations presented in sections 2 and 3 to convert from screen coordinates to graph coordinates.

8.1 Convert() Code

```
inline vector<vector<double> > Convert(//<====FROM SCREEN TO GRAPH COORDINATES
    AXES A,//<-----------------------------------------------AXES DEFINITION
    vector<COORD> P){//<--------------------------COORDINATES TO BE CONVERTED
  if(A.xlog)A.x1x=log(A.x1x)/log(A.xbase),A.x2x=log(A.x2x)/log(A.xbase);
  if(A.ylog)A.y1y=log(A.y1y)/log(A.ybase),A.y2y=log(A.y2y)/log(A.ybase);
  double xh=(A.x2.h-A.x1.h)/(A.x2x-A.x1x);
  double xv=(A.x2.v-A.x1.v)/(A.x2x-A.x1x);
  double yh=(A.y2.h-A.y1.h)/(A.y2y-A.y1y);
  double yv=(A.y2.v-A.y1.v)/(A.y2y-A.y1y);
  double t0=((A.y2.v-A.y1.v)*(A.x1.h-A.y1.h)-(A.y2.h-A.y1.h)*(A.x1.v-A.y1.v))/
    double((A.x1.h-A.x2.h)*(A.y2.v-A.y1.v)-(A.y2.h-A.y1.h)*(A.x1.v-A.x2.v));
  double t1=((A.x1.h-A.x2.h)*(A.x1.v-A.y1.v)-(A.x1.v-A.x2.v)*(A.x1.h-A.y1.h))/
    double((A.x1.h-A.x2.h)*(A.y2.v-A.y1.v)-(A.y2.h-A.y1.h)*(A.x1.v-A.x2.v));
  double Oh=A.x1.h+(A.x2.h-A.x1.h)*t0;
  double Ov=A.x1.v+(A.x2.v-A.x1.v)*t0;
  double Ox=A.x1x+(A.x2x-A.x1x)*t0;
  double Oy=A.y1y+(A.y2y-A.y1y)*t1;
  vector<vector<double> > p(P.size(),vector<double>(2));
  for(int i=0,n=p.size();i<n;++i){
    p[i][0]=(yv*(P[i].h-Oh)-yh*(P[i].v-Ov))/(xh*yv-xv*yh)+Ox;
    p[i][1]=(xh*(P[i].v-Ov)-xv*(P[i].h-Oh))/(xh*yv-xv*yh)+Oy;
    if(A.xlog)p[i][0]=pow(A.xbase,p[i][0]);
    if(A.ylog)p[i][1]=pow(A.ybase,p[i][1]);}
  return p;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~16JUL2013~~~~~~
```

8.2 Convert() Parameters

A **A** specifies an AXES struct. AXES structs can be populated using the GetAxes() function.

P **P** specifies a set of screen coordinates that will be converted to graph coordinates.

8.3 Convert() Return Value

The Convert() function returns a two-dimensional vector of doubles that is used to store locations in graph coordinates. The first index is used to specify a point. The second index specifies a coordinate that relates to an axis specified by the input AXES struct. For example, suppose that the output is stored in the variable Q. Q[5][0] specifies the x-axis value for the fifth point specified by the input variable **P**; Q[5][1] specifies the y-axis value.

9. main() Implementation

The following code uses the previously described structs and functions to convert points on a graph from screen coordinate to graph coordinates. The code leads the user through the process of entering axes information and clicking on relevant portions of the graph. The code creates an output file named “digitizer.txt.” Figure 2 presents a sample output file.

```
# AXES INFORMATION
x1.h=321
x1.v=1101
x2.h=2200
x2.v=944
y1.h=213
y1.v=1332
y2.h=643
y2.v=406
x1x=100.000000
x2x=1000.000000
y1y=0.010000
y2y=1000000.000000
xlog=true
ylog=true
xbase=10.000000
ybase=10.000000

# CLICKED COORDINATES (h,v)
231,1292
812,1213
1167,1135
1436,1045
1661,938
1862,814
2049,673
2229,512
2406,332

# TRANSFORMED COORDINATES (x,y)
99.833548,0.022182
199.826653,0.041661
300.023717,0.113291
400.826027,0.458247
501.214283,2.843453
601.771725,26.146471
702.615518,350.179622
803.499771,7181.026263
905.123141,219335.022371
```

Figure 2. Sample output file.

9.1 main() Code

```
#include "y_digitizer.h"//.....................................<vector>,<cstdio>
int main(){
  yDigitizer::AXES A=yDigitizer::GetAxes();
  printf("\nTrace a curve with mouse clicks (press escape to quit):\n");
  std::vector<yDigitizer::COORD> P;
  yDigitizer::COORD X;
  while(yDigitizer::GetLClick(X,""))P.push_back(X);
  printf("\r                    \n");
  std::vector<std::vector<double> > p=Convert(A,P);
  FILE *f=freopen("digitizer.txt","w",stdout);//.......redirect output to a file
  printf("# AXES INFORMATION\nx1.h=%d\nx1.v=%d\nx2.h=%d\nx2.v=%d\ny1.h=%d\ny1.v"
    "=%d\ny2.h=%d\ny2.v=%d\nx1x=%f\nx2x=%f\ny1y=%f\ny2y=%f\nxlog=%s\nylog=%s\n",
    A.x1.h,A.x1.v,A.x2.h,A.x2.v,A.y1.h,A.y1.v,A.y2.h,A.y2.v,
    A.x1x,A.x2x,A.y1y,A.y2y,A.xlog?"true":"false",A.ylog?"true":"false");
  if(A.xlog)printf("xbase=%f\n",A.xbase);
  if(A.ylog)printf("ybase=%f\n",A.ybase);
  printf("\n# CLICKED COORDINATES (h,v)\n");
  for(int i=0,n=p.size();i<n;++i)printf("%d,%d\n",P[i].h,P[i].v);
  printf("\n# TRANSFORMED COORDINATES (x,y)\n");
  for(int i=0,n=p.size();i<n;++i)printf("%f,%f\n",p[i][0],p[i][1]);
  fclose(f);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~16JUL2013~~~~~~
```

---

10. Example: Digitizing a Log-Log Plot With Rotated and Skewed Axes

---

To test the functionality of the digitizing code, a sample graph was created, printed, scanned, and digitized.

10.1 Data Generation

Using equation 36, nine data points were calculated and plotted on a base 10, log-log plot, as shown in figure 3.

$$y = e^{0.00002x^2+0.00012x-4}. \tag{36}$$

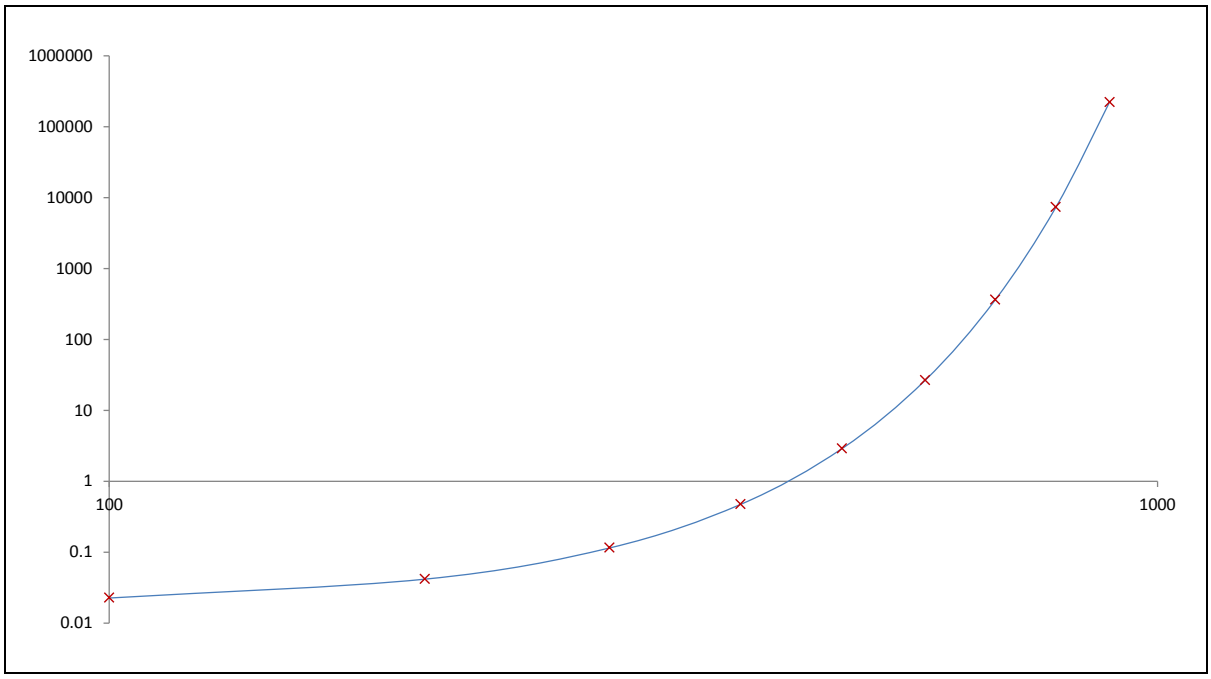

Figure 3. Nine sample data points plotted on a base-10, log-log plot.

10.2 Creating the Sample Graph

To simulate the type of graph that the code described in this report might be called upon to digitize, the graph that is presented in figure 3 was printed, then scanned back into the computer, then rotated and skewed. The final result is shown in figure 4.

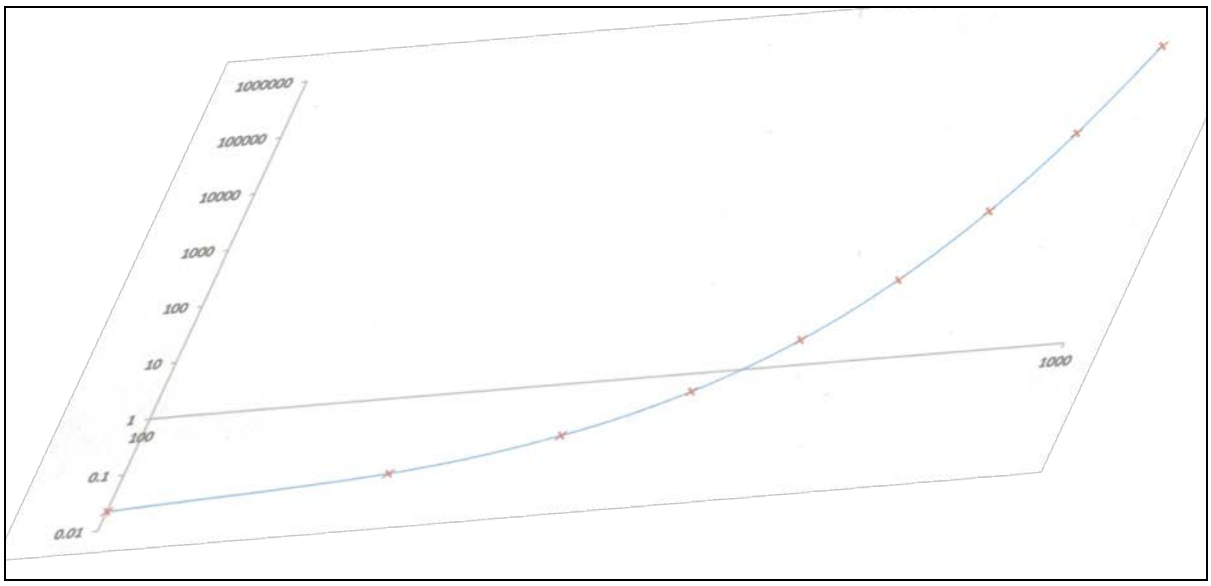

Figure 4. Sample graph after being printed, scanned, rotated, and skewed.

10.3 Digitized Results

Table 1 summarizes the results of the digitizing of the graph shown in figure 4. For all measured values, the percent error was less than 3%.

Table 1. Tabulated values (calculated and measured) for data points on sample log-log plot.

x			***y***		
Calculated	**Measured**	**% Error**	**Calculated**	**Measured**	**% Error**
100	99.8	0.17	0.0226	0.02218	1.85
200	199.8	0.09	0.0418	0.04166	0.22
300	300.0	0.01	0.1149	0.11329	1.37
400	400.8	0.21	0.4714	0.4582	2.79
500	501.2	0.24	2.8864	2.843	1.49
600	601.8	0.30	26.3640	26.15	0.83
700	702.6	0.37	359.2433	350.2	2.52
800	803.5	0.44	7302.7042	7181	1.67
900	905.1	0.57	221460.6056	219300	0.98

11. Summary

A summary sheet is provided at the end of this report. It presents the yDigitizer namespace, which contains the two structs and four functions that are described in this report.

yDigitizer Summary

y_digitizer.h

```
#ifndef Y_DIGITIZER_GUARD
#define Y_DIGITIZER_GUARD
#include <vector>
#include <iostream>//..................................................cin,cout
#include <cstdio>//....................................................printf()
#include <cmath>//.................................................pow(),log()
#include <windows.h>//....POINT,GetAsyncKeyState(),GetCursorPos(),VK_LBUTTON,...
namespace yDigitizer{
  using namespace std;
  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  struct COORD{//<================================STORAGE FOR SCREEN COORDINATES
    int h,v;//<-----------HORIZONTAL AND VERTICAL SCREEN COORDINATES (IN PIXELS)
  };//~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~03JUL2013~~~~~~
  struct AXES{//<===================================STORAGE FOR AXES INFORMATION
    COORD x1,x2,y1,y2;//<---------------COORDINATES FOR AXES REFERENCE LOCATIONS
    double x1x,x2x,y1y,y2y;//<----------------------------------ASSOCIATED VALUES
    bool xlog,ylog;//<------------ARE AXES LOGARITHMIC (TRUE) OR LINEAR (FALSE)?
    double xbase,ybase;//<---------------------------BASES FOR LOGARITHMIC AXES
  };//~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~03JUL2013~~~~~~
  inline bool GetLClick(//<=RECORD LEFT-MOUSE-CLICK POSITION(MICROSOFT SPECIFIC)
      COORD &C,//<-------------------------------STORAGE FOR OUTPUT COORDINATES
      const char* s){//<-------------TEXT TO DISPLAY, ALONG WITH CURSOR POSITION
    POINT P;
    while(!GetAsyncKeyState(VK_LBUTTON)){//..while left mouse button not pressed
      if(GetAsyncKeyState(VK_ESCAPE))return false;
      GetCursorPos(&P),printf("\r%s%5d,%5d",s,P.x,P.y);}
    C.h=P.x,C.v=P.y;
    while(GetAsyncKeyState(VK_LBUTTON));//....wait for left-mouse-button release
    printf("\n");
    return true;
  }//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~16JUL2013~~~~~~
  inline AXES GetAxes(){//<============================RECORDS AXES INFORMATION
    AXES A;
    cout<<"Enter value for axis #1, reference #1: ",cin>>A.x1x;
    cout<<"Enter value for axis #1, reference #2: ",cin>>A.x2x;
    cout<<"Is axis #1 log based? (y or n):";
    char c;/*<-*/cin>>c;A.xlog=(c=='Y'||c=='y');
    if(A.xlog)std::cout<<"Enter base for axis #1: ",cin>>A.xbase;
    cout<<"\nEnter value for axis #2, reference #1: ",cin>>A.y1y;
    cout<<"Enter value for axis #2, reference #2: ",cin>>A.y2y;
    cout<<"Is axis #2 log based? (y or n):";
    cin>>c;A.ylog=(c=='Y'||c=='y');
    if(A.ylog)cout<<"Enter base for axis #2: ",cin>>A.ybase;
    printf("\n");
    GetLClick(A.x1,"Click at axis #1, reference #1:");
    GetLClick(A.x2,"Click at axis #1, reference #2:");
    GetLClick(A.y1,"Click at axis #2, reference #1:");
    GetLClick(A.y2,"Click at axis #2, reference #2:");
    return A;
  }//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~16JUL2013~~~~~~
  inline vector<vector<double> > Convert(//<====FROM SCREEN TO GRAPH COORDINATES
      AXES A,//<-----------------------------------------------AXES DEFINITION
      vector<COORD> P){//<---------------------------COORDINATES TO BE CONVERTED
    if(A.xlog)A.x1x=log(A.x1x)/log(A.xbase),A.x2x=log(A.x2x)/log(A.xbase);
    if(A.ylog)A.y1y=log(A.y1y)/log(A.ybase),A.y2y=log(A.y2y)/log(A.ybase);
    double xh=(A.x2.h-A.x1.h)/(A.x2x-A.x1x);
    double xv=(A.x2.v-A.x1.v)/(A.x2x-A.x1x);
    double yh=(A.y2.h-A.y1.h)/(A.y2y-A.y1y);
    double yv=(A.y2.v-A.y1.v)/(A.y2y-A.y1y);
    double t0=((A.y2.v-A.y1.v)*(A.x1.h-A.y1.h)-(A.y2.h-A.y1.h)*(A.x1.v-A.y1.v))/
      double((A.x1.h-A.x2.h)*(A.y2.v-A.y1.v)-(A.y2.h-A.y1.h)*(A.x1.v-A.x2.v));
    double t1=((A.x1.h-A.x2.h)*(A.x1.v-A.y1.v)-(A.x1.v-A.x2.v)*(A.x1.h-A.y1.h))/
      double((A.x1.h-A.x2.h)*(A.y2.v-A.y1.v)-(A.y2.h-A.y1.h)*(A.x1.v-A.x2.v));
    double Oh=A.x1.h+(A.x2.h-A.x1.h)*t0;
    double Ov=A.x1.v+(A.x2.v-A.x1.v)*t0;
    double Ox=A.x1x+(A.x2x-A.x1x)*t0;
    double Oy=A.y1y+(A.y2y-A.y1y)*t1;
    vector<vector<double> > p(P.size(),vector<double>(2));
    for(int i=0,n=p.size();i<n;++i){
      p[i][0]=(yv*(P[i].h-Oh)-yh*(P[i].v-Ov))/(xh*yv-xv*yh)+Ox;
      p[i][1]=(xh*(P[i].v-Ov)-xv*(P[i].h-Oh))/(xh*yv-xv*yh)+Oy;
      if(A.xlog)p[i][0]=pow(A.xbase,p[i][0]);
      if(A.ylog)p[i][1]=pow(A.ybase,p[i][1]);}
    return p;
  }//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~16JUL2013~~~~~~
}//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
#endif
```

Implementation

```
#include "y_digitizer.h"//...................................<vector>,<cstdio>
int main(){
  yDigitizer::AXES A=yDigitizer::GetAxes();
  printf("\nTrace a curve with mouse clicks (press escape to quit):\n");
  std::vector<yDigitizer::COORD> P;
  yDigitizer::COORD X;
  while(yDigitizer::GetLClick(X,""))P.push_back(X);
  printf("\r                 \n");
  std::vector<std::vector<double> > p=Convert(A,P);
  FILE *f=freopen("digitizer.txt","w",stdout);//.......redirect output to a file
  printf("# AXES INFORMATION\nx1.h=%d\nx1.v=%d\nx2.h=%d\nx2.v=%d\ny1.h=%d\ny1.v"
    "=%d\ny2.h=%d\ny2.v=%d\nx1x=%f\nx2x=%f\ny1y=%f\ny2y=%f\nxlog=%s\nylog=%s\n",
    A.x1.h,A.x1.v,A.x2.h,A.x2.v,A.y1.h,A.y1.v,A.y2.h,A.y2.v,
    A.x1x,A.x2x,A.y1y,A.y2y,A.xlog?"true":"false",A.ylog?"true":"false");
  if(A.xlog)printf("xbase=%f\n",A.xbase);
  if(A.ylog)printf("ybase=%f\n",A.ybase);
  printf("\n# CLICKED COORDINATES (h,v)\n");
  for(int i=0,n=p.size();i<n;++i)printf("%d,%d\n",P[i].h,P[i].v);
  printf("\n# TRANSFORMED COORDINATES (x,y)\n");
  for(int i=0,n=p.size();i<n;++i)printf("%f,%f\n",p[i][0],p[i][1]);
  fclose(f);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~16JUL2013~~~~~~
```

NO. OF
COPIES ORGANIZATION

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

1 (PDF) DIRECTOR US ARMY RESEARCH LAB IMAL HRA

1 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL

1 (PDF) GOVT PRINTG OFC A MALHOTRA

ABERDEEN PROVING GROUND

1 (PDF) DIR USARL RDRL WML A R YAGER

INTENTIONALLY LEFT BLANK.