122nd ASEE Annual Conference & Exposition
June 14 - 17, 2015
Seattle, WA
Making Value for Society
Paper ID #12554

Motivating non-electrical and computer engineering students to learn C Programming

Dr. Nicholas A Baine, Grand Valley State University

Nicholas Baine, Ph.D., is an Assistant Professor in the School of Engineering. His expertise is in the design of electrical control systems and sensor data fusion. As an instructor, he specializes in teaching freshman courses as well as control systems and design of digital and embedded systems. While at Wright State University, he was part of the group which developed a new model to teach mathematics to engineering students. As a new faculty member at Grand Valley State University, he is working with faculty to develop and improve the relatively new freshman design courses.

Dr. Hugh Jack, Grand Valley State University

Dr. Jack is a professor of Product Design and Manufacturing Engineering at Grand Valley State University in Grand Rapids, MI. His interests include robotics, automation, and design.

Motivating non-electrical and computer engineering students to learn C Programming

Abstract

All too often, non-electrical and computer engineering students (in particular, manufacturing and mechanical engineering students) perceive computer programming as irrelevant to their studies [1, 2]. This is not surprising in that computer programming is taught almost exclusively with non-manufacturing examples. To address this problem a programming project was created for a freshman class. The project focused on a tolerance stack with a gap specification. The paper outlines the tolerance problem, core programming concepts, and the final solution. The programming assignment and a sample solution are provided in the appendices for reuse or modification for other engineering schools.

Introduction

The freshman year at Grand Valley State University is common for all disciplines: Computer, Electrical, Mechanical, and Product Design and Manufacturing Engineering. In the first semester, the students take a course, EGR 106 Introduction to Engineering Design I, the first of a course sequence that combines basic CAD/CAM design with C programming. In the second semester, students take EGR 107 Introduction to Engineering Design II, and EGR 220 Engineering Measurement and Data Analysis. The project described in this paper was used in EGR 107. Students in the course have intermediate programming abilities. In addition, they were taught a tolerance stack analysis technique earlier in the semester of the same course. Simultaneously the students have been exposed to statistical variations in EGR 220. Up to this point many of the freshmen see the topics as disconnected. Anecdotally this attitude persists for many students until the junior year.

The remainder of this paper outlines the assignment provided in Appendix A, and the solution provided in Appendix B. Other resources include kinematics, structures, and digital logic [3, 4].

Tolerance Stacks and Simple Analysis

The tolerance analysis problem chosen was a linear stack of parts with nominal dimensions and statistical variation. The first problem-solving step is to identify a tolerance loop in an assembly. A table of values is constructed including the nominal dimension and tolerance for each part. The table also indicates whether each part has a positive or negative effect on the characteristic feature, often a gap. In addition, all parts are identified as fixed or variable based on the cost of specification modification. Eventually the variable parts will be changed to make the design match the gap specification. A sample tolerance case is shown in Figure 1.

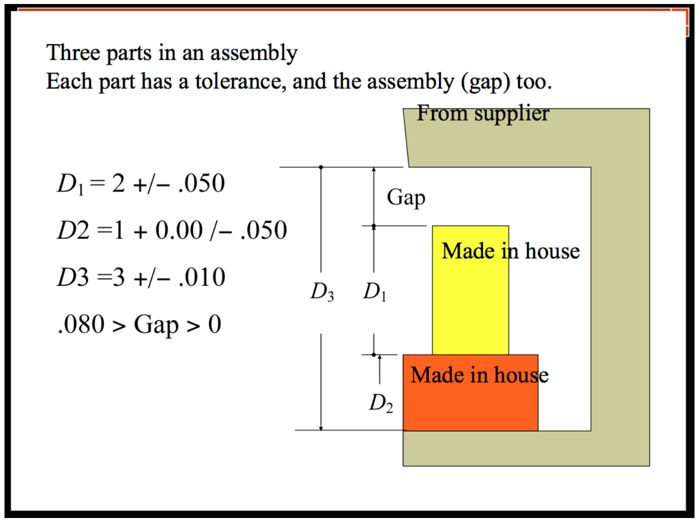


Figure 1 - A sample tolerance stack problem (lecture slide)

As always the first step for students is to convert the problem to a normalized form. In this case the tolerance table shown in Figure 2. The dimensions of the part and bilateral tolerances are listed in columns 1 and 3. Note that D2 was a given as a unilateral tolerance, but was converted to a bilateral tolerance for the table. The sensitivity is a sign that dictates whether a dimension adds or subtracts from the characteristic feature (gap). D1 and D2 have negative sensitivity because if they get larger the gap becomes smaller. However when D3 becomes larger the gap does too, so the sign is positive. Two of the parts, D1 and D2, are made in-house and will cost less to change, so they are marked a 'V' variable. The outside part, D3, is 'F' fixed because is made elsewhere and would have a higher cost to change. The table is then used to calculate the gap size and tolerance based on the part dimensions only. It is important to note that this is not the gap specified. At this point, there is a difference between the gap specified (0.000" to 0.080") and the resultant gap (0.025" +/- 0.085"). Therefore, the tolerances on the individual parts are too large and the resultant gap will not satisfy the specified gap.

Tolerance Analysis

Dimension.	Sensitivity	Bilateral Tolerance	Fixed Or Variable
D1 = 2	-1	.050	V
D2 =.975	-1	.025	V
D3 =3	+1	.010	F

Mean Value of the Gap

$$Gap = D_3 - (D_1 + D_2) \quad = 3 - (2 + .975) \quad = 0.025$$

Effective gap tolerance

$$t_{gap} = \pm(.050 + .025 + .010) = \pm.085$$

Figure 2 - Tolerance table with values and gap values (lecture slide)

The assignment specifies that students must read a tolerance table from a file in the format below. The numbers in the files, and in the following examples are based on the example in Figure 1. During this step, the students work with basic file pointers, string reading, and parsing. They must also create a basic data structure. The example program in Appendix B does this, using standard C functions.

```
PART,2.000,-1,0.050,V
PART,0.975,-1,0.025,V
PART,3.000,+1,0.010,F
GAP,0.000,0.080
```

Once the data array has been populated, a simple analysis is done, including the calculation seen in Figure 2. The sample output below shows that the upper and lower specified limits are both violated. Essentially the part tolerances and dimensions will not satisfy the gap specification.

```
Actual Gap Mean: 0.025”
Actual Gap Tolerance: 0.085”
The Maximum Gap (0.110”) is (Greater) than specified
(0.080”)
The Minimum Gap (-0.060”) is (Less) than the specified
(0.000”)
```

Tolerance and Dimension Correction

In a very good design, the part dimensions and tolerances would sum up to equal the specified gap values. In this case the total is larger than the specified gap, and the tolerances will have to be reduced. In this case the students must change the ‘V’ variable part dimensions and tolerances. Figure 3 shows the equations for recalculating dimensions and tolerances. The example decides to only change D2. D1 could have also been changed, but was fixed for simplicity. The tolerance equation shows that $x1 + x2 = 0.030$ for an exact solution. It is then up to the student to select a solution.

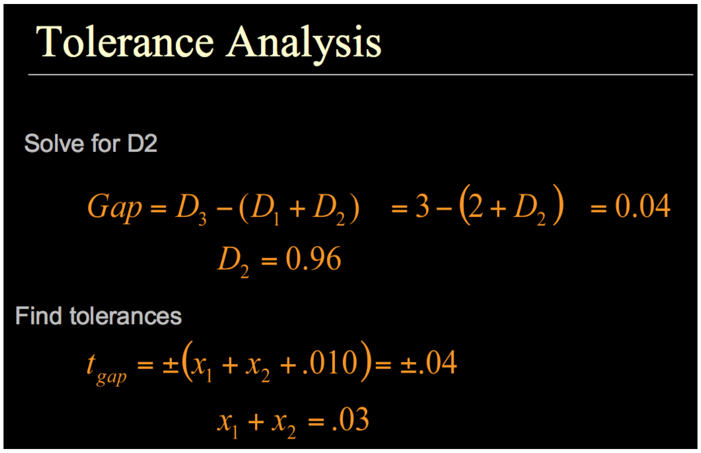


Figure 3 - Fixed versus variable constraint equations (lecture slide)

Ideally, the new dimensions and tolerances will exactly match the specification. The worst acceptable case would be to set all variable tolerances to zero. To add complexity, sometimes the fixed part dimensions and tolerances make it impossible to meet the specifications. This step of the process solution allows the most flexibility and technical problem solving on the part of the students. The example provided in Appendix B adds or subtracts the same amount from all variable tolerances to meet the specification. Likewise the same value is added/subtracted from all nominal dimensions to meet the mean gap value. Sample output from the program is shown after the correction routine.

```
Actual Gap Mean: 0.040000"
Actual Gap Tolerance: 0.040000"
The Maximum Gap (0.080000") is (Equal) to specified
(0.080000")
The Minimum Gap (0.000000") is (Equal) to specified
(0.000000")

Parts table
 PART 0 : dimension = 1.992500 +/- 0.027500, sensitivity =
-1.000000, type = Variable
 PART 1 : dimension = 0.967500 +/- 0.002500, sensitivity =
-1.000000, type = Variable
 PART 2 : dimension = 3.000000 +/- 0.010000, sensitivity =
1.000000, type = Fixed
Gap min=0.000000, max=0.080000
```

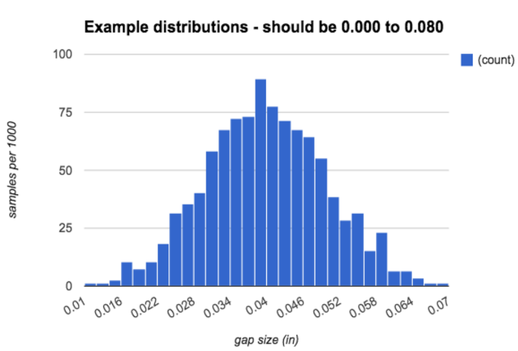
Monte Carlo Simulation

The final stage of the assignment is to do a statistical analysis of the tolerance stack using random variation. The students were provided with a function that would take the part dimension and tolerance and return a random value. It was assumed that the tolerance was equal to 3 standard deviations. The function uses the Box-Muller technique to generate a Gaussian distribution given that the random number generator included in standard C libraries is limited to a uniform distribution.

```
void random_dimension(double nominal, double tolerance,
double *random_value){
     double r1, r2, w, r;

     do {
            r1 = 2.0 * (( rand() % 10001 ) / 10000.0) -
1.0;
            r2 = 2.0 * (( rand() % 10001 ) / 10000.0) -
1.0;
            w = r1 * r1 + r2 * r2;
     } while ( w >= 1.0 );
     r = sqrt(-2 * log(w) / w );
     *random_value = nominal + (tolerance/3) * r * r1;
}
```

The programming required for this technique is trivial, but it reveals the important properties of a non-uniform distribution. Students are able to simulate the statistical distributions before and after the correction algorithm. The assignment requires students to output gap values to a text file and then use other software to graph the results. The two distributions from the program in Appendix B are shown in Figure 4.



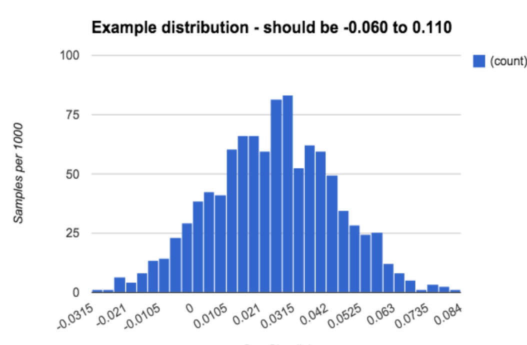


Figure 4 - Statistical Distribution of Tolerance Stack Data

Discussion

Most engineering students learn programming in isolation of engineering problems. This is mitigated by using resources and texts for problem solving with programming. Most of these texts have resources that relate to either mechanical or electrical engineering, but there is little to no resources for manufacturing engineering students. This paper walked through a detailed programming project that can be used to improve manufacturing student motivation to learn programming. It also illustrates to students of other disciplines how versatile programming is and how important a tool it is for an engineer.

Conclusions

The project outlined in this assignment was developed by manufacturing engineering faculty but used by electrical and computer engineering faculty to teach C programming. All too often faculty teaching programming fundamentals meet student resistance based on their perception of irrelevance to their chosen discipline. This assignment provides a resource for those programming instructors. The intent is for this to be shared with faculty teaching manufacturing/mechanical students to program.

References

1. Cole, W., Finn., C.W.P., "MET Graduate Survey Results", ASEE Annual Conference, June 1998, [web PDF - Accessed Mar., 5, 2015].
2. Sun, W., Sun, X., "Teaching Computer Programming Skills to Engineering and Technology Students with a Modular Programming Strategy", ASEE National Convention, June 2011, [web PDF - Accessed Mar., 5, 2015].
3. Myszka, D., "Motivating Students in an Introduction to Computing Course by Requiring Animated Solutions", ASEE National Conference, June 2006. [web PDF - Accessed Mar., 5, 2015].
4. Bowen, J.D., "Motivating Civil Engineering Students to Learn Computer Programming with a Structural Design Project", ASEE Annual Conference, June 2004, [web PDF - Accessed Mar., 5, 2015].

Appendix A - Tolerance Analysis Programming Project

Background

Computer Aided Engineering (CAE) is an important part of the design process. There are many different software packages used for design analysis. Mechanical engineers often use Finite Element Analysis (FEA). Electrical engineers will use programs like Spice to simulate circuits. Computer engineers use simulated operating systems and hardware to test design programs. PDM engineers will use software to simulate control systems. This software is always based on a solid mathematical model of real problems. An engineer will extract the critical elements of the design for the input to the software. The CAE software is then used to test various application cases. The outputs from these simulations are used to verify design decisions, or guide redesign.

This project will focus on the creation of CAE software for tolerance analysis. The basis of this analysis was taught at the beginning of the semester. The project is being broken into steps, as is done with any software engineering project. The result of the first stage will be a program that will load a tolerance table and estimate gap sizes and tolerances. The second stage will suggest options for changing tolerances and dimensions to meet the design specification, the characteristic feature. The third stage will be a statistical analysis that will numerically determine the distribution of the variances.

Project Description

Part I

The input to the program will be a text file containing the information for a tolerance table. An example follows using the values from the first lecture on tolerance analysis. These values will be stored in a text file. The data is comma delimited, which means that each data field is separated by a comma. If the first word is 'PART' the following values are the nominal size, +/- impact, tolerance, and fixed/variable. If the first word is 'GAP' the following values are the minimum and maximum sizes. (Note: assume all units are inches.)

```
PART,2.000,-1,0.050,V
PART,0.975,-1,0.025,V
PART,3.000,+1,0.010,F
GAP,0.000,0.080
```

These values will be processed using the method taught in class. A sample output for the first stage is given.

```
Actual Gap Mean: 0.025”
Actual Gap Tolerance: 0.085”
The Maximum Gap (0.110”) is (Greater) than specified (0.080”)
The Minimum Gap (-0.060”) is (Less) than the specified (0.000”)
```

Part II

The second phase extends the Stage 1 analysis. In this stage the program will suggest various combinations of part dimensions and tolerances to meet the gap specifications. Some of the options for suggestions include:

1. For each variable (V) part find the largest tolerance value, and new nominal value. Note: This is done for one part at a time.
2. For all variable parts use a scaling factor to reduce/multiply all tolerance values to fit. Use the results from the previous step to determine the largest and smallest scaling factors that make sense. Then use those values as upper and lower bounds for a search for the best scaling factor.

Part III

The third stage involves a statistical analysis called Monte Carlo simulation. Basically each of the dimensions is varied randomly and the gap is calculated. This random calculation is repeated hundreds or thousands of times. For each iteration, the individual gap value is calculated and stored in an array and in a file. The array of values will be used to compute the mean and standard deviation of the gap. The file will be opened using a spreadsheet program to graph a histogram, calculate an average, and calculate the standard deviation, which should match the result from your program. These values will then be used to estimate the number of rejected assemblies during production.

It is reasonable to assume that the tolerance for a part is 3 standard deviations (99.73% of parts will fall within the tolerance). So for any part, we can generate random realistic values by taking the nominal value and adding/subtracting a random number, which represents variations due to the tolerance. To do this, we will need to generate a specific type of random numbers.

The standard random number function in most programming language (`rand()` in C included) has a uniform distribution. This means that if we are finding random numbers from 0.0 to 1.0, the probability of getting 0.5 is the same as getting 0.1. This is not realistic for our application, since it will be more likely that we have our dimensions change by 0.1 than it is 0.5. For this reason, we need to modify the numbers from the `rand()` function so that they have a Gaussian (Normal) Distribution, which is sometimes referred to as the "bell curve". This can be accomplished by using the Box-Muller Transformation. Although it sounds complicated, this transformation can be easily done and applied to generate a random dimension with the code below.

```
void random_dimension(double nominal, double tolerance, double *random_value){
        double r1, r2, w, r;

        do {
             r1 = 2.0 * (( rand() % 10001 ) / 10000.0) - 1.0;
             r2 = 2.0 * (( rand() % 10001 ) / 10000.0) - 1.0;
             w = r1 * r1 + r2 * r2;
        } while ( w >= 1.0 );
        r = sqrt(-2 * log(w) / w );
        *random_value = nominal + (tolerance/3) * r * r1;
}
```

Deliverables

Part I (2 weeks)
Demonstrate and submit the program for basic gap analysis. (Note: You are encouraged to work ahead, but the grading will focus on the work for the stage.)

Part II (2 weeks)
Demonstrate and submit the program to make suggestions for changes to part nominal sizes and dimensions.

Part III (2 weeks)
Demonstrate and submit the final program with a menu that allows the user to select the Stage 1, Stage 2, or Stage 3 analysis. Don't forget that Stage 1 must be done before Stage 2.

Appendix B - Source Code Solution

```
// A tolerance analysis program for EGR 107
// The program is written at a secondary level of C programming.
// Note: gap is used for description, but a -ve gap is an overlap of material.
// H. Jack - 3_1_15

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h> // this is only being added to get a seed value for random numbers



// These matrices store the basic tolerance table values for the parts
#define MAX_PARTS    10 // Set a maximum matrix size, increase for more parts
double  part_nominal[MAX_PARTS];
double  part_sign[MAX_PARTS];
double  part_tolerance[MAX_PARTS];
char    part_variability[MAX_PARTS];
int     part_cnt = 0; // how many parts are in the table



// gap/overlap range - note that -ve values indicate overlap
double gap_min, gap_max; // the specified gap range
double gap_mean, gap_tolerance;  // the gap mean and tolerance calculated from parts



// Some definitions to keep things orderly
#define  INPUT_FILE_NAME  "tolerances.csv"
#define  OUTPUT_FILE_NAME "simulation.csv"
#define  SIMULATION_STEPS   1000
#define  ERROR        1
#define  NO_ERROR     0
#define  TRUE     1
#define  FALSE    0



// Prototypes so that we can have a main function at the top of the program
int load_parts();
int parse_input_file( FILE *fp );
int dump_table();
int update_gap();
int simple_analysis();
int adjust_parts();
int monte_carlo();
void random_dimension(double nominal, double tolerance, double *random_value);
int random_gap(double *gap_random);
```

```
int main(){
    int error = NO_ERROR;
    char menu_choice;

    printf("\nTolerance Program\n");
    printf("1=load, 2=simple_analysis, 3=adjust_tolerances, 4=monte_carlo, 5=print_parts,
0=quit\n");

    while( error == NO_ERROR ){

        scanf("%c", &menu_choice);

        switch(menu_choice){
            case '0':
                return NO_ERROR;
                break;
            case '1':
                error = load_parts();
                break;
            case '2':
                error = simple_analysis();
                break;
            case '3':
                error = adjust_parts();
                break;
            case '4':
                error = monte_carlo();
                break;
            case '5':
                error = dump_table();
                break;
            case '\n':
                break;
            default:
                printf("Input not recognized");
                printf("1=load, 2=simple_analysis, 3=adjust_tolerances, 4=monte_carlo,
5=print_parts, 0=quit\n");
        }
    }

    return ERROR;
}



//////////////////////////////////
// Functions for Phase I

int load_parts(){
    FILE *fp_in;

    printf("\nLoading Parts\n");
    if( ( fp_in = fopen( INPUT_FILE_NAME , "r" ) ) != NULL ){
        parse_input_file(fp_in);
        fclose(fp_in);
    } else {
         printf("Error: File would not open\n");
         return ERROR;
```

```
    }
    printf("Parts loaded\n");
    return NO_ERROR;
}



// this function reads the file one line at a time and populates the parts matrices
int parse_input_file( FILE *fp ){
    int    max_string_length = 30; // cap the string length - please no buffer overflow hacks
    char   new_line[max_string_length + 1];

    part_cnt = 0;
    for( int i = 0; i < MAX_PARTS; i++){
        if( fgets( new_line, max_string_length, fp ) != NULL ){
            // got a string from the file, time to parse
            if( strncmp( new_line, "PART", 4) == 0 ){ // got a part line
                 // Reads a line from the file in PART format
                 sscanf( &(new_line[5]), "%lf, %lf, %lf, %c\n",
                      &part_nominal[part_cnt], &part_sign[part_cnt],
                      &part_tolerance[part_cnt], &part_variability[part_cnt]);
                 part_cnt++;
            } else if( strncmp( new_line, "GAP", 3) == 0 ){ // got a gap
                 // reads a line from the file in GAP format
                 sscanf( &(new_line[4]), "%lf, %lf", &gap_min, &gap_max);
            } else if (strlen( new_line ) <= 1 ){ // allow for a <cr> or <lf>
                 // empty line, do nothing
            } else {
                 printf("ERROR: unexpected input file content \n");
                 return ERROR;
            }
        }
    }

    return NO_ERROR;
}



// this prints the contents of the tolerance table, mainly for debugging
int dump_table(){
    printf("\nParts table\n");
    for( int i = 0; i < part_cnt; i++){
        printf(" PART %d : dimension = %lf +/- %lf, sensitivity = %lf, type = ", i,
             part_nominal[i], part_tolerance[i], part_sign[i]);
        if( part_variability[i] == 'F' ){
             printf("Fixed \n");
        } else {
             printf("Variable \n");
        }
    }
    printf("Gap min=%f, max=%f\n", gap_min, gap_max);

    return NO_ERROR;
}



int update_gap(){
    gap_mean = 0.0;
    gap_tolerance = 0.0;
```

```
    for(int i = 0; i < part_cnt; i++){
        gap_mean += part_nominal[i] * part_sign[i];
        gap_tolerance += part_tolerance[i];
    }
    return NO_ERROR;
}



int simple_analysis(){
    printf("\nAnalysis of current tolerances\n");
    update_gap();
    printf("Actual Gap Mean: %lf\" \n", gap_mean);
    printf("Actual Gap Tolerance: %lf\" \n", gap_tolerance);
    if( gap_mean+gap_tolerance > gap_max + 0.000001 ){
        printf("The Maximum Gap (%lf\") is (Greater) than specified (%lf\")\n",
gap_mean+gap_tolerance, gap_max);
    } else if ( gap_mean+gap_tolerance < gap_min - 0.000001 ){
        printf("The Maximum Gap (%lf\") is (Less) than specified (%lf\")\n",
gap_mean+gap_tolerance, gap_max);
    } else {
        printf("The Maximum Gap (%lf\") is (Equal) to specified (%lf\")\n",
gap_mean+gap_tolerance, gap_max);
    }

    if( gap_mean-gap_tolerance < gap_min - 0.000001 ){
        printf("The Minimum Gap (%lf\") is (Less) than specified (%lf\")\n", gap_mean-
gap_tolerance, gap_min);
    } else if (gap_mean-gap_tolerance > gap_min + 0.000001){
        printf("The Minimum Gap (%lf\") is (Greater) than specified (%lf\")\n", gap_mean-
gap_tolerance, gap_min);
    } else {
        printf("The Minimum Gap (%lf\") is (Equal) to specified (%lf\")\n", gap_mean-
gap_tolerance, gap_min);
    }

    return NO_ERROR;
}




///////////////////////
// Functions for Phase II



int adjust_parts(){
    double gap_tolerance, gap_tolerance_min, gap_tolerance_max;
    int variable_part_cnt;
    double wiggle_room;
    int i, j;

    printf("\nAdjusting parts values\n");

    // getting numbers first
    gap_tolerance = (gap_max - gap_min) / 2.0;
    gap_tolerance_min = gap_tolerance_max = 0.0;
    variable_part_cnt = 0;
```

```
    srand(time(NULL)); // changing the seed so that the random numbers are not the same each
time.

    for( i = 0; i < part_cnt; i++ ){
        // gap_nominal += part_nominal[i] * part_sign[i];
        gap_tolerance_max += part_tolerance[i];
        if( part_variability[i] == 'F' ){
            gap_tolerance_min += part_tolerance[i];
        } else {
               variable_part_cnt++;
        }
    }
    if( gap_tolerance_min > gap_tolerance ){
        printf("Error: The tolerance sum of the fixed parts is too large to work\n");
        return ERROR;
    }

    // Adjust the tolerances
    wiggle_room = (gap_tolerance_max - gap_tolerance) / variable_part_cnt;
    for( i = 0; i < part_cnt; i++ ){
        if( part_variability[i] == 'V' ) part_tolerance[i] -= wiggle_room;;
    }

    // now that the tolerances are adjusted, time for the dimensions
    wiggle_room = ((gap_max - gap_min)/2.0 - gap_mean) / variable_part_cnt;
    for( i = 0; i < part_cnt; i++ ){
        if( part_variability[i] == 'V' ) part_nominal[i] += wiggle_room * part_sign[i];
    }
    update_gap();

    return NO_ERROR;
}




//////////////////////
// Functions for Phase III


int monte_carlo(){
    FILE *fp_in, *fp_out;
    double gap_size;

    printf("\nDoing Monte Carlo Calcs and dumping to file [%s] \n", OUTPUT_FILE_NAME);
    if( ( fp_out = fopen( OUTPUT_FILE_NAME , "w" ) ) != NULL ){
        for(int i = 0; i < SIMULATION_STEPS; i++){
            random_gap(&gap_size);
            fprintf(fp_out, "%lf\n", gap_size);
        }
        fclose(fp_out);
    } else {
         printf("Error: File would not open\n");
         return ERROR;
    }
    return NO_ERROR;
}
```

```
// Calculates a random dimension size given the nominal and tolerance
// This algorithm uses the Box-Muller transformation
void random_dimension(double nominal, double tolerance, double *random_value){
        double r1, r2, w, r;

        do {
             r1 = 2.0 * (( rand() % 10001 ) / 10000.0) - 1.0;
             r2 = 2.0 * (( rand() % 10001 ) / 10000.0) - 1.0;
             w = r1 * r1 + r2 * r2;
         } while ( w >= 1.0 );
        r = sqrt(-2 * log(w) / w );
//printf("rand %f \n", r * r1);
        *random_value = nominal + (tolerance/3) * r * r1;
}



int random_gap(double *gap_random){
    double dimension_random;
    *gap_random = 0.0;

    for(int i = 0; i < part_cnt; i++){
        random_dimension( part_nominal[i], part_tolerance[i], &dimension_random);
        *gap_random += dimension_random * part_sign[i];
    }

    return NO_ERROR;
}
```