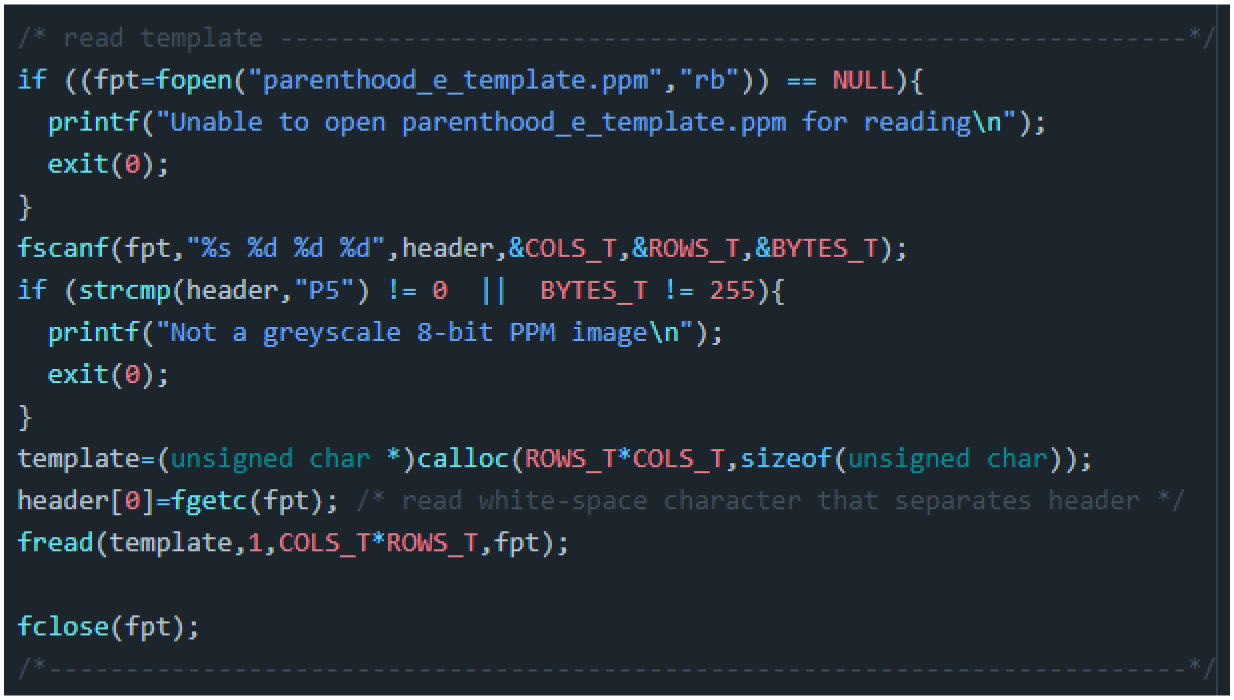
Brandon Shumin
ECE 6310
Lab 2

Reading Input Data, Template Image, and Ground Truth Files:

This step was very easy and carried out essentially the same way as in previous labs so not much detail needs to be mentioned here. The only thing that might be slightly strange was that I decided to read in my ground truth file as three separate arrays, such that the first array contains the correct letter at an index and the second and third arrays contain the columns and rows respectively for said letter at a given index. This is shown in the code below. This program does also take a threshold as a command-line argument, this threshold will be calculated along with a preset range of thresholds.



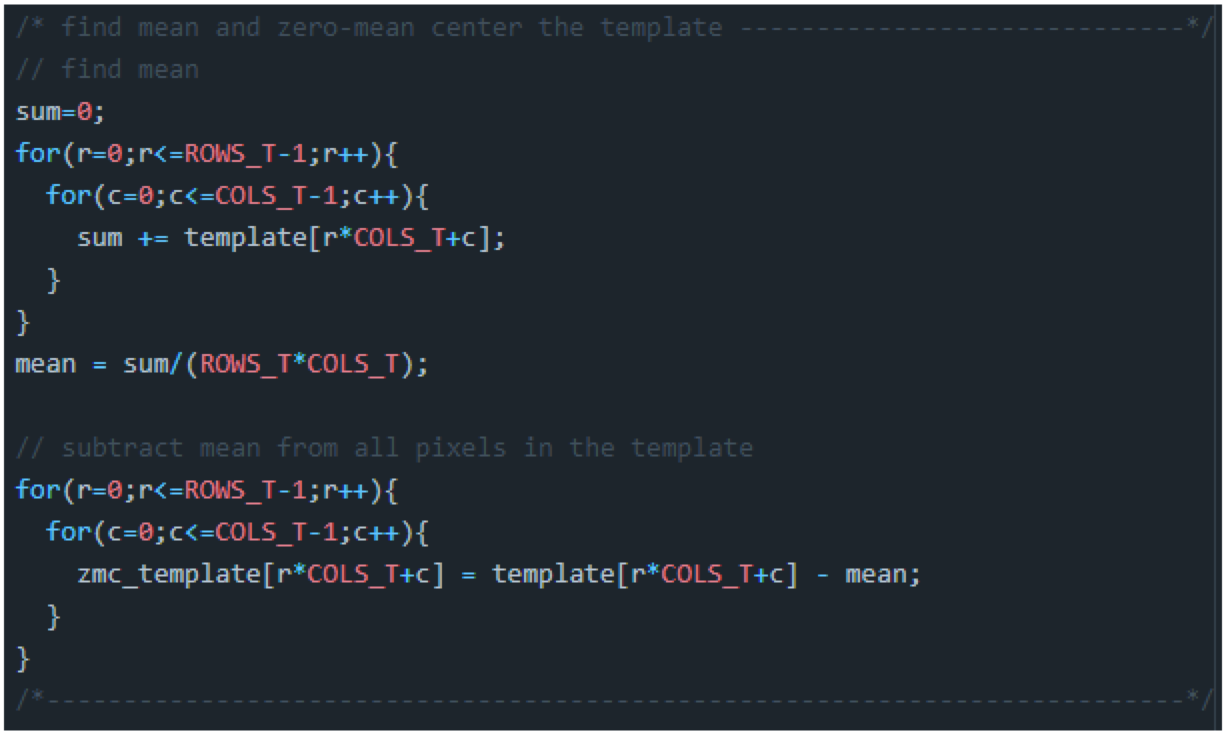
```c
/* read template -----------------------------------------------------------*/
if ((fpt=fopen("parenthood_e_template.ppm","rb")) == NULL){
  printf("Unable to open parenthood_e_template.ppm for reading\n");
  exit(0);
}
fscanf(fpt,"%s %d %d %d",header,&COLS_T,&ROWS_T,&BYTES_T);
if (strcmp(header,"P5") != 0  ||  BYTES_T != 255){
  printf("Not a greyscale 8-bit PPM image\n");
  exit(0);
}
template=(unsigned char *)calloc(ROWS_T*COLS_T,sizeof(unsigned char));
header[0]=fgetc(fpt); /* read white-space character that separates header */
fread(template,1,COLS_T*ROWS_T,fpt);

fclose(fpt);
/*-------------------------------------------------------------------------*/
```

Zero-Mean-Center:

To Zero-Mean-Center the template image for use in calculating the MSF image, first the mean of the template is calculated and then this mean is subtracted from each pixel value in the template to create the ZMC template image. The code for this is shown below:

Brandon Shumin
ECE 6310
Lab 2



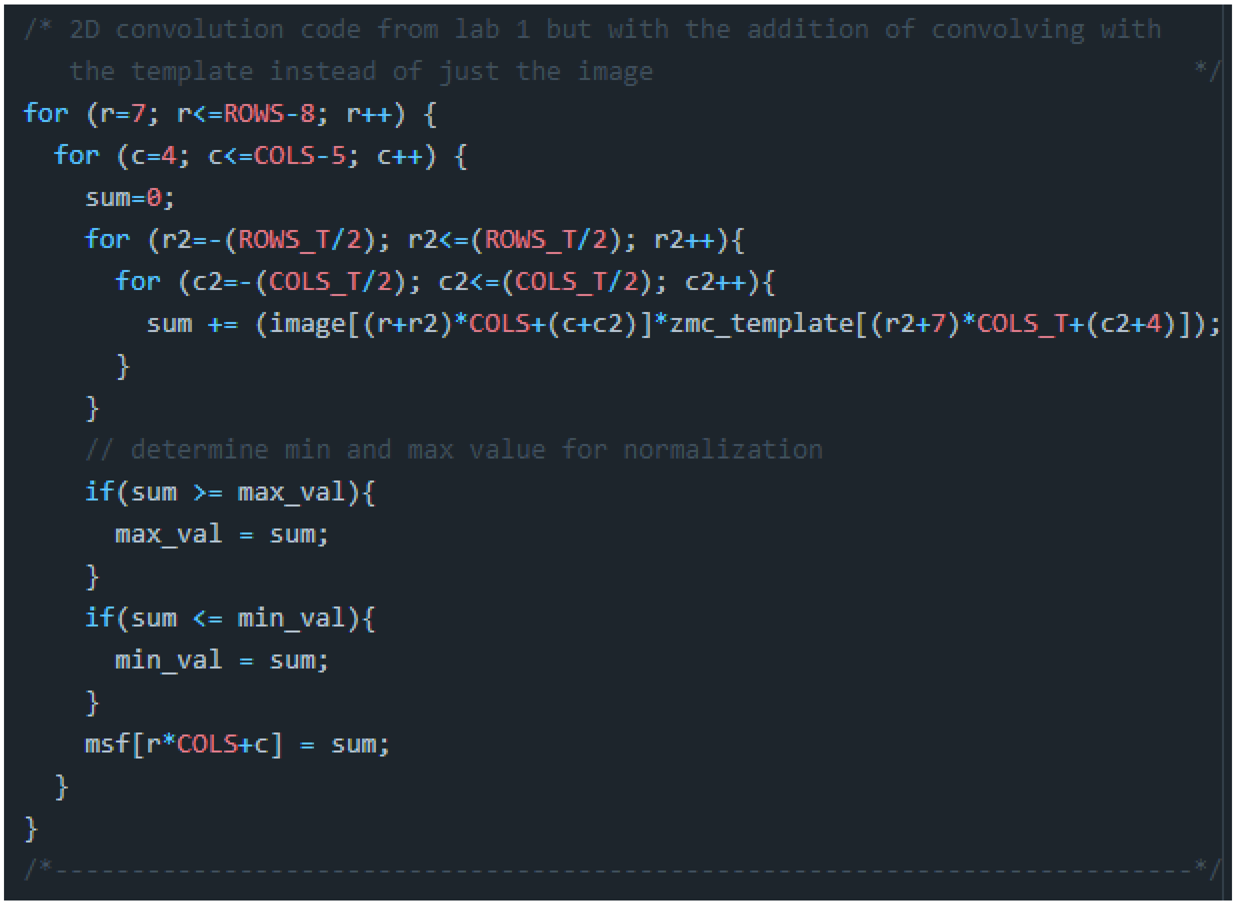
```
/* find mean and zero-mean center the template ----------------------------*/
// find mean
sum=0;
for(r=0;r<=ROWS_T-1;r++){
  for(c=0;c<=COLS_T-1;c++){
    sum += template[r*COLS_T+c];
  }
}
mean = sum/(ROWS_T*COLS_T);

// subtract mean from all pixels in the template
for(r=0;r<=ROWS_T-1;r++){
  for(c=0;c<=COLS_T-1;c++){
    zmc_template[r*COLS_T+c] = template[r*COLS_T+c] - mean;
  }
}
/*-----------------------------------------------------------------------*/
```

Calculating MSF Image:

Using the 2D convolution code from Lab1, the MSF image was created by convolving the image read in with the Zero-Mean-Centered template with a 9x15 area of convolution. During this step, the min and max value of the MSF is determined for later use in normalizing the MSF. The 2D convolution code was chosen since it was the easiest to implement with this new project, and the difference in run-time between this method and the other two methods tested in the previous lab was negligible given the size of this image, so a faster performing method was not needed. The code used to create the MSF and find the min and max values is shown below:

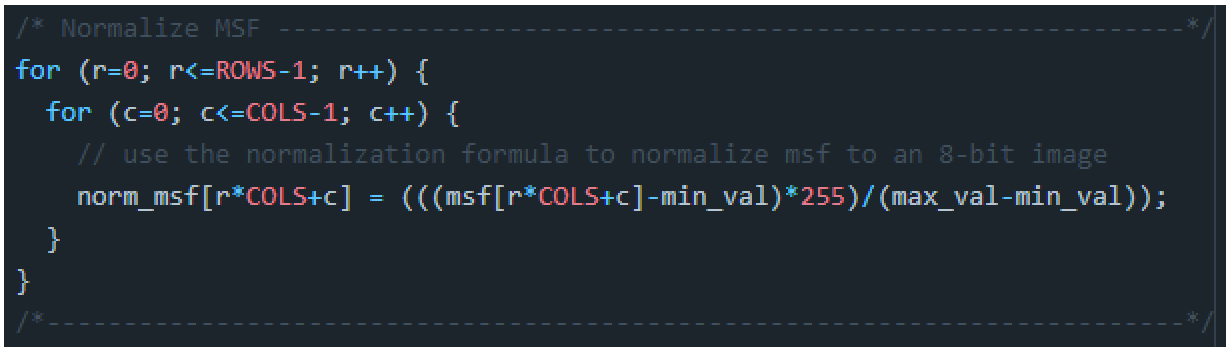
Brandon Shumin
ECE 6310
Lab 2



```
/* 2D convolution code from lab 1 but with the addition of convolving with
   the template instead of just the image                                 */
for (r=7; r<=ROWS-8; r++) {
  for (c=4; c<=COLS-5; c++) {
    sum=0;
    for (r2=-(ROWS_T/2); r2<=(ROWS_T/2); r2++){
      for (c2=-(COLS_T/2); c2<=(COLS_T/2); c2++){
        sum += (image[(r+r2)*COLS+(c+c2)]*zmc_template[(r2+7)*COLS_T+(c2+4)]);
      }
    }
    // determine min and max value for normalization
    if(sum >= max_val){
      max_val = sum;
    }
    if(sum <= min_val){
      min_val = sum;
    }
    msf[r*COLS+c] = sum;
  }
}
/*-----------------------------------------------------------------------*/
```

Normalizing the MSF:

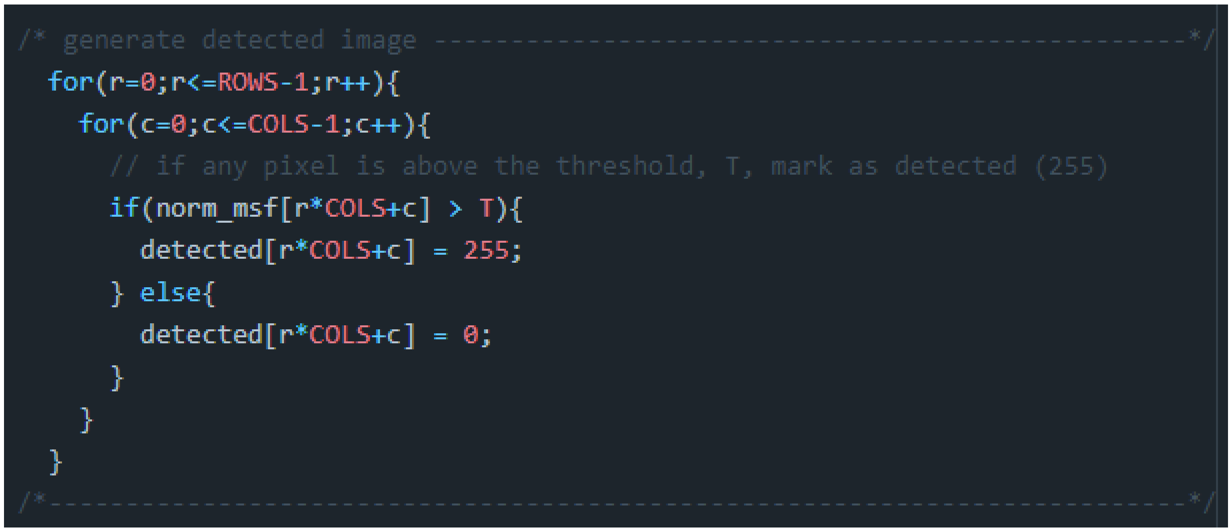
Since the MSF image generated in the above image is not normalized to be 8-bits, the formula $MSF_{normalized}[i] = \frac{(MSF[i]-min)*255}{max-min}$, where “i” is an index of the arrays, is used to normalize the MSF image to be 8-bits. The normalized MSF image is shown in Figure 3. The code for accomplishing this is shown below:



```
/* Normalize MSF --------------------------------------------------------*/
for (r=0; r<=ROWS-1; r++) {
  for (c=0; c<=COLS-1; c++) {
    // use the normalization formula to normalize msf to an 8-bit image
    norm_msf[r*COLS+c] = (((msf[r*COLS+c]-min_val)*255)/(max_val-min_val));
  }
}
/*-----------------------------------------------------------------------*/
```

Generate Binary File for Detections:

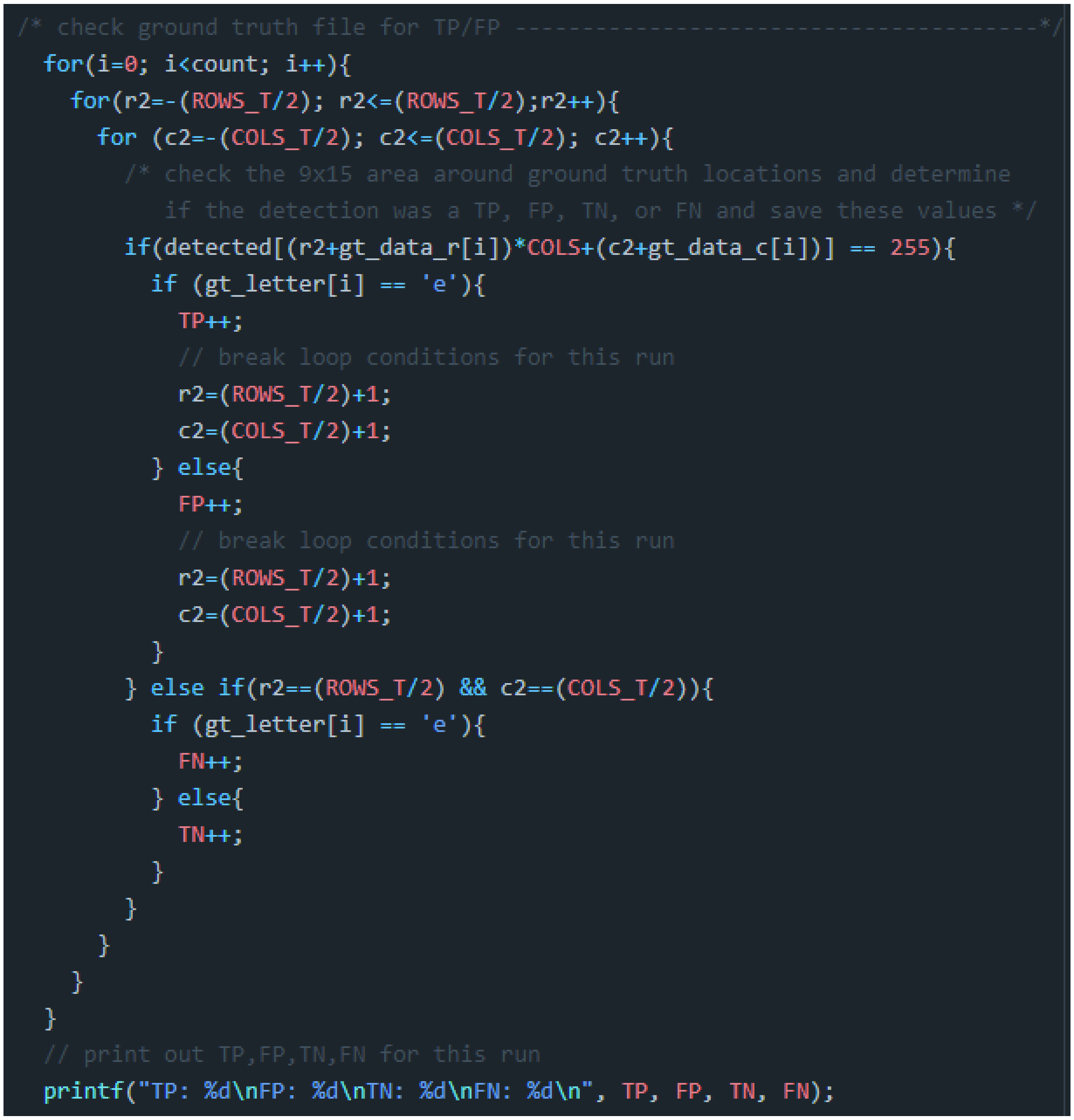
Next, an image of the pixels that are detected at a given threshold is generated. This is accomplished by checking if the values of the normalized MSF are above the threshold value defined. If a pixel value is greater than the threshold, then it is set to 255, else it is 0. In my program, all thresholds between 160 and 240 going by 10 are checked as well as the threshold defined by the user as a command-line argument. This is also the threshold used to print out the binary image at the end of the program. The generated binary image for the user's command-line threshold is shown in Figure 2. This code used to accomplish this is shown below:


```
/* generate detected image ---------------------------------------------*/
  for(r=0;r<=ROWS-1;r++){
    for(c=0;c<=COLS-1;c++){
      // if any pixel is above the threshold, T, mark as detected (255)
      if(norm_msf[r*COLS+c] > T){
        detected[r*COLS+c] = 255;
      } else{
        detected[r*COLS+c] = 0;
      }
    }
  }
/*---------------------------------------------------------------------*/
```

Compare Detections with Ground Truth:

The final step for the program is to check which detections generated in the previous step constitute a True Positive (TP), False Positive (FP), True Negative (TN), or False Negative (FN). To do this, the coordinates for the ground truths are found and then the a 9x15 area around it is checked. If any of the pixels are found to be "detected", or have a value of 255, the letter "e" is said to be detected and the ground truth letter is consulted, if it is also "e" then it is a TP, else it's a FP. If no pixel was "detected" in the 9x15 area around the ground truth, then if the ground truth letter is an "e" it is a FN and if it was not an "e" than it is TN. The code for this is shown below:

Brandon Shumin
ECE 6310
Lab 2

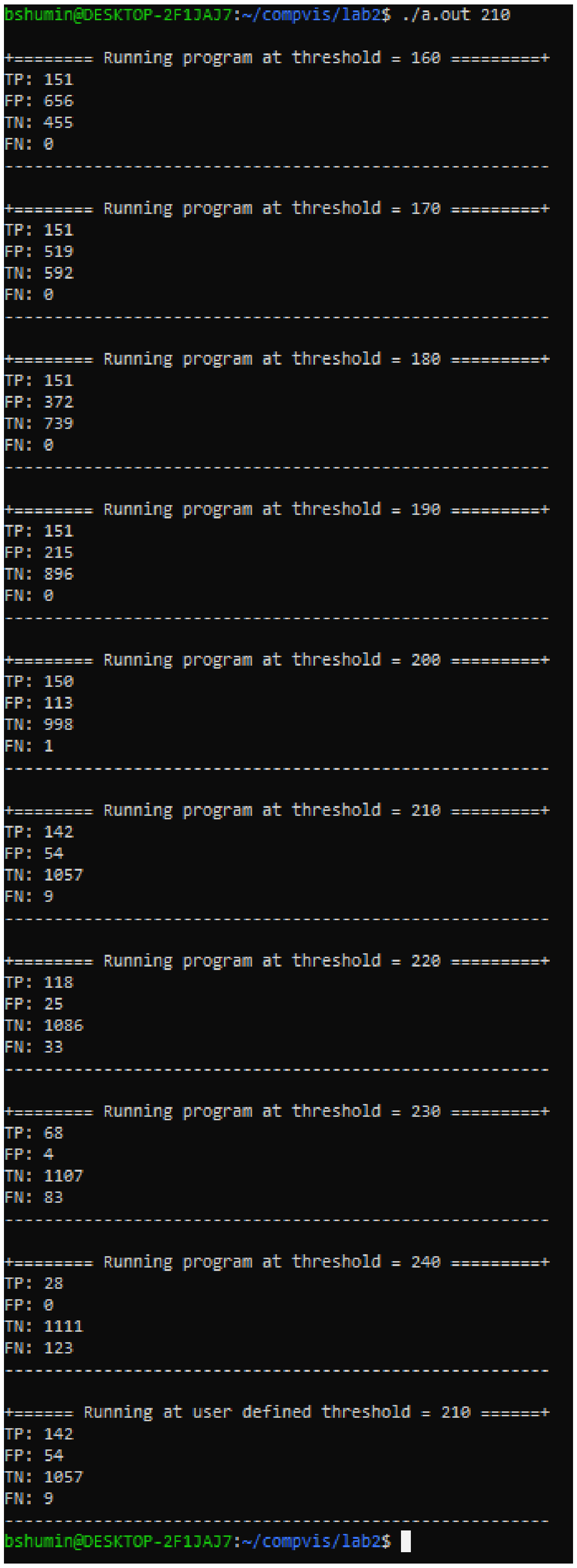


```
/* check ground truth file for TP/FP -------------------------------------*/
  for(i=0; i<count; i++){
    for(r2=-(ROWS_T/2); r2<=(ROWS_T/2);r2++){
      for (c2=-(COLS_T/2); c2<=(COLS_T/2); c2++){
        /* check the 9x15 area around ground truth locations and determine
           if the detection was a TP, FP, TN, or FN and save these values */
        if(detected[(r2+gt_data_r[i])*COLS+(c2+gt_data_c[i])] == 255){
          if (gt_letter[i] == 'e'){
            TP++;
            // break loop conditions for this run
            r2=(ROWS_T/2)+1;
            c2=(COLS_T/2)+1;
          } else{
            FP++;
            // break loop conditions for this run
            r2=(ROWS_T/2)+1;
            c2=(COLS_T/2)+1;
          }
        } else if(r2==(ROWS_T/2) && c2==(COLS_T/2)){
          if (gt_letter[i] == 'e'){
            FN++;
          } else{
            TN++;
          }
        }
      }
    }
  }
  // print out TP,FP,TN,FN for this run
  printf("TP: %d\nFP: %d\nTN: %d\nFN: %d\n", TP, FP, TN, FN);
```

ROC Curve:

Using the outputs generated in Figure 1 below, a table of data, shown in Figure 4, was used to generate the ROC Curve shown in Figure 5. This curve shows that for my program and given the thresholds that I checked, the most appropriate T, or threshold values, would be 210 which would mean the TP count would be 142 and the FP count would be 54. The optimal threshold is 210 because it is the closest threshold to having a TPR of 1 and FPR of 0, which would be a perfect model.

Brandon Shumin
ECE 6310
Lab 2



```
bshumin@DESKTOP-2F1JAJ7:~/compvis/lab2$ ./a.out 210

+======== Running program at threshold = 160 =========+
TP: 151
FP: 656
TN: 455
FN: 0
-------------------------------------------------------

+======== Running program at threshold = 170 =========+
TP: 151
FP: 519
TN: 592
FN: 0
-------------------------------------------------------

+======== Running program at threshold = 180 =========+
TP: 151
FP: 372
TN: 739
FN: 0
-------------------------------------------------------

+======== Running program at threshold = 190 =========+
TP: 151
FP: 215
TN: 896
FN: 0
-------------------------------------------------------

+======== Running program at threshold = 200 =========+
TP: 150
FP: 113
TN: 998
FN: 1
-------------------------------------------------------

+======== Running program at threshold = 210 =========+
TP: 142
FP: 54
TN: 1057
FN: 9
-------------------------------------------------------

+======== Running program at threshold = 220 =========+
TP: 118
FP: 25
TN: 1086
FN: 33
-------------------------------------------------------

+======== Running program at threshold = 230 =========+
TP: 68
FP: 4
TN: 1107
FN: 83
-------------------------------------------------------

+======== Running program at threshold = 240 =========+
TP: 28
FP: 0
TN: 1111
FN: 123
-------------------------------------------------------

+====== Running at user defined threshold = 210 ======+
TP: 142
FP: 54
TN: 1057
FN: 9
-------------------------------------------------------
bshumin@DESKTOP-2F1JAJ7:~/compvis/lab2$
```

Figure 1: Example of running this code with command-line argument and output at determined ideal threshold

Brandon Shumin
ECE 6310
Lab 2

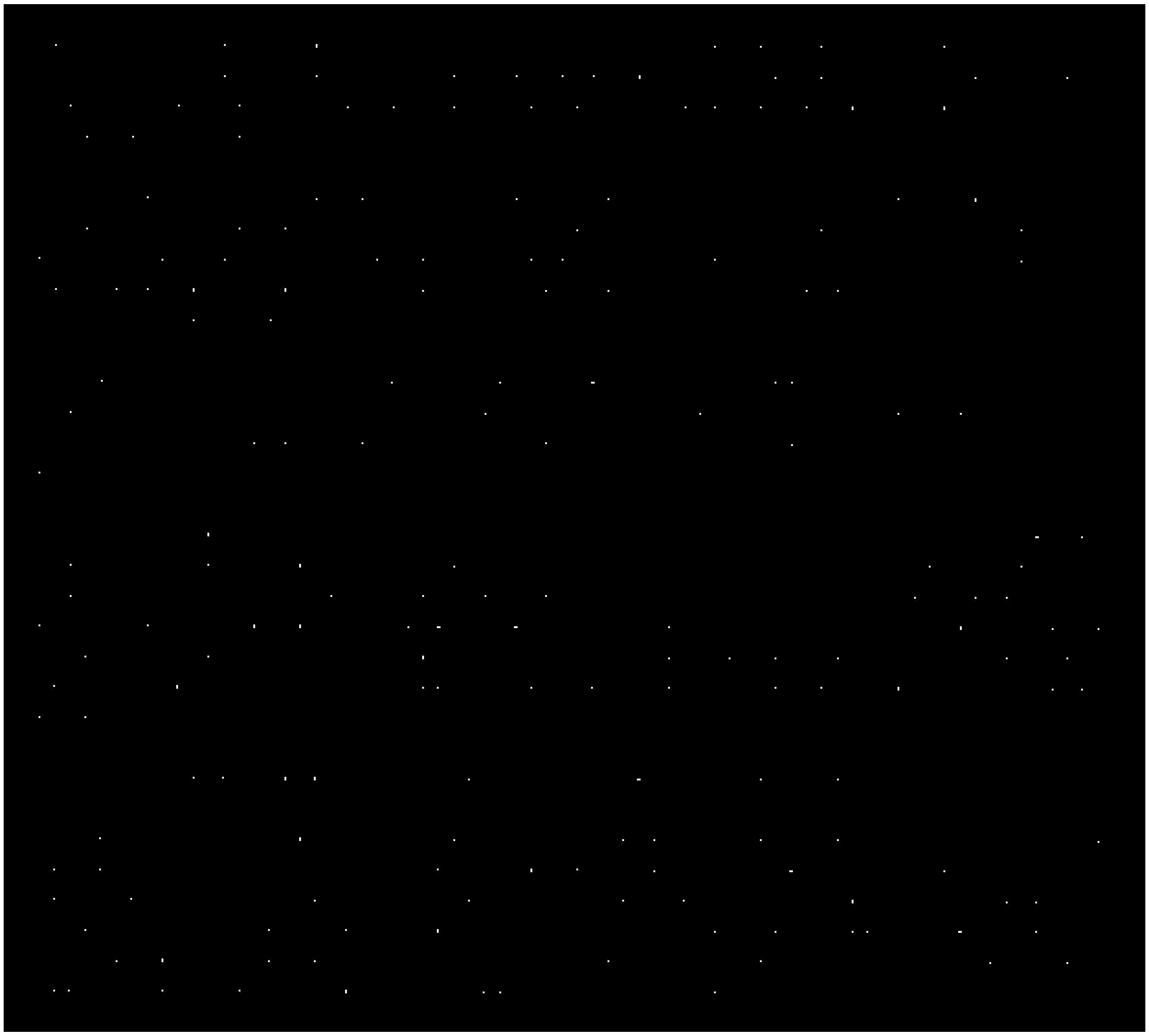


Figure 2: Binary detected pixel image run at threshold = 210

Brandon Shumin
ECE 6310
Lab 2

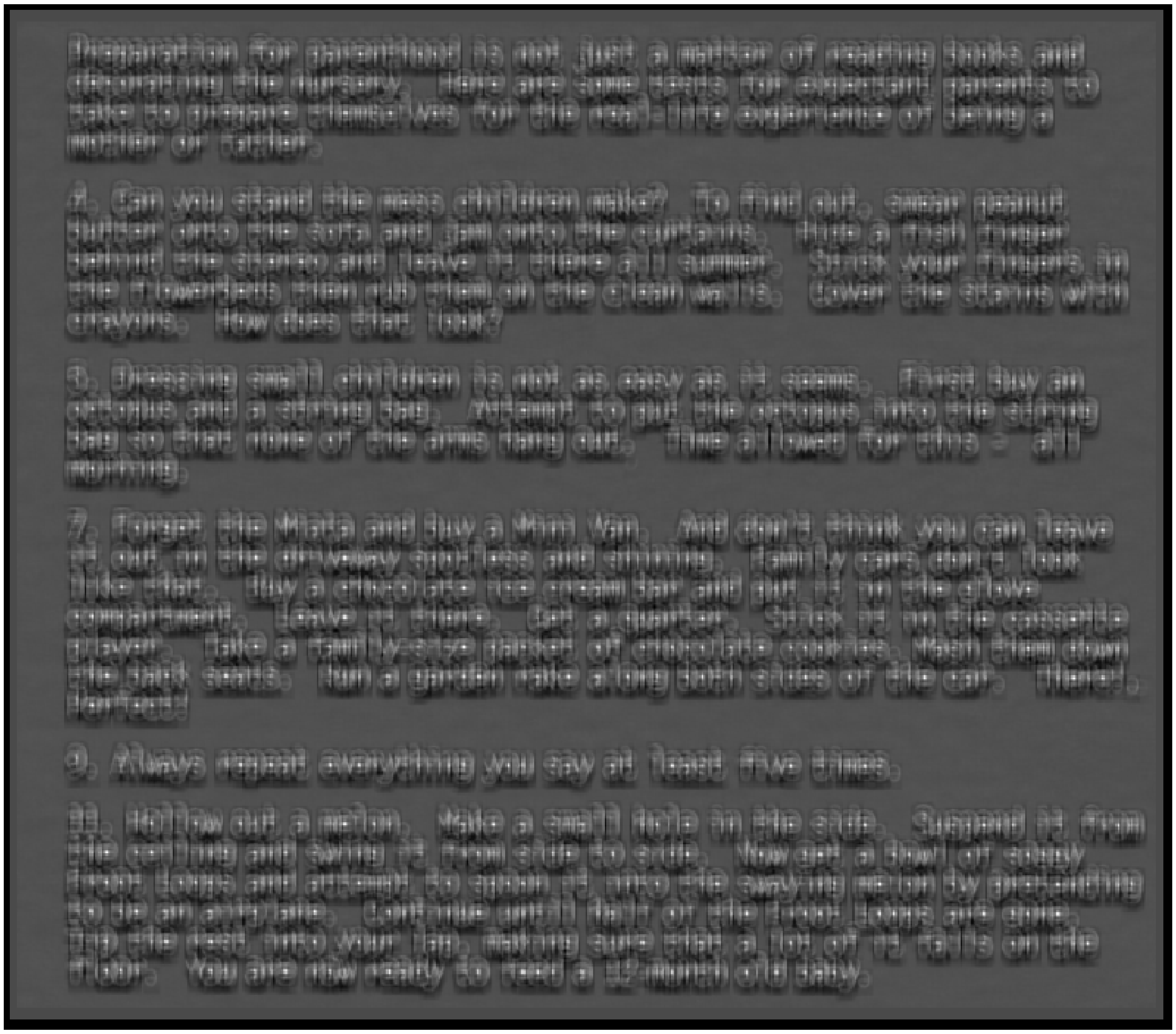


Figure 3: Normalized MSF image (NOTE: black edges are from Irfan's background and are not image borders)

Threshold	TP	FP	TN	FN	FPR	TPR
160	151	656	455	0	0.590459	1
170	151	519	592	0	0.467147	1
180	151	372	739	0	0.334833	1
190	151	215	896	0	0.193519	1
200	150	113	998	1	0.10171	0.993377
210	142	54	1057	9	0.048605	0.940397
220	118	25	1086	33	0.022502	0.781457
230	68	4	1107	83	0.0036	0.450331
240	28	0	1111	123	0	0.18543

Figure 4: Table of TP, FP, TN, FN, TPR, and FPR

Brandon Shumin
ECE 6310
Lab 2

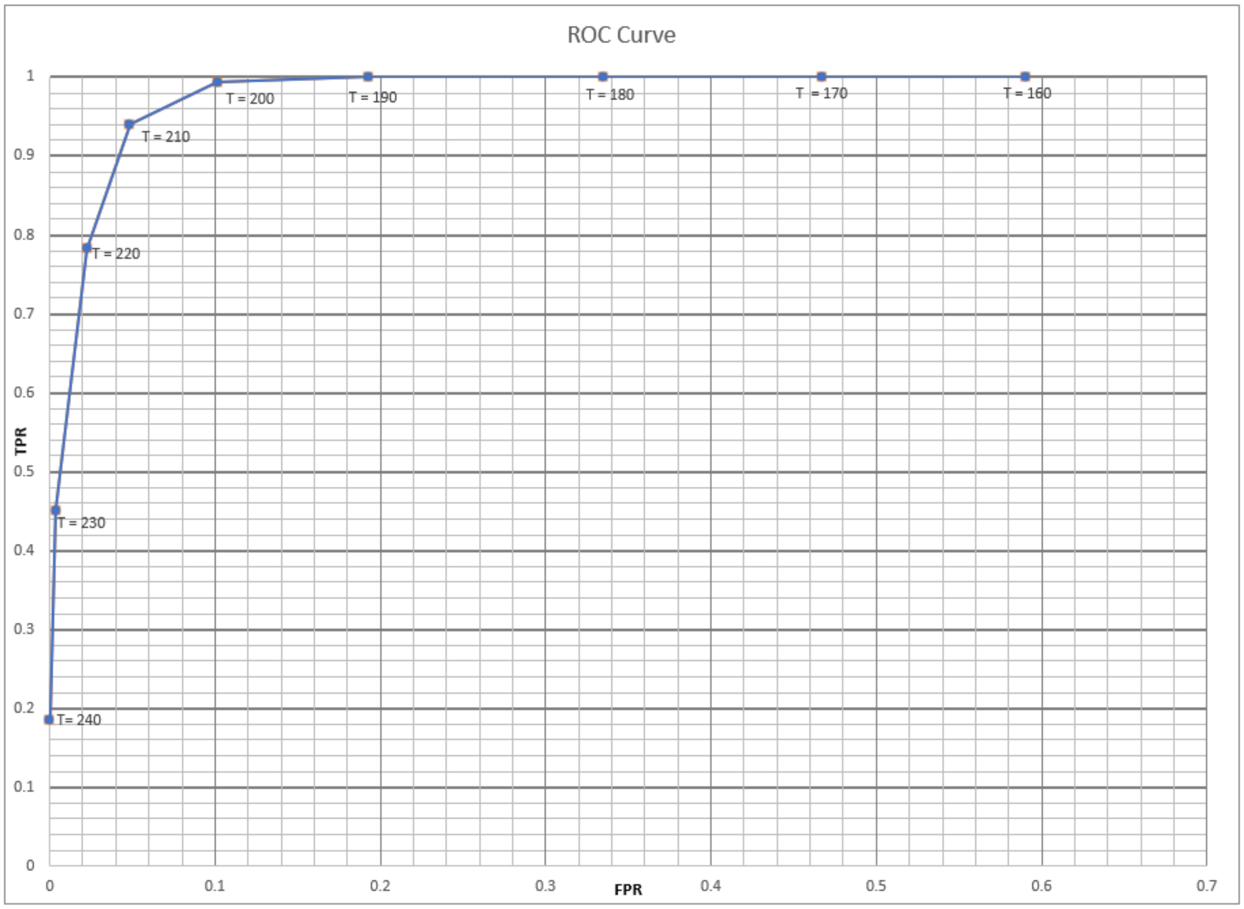


Figure 5: ROC Curve generated from data in Figure 4