Shashi Shivaraju
XID:C88650674

# REPORT
## ECE 6310 Semester Project –Segmenting Foods

**Objective**: To implement an active contour algorithm that allows a user to semi-automatically segment food items in a photograph of multiple foods on a dinner plate.

**Implementation:** The segmentation process of the provided image follows the below steps:

### a) Load a colour PNM image and display its grayscale version.

Images with resolution greater than the screen resolution were down sampled and were displayed to the user in reduced resolution.

[Code snippet]:

```
LRESULT CALLBACK WndProc (HWND hWnd, UINT uMsg,
                                    WPARAM wParam, LPARAM lParam)
{

	switch (uMsg)
	{
	case WM_COMMAND:
		switch (LOWORD(wParam))
		{
			case ID_FILE_LOAD:

			/*Reduce the image resolution to fit to screen*/
				while(screen_width < COLS || screen_height < ROWS)
				{
					temp = (unsigned char *)calloc(3*(ROWS/2)*(COLS/2),1);
					/*reduce the resolution by factor of 2*/
					downsample_rgb_image(FileImage,temp,ROWS,COLS);
					COLS = COLS/2;
					ROWS = ROWS/2;

					free(FileImage);
					FileImage = temp;
				}
				/*Grayscale representation*/
				OriginalImage = (unsigned char *)calloc(ROWS*COLS,1);
				convert_rgb_to_grey(FileImage,OriginalImage,ROWS,COLS);
		}
	}
}
```

Shashi Shivaraju
XID:C88650674

```
void downsample_rgb_image(unsigned char *image,unsigned char *output,int ROW,int COL)
{
	int i=0,j = 0;
	int r = 0,c = 0;

	for(r=0;r<ROW;r=r+2)
	{
		for(c=0;c<COL*3;c=c+6)
		{
			output[j]=image[r*COL*3+c];
			output[j+1]=image[r*COL*3+c+1];
			output[j+2]=image[r*COL*3+c+2];

			j=j+3;
		}
	}

}

void convert_rgb_to_grey(unsigned char *image,unsigned char *grey_image,int ROW,int COL)
{
	int i=0,j = 0;
	float pixel_avg = 0;

	while(i+2 < ROW*COL*3)
	{
		pixel_avg =(float)(image[i]+image[i+1]+image[i+2])/3;
		if(255 <pixel_avg)
			pixel_avg = 255;
		grey_image[j]= (unsigned char)pixel_avg;

		i = i+3;
		j = j+1;
	}
}
```

[Greyscale version of colour image displayed]:

Shashi Shivaraju
XID:C88650674

**b) Active Contour Models:**

Three active contours were used in the project based on Rubber Band Active Contour Model, Balloon Active Contour Model and Neutral Size Active Contour Model.

The Active Contour based on the Rubber Band Active Contour Model was used to segment the food item where in the user draws the initial contour around the food item and then the active contour iteratively shrink to the boundary of the food item. The energy terms used for this active contour were:

- [Internal Energy]Minimizing the average distance between points (rewarding even spacing of points).
- [Internal Energy]Minimizing the distance between points (rewarding the points which shrink the contour).
- [External Energy]Minimizing the negative of the gradient (rewarding points that move towards edges) .
- [External Energy]Minimizing variance of colour with respect to the colour of the centroid of the enclosed contour (rewarding points that homogenize the enclosed contour).

The Active Contour based on the Balloon Active Contour Model was used to segment the food item where in the user selects the initial seed location on the food item and then the active contour iteratively expands to the boundary of the food item. The energy terms used for this active contour were:

- [Internal Energy]Minimizing the average distance between points (rewarding even spacing of points).
- [Internal Energy]Minimizing the curvature of the points from the centroid of the enclosed contour (rewarding the points which expand the contour).
- [External Energy]Minimizing the negative of the gradient (rewarding points that move towards edges) .
- [External Energy]Minimizing the variance of colour with the colour of the centroid of the enclosed contour (rewarding points that homogenize the enclosed contour).

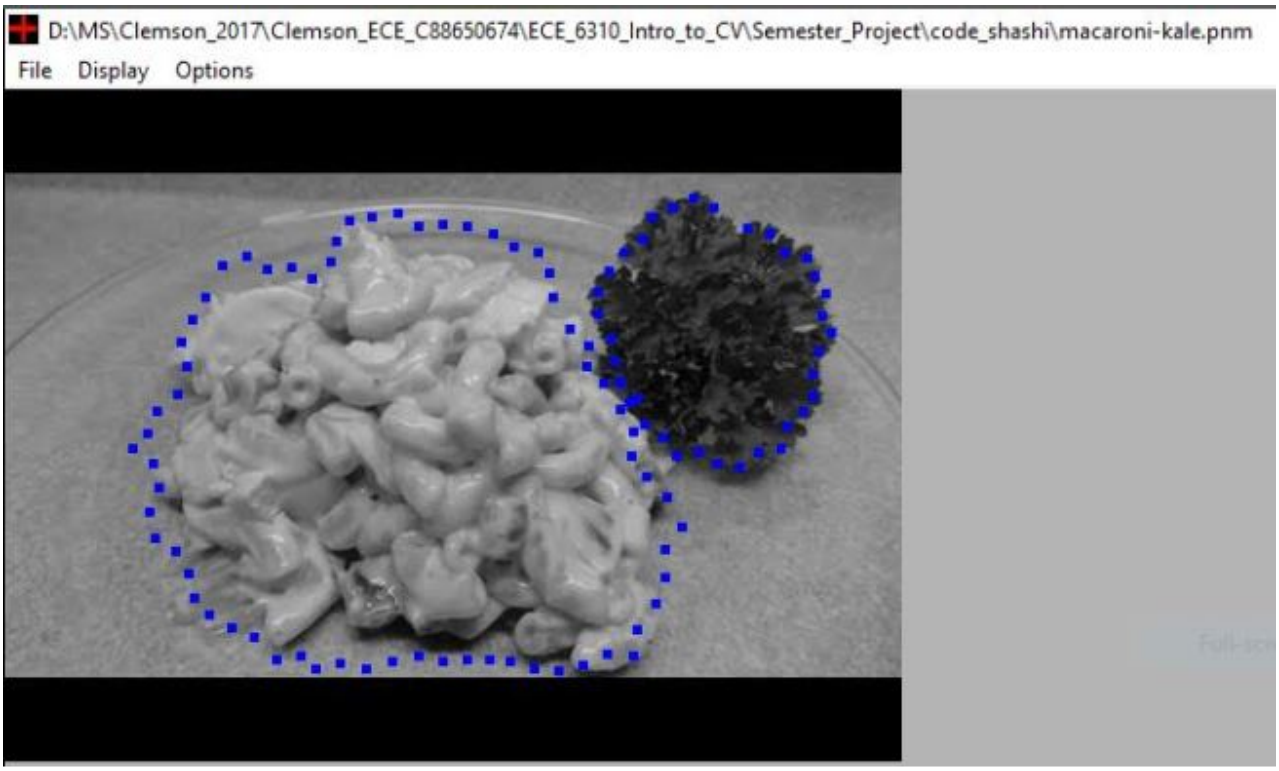
The neutral size active contour Active Contour has the following energy terms with the internal energy term weighted more than the external energy term:

- [Internal Energy]Minimizing the average distance between points (rewarding even spacing of points).
- [External Energy]Minimizing the negative of the gradient (rewarding points that move towards edges) .
- [External Energy]Minimizing the variance of colour with the colour of the centroid of the enclosed contour (rewarding points that homogenize the enclosed contour).

[Code for each of the Active contour implementation has been attached at the end of the document.]

Shashi Shivaraju
XID:C88650674

**Results:**
[Image 1: macaroni-kale.pnm]

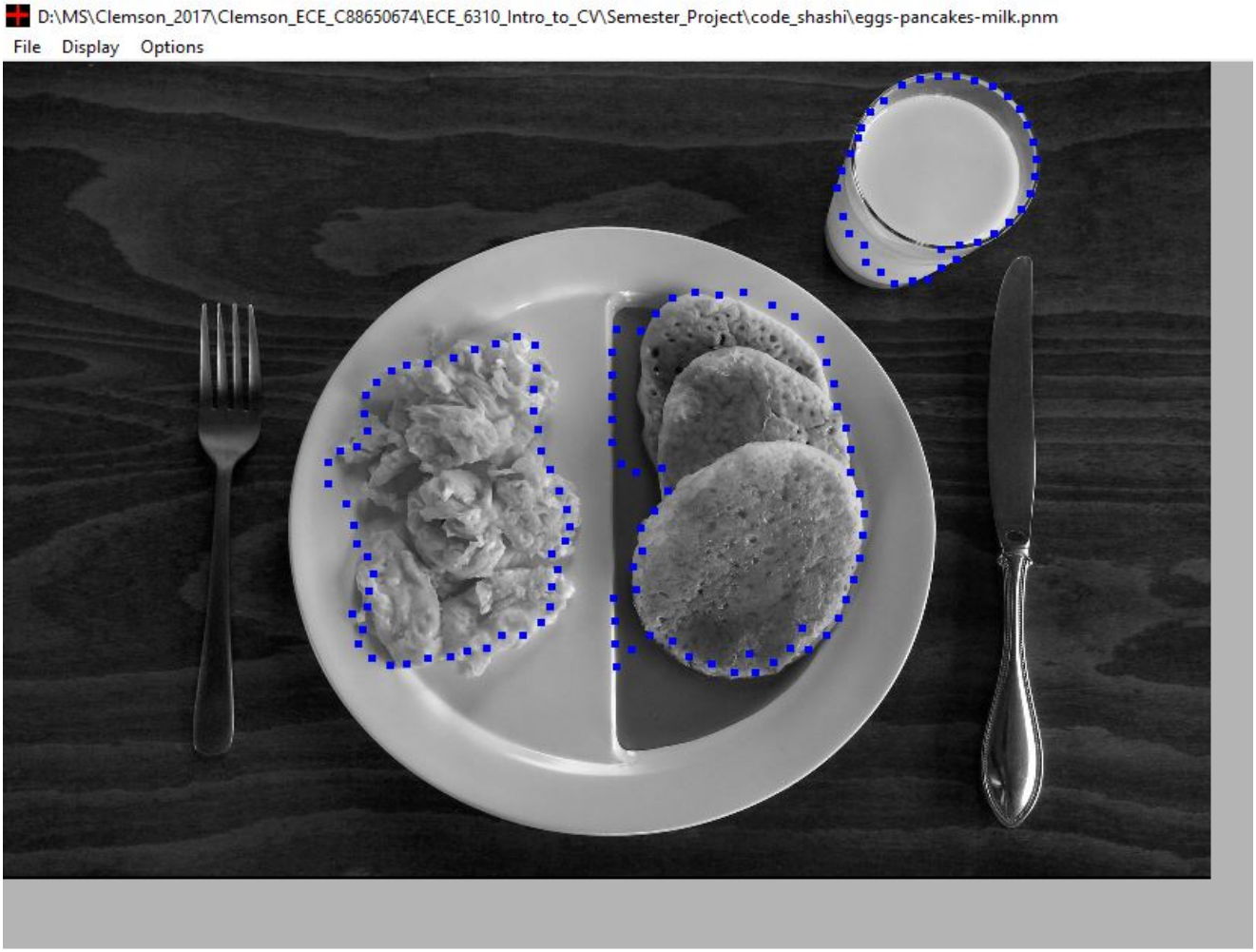



Observations:

- Balloon Active contour model worked well for segmenting Kale.
- Kale has good edge gradient values compared to Macaroni, hence it was much easier to segment.
- The RGB components of the Macaroni and the background were having similar values and hence less variations in the External energy. Due to this the balloon model did not perform well.
- Contour did not deform at very sharp variations in shape.

Shashi Shivaraju
XID:C88650674

[Image 2: eggs-pancakes-milk.pnm]

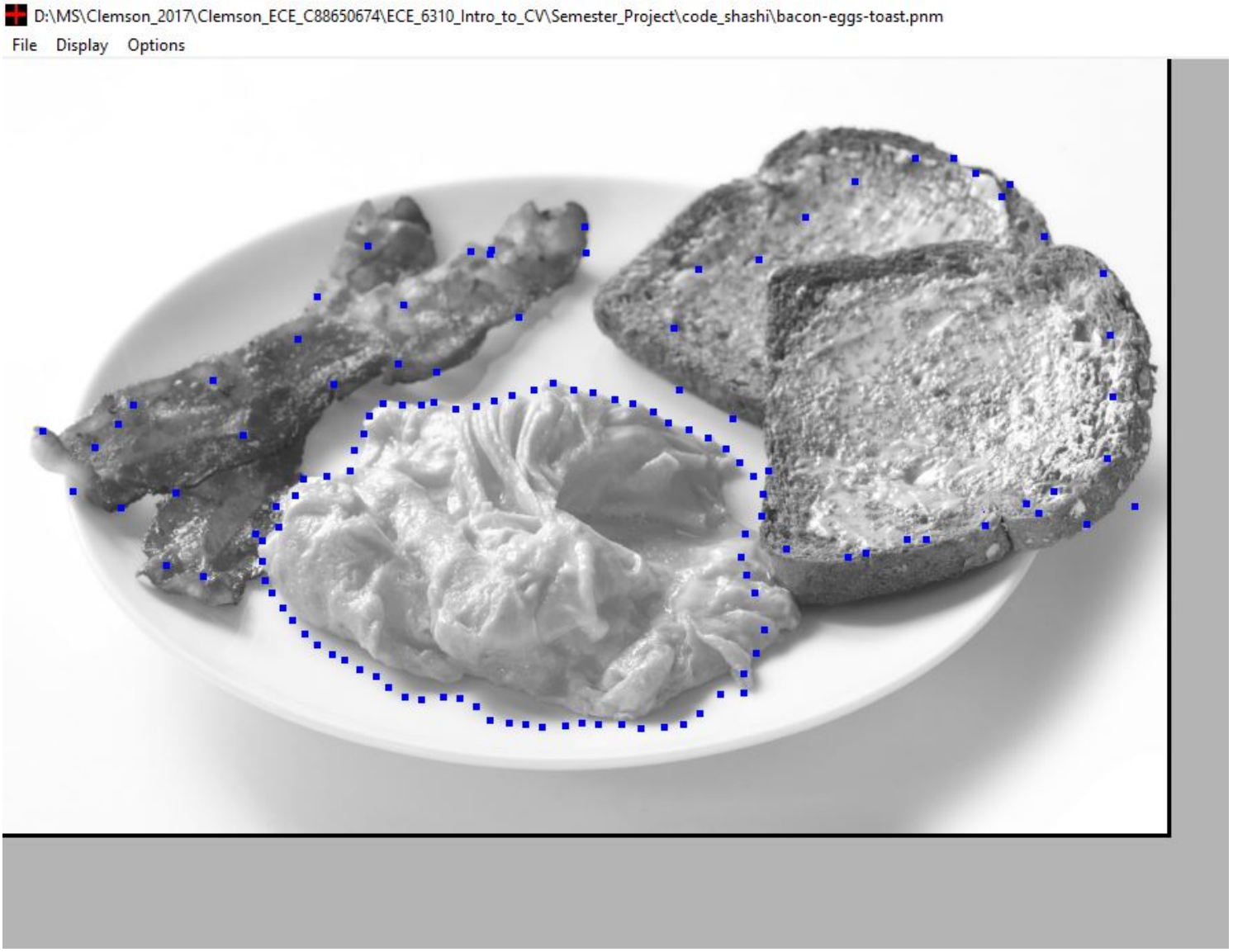



Observations:

- Rubber Band Active contour model worked well for segmenting Egg and Milk.
- Segmentation of Pancake was difficult with varied results due to the design of the plate(the middle partition for separating the eggs from the pancake syrup).
- Balloon model worked satisfactorily for milk but performance were bad for eggs and pancake.

Shashi Shivaraju
XID:C88650674

[Image 3: bacon-eggs-toast.pnm]

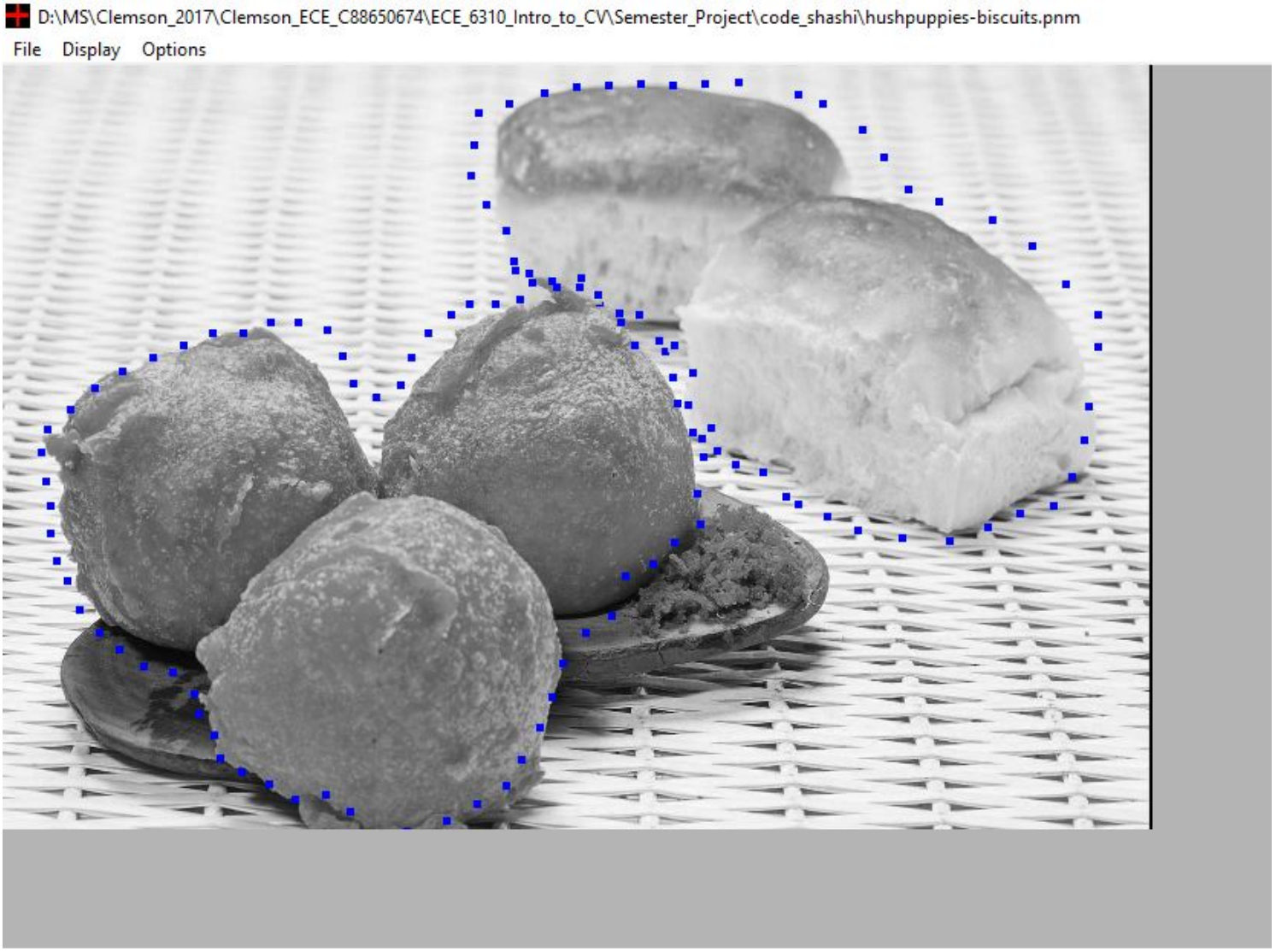



Observations:

- Balloon active contour model worked well for segmenting Bacon and satisfactorily for Bread and both required few manual contour point adjustments.
- Rubber band model worked well for eggs than compared to Balloon model.
- Overlapping of the food items caused performance degradation for Balloon model and the shape/bends of bacon caused issues for the Rubber band model to properly deform.
- Also position of the food items at the extremities of the picture caused problems for both contour models.

Shashi Shivaraju
XID:C88650674

[Image 4: hushpuppies-biscuits.pnm]




Observations:

- Rubber Band Model worked well compared to Balloon Model.
- Overlapping of the food items caused performance degradation for both Rubber Band and Balloon Active Contour Models.
- Position of the food item at the extremities of the picture caused problem.

Shashi Shivaraju
XID:C88650674

[Image 4: hushpuppies-biscuits.pnm]


Observations:

- Rubber Band Model worked well compared to Balloon Model.
- Overlapping of the food items caused performance degradation for Balloon Active Contour Model.

.

Shashi Shivaraju
XID:C88650674

Generic Issues:

- For Balloon Model, the performance varies with the position of the seed location clicked.
- For Rubber Band Model, the performance varies as the location of the initial contour is drawn away from the object which has to be segmented.

Possible Techniques to improve the performance:

- Use better edge detection Algorithm like Canny's Edge detection to get better gradient detections.
- Use gradient vector flow (GVF) snake an enhancement of the traditional snakes.
- For segmentation process, if the food items are placed separated by each other then the process will be much easier to accomplish.

**[Code Snippet for Rubber Band Active Contour Model]:**

```
void RubberBandActiveContourAlgo(unsigned char *image,unsigned char *sobel_image,unsigned char
*rgb_image,contour_position* contour_head,int ROW,int COL, Algo_options* AlgoOpt)
{
	/*calculate the max and min Sobel gradient*/
	for(i=0;i<ROW*COL;i++)
	{
		sobel_pixel_square = SQR(sobel_image[i]);
		if(!i || sobel_pixel_square < min_ext_energy)
			min_ext_energy = sobel_pixel_square;

		if(!i|| sobel_pixel_square > max_ext_energy)
			max_ext_energy = sobel_pixel_square;

	}

	for(i=0;i<Algo_Opt.left_contour_iteration;i++)
	{
		contour_head_xpos = contour_head->x_pos;
		contour_head_ypos = contour_head->y_pos;

		/*To calculate the average distance between the contour points*/
		CalculateAverageContourDistance(contour_head,&avrg_contour_distance);

		/*To claculate the centroid of the countour points and find its rgb componets*/
		CalculateContourCentroid(contour_head,&centroid_xpos,&centroid_ypos);
		ret = convert_height_width2rgbindex(&index,ROW,COL,centroid_xpos,centroid_ypos);
		if(ret != 0 || 0 > index )
			continue;

		red_avg = rgb_image[index];
		green_avg = rgb_image[index+1];
		blue_avg = rgb_image[index+2];
```

```
/*To calculate the internal energies and
external energy at each contour point around pixel window*/
curr_cont = next_cont = contour_head;
while(next_cont)
{

      curr_cont = next_cont;
      next_cont = curr_cont->next;

      cur_xpos = curr_cont->x_pos;
      cur_ypos = curr_cont->y_pos;


      if(next_cont)
      {
            next_xpos = next_cont->x_pos;
            next_ypos = next_cont->y_pos;
      }
      else /*Assume the last contour point connects first contour point*/
      {
            next_xpos = contour_head_xpos;
            next_ypos = contour_head_ypos;
      }

      count = 0;
      for(r1=-(LEFT_CLICK_WINDOW/2);r1<=(LEFT_CLICK_WINDOW/2);r1++)
      {
            for(c1=-(LEFT_CLICK_WINDOW/2);c1<=(LEFT_CLICK_WINDOW/2);c1++)
            {
                  /*Internal Energy 1
                  (Square of distance between contour points)*/
                  distance_square = SQR(next_xpos-(cur_xpos+c1))+\
                        SQR(next_ypos-(cur_ypos+r1));
                  internal_energy_1[count] = distance_square;

                  if(!count|| distance_square < min_energy1)
                        min_energy1 = distance_square;

                  if(!count|| distance_square > max_energy1)
                        max_energy1 = distance_square;

                  /*Internal Energy 2
                  (Square of difference between average contour distance
                  and the distance between current contour points )*/
                  distance_diff_square = SQR(avrg_contour_distance -
sqrt(distance_square));
                  internal_energy_2[count] = distance_diff_square;

                  if(!count || distance_diff_square < min_energy2)
                        min_energy2 = distance_diff_square;

                  if(!count|| distance_diff_square > max_energy2)
                        max_energy2 = distance_diff_square;
```

```
                        /*External Energy
                        (Square of the image gradient magnitude(Sobel convoluted image))*/
                              ret =
convert_height_width2index(&index,ROW,COL,cur_xpos+c1,cur_ypos+r1);
                              if(ret != 0 || 0 > index )
                                    continue;

                              sobel_pixel_square = SQR(sobel_image[index]);
                              external_energy[count] = sobel_pixel_square;

                  /*External Energy 2
      (Variance of the rgb image intensity from the rgb image intensity of the centroid)*/
                              ret =
convert_height_width2rgbindex(&index,ROW,COL,cur_xpos+c1,cur_ypos+r1);
                              if(ret != 0 || 0 > index )
                                    continue;

                              variance = SQR(abs(red_avg-rgb_image[index])+\
                                    abs(green_avg-rgb_image[index+1])+abs(blue_avg-
rgb_image[index+2]));
                              external_energy_2[count] = variance;
                              if(!count || variance < min_ext_eng_2)
                                    min_ext_eng_2 = variance;

                              if(!count|| variance > max_ext_eng_2)
                                    max_ext_eng_2 = variance;

                              count ++;
                        }
                  }

                  count = 0;min_total_energy = 0;
                  ext_eng =0;ext_energy_2 = 0;int_eng1 = 0;int_eng2 = 0;total_eng = 0;

                  /*Normalise the individual energies and find total energy*/
                  energy1_range       = max_energy1 - min_energy1;
                  energy2_range       = max_energy2 - min_energy2;
                  ext_energy_range  = max_ext_energy - min_ext_energy;
                  ext_energy_2_range = max_ext_eng_2 - min_ext_eng_2;

                  for(j=0;j<(SQR(LEFT_CLICK_WINDOW));j++)
                  {
                        int_eng1 = ((internal_energy_1[j]-min_energy1)/energy1_range)* AlgoOpt-
>left_IE1_W;
                        int_eng2 = ((internal_energy_2[j]-min_energy2)/energy2_range)* AlgoOpt-
>left_IE2_W;
                        ext_eng = ((external_energy[j]-min_ext_energy)/ext_energy_range)*
AlgoOpt->left_EE1_W;
                        ext_energy_2 = ((external_energy_2[j]-
min_ext_eng_2)/ext_energy_2_range)* AlgoOpt->left_EE2_W;

                        total_eng = int_eng1+int_eng2-ext_eng+ext_energy_2;
                        total_energy[j] = total_eng;
                        if(!j || total_eng < min_total_energy)
                        {
                              min_total_energy = total_eng;
                              count = j;/*location of lowest total energy in window*/
                        }
                  }
```

Shashi Shivaraju
XID:C88650674

```
                  /*udpate the new contour position to the lowest total energy location in
window*/
                  ret =
convert_index2height_width(count,LEFT_CLICK_WINDOW,LEFT_CLICK_WINDOW,&next_xpos,&next_ypos);
                  if(ret != 0 || 0 > next_xpos || 0 > next_ypos )
                        continue;

                  curr_cont->last_x_pos = curr_cont->x_pos;
                  curr_cont->last_y_pos = curr_cont->y_pos;

                  if(next_xpos>(LEFT_CLICK_WINDOW/2))
                        curr_cont->x_pos = curr_cont->x_pos + (next_xpos-
(LEFT_CLICK_WINDOW/2));
                  else if(next_xpos<(LEFT_CLICK_WINDOW/2))
                        curr_cont->x_pos = curr_cont->x_pos -
(next_xpos+(LEFT_CLICK_WINDOW/2));

                  if(next_ypos>(LEFT_CLICK_WINDOW/2))
                        curr_cont->y_pos = curr_cont->y_pos + (next_ypos-
(LEFT_CLICK_WINDOW/2));
                  else if(next_ypos<(LEFT_CLICK_WINDOW/2))
                        curr_cont->y_pos = curr_cont->y_pos + (next_ypos-
(LEFT_CLICK_WINDOW/2));

                  ret = convert_height_width2index(&curr_cont->index,ROW,COL,curr_cont-
>x_pos,curr_cont->y_pos);
                  if(ret != 0 || 0 > curr_cont->index )
                        continue;

            }
      }

}
```

Shashi Shivaraju
XID:C88650674

**[Code Snippet for Balloon Active Contour Model]:**

```
void BalloonActiveContourAlgo(unsigned char *image,
                                        unsigned char *sobel_image,
                                        unsigned char *rgb_image,
                                        contour_position* contour_head,
                                        int ROW,int COL,
                                        int seed_xpos,int seed_ypos,
                                         Algo_options* AlgoOpt)
{

      /*To calculate the max and min  Sobel gradient of the image*/
      for(i=0;i<ROW*COL;i++)
      {
            sobel_pixel_square = SQR(sobel_image[i]);
            if(!i || sobel_pixel_square < min_ext_energy)
                  min_ext_energy = sobel_pixel_square;

            if(!i|| sobel_pixel_square > max_ext_energy)
                  max_ext_energy = sobel_pixel_square;

      }

      for(i=0;i<Algo_Opt.right_contour_iteration;i++)
      {
            contour_head_xpos = contour_head->x_pos;
            contour_head_ypos = contour_head->y_pos;

            /*To calculate the average distance between the contour points*/
            CalculateAverageContourDistance(contour_head,&avrg_contour_distance);

            /*To claculate the centroid of the countour points*/
            CalculateContourCentroid(contour_head,&centroid_xpos,&centroid_ypos);


            ret = convert_height_width2rgbindex(&index,ROW,COL,centroid_xpos,centroid_ypos);
            if(ret != 0 || 0 > index )
                  continue;

            red_avg = rgb_image[index];
            green_avg = rgb_image[index+1];
            blue_avg = rgb_image[index+2];


            /*To calculate the internal energies and
            external energy at each contour point around  pixel window*/
            curr_cont = next_cont = contour_head;
            while(next_cont)
            {

                  curr_cont = next_cont;
                  next_cont = curr_cont->next;

                  cur_xpos = curr_cont->x_pos;
                  cur_ypos = curr_cont->y_pos;

                  if(next_cont)
                  {
                        next_xpos = next_cont->x_pos;
```

```
                        next_ypos = next_cont->y_pos;
                }
                else /*Assume the last contour point connects first contour point*/
                {
                        next_xpos = contour_head_xpos;
                        next_ypos = contour_head_ypos;
                }

                count = 0;
                for(r1=-(RIGHT_CLICK_WINDOW/2);r1<=(RIGHT_CLICK_WINDOW/2);r1++)
                {
                        for(c1=-(RIGHT_CLICK_WINDOW/2);c1<=(RIGHT_CLICK_WINDOW/2);c1++)
                        {
                                /*Internal Energy 1
                                (curvature of the contour point from the centroid)*/
                                radius_square = SQR(centroid_xpos-(cur_xpos+c1))+\
                                        SQR(centroid_ypos-(cur_ypos+r1));
                                curve_square = 1/radius_square;
                                internal_energy_1[count] = curve_square;

                                if(!count|| curve_square < min_energy1)
                                        min_energy1 = curve_square;

                                if(!count|| curve_square > max_energy1)
                                        max_energy1 = curve_square;

                                /*Internal Energy 2
                                (Square of difference between average contour distance
                                and the distance between current contour points )*/
                                distance_square = SQR(next_xpos-(cur_xpos+c1))+\
                                        SQR(next_ypos-(cur_ypos+r1));
                                distance_diff_square = SQR(avrg_contour_distance -
sqrt(distance_square));
                                internal_energy_2[count] = distance_diff_square;

                                if(!count || distance_diff_square < min_energy2)
                                        min_energy2 = distance_diff_square;

                                if(!count|| distance_diff_square > max_energy2)
                                        max_energy2 = distance_diff_square;

                                /*External Energy
                                (Square of the image gradient magnitude(Sobel convoluted
image))*/
                                ret =
convert_height_width2index(&index,ROW,COL,cur_xpos+c1,cur_ypos+r1);
                                if(ret != 0 || 0 > index )
                                        continue;
                                sobel_pixel_square = SQR(sobel_image[index]);
                                external_energy[count] = sobel_pixel_square;


                                /*External Energy 2
                                (Variance of the image rgb intensity from
                                the  rgb intensity of the centroid points)*/

                                ret =
convert_height_width2rgbindex(&index,ROW,COL,cur_xpos+c1,cur_ypos+r1);
                                if(ret != 0 || 0 > index )
```

```
                                    continue;

                              variance = SQR(abs(red_avg-rgb_image[index])+\
                                    abs(green_avg-rgb_image[index+1])+abs(blue_avg-
rgb_image[index+2]));
                              external_energy_2[count] = variance;
                              if(!count || variance < min_ext_eng_2)
                                    min_ext_eng_2 = variance;

                              if(!count|| variance > max_ext_eng_2)
                                    max_ext_eng_2 = variance;


                              count ++;
                        }
                  }

                  count = 0;min_total_energy = 0;variance = 0;
                  ext_eng =0;ext_energy_2 = 0;ext_energy_3 = 0;int_eng1 = 0;int_eng2 =
0;total_eng = 0;

                  /*Normalise the individual energies and find total energy*/
                  energy1_range       = max_energy1 - min_energy1;
                  energy2_range       = max_energy2 - min_energy2;
                  ext_energy_range  = max_ext_energy - min_ext_energy;
                  ext_energy_2_range = max_ext_eng_2 - min_ext_eng_2;

                  for(j=0;j<(SQR(RIGHT_CLICK_WINDOW));j++)
                  {
                        int_eng1 = ((internal_energy_1[j]-min_energy1)/energy1_range)*AlgoOpt-
>right_IE1_W;
                        int_eng2 = ((internal_energy_2[j]-min_energy2)/energy2_range)*AlgoOpt-
>right_IE2_W;
                        ext_eng = ((external_energy[j]-
min_ext_energy)/ext_energy_range)*AlgoOpt->right_EE1_W;
                        ext_energy_2 = ((external_energy_2[j]-
min_ext_eng_2)/ext_energy_2_range)*AlgoOpt->right_EE2_W;

                        total_eng = int_eng1+int_eng2+ext_energy_2-ext_eng;
                        total_energy[j] = total_eng;
                        if(!j || total_eng < min_total_energy)
                        {
                              min_total_energy = total_eng;
                              count = j;/*location of lowest total energy in window*/
                        }
                  }

                  /*udpate the new contour position to the lowest total energy location in 7x7
window*/
                  ret =
convert_index2height_width(count,RIGHT_CLICK_WINDOW,RIGHT_CLICK_WINDOW,&next_xpos,&next_ypos);
                  if(ret != 0 || 0 > next_xpos || 0>next_ypos )
                        continue;
                  curr_cont->last_x_pos = curr_cont->x_pos;
                  curr_cont->last_y_pos = curr_cont->y_pos;

                  if(next_xpos>(RIGHT_CLICK_WINDOW/2))
                        curr_cont->x_pos = curr_cont->x_pos + (next_xpos-
(RIGHT_CLICK_WINDOW/2));
```

```
                  else if(next_xpos<(RIGHT_CLICK_WINDOW/2))
                        curr_cont->x_pos = curr_cont->x_pos -
(next_xpos+(RIGHT_CLICK_WINDOW/2));

                  if(next_ypos>(RIGHT_CLICK_WINDOW/2))
                        curr_cont->y_pos = curr_cont->y_pos + (next_ypos-
(RIGHT_CLICK_WINDOW/2));
                  else if(next_ypos<(RIGHT_CLICK_WINDOW/2))
                        curr_cont->y_pos = curr_cont->y_pos + (next_ypos-
(RIGHT_CLICK_WINDOW/2));

                  ret = convert_height_width2index(&curr_cont->index,ROW,COL,curr_cont-
>x_pos,curr_cont->y_pos);
                  if(ret != 0 || 0 > curr_cont->index )
                        continue;
            }
      }
}

void AlterActiveContourAlgo(unsigned char *image,unsigned char *sobel_image,
                                                unsigned char *rgb_image,contour_position*
contour_head,int ROW,int COL,Algo_options* AlgoOpt)
{

      for(i=0;i<ROW*COL;i++)
      {
            sobel_pixel_square = SQR(sobel_image[i]);
            if(!i || sobel_pixel_square < min_ext_energy)
                  min_ext_energy = sobel_pixel_square;

            if(!i|| sobel_pixel_square > max_ext_energy)
                  max_ext_energy = sobel_pixel_square;

      }

      for(i=0;i<Algo_Opt.alter_contour_iteration;i++)
      {
            contour_head_xpos = contour_head->x_pos;
            contour_head_ypos = contour_head->y_pos;

            /*To calculate the average distance between the contour points*/
            CalculateAverageContourDistance(contour_head,&avrg_contour_distance);

            /*To claculate the centroid of the countour points*/
            CalculateContourCentroid(contour_head,&centroid_xpos,&centroid_ypos);
            ret = convert_height_width2rgbindex(&index,ROW,COL,centroid_xpos,centroid_ypos);
            if(ret != 0 || 0 > index )
                  continue;

            red_avg = rgb_image[index];
            green_avg = rgb_image[index+1];
            blue_avg = rgb_image[index+2];

            /*To calculate the internal energies and
            external energy at each contour point around 7x7 pixel window*/
            curr_cont = next_cont = contour_head;
            while(next_cont)
            {
```

```
curr_cont = next_cont;
next_cont = curr_cont->next;

cur_xpos = curr_cont->x_pos;
cur_ypos = curr_cont->y_pos;

/*skip if the point if fixed by the user*/
 if(curr_cont->pos_state == eFixed)
       continue;

if(next_cont)
{
      next_xpos = next_cont->x_pos;
      next_ypos = next_cont->y_pos;
}
else /*Assume the last contour point connects first contour point*/
{
      next_xpos = contour_head_xpos;
      next_ypos = contour_head_ypos;
}

count = 0;
for(r1=-(ALTER_CLICK_WINDOW/2);r1<=(ALTER_CLICK_WINDOW/2);r1++)
{
      for(c1=-(ALTER_CLICK_WINDOW/2);c1<=(ALTER_CLICK_WINDOW/2);c1++)
      {
            /*Internal Energy 1
            (Square of distance between contour points)*/
            distance_square = SQR(next_xpos-(cur_xpos+c1))+\
                  SQR(next_ypos-(cur_ypos+r1));
            internal_energy_1[count] = distance_square;

            if(!count|| distance_square < min_energy1)
                  min_energy1 = distance_square;

            if(!count|| distance_square > max_energy1)
                  max_energy1 = distance_square;

            /*Internal Energy 2
            (Square of difference between average contour distance
            and the distance between current contour points )*/
            distance_diff_square = SQR(avrg_contour_distance -
sqrt(distance_square));
            internal_energy_2[count] = distance_diff_square;

            if(!count || distance_diff_square < min_energy2)
                  min_energy2 = distance_diff_square;

            if(!count|| distance_diff_square > max_energy2)
                  max_energy2 = distance_diff_square;

            /*External Energy
            (Square of the image gradient magnitude(Sobel convoluted
image))*/
            ret =
convert_height_width2index(&index,ROW,COL,cur_xpos+c1,cur_ypos+r1);
            if(ret != 0 || 0 > index )
                  continue;
```

```
                              sobel_pixel_square = SQR(sobel_image[index]);
                              external_energy[count] = sobel_pixel_square;

                              /*External Energy 2
                              (Variance of the rgb image intensity from the rgb image intensity
of the centroid)*/
                              ret =
convert_height_width2rgbindex(&index,ROW,COL,cur_xpos+c1,cur_ypos+r1);
                              if(ret != 0 || 0 > index )
                                    continue;

                              variance = SQR(abs(red_avg-rgb_image[index])+\
                                    abs(green_avg-rgb_image[index+1])+abs(blue_avg-
rgb_image[index+2]));
                              external_energy_2[count] = variance;
                              if(!count || variance < min_ext_eng_2)
                                    min_ext_eng_2 = variance;

                              if(!count|| variance > max_ext_eng_2)
                                    max_ext_eng_2 = variance;

                              count ++;
                        }
                  }

                  count = 0;min_total_energy = 0;
                  ext_eng =0;ext_energy_2 = 0;int_eng1 = 0;int_eng2 = 0;total_eng = 0;

                  /*Normalise the individual energies and find total energy*/
                  energy1_range       = max_energy1 - min_energy1;
                  energy2_range       = max_energy2 - min_energy2;
                  ext_energy_range  = max_ext_energy - min_ext_energy;
                  ext_energy_2_range = max_ext_eng_2 - min_ext_eng_2;

                  for(j=0;j<(SQR(ALTER_CLICK_WINDOW));j++)
                  {
                        int_eng1 = ((internal_energy_1[j]-min_energy1)/energy1_range)*AlgoOpt-
>alter_IE1_W;
                        int_eng2 = ((internal_energy_2[j]-min_energy2)/energy2_range)*AlgoOpt-
>alter_IE2_W;
                        ext_eng = ((external_energy[j]-
min_ext_energy)/ext_energy_range)*AlgoOpt->alter_EE1_W;
                        ext_energy_2 = ((external_energy_2[j]-
min_ext_eng_2)/ext_energy_2_range)*AlgoOpt->alter_EE2_W;

                        total_eng = int_eng1+int_eng2-ext_eng+ext_energy_2;
                        total_energy[j] = total_eng;
                        if(!j || total_eng < min_total_energy)
                        {
                              min_total_energy = total_eng;
                              count = j;/*location of lowest total energy in window*/
                        }
                  }
```

```
                  /*udpate the new contour position to the lowest total energy location in 7x7
window*/
                  ret =
convert_index2height_width(count,ALTER_CLICK_WINDOW,ALTER_CLICK_WINDOW,&next_xpos,&next_ypos);
                  if(ret != 0 || 0 > next_xpos || 0 > next_ypos )
                        continue;

                  curr_cont->last_x_pos = curr_cont->x_pos;
                  curr_cont->last_y_pos = curr_cont->y_pos;

                  if(next_xpos>(ALTER_CLICK_WINDOW/2))
                        curr_cont->x_pos = curr_cont->x_pos + (next_xpos-
(ALTER_CLICK_WINDOW/2));
                  else if(next_xpos<(ALTER_CLICK_WINDOW/2))
                        curr_cont->x_pos = curr_cont->x_pos -
(next_xpos+(ALTER_CLICK_WINDOW/2));

                  if(next_ypos>(ALTER_CLICK_WINDOW/2))
                        curr_cont->y_pos = curr_cont->y_pos + (next_ypos-
(ALTER_CLICK_WINDOW/2));
                  else if(next_ypos<(ALTER_CLICK_WINDOW/2))
                        curr_cont->y_pos = curr_cont->y_pos + (next_ypos-
(ALTER_CLICK_WINDOW/2));

                  ret = convert_height_width2index(&curr_cont->index,ROW,COL,curr_cont-
>x_pos,curr_cont->y_pos);
                  if(ret != 0 || 0 > curr_cont->index )
                        continue;

}
```