CS101_GROUP6_ProjectCodeDocumentation

Generated by Doxygen 1.8.6

Sun Apr 19 2015 01:44:49

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 boundry Struct Reference

This structure stores all the information of a detected blob.

Public Member Functions

- void set_center ()

  Sets the center of the calculated boundry of the blob.
- void set_distance_from_center ()

  Intializes the distance_from_center array and calculates the average distance.
- float calculate_standard_deviation ()

  Calcualtes standard deviation of the distances of points on the boundary from the centre.
- void check_whether_circle ()

  Checks whether the boundary is an approximate circle.

Public Attributes

- bool whether_object

  Set to 0 if the blob cannot be a circle.
- pixel boundry [5000]

  Array which stores the pixels of the boundary of the setected blob.
- int total_boundry_pixels

  Stores the number of boundary pixels of the detected blob.
- pixel center

  Stores the center pixel of the detected blob.
- float distance_from_center [5000]

  Array which stores the distance of each point on the boundary from the centre of the blob.
- float average_distance_from_center

  Stores the average distance of the boundary points from the centre of the blob.
- float standard_deviation

  Stores the standard deviation of the distance of each point on the boundary from the centre of the blob.

3.1.1 Detailed Description

This structure stores all the information of a detected blob.

3.1.2 Member Function Documentation

3.1.2.1 float boundry::calculate_standard_deviation ( ) [inline]

Calcualtes standard deviation of the distances of points on the boundary from the centre.

```
156     {
157         standard_deviation = 0;
158         for(int i = 0; i < total_boundry_pixels; ++i)
159         {
160             standard_deviation += pow((
    average_distance_from_center - distance_from_center[i]),2);
161         }
162         standard_deviation /= total_boundry_pixels;
163         standard_deviation = sqrt(standard_deviation);
164     }
```

3.1.2.2 void boundry::check_whether_circle ( ) [inline]

Checks whether the boundary is an approximate circle.

```
168     {
169         set_center();
170         set_distance_from_center();
171 
172         if(average_distance_from_center <
    threshold_average_distance_from_center)
173         {
174             whether_object = false;
175             return;
176         }
177         else
178             whether_object = true;
179         calculate_standard_deviation();
180         if(standard_deviation < threshold_standard_deviation)
181             whether_object = true;
182         else
183             whether_object = false;
184 
185         if (whether_object == true)
186             total_circles_detected++;
187     }
```

3.1.2.3 void boundry::set_center ( ) [inline]

Sets the center of the calculated boundry of the blob.

```
130     {
131         center.set(0,0);
132         for(int i=0; i<total_boundry_pixels; ++i)
133         {
134             center.x += boundry[i].x;
135             center.y += boundry[i].y;
136         }
137         center.x /= total_boundry_pixels;
138         center.y /= total_boundry_pixels;
139     }
```

3.1.2.4 void boundry::set_distance_from_center ( ) [inline]

Intializes the distance_from_center array and calculates the average distance.

```
143     {
144         average_distance_from_center = 0;
145         for(int i=0; i<total_boundry_pixels; ++i)
146         {
147             distance_from_center[i] = distance_pixels(
    center, boundry[i]);
```

```
148             average_distance_from_center +=
      distance_from_center[i];
149           }
150           average_distance_from_center /=
      total_boundry_pixels;
151       }
```

3.1.3 Member Data Documentation

3.1.3.1 float boundry::average_distance_from_center

Stores the average distance of the boundary points from the centre of the blob.

3.1.3.2 pixel boundry::boundry[5000]

Array which stores the pixels of the boundary of the setected blob.

3.1.3.3 pixel boundry::center

Stores the center pixel of the detected blob.

3.1.3.4 float boundry::distance_from_center[5000]

Array which stores the distance of each point on the boundary from the centre of the blob.

3.1.3.5 float boundry::standard_deviation

Stores the standard deviation of the distance of each point on the boundary from the centre of the blob.

3.1.3.6 int boundry::total_boundry_pixels

Stores the number of boundary pixels of the detected blob.

3.1.3.7 bool boundry::whether_object

Set to 0 if the blob cannot be a circle.

The documentation for this struct was generated from the following file:

- image_processing_and_serial_communcation/main.cpp

3.2 pixel Struct Reference

Stores the coordinate of a pixel.

Public Member Functions

- void set (int r, int c)

  Function to initialize the variables of the struct "pixel".

Public Attributes

- int x
- int y

3.2.1 Detailed Description

Stores the coordinate of a pixel.

3.2.2 Member Function Documentation

3.2.2.1 void pixel::set (int *r*, int *c*) [inline]

Function to initialize the variables of the struct "pixel".

```
92      {
93          x = r;
94          y = c;
95      }
```

3.2.3 Member Data Documentation

3.2.3.1 int pixel::x

3.2.3.2 int pixel::y

The documentation for this struct was generated from the following file:

- image_processing_and_serial_communcation/main.cpp

Chapter 4

File Documentation

4.1 FireBird_V_Final.c File Reference

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <math.h>
```

Functions

- void uart0_init (void)

  Function to initialize UART0.
- void motion_pin_config (void)

  Function to confifure the motion pins.
- void adc_pin_config (void)

  Function to confifure the adc pins.
- void left_encoder_pin_config (void)

  Function to configure INT4 (PORTE 4) pin as input for the left position encoder.
- void right_encoder_pin_config (void)

  Function to configure INT5 (PORTE 5) pin as input for the right position encoder.
- void adc_init ()

  Function to Initialize ADC.
- void port_init ()

  Function to Initialize PORTS.
- void left_position_encoder_interrupt_init (void)

  Interrupt 4 enable.
- void right_position_encoder_interrupt_init (void)

  Interrupt 5 enable.
- ISR (INT5_vect)

  ISR for right position encoder.
- ISR (INT4_vect)

  ISR for left position encoder.
- void motion_set (unsigned char Direction)
- void angle_rotate (unsigned int Degrees)
- void linear_distance_with_update_of_coordinates (unsigned int DistanceInMM)
- void linear_distance_without_update_of_coordinates (unsigned int DistanceInMM)
- void turn_left_with_update_of_coordinates (void)

Turns left through 90 degrees as well as updates the global orientation.

- void turn_right_with_update_of_coordinates (void)

  Turns right through 90 degrees as well as updates the global orientation.
- void turn_right1 (void)

  Turns right through 88 degrees as well as updates the global orientation.
- void turn_left1 (void)

  Turns left through 95 degrees as well as updates the global orientation.
- unsigned char ADC_Conversion (unsigned char Ch)
- unsigned int Sharp_GP2D12_estimation (unsigned char adc_reading)
- void retrace_back ()
- void move_towards_ball ()
- SIGNAL (SIG_USART0_RECV)
- void send_data (unsigned char send)
- void rotate_360_scanning_for_ball (int angle_rotated_in_each_turn)
- int check_obstacles (unsigned int check_value)
- void restore_orientation_after_overcoming_obstacle (int initial)
- void overcome_obstacle (void)
- void move_forward_by_specific_distance_avoiding_obstacles (unsigned int distance)
- void update_coordinates (int angle)
- void rotate_to_centre_of_the_ball_in_the_image (int angle)
- void init_devices (void)

  This function initializes all ports, adc, UART ports and the left and right position encoders.
- int main ()

  Main function.

Variables

- int global_orientation_of_bot = 0
- int flag_goes_into_interrupt = 0

  Flag goes high when code goes into interrupt.
- int distance_of_the_centre_of_ball_from_centre_of_image

  Stores the distance of the centre of the ball from the centre of the image.
- int net_angle_rotated_after_detecting_ball =0

  Keeps storage of the net angle rotated by the bot,with respect to y-axis, after detecting the ball.
- unsigned char data_received_by_bot_from_laptop =0x00

  Stores the data received by the bot from the laptop.
- int turn_from_global_orientation_at_which_ball_was_detected =0

  Keeps count of the turn,while rotating, from global orientation at which ball was detected.
- int y_coordinate_of_bot = 0

  Stores the displacement of the bot ONLY in the Y direction.
- int x_coordinate_of_bot = 0

  Stores the displacement of the bot ONLY in the X direction.
- int y_coordinate_of_bot_at_which_ball_was_detected = 0

  Stores the Y co-ordinate of the bot when it sees the ball for the first ball.
- int x_coordinate_of_bot_at_which_ball_was_detected = 0

  Stores the X co-ordinate of the bot when it sees the ball for the first ball.
- unsigned long int ShaftCountLeft = 0

  To keep track of left position encoder.
- unsigned long int ShaftCountRight = 0

  To keep track of right position encoder.

4.1.1 Function Documentation

4.1.1.1 unsigned char ADC_Conversion (unsigned char *Ch*)

Precondition : "Ch" is the channel number

This Function accepts the Channel Number and returns the corresponding Analog Value

```
322 {
323     unsigned char a;
324     if(Ch>7)
325     {
326         ADCSRB = 0x08;
327     }
328     Ch = Ch & 0x07;
329     ADMUX= 0x20| Ch;
330     ADCSRA = ADCSRA | 0x40;     //Set start conversion bit
331     while((ADCSRA&0x10)==0);    //Wait for ADC conversion to complete
332     a=ADCH;
333     ADCSRA = ADCSRA|0x10; //clear ADIF (ADC Interrupt Flag) by writing 1 to it
334     ADCSRB = 0x00;
335     return a;
336 }
```

4.1.1.2 void adc_init ( )

Function to Initialize ADC.

```
116 {
117     ADCSRA = 0x00;
118     ADCSRB = 0x00;      //MUX5 = 0
119     ADMUX = 0x20;       //Vref=5V external --- ADLAR=1 --- MUX4:0 = 0000
120     ACSR = 0x80;
121     ADCSRA = 0x86;      //ADEN=1 --- ADIE=1 --- ADPS2:0 = 1 1 0
122 }
```

4.1.1.3 void adc_pin_config (void)

Function to confifure the adc pins.

```
93 {
94  DDRF = 0x00;  //set PORTF direction as input
95  PORTF = 0x00; //set PORTF pins floating
96  DDRK = 0x00;  //set PORTK direction as input
97  PORTK = 0x00; //set PORTK pins floating
98 }
```

4.1.1.4 void angle_rotate (unsigned int *Degrees*)

Precondition : "Degrees" specifies the angle in degrees through which the bot is to be rotated

Function to rotate the bot by a specified angle

```
194 {
195  float ReqdShaftCount = 0;
196  unsigned long int ReqdShaftCountInt = 0;
197
198  ReqdShaftCount = (float) Degrees/ 4.5;             // division by resolution to get shaft count
199  ReqdShaftCountInt = (unsigned int) ReqdShaftCount;
200  ShaftCountRight = 0;
201  ShaftCountLeft = 0;
202
203  while (1)
204  {
205   if((ShaftCountRight >= ReqdShaftCountInt) | (ShaftCountLeft >=
     ReqdShaftCountInt))
206   break;
207  }
208  motion_set(0x00); //Stop robot
209 }
```

4.1.1.5 int check_obstacles (unsigned int *check_value*)

Precondition : "check_value" takes the minimum distance of the bot from the obstacle for the obstacle to be detected.

This function returns 1 when the distance of the obstacle from the bot is less than "check_value" and returns 0 otherwise.

```
521 {
522     unsigned int value;
523     unsigned char sharp;
524     sharp = ADC_Conversion(11);
525     value = Sharp_GP2D12_estimation(sharp);
526     if(value<check_value)
527      {  motion_set(0x00);
528         _delay_ms(1000);
529         return 1;
530      }
531     return 0;
532 }
```

4.1.1.6 void init_devices (void)

This function initializes all ports, adc, UART ports and the left and right position encoders.

```
649 {
650  cli();
651  port_init();
652  adc_init();
653  uart0_init();
654  left_position_encoder_interrupt_init();
655  right_position_encoder_interrupt_init();
656  sei();
657 }
```

4.1.1.7 ISR (INT5_vect)

ISR for right position encoder.

```
153 {
154  ShaftCountRight++;  //increment right shaft position count
155 }
```

4.1.1.8 ISR (INT4_vect)

ISR for left position encoder.

```
160 {
161  ShaftCountLeft++;  //increment left shaft position count
162 }
```

4.1.1.9 void left_encoder_pin_config (void)

Function to configure INT4 (PORTE 4) pin as input for the left position encoder.

```
102 {
103  DDRE  = DDRE & 0xEF;  //Set the direction of the PORTE 4 pin as input
104  PORTE = PORTE | 0x10; //Enable internal pull-up for PORTE 4 pin
105 }
```

4.1.1.10 void left_position_encoder_interrupt_init (void)

Interrupt 4 enable.

```
135 {
136  cli();                  //Clears the global interrupt
137  EICRB = EICRB | 0x02;  // INT4 is set to trigger with falling edge
138  EIMSK = EIMSK | 0x10;  // Enable Interrupt INT4 for left position encoder
139  sei();                  // Enables the global interrupt
140 }
```

4.1.1.11 void linear_distance_with_update_of_coordinates (unsigned int *DistanceInMM*)

Precondition : "DistanceInMM" gives the distance to be travelled by the bot in mm

Function used for moving robot forward by specified distance and updating the coordinates of the bot in the arena

```
215 {
216     motion_set(0x06);
217     float ReqdShaftCount = 0;
218     unsigned long int ReqdShaftCountInt = 0;
219
220     ReqdShaftCount = DistanceInMM / 5.338; // division by resolution to get shaft count
221     ReqdShaftCountInt = (unsigned long int) ReqdShaftCount;
222
223     ShaftCountRight = 0;
224     while(1)
225     {
226         if(ShaftCountRight > ReqdShaftCountInt)
227         {
228             break;
229         }
230     }
231
232     /*
233     Updates the value of "y_coordinate_of_bot" and "x_coordinate_of_bot" according to the orientation.

234     ie updates the displacement of the bot  */
235
236     int mod = global_orientation_of_bot % 4;
237     switch(mod)
238     {
239         case 0 :    y_coordinate_of_bot = y_coordinate_of_bot +
      DistanceInMM; //in this case the bot is facing forward
240                     break;
241         case 1 :    x_coordinate_of_bot = x_coordinate_of_bot +
      DistanceInMM; //in this case bot is facing rightward
242                     break;
243         case 2 :    y_coordinate_of_bot = y_coordinate_of_bot -
      DistanceInMM; //in this case bot is facing backward
244                     break;
245         case 3 :    x_coordinate_of_bot = x_coordinate_of_bot -
      DistanceInMM; //in this case bot is facing leftward
246                     break;
247     }
248
249     motion_set(0x00);
250     _delay_ms(10);
251
252 }
```

4.1.1.12 void linear_distance_without_update_of_coordinates (unsigned int *DistanceInMM*)

Precondition : "DistanceInMM" gives the distance to be travelled by the bot in mm

Function used for moving robot forward by specified distance without uppdating the coordinates of the bot in the arena

```
259 {
260     motion_set(0x06);
261     float ReqdShaftCount = 0;
262     unsigned long int ReqdShaftCountInt = 0;
263
264     ReqdShaftCount = DistanceInMM / 5.338; // division by resolution to get shaft count
```

```
265     ReqdShaftCountInt = (unsigned long int) ReqdShaftCount;
266
267     ShaftCountRight = 0;
268     while(1)
269     {
270         if(ShaftCountRight > ReqdShaftCountInt)
271         {
272             break;
273         }
274     }
275
276     motion_set(0x00);
277     _delay_ms(10);
278
279 }
```

4.1.1.13 int main ( )

Main function.

```
661 {
662   init_devices();
663
664   unsigned int value;
665   unsigned char sharp;
666
667   unsigned int angle_rotated_in_each_turn = 90;
668   unsigned int arena_breadth= 930 , arena_length= 930 ;
669   unsigned int distance_moved_forward_in_one_go = arena_breadth/4, distance_moved_sideways_in_one_go =
      arena_length/3;
670
671   unsigned int left_or_right = 0;
672
673   while(1)
674   {
675      while((y_coordinate_of_bot + distance_moved_forward_in_one_go <= arena_breadth &&
      global_orientation_of_bot ==0) || (y_coordinate_of_bot >=
      distance_moved_forward_in_one_go && global_orientation_of_bot ==2 ))
676       {
677           rotate_360_scanning_for_ball(angle_rotated_in_each_turn);
678           move_forward_by_specific_distance_avoiding_obstacles
      (distance_moved_forward_in_one_go);
679           motion_set(0x00);
680           _delay_ms(1000);
681       }
682
683       if( (x_coordinate_of_bot + distance_moved_sideways_in_one_go) >= arena_length )
684       {
685           motion_set(0x00);
686           while(1);
687       }
688
689       if(left_or_right == 0)
690       {
691           turn_right1();
692           _delay_ms(1000);
693           move_forward_by_specific_distance_avoiding_obstacles
      (distance_moved_sideways_in_one_go);
694           motion_set(0x00);
695           _delay_ms(1000);
696           turn_right_with_update_of_coordinates();
697           _delay_ms(1000);
698           left_or_right = 1;
699
700       }
701       else
702       {
703           turn_left_with_update_of_coordinates();
704           _delay_ms(1000);
705           move_forward_by_specific_distance_avoiding_obstacles
      (distance_moved_sideways_in_one_go);
706           motion_set(0x00);
707           _delay_ms(1000);
708           turn_left_with_update_of_coordinates();
709           _delay_ms(1000);
710           left_or_right = 0;
711       }
712
713
714   }
715 }
```

4.1.1.14 void motion_pin_config (void)

Function to confifure the motion pins.

```
84 {
85  DDRA = DDRA | 0x0F;   //set direction of the PORTA 3 to PORTA 0 pins as output
86  PORTA = PORTA & 0xF0; // set initial value of the PORTA 3 to PORTA 0 pins to logic 0
87  DDRL = DDRL | 0x18;   //Setting PL3 and PL4 pins as output for PWM generation
88  PORTL = PORTL | 0x18; //PL3 and PL4 pins are for velocity control using PWM
89 }
```

4.1.1.15 void motion_set (unsigned char *Direction*)

Precondition: "Direction" takes the following values for corresponding motion

```
        0x06 - Forward

        0x09 - Backward

        0x05 - Left

        0x0A - Right

        0x00 - Stop
```

Function used for setting motor's direction

```
179 {
180  unsigned char PortARestore = 0;
181
182  Direction &= 0x0F;             // removing upper nibbel as it is not needed
183  PortARestore = PORTA;          // reading the PORTA's original status
184  PortARestore &= 0xF0;          // setting lower direction nibbel to 0
185  PortARestore |= Direction;     // adding lower nibbel for direction command and restoring the PORTA status
186  PORTA = PortARestore;          // setting the command to the port
187 }
```

4.1.1.16 void move_forward_by_specific_distance_avoiding_obstacles (unsigned int *distance*)

Precondition : "distance" specifies the total distance to be moved

This function moves the bot forward by a specific distance, avoiding all obstacles.

```
599 {
600     int number_of_iters = distance/60;
601     for(int i  = 0; i < number_of_iters; i++)
602     {
603         if(check_obstacles(200))
604         {
605             overcome_obstacle();
606
607         }
608
609         linear_distance_with_update_of_coordinates(60);
610     }
611 }
```

4.1.1.17 void move_towards_ball ( )

Precondition : This function is called when the object is detected

The function makes the bot move towards the detected object

```
394 { double proportionality_factor=45.0/64.0;
395   int reqd_angle;
396   unsigned int value;
```

```
397   unsigned char sharp;
398     sharp = ADC_Conversion(11);
399     value = Sharp_GP2D12_estimation(sharp);
400   while(value>=200)
401   {
402
403   flag_goes_into_interrupt=0;
404   reqd_angle = distance_of_the_centre_of_ball_from_centre_of_image
       * proportionality_factor;
405   rotate_to_centre_of_the_ball_in_the_image(reqd_angle);
406   linear_distance_without_update_of_coordinates(200);
407   update_coordinates(reqd_angle);
408
409   send_data(0x00);
410   while(!flag_goes_into_interrupt);
411   if(data_received_by_bot_from_laptop==0x00)
412     { retrace_back();
413       y_coordinate_of_bot=
      y_coordinate_of_bot_at_which_ball_was_detected;
414       x_coordinate_of_bot=
      y_coordinate_of_bot_at_which_ball_was_detected;
415       return;
416      }
417
418     sharp = ADC_Conversion(11);
419     value = Sharp_GP2D12_estimation(sharp);
420
421   }
422   motion_set(0x00);
423   while(1);
424 }
```

4.1.1.18 void overcome_obstacle (void)

Precondition : This function is called when an obstacle is detected.

This function overcomes the obstacles in the way of the bot and then continues its normal motion

```
555 {
556     int init_orientatn = global_orientation_of_bot;
557     _delay_ms(200);
558     rotate_360_scanning_for_ball(90);
559
560     do
561     {
562
563         while(check_obstacles(200)==1)
564         {
565             turn_right_with_update_of_coordinates();
566             motion_set(0x00);
567             _delay_ms(500);
568         }
569
570         linear_distance_with_update_of_coordinates(200);
571         motion_set(0x00);
572         _delay_ms(700);
573
574         turn_left_with_update_of_coordinates();
575         motion_set(0x00);
576         _delay_ms(500);
577     }
578     while (check_obstacles(200)==1);
579
580     _delay_ms(500);
581     turn_right_with_update_of_coordinates();
582     _delay_ms(500);
583     linear_distance_with_update_of_coordinates(50);
584     turn_left_with_update_of_coordinates();
585     _delay_ms(1000);
586
587     int k = (global_orientation_of_bot - init_orientatn)%4;
588     restore_orientation_after_overcoming_obstacle(
    init_orientatn);
589
590     if(k>=2)
591      rotate_360_scanning_for_ball(90);
592
593 }
```

4.1.1.19 void port_init ( )

Function to Initialize PORTS.

```
126 {
127  motion_pin_config();
128  adc_pin_config();
129  left_encoder_pin_config();  //left encoder pin configuration
130  right_encoder_pin_config(); //right encoder pin configuration
131 }
```

4.1.1.20 void restore_orientation_after_overcoming_obstacle (int *initial*)

Precondition : "initial" takes the value of the initial orientation of the bot before the obstacle was detected.

This function restores the orientation of the bot after the obstacle is overcome.

```
539 {
540     int curr_orientatn = global_orientation_of_bot;
541     for(int i = 0; i < (curr_orientatn - initial) % 4; i++)
542     {
543         linear_distance_with_update_of_coordinates(300);
544         _delay_ms(500);
545         turn_left_with_update_of_coordinates();
546         _delay_ms(500);
547     }
548 }
```

4.1.1.21 void retrace_back ( )

Precondition : This function is called when the bot is moving towards the detected object

```
          but it turns out that it is not the required object due to ineffieciency in image processing
```

This function takes the bot back to its initial position from which the object was first detected.

```
363 {
364     if(net_angle_rotated_after_detecting_ball >= 0)
365     motion_set(0x05);
366
367     else
368     {
369         motion_set(0x0A);
370         net_angle_rotated_after_detecting_ball =
      net_angle_rotated_after_detecting_ball*(-1);
371     }
372
373     angle_rotate(net_angle_rotated_after_detecting_ball);
374     _delay_ms(2000);
375     motion_set(0x05);
376
377     double y = (y_coordinate_of_bot -
      y_coordinate_of_bot_at_which_ball_was_detected);
378     double x = (x_coordinate_of_bot -
      y_coordinate_of_bot_at_which_ball_was_detected);
379     double z = atan2( y , x );
380     double degrees = ((z / 3.142 + 0.5)*180);
381     angle_rotate( degrees );
382
383     linear_distance_without_update_of_coordinates( sqrt((x*x)+
      (y*y)) );
384     motion_set(0x0A);
385     angle_rotate(degrees);
386
387 }
```

4.1.1.22 void right_encoder_pin_config (void)

Function to configure INT5 (PORTE 5) pin as input for the right position encoder.

```
109 {
110  DDRE  = DDRE & 0xDF;  //Set the direction of the PORTE 4 pin as input
111  PORTE = PORTE | 0x20; //Enable internal pull-up for PORTE 4 pin
112 }
```

4.1.1.23 void right_position_encoder_interrupt_init (void)

Interrupt 5 enable.

```
144 {
145  cli();                 //Clears the global interrupt
146  EICRB = EICRB | 0x08;  // INT5 is set to trigger with falling edge
147  EIMSK = EIMSK | 0x20;  // Enable Interrupt INT5 for right position encoder
148  sei();                 // Enables the global interrupt
149 }
```

4.1.1.24 void rotate_360_scanning_for_ball (int *angle_rotated_in_each_turn*)

Precondition : "angle_rotated_in_each_turn" gives the angle through which the bot rotates in each turn

This function scans 360 degrees looking for the specified object

```
482  {  turn_from_global_orientation_at_which_ball_was_detected
      =0;
483    int first_bit;
484     for (int i = 0 ; i < 360 / angle_rotated_in_each_turn ; i++)
485         {
486          flag_goes_into_interrupt=0;
487          motion_set(0x0A);
488          angle_rotate(angle_rotated_in_each_turn);
489          motion_set(0x00);
490          send_data(0x00);
491
492          while(!flag_goes_into_interrupt); //Will wait until
       data_received_by_bot_from_laptop is received
493
494          //Goes into interrupt
495          flag_goes_into_interrupt = 0;
496          first_bit = (data_received_by_bot_from_laptop & 0x80)/128;
497          if(first_bit == 1)
498          {
499                net_angle_rotated_after_detecting_ball = (((
      global_orientation_of_bot % 4)*90 + (
      turn_from_global_orientation_at_which_ball_was_detected
      *angle_rotated_in_each_turn)) % 360 );
500                y_coordinate_of_bot_at_which_ball_was_detected
       = y_coordinate_of_bot;
501                y_coordinate_of_bot_at_which_ball_was_detected
       = x_coordinate_of_bot;
502                move_towards_ball();
503                turn_from_global_orientation_at_which_ball_was_detected
      ++;
504                continue;
505           }
506          while(data_received_by_bot_from_laptop != 0x00);
507
508          _delay_ms(500);
509          turn_from_global_orientation_at_which_ball_was_detected
      ++;
510
511         }
512  }
```

4.1.1.25 void rotate_to_centre_of_the_ball_in_the_image (int *angle*)

Precondition : "angle" specifies the angle rotated by the bot towards the centre of the detected ball

This function rotates the bot such that it faces the centre of the detected ball in the image

```
632 {
633
```

```
634     if(angle >= 0)
635     motion_set(0x0A);
636
637     else
638     {
639     motion_set(0x05);
640     angle = angle*(-1);
641     }
642
643     angle_rotate(angle);
644     motion_set(0x00);
645 }
```

4.1.1.26 void send_data (unsigned char *send*)

Precondition : "send" is the data to be sent to the laptop

Sends 0x00 which tells the laptop to click an image and process it

```
471 {
472     while(data_received_by_bot_from_laptop != 0x03)
473     {
474         UDR0 = send;
475     }
476 }
```

4.1.1.27 unsigned int Sharp_GP2D12_estimation (unsigned char *adc_reading*)

Precondition : "adc_reading" is the analog value of the sharp sensor

This Function calculates the actual distance in millimeters(mm) from the input analog value of Sharp Sensor.

```
343 {
344     float distance;
345     unsigned int distanceInt;
346     distance = (int)(10.00*(2799.6*(1.00/(pow(adc_reading,1.1546)))));
347     distanceInt = (int)distance;
348     if(distanceInt>800)
349     {
350         distanceInt=800;
351     }
352     return distanceInt;
353 }
```

4.1.1.28 SIGNAL (SIG_USART0_RECV)

ISR which is trigerred when data is returned to the received buffer

The data received from the laptop is distinguised as follows:

```
0x03 - Confirmation that the sent byte has been received by the laptop

0x80 - To relay to the laptop that the sent byte has been received by the bot

First bit is 0 - object not found

First bit is 1 - object found

If first bit is 1 ,

 second bit is 0 - Centre of ball is towards the right of the centre of the image

 second bit is 1 - Centre of ball is towards the left of the centre of the image
```

```
444 {
445   distance_of_the_centre_of_ball_from_centre_of_image =
      0;
446   data_received_by_bot_from_laptop=UDR0;
```

```
447   if(data_received_by_bot_from_laptop!=0x03)
448    { flag_goes_into_interrupt=1;
449      _delay_ms(10);
450      UDR0 = 0x80;
451      _delay_ms(10);
452    }
453   int first_bit = (data_received_by_bot_from_laptop & 0x80)/128; //Check
       first bit to see if ball was found
454
455   if(first_bit == 1)
456   {
457     int second_bit = (data_received_by_bot_from_laptop & 0x40)/64;
458     distance_of_the_centre_of_ball_from_centre_of_image
      = data_received_by_bot_from_laptop & 0x3F;
459     if(second_bit == 1)
460     {
461         distance_of_the_centre_of_ball_from_centre_of_image
       *= (-1);
462     }
463   }
464
465 }
```

4.1.1.29 void turn_left1 (void)

Turns left through 95 degrees as well as updates the global orientation.

```
311 {
312     motion_set(0x05);
313     angle_rotate(95);
314     global_orientation_of_bot--;
315     motion_set(0x00);
316 }
```

4.1.1.30 void turn_left_with_update_of_coordinates (void)

Turns left through 90 degrees as well as updates the global orientation.

```
284 {
285     motion_set(0x05);
286     angle_rotate(90);
287     global_orientation_of_bot--;
288     motion_set(0x00);
289 }
```

4.1.1.31 void turn_right1 (void)

Turns right through 88 degrees as well as updates the global orientation.

```
302 {
303     motion_set(0x0A);
304     angle_rotate(88);
305     global_orientation_of_bot++;
306     motion_set(0x00);
307 }
```

4.1.1.32 void turn_right_with_update_of_coordinates (void)

Turns right through 90 degrees as well as updates the global orientation.

```
293 {
294     motion_set(0x0A);
295     angle_rotate(90);
296     global_orientation_of_bot++;
297     motion_set(0x00);
298 }
```

4.1.1.33 void uart0_init (void)

Function to initialize UART0.

```
73 {
74  UCSR0B = 0x00; //disable while setting baud rate
75  UCSR0A = 0x00;
76  UCSR0C = 0x06;
77  UBRR0L = 0x5F; //set baud rate lo
78  UBRR0H = 0x00; //set baud rate hi
79  UCSR0B = 0x98;
80 }
```

4.1.1.34 void update_coordinates (int *angle*)

Precondition : "angle" specifies the angle rotated by the bot towards the centre of the detected ball

This function updates the coordinates of the bot during its motion towards the ball

```
618 {
619
620     net_angle_rotated_after_detecting_ball =
      net_angle_rotated_after_detecting_ball + angle;  //Right side
       positive
621     y_coordinate_of_bot = y_coordinate_of_bot + 200*cos(
      net_angle_rotated_after_detecting_ball*3.14/180);
622     x_coordinate_of_bot = x_coordinate_of_bot + 200*sin(
      net_angle_rotated_after_detecting_ball*3.14/180);
623
624
625 }
```

4.1.2 Variable Documentation

4.1.2.1 unsigned char data_received_by_bot_from_laptop =0x00

Stores the data received by the bot from the laptop.

4.1.2.2 int distance_of_the_centre_of_ball_from_centre_of_image

Stores the distance of the centre of the ball from the centre of the image.

4.1.2.3 int flag_goes_into_interrupt = 0

Flag goes high when code goes into interrupt.

4.1.2.4 int global_orientation_of_bot = 0

Written by : Group 6, CS101 Embedded Systems Project

AVR Studio Version 4.17, Build 666

Date : 18-04-15

This code demonstrates the motion of the bot inside the arena looking for the object. The bot will capture images with the help of a webcam at various instants from different positions and sent it to the laptop for image processing. Once the ball is detected the bot will move towards the ball and stop at a certain distance from it.

Note:

1. Make sure that in the configuration options following settings are done for proper operation of the code Microcontroller: atmega2560 Frequency: 14745600 Optimization: -O0 Global orientatn specifies the orientatn of the bot in the arena.

i.e. whether the bot is facing forward or x_coordinate_of_bot or left or backward.

4.1.2.5 int net_angle_rotated_after_detecting_ball =0

Keeps storage of the net angle rotated by the bot,with respect to y-axis, after detecting the ball.

4.1.2.6 unsigned long int ShaftCountLeft = 0

To keep track of left position encoder.

4.1.2.7 unsigned long int ShaftCountRight = 0

To keep track of right position encoder.

4.1.2.8 int turn_from_global_orientation_at_which_ball_was_detected =0

Keeps count of the turn,while rotating, from global orientation at which ball was detected.

4.1.2.9 int x_coordinate_of_bot = 0

Stores the displacement of the bot ONLY in the X direction.

4.1.2.10 int x_coordinate_of_bot_at_which_ball_was_detected = 0

Stores the X co-ordinate of the bot when it sees the ball for the first ball.

4.1.2.11 int y_coordinate_of_bot = 0

Stores the displacement of the bot ONLY in the Y direction.

4.1.2.12 int y_coordinate_of_bot_at_which_ball_was_detected = 0

Stores the Y co-ordinate of the bot when it sees the ball for the first ball.

4.2 image_processing_and_serial_communcation/main.cpp File Reference

```
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/highgui/highgui_c.h>
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <windows.h>
```

Classes

- struct pixel

  Stores the coordinate of a pixel.
- struct boundry

  This structure stores all the information of a detected blob.

Functions

- bool write_data_to_XBEE ()

  This function writes data from the Xbee and returns true if the data is sucessfully written to the port.
- float distance_pixels (pixel p1, pixel p2)
- bool satisfy_ballcolor (Vec3b bgr_value)
- void convert_only_ballcolor (const Mat &original_image, Mat &only_ballcolor)
- void check_for_circles (struct boundry boundry[], int total_boundries)
- void draw_detected_object (Mat &img, struct boundry boundry[], int total_boundries)

  This function draws the detected object onto the matrix "img".
- void detect_all_boundries (Mat &img, const Mat &only_ballcolor)

  This function detects the boundary point of the blobs.
- void store_a_boundary (struct boundry &boundry, Mat &img, int r, int c)

  This function finds boundaries and stores them into the struct "boundry".
- void store_all_boundries (struct boundry boundry[], int &total_boundries, Mat &img)

  This function stores all the boundaries.
- void check_number_of_circles_detected ()
- bool read_data_from_XBEE ()

  This function reads data from the Xbee and returns true if the data is received else returns false.
- bool set_up_serial_connection ()

  This function sets up serial communication between the bot and the laptop and returns true if successful.
- void send_distance_from_center (int center_column_of_the_image)

  This function sends the distance of the centre of the detected object from the centre of the image to the bot.
- int main ()

  Main Function.

Variables

- HANDLE hSerial = CreateFile("COM1", GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0)
- DCB dcb
- uchar ball_color [3][2] = {{'<', 60}, {'<', 60}, {'<', 60}}
- float threshold_standard_deviation = 20

  Set the threshold for checking whether the given boundry is a circle or not.
- float threshold_average_distance_from_center = 25

  Set the threshold for checking whether the circle is big enough.
- int total_circles_detected = 0

  Will store the total number of detected circles finally.
- uchar input_data_from_xbee [2] = {0xFF,0}

  Stores the bytes received from the Xbee on the bot.
- uchar output_data_to_xbee [2] = {0x8E,0}

  Stores the bytes to be written onto the port.
- int x_coordinate_of_bot = 0

  Stores the displacement of the bot in the X direction.
- int y_coordinate_of_bot = 0

  Stores the displacement of the bot in the Y direction.
- bool whether_sending_x_coordinate_over = 0

  True when sending X-coordinate is over and sending the Y-coordinate is being sent.
- int is_the_coordinate_Y = 0

  Ture when the data being sent is the Y-coordinate of the bot.
- bool whether_any_object_detected = 0

True when the object is detected.

- boundry boundry [10000]

  Array of type struct "boundry" which stores all closed boundaries detected in the image.

- int total_boundries = 0

  Stores the number of boundaries detected in the image.

- pixel surrounding [8] = {{0,1},{0,-1},{1,1},{1,-1},{1,0},{-1,-1},{-1,0},{-1,1}}

  Array of type "pixel" which defines the relative locations of the eight adjecent pixels to every pixel in the image.

4.2.1 Function Documentation

4.2.1.1 void check_for_circles (struct boundry *boundry[]*, int *total_boundries*)

Precondition : Takes as argument the structure storing the boundaries of the detected blobs and the total number of boundaries

This function checks whether the detected boundaries are cicles or not

```
257 {
258     total_circles_detected = 0;
259     for(int i=0; i<total_boundries; ++i)
260     {
261         boundry[i].check_whether_circle();
262     }
263 }
```

4.2.1.2 void check_number_of_circles_detected ( )

This function checks if the number of circles detected is exactly one. If the number of detected circles is greater than one, it resets the totoal boundaries to zero

```
365 {
366     if(total_circles_detected!=1)                    //If no or more than 1 circle
      detected
367     {
368         total_boundries = 0;                           //So that no further action is
      taken
369                                                      //For drawing or sending data we run a loop from 0
      to total_boundries
370     }
371 }
```

4.2.1.3 void convert_only_ballcolor (const Mat & *original_image*, Mat & *only_ballcolor*)

Precondition : Takes as arguments a martix of the original image and the matrix storing pixels having the colour specified

This function sets the pixels having the colour specified by the user to black and the other pixels to white in the matrix "only_ballcolor"

```
238 {
239     Vec3b bgr_value;
240     for(int i=0; i<original_image.rows; ++i)
241         for(int j=0; j<original_image.cols; ++j)
242         {
243             bgr_value = original_image.at<Vec3b>(i, j);
244             if(satisfy_ballcolor(bgr_value))
245             {
246                 only_ballcolor.at<uchar>(i, j) = 0 ;
247             }
248         }
249 }
```

4.2.1.4 void detect_all_boundries (Mat & *img*, const Mat & *only_ballcolor*)

This function detects the boundary point of the blobs.

```
283 {
284     int total_black_surrounding_pixels;
285     for(int i = 0; i < img.rows; ++i)
286         for(int j = 0; j < img.cols; ++j)
287         {
288             if(only_ballcolor.at<uchar>(i, j)==0)
289             {
290                 total_black_surrounding_pixels = 0;
291                 for(int k = 0; k < 8; ++k)
292                 {
293                     if(only_ballcolor.at<uchar>(i + surrounding[k].x, j +
    surrounding[k].y)==0)
294                         total_black_surrounding_pixels++;
295                 }
296                 if(total_black_surrounding_pixels < 8)
297                 {
298                     img.at<uchar>(i, j) = 0;
299                 }
300             }
301         }
302 }
```

4.2.1.5 float distance_pixels (pixel *p1*, pixel *p2*)

Precondition : p1 and p2 are variables of "pixel" type

This function returns the distance between the two input pixels p1 and p2

```
103 {
104     float distance_x = pow((p1.x-p2.x), 2);
105     float distance_y = pow((p1.y-p2.y), 2);
106     float distance = sqrt(distance_x + distance_y);
107     return distance;
108 }
```

4.2.1.6 void draw_detected_object (Mat & *img*, struct boundry *boundry[]*, int *total_boundries*)

This function draws the detected object onto the matrix "img".

```
267 {
268     for(int i = 0; i < total_boundries; ++i)
269     {
270         if(boundry[i].whether_object)
271         {
272             for(int j = 0; j < boundry[i].total_boundry_pixels; ++j)
273             {
274                 img.at<uchar>(boundry[i].boundry[j].x, boundry[i].
    boundry[j].y) = 0;
275             }
276         img.at<uchar>(boundry[i].center.x, boundry[i].center.y)=0;
277     }
278     }
279 }
```

4.2.1.7 int main ( )

Main Function.

```
497 {
498
499     VideoCapture camera(0);
500     Mat original_image;
501     if(!camera.isOpened())
502     {
503         cout<<"Can't open Camera!"<<endl;
```

```
504         output_data_to_xbee[0]=0x02;
505         write_data_to_XBEE();
506         return -1;
507     }
508
509     namedWindow("original_image", CV_WINDOW_NORMAL); //Opens a window displaying the original image
      currently visible to the bot
510     namedWindow("only_ballcolor", CV_WINDOW_NORMAL); //Opens a window displaying only those pixels from the
      original image whose color matches the color of the object to be detected
511     namedWindow("only_boundries", CV_WINDOW_NORMAL); //Opens a window displaying only the boundaries of the
      detected blobs
512     namedWindow("detected_object", CV_WINDOW_NORMAL); //Opens a window displaying the detected object
513     if(!set_up_serial_connection())
514     {
515         cout<<"Error in setting up XBEEs"<<endl;
516         return -1;
517     }
518     while(1)
519     {
520         read_data_from_XBEE();
521         if(input_data_from_xbee[0] == 0x00)
522         {
523             if(!camera.read(original_image))
524             {
525                 cout<<"Can't read frames from camera.";
526                 output_data_to_xbee[0]=0x02;
527                 write_data_to_XBEE();
528                 return -1;
529             }
530             camera.read(original_image);
531             imshow("original_image", original_image);
532             GaussianBlur(original_image, original_image, Size(13, 13), 0, 0 );
533             Mat only_ballcolor(original_image.rows, original_image.cols, CV_8UC1, 255);
534             convert_only_ballcolor(original_image, only_ballcolor);
535             imshow("only_ballcolor", only_ballcolor);
536             Mat only_boundries(original_image.rows, original_image.cols, CV_8UC1, 255);
537             detect_all_boundries(only_boundries, only_ballcolor);
538             imshow("only_boundries", only_boundries);
539             Mat detected_object = only_boundries.clone();
540             store_all_boundries(boundry,
      total_boundries, detected_object);
541             check_for_circles(boundry, total_boundries);
542             check_number_of_circles_detected();
543             draw_detected_object(detected_object, boundry,
      total_boundries);
544             imshow("detected_object", detected_object);
545             send_distance_from_center(original_image.cols/2);
546             if(waitKey(1)==27)
547             {
548                 destroyWindow("original_image");
549                 destroyWindow("only_ballcolor");
550                 destroyWindow("only_boundries");
551                 destroyWindow("detected_object");
552                 return 1;
553             }
554         }
555     }
556 }
```

4.2.1.8 bool read_data_from_XBEE ( )

This function reads data from the Xbee and returns true if the data is received else returns false.

```
375 {
376     DWORD dwBytesTransferred = 0;
377     DWORD dwCommModemStatus = 0;
378     bool retVal;
379     if(!GetCommState(hSerial, &dcb))
380     {
381         cout<<"Serial port can't be opened"<<endl;
382         return 0;
383     }
384
385     SetCommMask(hSerial, EV_RXCHAR|EV_ERR);
386     WaitCommEvent(hSerial, &dwCommModemStatus, 0);
387     if (dwCommModemStatus& EV_RXCHAR)
388     {
389         retVal = (ReadFile(hSerial, input_data_from_xbee, 1, &dwBytesTransferred
      , NULL));
390     }
391
392     else
```

```
393     {
394         cout<<"Some error has occured"<<endl;
395         return 0;
396     }
397
398     if(retVal && input_data_from_xbee[0] != 0x80)
399     {
400         output_data_to_xbee[0] = 0x03;
401         write_data_to_XBEE();
402         Sleep(10);
403     }
404     return retVal;
405 }
```

4.2.1.9 bool satisfy_ballcolor (Vec3b *bgr_value*)

Precondition : "bgr_value" specifies the BGR value of the colour of the ball to be detected

This function returns true if an object of the specified colur is found else returns false.

```
205 {
206     bool return_value = 1;
207     for(int i = 0; i<3; ++i) //For B, G, R
208     {
209         switch(ball_color[i][0])
210         {
211             case '~':   if(ball_color[i][1]-10 > bgr_value[i] ||
    ball_color[i][1] + 10 < bgr_value[i])
212                         return_value = 0;
213
214             case '<':   if(bgr_value[i] >= ball_color[i][1])
215                         return_value = 0;
216                         break;
217
218             case '>':   if(bgr_value[i] <= ball_color[i][1])
219                         return_value = 0;
220                         break;
221
222             case '=':   if(bgr_value[i] != ball_color[i][1])
223                         return_value = 0;
224                         break;
225
226             default:    break;
227         }
228     }
229     return return_value;
230 }
```

4.2.1.10 void send_distance_from_center (int *center_column_of_the_image*)

This function sends the distance of the centre of the detected object from the centre of the image to the bot.

```
460 {
461     whether_any_object_detected = 0;
462     int distance_of_the_center_of_the_object_from_the_center_of_the_image;
463     for(int i = 0; i < total_boundries; ++i)
464     {
465         if(boundry[i].whether_object)
466         {
467             whether_any_object_detected = 1;
468             distance_of_the_center_of_the_object_from_the_center_of_the_image =
    boundry[i].center.y-center_column_of_the_image;
469             distance_of_the_center_of_the_object_from_the_center_of_the_image/=5;
470             cout<<"Distance of the ball from the center of the screen is: "
471                 <<distance_of_the_center_of_the_object_from_the_center_of_the_image<<endl;
472             if(distance_of_the_center_of_the_object_from_the_center_of_the_image>=0)
473             {
474                 distance_of_the_center_of_the_object_from_the_center_of_the_image|=0x80;
475                 output_data_to_xbee[0]=
    distance_of_the_center_of_the_object_from_the_center_of_the_image;
476                 write_data_to_XBEE();
477             }
478             else
479             {
480                 distance_of_the_center_of_the_object_from_the_center_of_the_image|=0xC0;
481                 output_data_to_xbee[0] =
    distance_of_the_center_of_the_object_from_the_center_of_the_image;
```

```
482                 write_data_to_XBEE();
483             }
484             break;
485         }
486     }
487     if(!whether_any_object_detected)
488     {
489         output_data_to_xbee[0] = 0x00;
490         write_data_to_XBEE();
491     }
492
493 }
```

4.2.1.11 bool set_up_serial_connection ( )

This function sets up serial communication between the bot and the laptop and returns true if successful.

```
445 {
446     if (!GetCommState(hSerial, &dcb))
447     {
448         cout<<"Serial port can't be opened"<<endl;
449         return false;
450     }
451     dcb.BaudRate = CBR_9600;
452     dcb.ByteSize = 8;
453     dcb.Parity = NOPARITY;
454     dcb.StopBits = ONESTOPBIT;
455     return SetCommState(hSerial, &dcb);
456 }
```

4.2.1.12 void store_a_boundary (struct boundry & *boundry,* Mat & *img,* int *r,* int *c*)

This function finds boundaries and stores them into the struct "boundry".

```
306 {
307     pixel current_frontier[10000], next_frontier[10000];
308     int total_pixels_in_current_frontier = 1, total_pixels_in_next_frontier = 0;
309     img.at<uchar>(r, c) = 255;
310     current_frontier[0].set(r, c);
311     boundry.total_boundry_pixels = 0;
312     while(total_pixels_in_current_frontier != 0)
313     {
314         for(int i = 0; i < total_pixels_in_current_frontier; ++i)
315         {
316             for(int j = 0; j < 8; ++j)
317             {
318                 if(current_frontier[i].x + surrounding[j].x < img.rows
319                     && current_frontier[i].y + surrounding[j].y < img.cols
320                     && current_frontier[i].x + surrounding[j].x >=0
321                     && current_frontier[i].y + surrounding[j].y >= 0)
322                 {
323                     if(img.at<uchar>(current_frontier[i].x + surrounding[j].
    x, current_frontier[i].y + surrounding[j].y) == 0)
324                     {
325                         img.at<uchar>(current_frontier[i].x + surrounding[j].
    x, current_frontier[i].y + surrounding[j].y) = 255;
326                         next_frontier[total_pixels_in_next_frontier].set(current_frontier[i].x +
    surrounding[j].x, current_frontier[i].y + surrounding[j].y);
327                         total_pixels_in_next_frontier++;
328                     }
329                 }
330             }
331         }
332         for(int i = 0; i < total_pixels_in_current_frontier; ++i)
333         {
334             boundry.boundry[boundry.total_boundry_pixels] = current_frontier[i];
335             boundry.total_boundry_pixels++;
336         }
337         total_pixels_in_current_frontier = total_pixels_in_next_frontier;
338         for(int i = 0; i < total_pixels_in_current_frontier; ++i)
339         {
340             current_frontier[i] = next_frontier[i];
341         }
342         total_pixels_in_next_frontier = 0;
343     }
344 }
```

4.2.1.13 void store_all_boundries (struct boundry *boundry[]*, int & *total_boundries*, Mat & *img*)

This function stores all the boundaries.

```
348 {
349     total_boundries = 0;
350     for(int i = 0; i < img.rows; i++)
351         for(int j = 0; j < img.cols; j++)
352         {
353             if(img.at<uchar>(i,j)==0)
354             {
355                 store_a_boundary(boundry[total_boundries], img, i, j);
356                 total_boundries++;
357             }
358         }
359 }
```

4.2.1.14 bool write_data_to_XBEE ( )

This function writes data from the Xbee and returns true if the data is sucessfully written to the port.

```
409 {
410     DWORD byteswritten;
411     DWORD received_byte;
412     uchar whether_received[2] = {0};
413     unsigned long long timer1,timer2;
414     bool has_the_received_byte_been_received = 0;
415
416     if (!GetCommState(hSerial, &dcb))
417     {
418         cout<<"\n Serial port can't be opened"<<endl;
419         return false;
420     }
421     Sleep(10);
422     bool retVal;
423     while(!has_the_received_byte_been_received)
424     {
425         timer1 = GetTickCount();
426         timer2 = timer1;
427         retVal = WriteFile(hSerial, output_data_to_xbee, 1, &byteswritten, NULL);
428         while(timer2 - timer1 < 1000)
429         {
430             if(!ReadFile(hSerial, whether_received, 1, &received_byte, NULL));
431             if(whether_received[0] == 0x80 || output_data_to_xbee[0] == 0x03)
432             {
433                 has_the_received_byte_been_received = 1;
434                 whether_received[0] = 0;
435                 break;
436             }
437             timer2 = GetTickCount();
438         }
439     }
440     return retVal;
441 }
```

4.2.2 Variable Documentation

4.2.2.1 uchar ball_color[3][2] = {{'<', 60}, {'<', 60}, {'<', 60}}

Stores the ball colour threshold in BGR and condition as per the given ball in bgr value can be changed according to the given ball

4.2.2.2 boundry boundry[10000]

Array of type struct "boundry" which stores all closed boundaries detected in the image.

4.2.2.3 DCB dcb

4.2.2.4 HANDLE hSerial = CreateFile("COM1", GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0)

4.2.2.5 uchar input_data_from_xbee[2] = {0xFF,0}

Stores the bytes received from the Xbee on the bot.

4.2.2.6 int is_the_coordinate_Y = 0

Ture when the data being sent is the Y-coordinate of the bot.

4.2.2.7 uchar output_data_to_xbee[2] = {0x8E,0}

Stores the bytes to be written onto the port.

4.2.2.8 pixel surrounding[8] = {{0,1},{0,-1},{1,1},{1,-1},{1,0},{-1,-1},{-1,0},{-1,1}}

Array of type "pixel" which defines the relative locations of the eight adjecent pixels to every pixel in the image.

4.2.2.9 float threshold_average_distance_from_center = 25

Set the threshold for checking whether the circle is big enough.

4.2.2.10 float threshold_standard_deviation = 20

Set the threshold for checking whether the given boundry is a circle or not.

4.2.2.11 int total_boundries = 0

Stores the number of boundaries detected in the image.

4.2.2.12 int total_circles_detected = 0

Will store the total number of detected circles finally.

4.2.2.13 bool whether_any_object_detected = 0

True when the object is detected.

4.2.2.14 bool whether_sending_x_coordinate_over = 0

True when sending X-coordinate is over and sending the Y-coordinate is being sent.

4.2.2.15 int x_coordinate_of_bot = 0

Stores the displacement of the bot in the X direction.

4.2.2.16 int y_coordinate_of_bot = 0

Stores the displacement of the bot in the Y direction.