CS 101 - PROJECT SPRING 2015

DOCUMENTATION

OPTIMISED BALL COLLECTOR

GROUP - 418

Team Members

***Sudeep Salgia*, 14D070011**

***Aviral Kumar*, 140070031**

***Sohum Dhar*, 140070001**

***Muthamizh Selvan*,140110091**

ACKNOWLEDGEMENTS

We would like to sincerely thank our professor Prof. Kavi Arya for giving us a golden opportunity to work on an embedded systems project with the e-Yantra Project, thus giving us the taste of embedded systems programming. We would also like to thank our TA, Chinmay Kulkarni for all the assistance given to us from choosing an apt project to helping us code and debug our project. We would also like to take this opportunity to thank the entire team of e-Yantra for their guidance and assistance at all stages of the project.

TABLE OF CONTENTS

PROBLEM STATEMENT

It is often necessary to go to a set of certain points in a network and minimization of effort/energy is always a matter of concern. Thus we aim to address this requirement in a prototype scenario.

Thus, using a bot, we intend to collect balls spread over an area ensuring that the distance that the bot travels is minimized.

In order to achieve this, a top view of the arena is provided to the code which connects with the bot and sends the path to the bot which it has to traverse in order to achieve the total path length minimization.

ANALYSIS OF THE CODE

Our code is mainly divided into four major mdoules.

❑ Image Processing
❑ Shortest Path Algorithm
❑ Instructions to the bot
❑ Wireless Data Transfer using X-Bee

In the subsequent chapters each of the modules is explained in detail.

MODULE 1: IMAGE PROCESSING

Header Files:

1. iostream
2. fstream
3. vector
4. openCV library:
    - ❑ opencv2/core/core.hpp
    - ❑ opencv2/highgui/highgui.hpp
    - ❑ opencv2/imgproc/imgproc.hpp

Structures:

1. struct Node { double x, double y};

This structure Node is used to access and modify the (x,y) coordinates of the points stored in the file, and also write coordinates of the points in the file.

Classes :

1. **class Points**

```
{
        double x, double y;
        public:      Points( double x, double y) { }
        };
```

This class is used to easily access points during runtime in the program. It has two constructors, one a default constructor and one a parameterized constructor which takes in two double values and initialises the object with those values.

Vectors :

1. **vector <Points> AllThePoints :** It is a vector of type Points to store points which are detected on the edge of a ball.

2. **vector <Points> Centre:** It is a vector of type Points to store the calculated centres of the balls.

Mat :

Mat is an OpenCV class used for image processing operations. It represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, histograms, etc.

Mat has two data parts: the matrix header (containing information such as the size of the matrix, the method used for storing, at which address is the matrix stored, and so on) and a pointer to the matrix containing the pixel values (taking any dimensionality depending on the method chosen for storing) . The matrix header size is constant, however the size of the matrix itself may vary from image to image and usually is larger by orders of magnitude.

By default any Mat object is allocated dynamically. Each object has a "***reference counter"*** variable associated with it which can be used for automatic allocation and deallocation of memory. Some of the functions of this class which have been used in our program are as follows:

- ❑ **uchar* data-** This is a pointer of type "uchar" (type defined for unsigned char in openCV) which can be used to access the pixel values at any coordinates of an image.

- ❑ **int rows -** Represents the number of rows in the Mat image matrix

- <u>**int cols -**</u>   Represents the number of cols in the Mat image matrix

Some of the constructors of **Mat** used by us in our code are:

- <u>**Mat::zeros(Size(img.cols, img.rows),CV_8UC1);:**</u> This constructor is used to initialise the pixels of a matrix with Size of the image as specified in cols and rows (in order), and then the channel number: CV_8UC1 is used to represent 1 channel image in openCV with zeros. i.e. the image is entirely black.
- <u>**Default constructor:**</u> Mat img;

for more details on Mat and datatypes of OpenCV, refer to the OpenCV documentation on :
http://docs.opencv.org/modules/core/doc/basic_structures.html#mat

SOME KEYWORDS:

- <u>**CV_WINDOW_AUTOSIZE-**</u> This keyword when used makes sure that the window created to display an image would be of the size of the image to be displayed.

- <u>**CV_BGR2HSV**</u>- This Keyword is used to convert the image from RGB format to HSV format.

- <u>**MORPH_ELLLIPSE-**</u> It denotes that the element to be created is ellipse. It is used in getStructuringelement function.

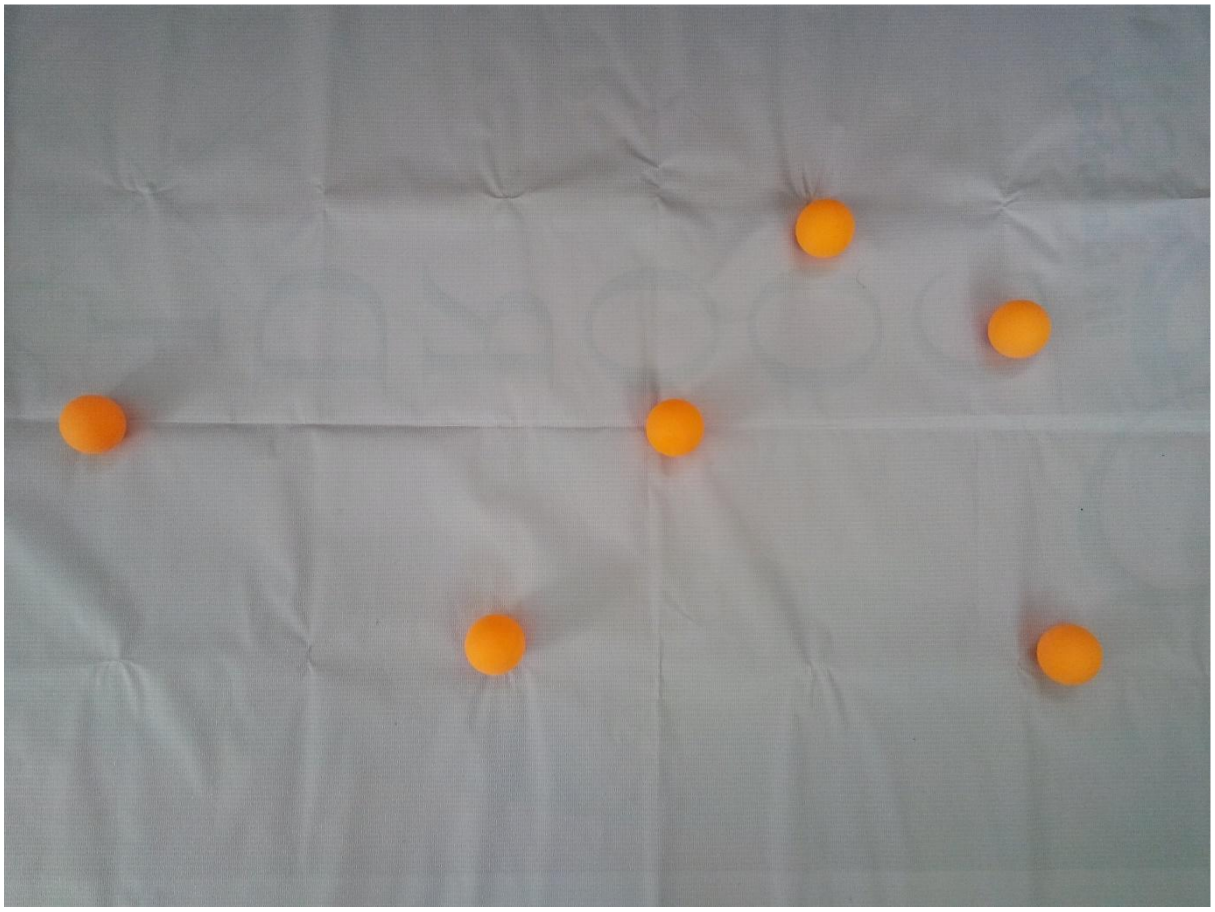
OpenCV FUNCTIONS USED:

- **imread-** The Function is used to read an image into a Mat data type .The Image must be present in the directory where the code is present.

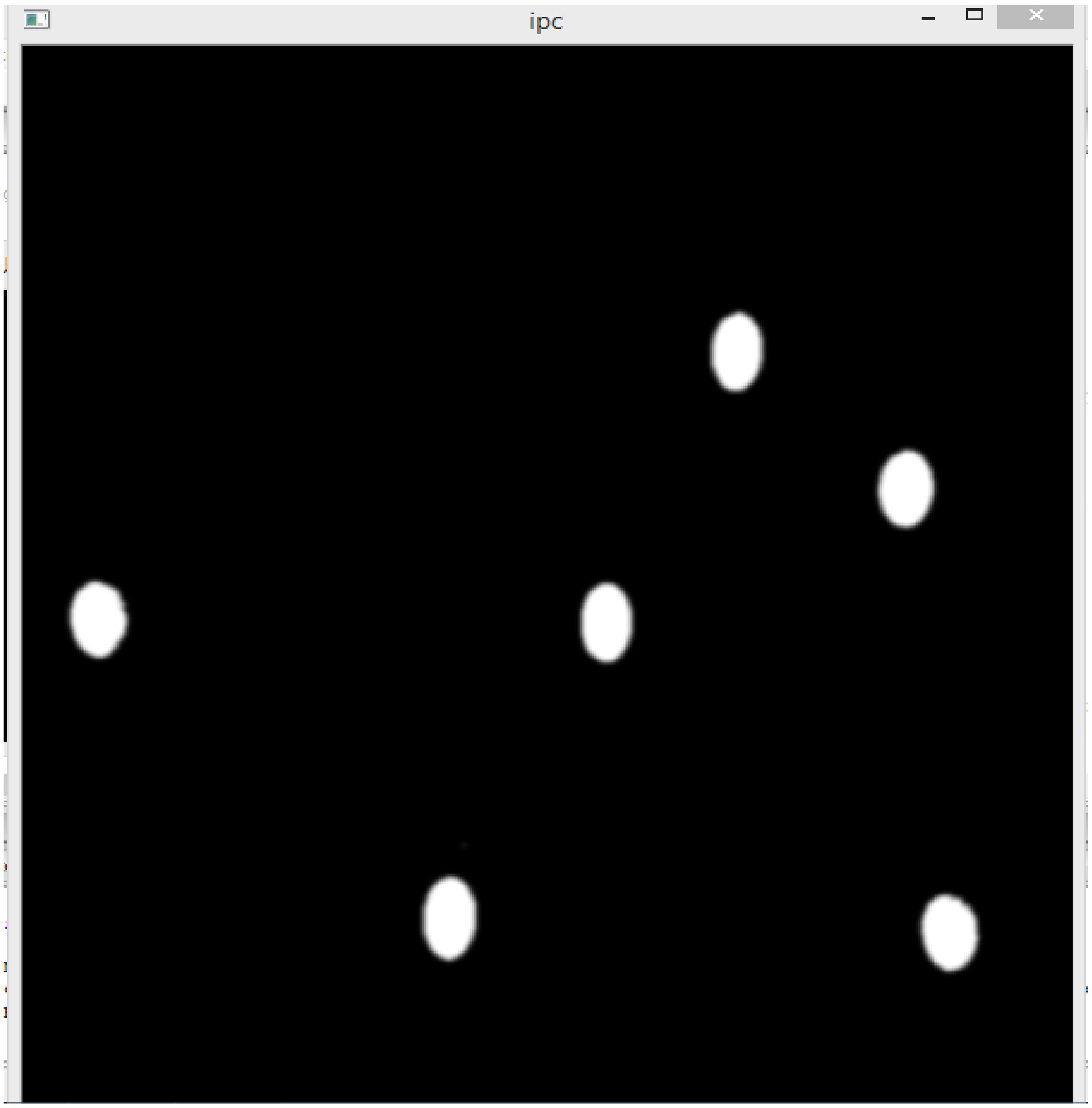
- ❑ **resize-** This Function is used resize the image read into the Mat data type.It is a must if the image goes out of the screen .
- ❑ **cvtColor-** This function is used for image transformations.For Example the image has been converted from RGB format to HSV format for efficient work of the code.
- ❑ **namedWindow-** This Function is used to create a window for displaying the image
- ❑ **imshow-** This Function is used to display the image in the created Display Window.
- ❑ **waitKey-** It waits for the press of a key.until the time specified in the parameter and if zero is provided the program waits forever for the press of a key.
- ❑ **GaussianBlur-** This is an in-built function used to create a transition in those area where the pixels values change abruptly. Thus it smooths the sharp change in pixel values.
- ❑ **getStructuringelement-** This function is used to create a Mat element that would be used in the functions erode and dilate.
- ❑ **Erode-** This function is used to expand the pixels around its neighbourhood.
- ❑ **Dilate-** This Function is used to reduce the size of the certain areas of pixels according to the structuring element .

USER DEFINED FUNCTIONS USED:

All examples shown here are on the following image:



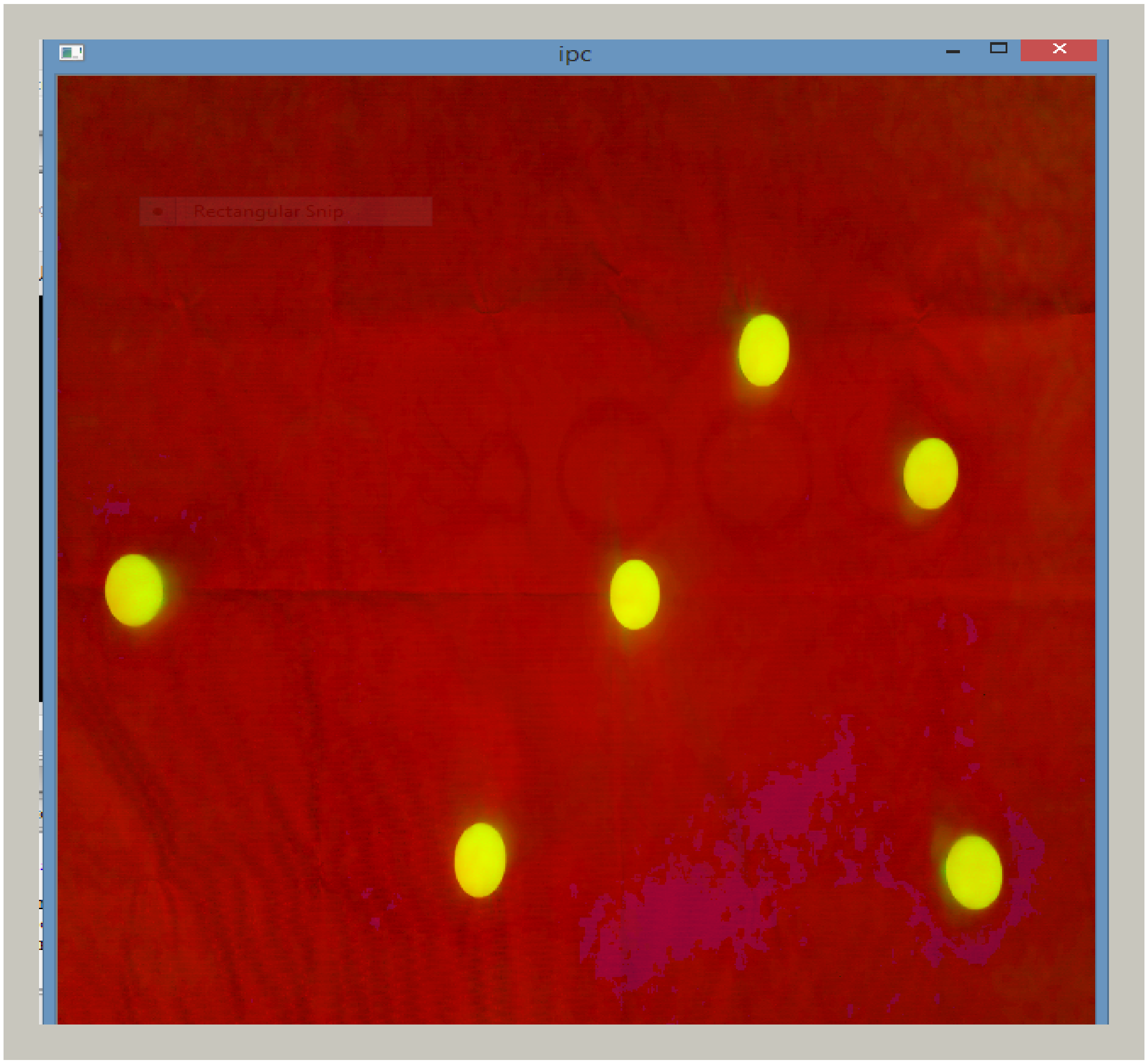
- **Function**- getBinaryImage(Mat &img)
  Input- an openCV 3 channel image matrix
  Output- none (just converts a coloured image may to a black and white binary image)
  Logic - Using proper threshold values for the image taken, the pixels with Red, Green and blue values corresponding to the balls are converted to white and the remaining are converted to black. In other words, other colours are filtered out from the image for easy processing after this.



- **Function -** convertToDigitalisedImage(Mat& img , Mat& img)
  Input - 2 Mat passed by reference: Mat A - 3 channeled, Mat B - single channeled
  Output - None (Just stores the 3-channel binary image developed earlier in a single channel format
  Logic - The three channeled input image is searched for white pixels and in

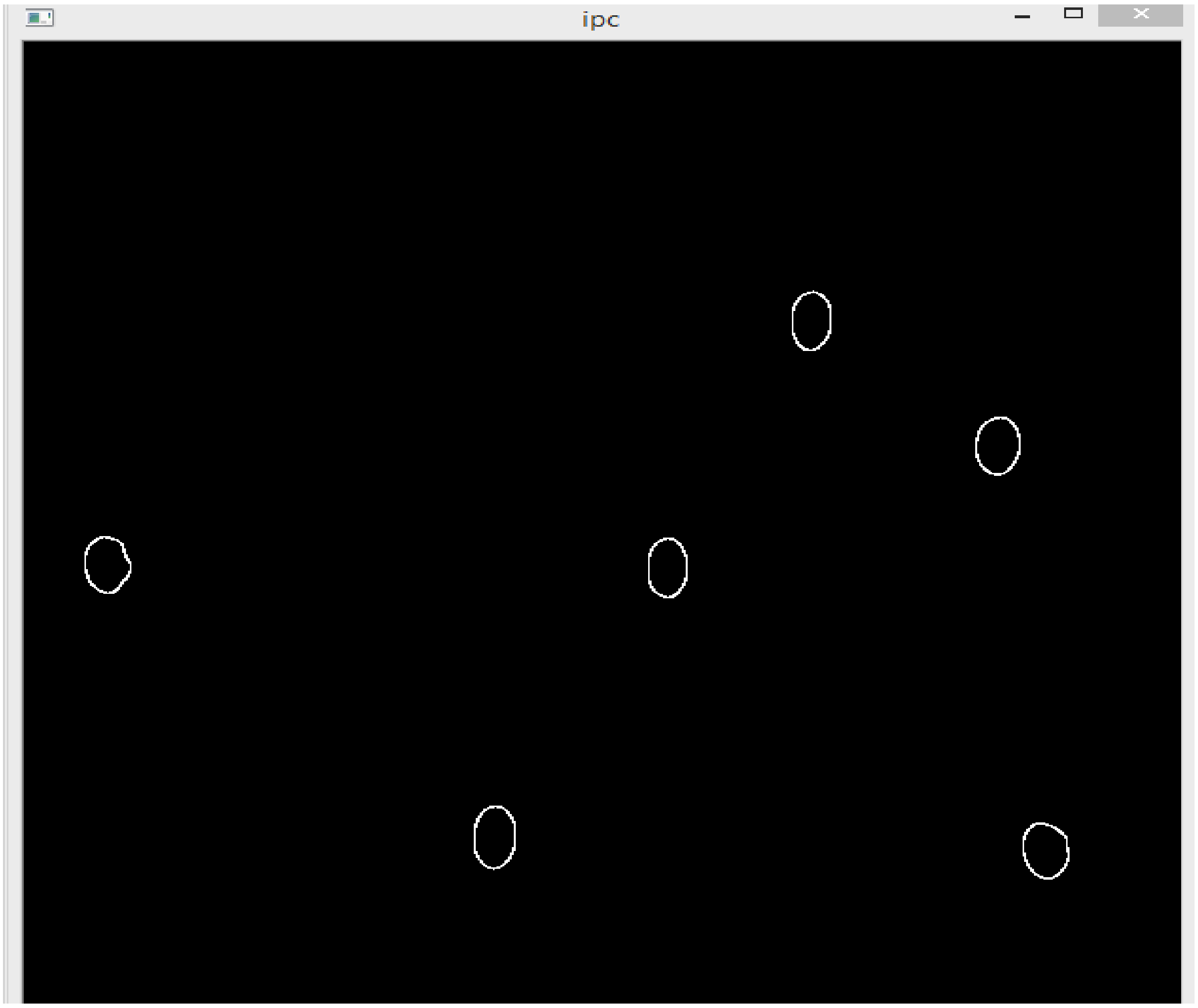
the corresponding pixels in the output single channeled image, white values are stored.

- ❑ **Function -** bgr2hsv2bgr(Mat& img)
  Input - a Mat 3-channeled image
  Output - Outputs an image such that the balls remain in their original colour and remaining background turns red.
  Logic - Converts the image to HSV and then converts those pixels which don’t have H values(hue values) between 10 and 40 to H = 0 (red), and let the balls be in their original colour.



- **Function-** void edgeDetect(Mat &)
  Inputs- Mat &ImgMatrix , i.e. reference to some image matrix
  Outputs- Changes the given matrix into matrix containing only the edges of the balls instead of completely filled balls.
  Logic- Checking the value of color of the pixel, after applying Gaussian blur, errosion and dilate to DigitalisedImageMatrix, to identify if it's an

edge point. The edge point is defined to have color value 50 - 150. This function drastically reduces the number of points to be accounted for, per ball.

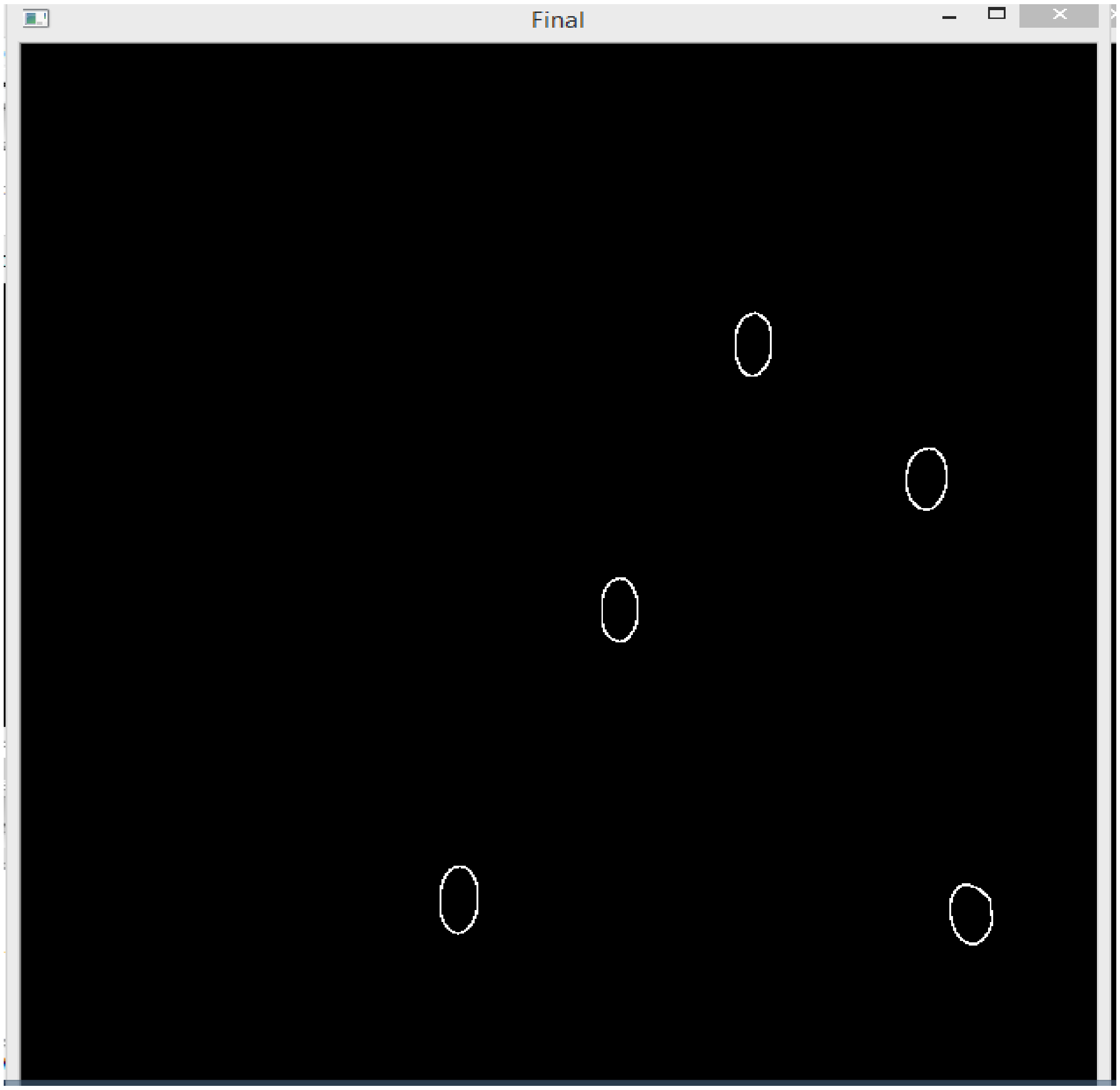


- **Function-** goToNextWhite (Mat& , int , int , vector<Points>&)
  Inputs- Mat ImgMatrix, int i, int j, vector<Points>& AllThePoints
  Outputs- Stores all the points corresponding to a ball in vector AllThePoints **Logic-** A ball is a set of continuous white pixels. goToNextWhite(...) is a recursive function which calls itself with a neighbouring white pixel as new base point. It turns the current white pixel int black so that it is not stored again. Recursion stops when no more white pixels are left in the immediate neighbourhood. Finally, when whole

ball is covered, all the points are stored in AllThePoints and control is returned.
Use - This function has been called inside the find_a_ball() function so there's no image that can be shown for this function.

- ❑ **Function** : void findCentre(vector<Points>& , vector<Points>&)
  Inputs : vector<Points>& AllThePoints, vector<Points>& TheCentres
  Outputs : Stores centre of the ball represented by the set of points in AllThePoints vector, into TheCentres vector
  Logic- The Centre of the ball, i.e. collection of points can be calculated by taking average of the x and y coordinates of all the points on the ball.
  Example : void findCentre(Allpoints, Centres);
  Use - This function has been called inside the find_a_ball function. As soon as the control returns from gotoNextWhite() to find_a_ball(), the findCentre function is called which stores the centre corresponding to the ball which was stored in the vector AllThePoints into TheCentres

- ❑ **Function-** void find_a_ball (Mat , vector<Points>& , vector<Points>&)
  Inputs- Mat ImgMatrix, vector<Points>& AllThePoints, vector<Points>& TheCentres
  Outputs- Locates a complete ball and finds its centre. Does the same for all the balls in the arena and stores their centres in vector TheCentres.
  Logic- Calls function goToNextWhite(Mat, int, int, vector) to find a ball in the arena, store all the points corresponding points for the ball in vector AllThePoints and calls function findCentre(vector &, vector &) to store the centres corresponding to various balls in vector TheCentres.
  Example -



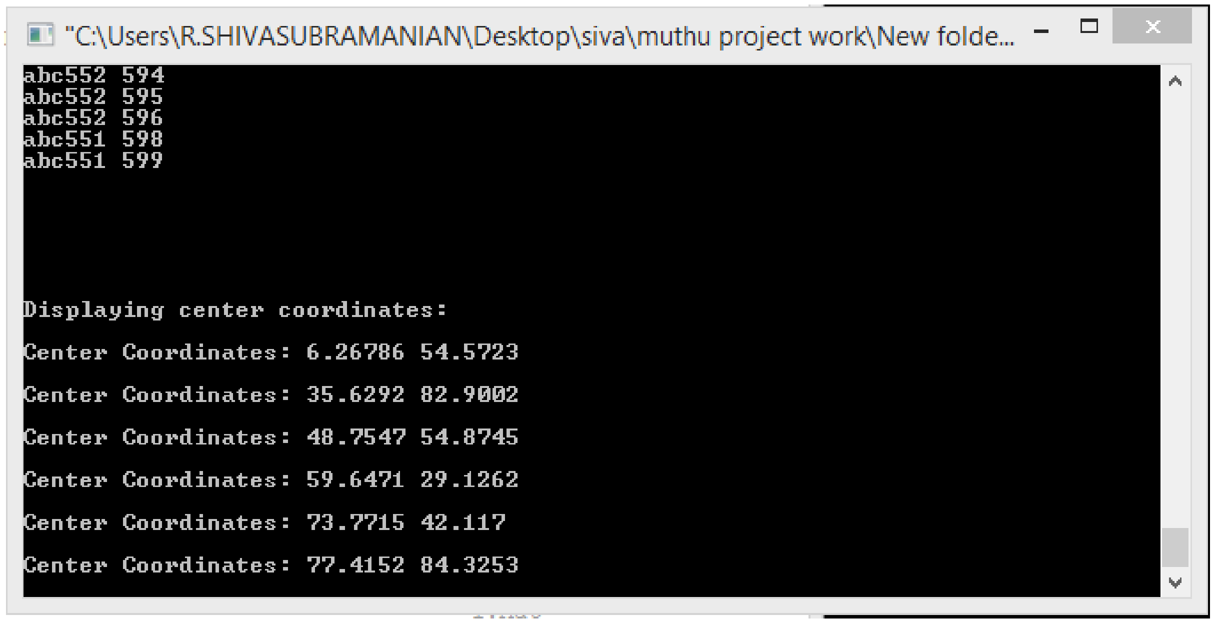
- **Function -** storevalues(vector<Points> &)

  Input - a vector of Points type

  Output - Creates a file named Points.dat and writes the coordinates of the centres of the balls into that file

  Logic - Uses a temporary instance of the struct Node and a Points iterator to access the vector that stores the Centre coordinates and then assigns the x and y of a record to the Temp Node and writes it in a file called Points.dat

Finally the centres are displayed:



"C:\Users\R.SHIVASUBRAMANIAN\Desktop\siva\muthu project work\New folde...

```
abc552 594
abc552 595
abc552 596
abc551 598
abc551 599




Displaying center coordinates:

Center Coordinates: 6.26786 54.5723

Center Coordinates: 35.6292 82.9002

Center Coordinates: 48.7547 54.8745

Center Coordinates: 59.6471 29.1262

Center Coordinates: 73.7715 42.117

Center Coordinates: 77.4152 84.3253
```

MODULE 2: SHORTEST PATH ALGORITHM

Header Files:

1. iostream
2. fstream
3. cmath

Structures:

1. struct Node { double x, double y;
```
                 Node* nextNode;
                 Node(){...}
                };
```

This structure Node is used to access and modify the (x,y) coordinates of the points stored in the file, and also write coordinates of the points in the file. It has a variable nextNode which is a pointer of type Node, which stores the address of the next node to implement linked list to store all the points.

2. struct BranchVector { double radius, double theta};

This structure BranchVector is used to store the (radius , theta), i.e. instruction in (distance, angle) format, which is then written into a file "Instr.dat" (Instr short for Instructions).

Classes :

1. class Path
```
  {
       Node *pathNode, *firstNode;
       int ballCount;
```

```
    double minPathLength;
  public:
    double branchLength(int nodePos1, int nodePos2);
    void sortForGreedyAlgorithm();
    void convertPathToArray();
    void pushBackAnotherNode();
    void swap2BranchEnds(int i1, int i2);
    bool checkPathLength();
    void optimalSolutionWithSwap();
    Node getPathNode(int i);
    int getBallCount();
    double branchLength(int nodePos1, int nodePos2);
    void sortForGreedyAlgorithm();
};
```

This class Path is used to store the points in (x,y) format/ Cartesian Coordinates, and manipulate these points by applying some operations to obtain the Optimal solution as a set of (x,y) coordinates.

2. class BotVectorPath

```
{
      BranchVector* pathVector;
      int branchCount;
    public:
      BotVectorPath( Path );
      ~BotVectorPath();
      BranchVector&  getBranch(int pos);
      int getBranchCount();
  };
```

This class BotVectorPath is used to access and modify the (x,y) coordinates of the points in order in the Optimal Path to get a set of instructions for the robot, Firebird V in our case, as a set of distance, angle, i.e. the distance to be moved forward and the angle to be rotated towards left if positive and right if negative.

FUNCTIONS :

Following is the list of functions used in the code-

- ❏ **Function** : swap2BranchEnds(int, int)

  Input : int i1, int i2, i.e. indices of the fixed end points where corresponding next, previous endpoints are to be swapped

  Output : pathNode[] representing a path with the two specified end points(i1+1, i2-1) swapped in position the sequence of nodes, keeping the corresponding i1 and i2 branch endpoints same

  Logic : Swapping nodes (i1+1) and (i2-1) by altering the path from i1, i1+1 to i2-1, i2 to i1, i2-1 to i1+1 , i2 by simple array operations, i.e. making a part of the array be stored in reverse order by simply swapping entires till the middle. Also i1<i2 is necessary for the loop and thus the code snippet for the same is implemented. (given with comments)

  Example : BotPath.swap2BranchEnds(0,3);

- ❏ **Function** : optimalSolutionWithSwap()

  Input : void

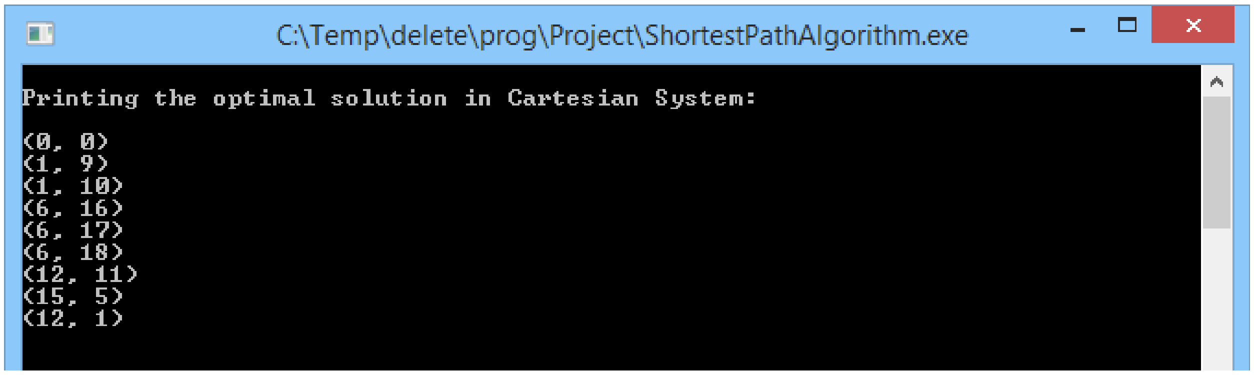
  Output : pathNode[] is set to a path close to the optimal solution for given set of points, and prints the points(x,y) in order as per the path.

  Logic : It tries out all possible swaps between end points of the branches (say with indices i1 and i2) keeping i1-1 and i2+1 fixed. Checks if new path length is lesser by only checking for the change in path length and seeing is its negative. And if so, sets this new path as the new solution using the function call to swap2BranchEnds(i1-1, i2+1) for above mentioned i1 and i2.

  With many iterations(in our case NoOfCycles=5, as mentioned in the namespace), we get the optimal solution.

  Example : BotPath.optimalSolutionWithSwap();

Using a set of points as test case - (0,0), (6,16), (6,17), (1,10), (1,9), (6,18), (12,1), (12,11), (15,5). We get-



```
Printing the optimal solution in Cartesian System:

(0, 0)
(1, 9)
(1, 10)
(6, 16)
(6, 17)
(6, 18)
(12, 11)
(15, 5)
(12, 1)
```

❑ **Function** : getPathNode(int)

Input : int i, i.e. index of the Node

Output : Node on the pathNode[] with index i

Logic : It's an accessor function.

If i is in range i.e. 0 to ballCount-1 return pathNode[i], i.e. Node at index i, else return (0,0), i.e. error point.

Example : Node N= BotPath.getPathNode(0);

❑ **Function** : getBallCount()

Input : void

Output : ballCount ,i.e. total number of balls in the arena

Logic : It's a simple accessor function. Returns ballCount variable defined in the class

Example : int N=BotPath.getBallCount();

❑ **Function** : pushBackAnotherNode()

Input : void

Output : Requests for heap memory and stores the address in pathNode, to be used as a new Node point.

Logic : Using new statement to request for memory from the heap for the new node. In case of entry for the first time, firstNode is initialised with the address of the first node in the linked list. For all other entries the previous node created stores the address of the new node in its data member -'nextNode'. Thus, implementing the function for the linked list of nodes, for insertion of new entry.
Example : pushBackAnotherNode();

- ❑ **Function** : void convertPathToArray()
  Input : void
  Output : pathNode[] is created containing all the points in the file, which were earlier stored as a linked list.
  Logic : Function requests for an array of heap memory of type Node, pointed to by pointer pathNode. All the entries of the linked list are copied into the pathNode array. Copying is implemented using simple accession of linked list elements, starting from first element - whose address is stored in firstNode. The address of the next element is stored in the data member 'nextNode'. Every next entry is accessed using its address stored in nextNode and then the current original node is deleted/freed using delete statement, since it's no longer required.
  Example : void convertPathToArray();

- ❑ **Function** : branchLength(int , int)
  Input : int nodePos1, int nodePos2, i.e. indices of nodes between which distance is to be calculated
  Output : (double) distance between the two nodes specified by the indices, on pathNode[]
  Logic : using distance between two points (x1,y1), (x2,y2) i.e. distance in Cartesian coordinates i.e. square root of (square of difference of(x) + square of difference of(y)). Square root function is implemented using 'cmath' library, included above.
  Example : double d0= BotPath.branchLength(0,1);

- ❑ **Function** : sortForGreedyAlgorithm()

  Input : void

  Output : pathNode[] set to a path decided by the Greedy Salesman Algorithm i.e., the Salesman travels to the next nearest town first.

  Logic : Use 'Selection sorting' algorithm to implement 'Greedy Salesman Algorithm' based on distances between nodes. (Given fixed initial point.) According to the Greedy Salesman algorithm/ Nearest Neighbour algorithm the closest Node/Point is set to be the next destination in the Path to cover all the points. This exercise is then repeated till all the Nodes/ Points are covered. The implementation for our case uses selection sort algorithm, in which the nearest neighbour is picked and its position is swapped with the element at the next Node in the array.

  Example : BotPath.sortForGreedyAlgorithm();

- ❑ **Function** : getBranch(int)

  Input : int pos , i.e. index of the branch you require

  Output : pathVector[pos], i.e. the BranchVector variable corresponding to index pos on pathVector[] array.

  Logic : It's an accessor function returning an element of an array pathVector[] at index 'pos' using []operator.

  Example : BranchVector B=BotVecPath.getBranch(0); // the first(i=0) branch in path is stored into B

- ❑ **Function** : getBranchCount()

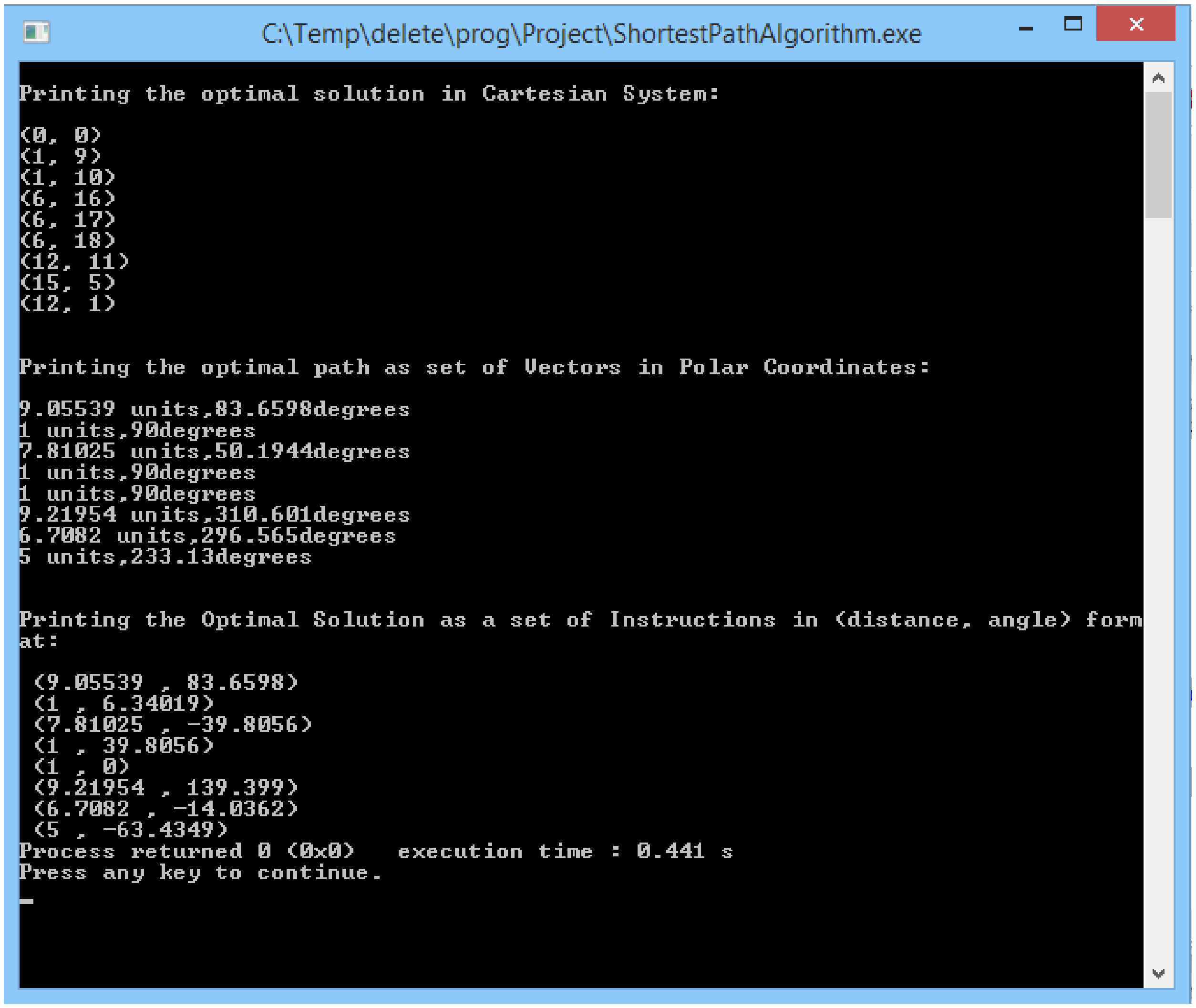
  Input : void

  Output : branchCount, i.e. total number of branch vectors in the optimal path found

  Logic : It's an accessor function returning branchCount i.e. a member of the class.

  Example : int N=BotVecPath.getBranchCount();

Finally, the console window displaying the optimal solution/ path first in Cartesian system, then in polar coordinates system and finally the set of instructions in (distance, angle) format are displayed. These ‘instruction coordinates’ will be stored in a file -”Instr.dat” (‘Instr’ short for Instruction), using simple data file handling.



```
Printing the optimal solution in Cartesian System:

(0, 0)
(1, 9)
(1, 10)
(6, 16)
(6, 17)
(6, 18)
(12, 11)
(15, 5)
(12, 1)


Printing the optimal path as set of Vectors in Polar Coordinates:

9.05539 units,83.6598degrees
1 units,90degrees
7.81025 units,50.1944degrees
1 units,90degrees
1 units,90degrees
9.21954 units,310.601degrees
6.7082 units,296.565degrees
5 units,233.13degrees


Printing the Optimal Solution as a set of Instructions in (distance, angle) form
at:

 (9.05539 , 83.6598)
 (1 , 6.34019)
 (7.81025 , -39.8056)
 (1 , 39.8056)
 (1 , 0)
 (9.21954 , 139.399)
 (6.7082 , -14.0362)
 (5 , -63.4349)
Process returned 0 (0x0)   execution time : 0.441 s
Press any key to continue.
```

MODULE 3: INSTRUCTIONS TO THE ROBOT

HEADER FILES USED:

1. avr / io.h
2. avr / interrupt.h
3. math.h
4. util / delay.h
5. ADC.h
6. lcd.h

CLASSES:

No classes were defined as only functions were required to implement the functionality of the bot.

FUNCTIONS:

1. **Function**: void buzzer_init(void)
   Inputs: void
   Outputs: Initialises buzzer
   Logic: Using DDR to set 4th bit as output bit and others as input bits.
   Example: buzzer_init();

2. **Function**: void buzzer_on (void)
   Inputs: void

Outputs: Starts the buzzer by setting the buzzer on PORTC PIN3 bit high as soon as the function is called.
Logic: Sets PINC 3 bit as high and remaining bits unchanged using masking.
Examples: buzzer_on ();

3. **Function**: void buzzer_off (void)
   Inputs: void
   Outputs: Stops the buzzer by setting the buzzer on PORTC PIN3 bit low as soon as the function is called.
   Logic: Sets PINC 3 bit as low and remaining bits unchanged using masking.
   Examples: buzzer_off ();

4. **Function**:void lcd_port_config();
   Inputs: void
   Outputs: Configures the port which is connected to the LCD
   Logic: Sets the direction of all the pins of the LCD as output and initializes the all LCD pins to logic 0
   Example: lcd_port_config();

5. **Function**: void motion_pin_config();
   Inputs: void
   Outputs: Configures the port A and L for motion and velocity control respectively
   Logic: Sets the third and fourth pins of the port L as output for PWM generation which in turn is used for velocity control and sets the direction lower nibble of port A as output and write a logical 0 to the port
   Example: motion_pin_config();

6. **Function**:void left_encoder_pin_config();
   Inputs: void
   Outputs: Configures the port which is connected to the encoder on the left wheel

Logic: Sets the direction of all the pin connected to the encoder as output and enables internal pull up for the pin (PORT E pin 4)
Example: left_encoder_pin_config();

7. **Function**: void right_encoder_pin_config();
   Inputs: void
   Outputs: Configures the port which is connected to the encoder on the right wheel
   Logic: Sets the direction of all the pin connected to the encoder as output and enables internal pull up for the pin (PORT E pin 5)
   Example: right_encoder_pin_config();

8. **Function**: void motion_set (unsigned char Direction);
   Inputs: one variable of datatype unsigned char
   Outputs: Sets the motion of both the wheels in a specific direction
   Logic: The lower nibble of the port A is replaced by that of the parameter. The parameter has a specific pattern of the lower nibble for specific movement. The 4 bits represent "left wheel forward" , "left wheel backward" , "right wheel forward" and "right wheel backward" in the order of the most significant bit to the least significant bit. A high on a specific bit means the kind of motion desired.
   Example: motion_set(0x06); It would mean both wheels would be moving forward

9. **Function**: void forward();
   Inputs: void
   Outputs: Makes the bot move forward
   Logic: Uses the motion_set(unsigned char) function above with the parameter to make both the wheels move forward
   Example: forward();

10. **Function**: void left();
    Inputs: void

Outputs: Makes the bot move left
Logic: Uses the motion_set(unsigned char) function above with the parameter to make the left wheel move backward and the right wheel forward in order to achieve a left turn using differential mechanism
Example: left();

11. **Function**:void right();
Inputs: void
Outputs: Makes the bot move left
Logic: Uses the motion_set(unsigned char) function above with the parameter to make the right wheel move backward and the left wheel forward in order to achieve a right turn using differential mechanism
Example: right();

12. **Function**: void stop(void)
Inputs: void
Outputs: Stops the bot
Logic: Calls function motion_set(...) to set PORTA to 0x00, i.e. all bits low to stop the bot.
Example: stop();

13. **Function**: void velocity(int left_motor, int right_motor)
Inputs:Two variables of the datatype int to which determine the velocities of the motors
Outputs: Sets the velocity of the bot accordingly
Logic: Writes the values passed as parameters to the respective registers linked to the motors
Example: velocity(100,100); It makes both the wheels to move with the same velocity which corresponds to 100 on a scale of 0-255 with 255 corresponding to the maximum velocity that the bot can achieve

14. **Function**: void port_init()
Inputs: void

Outputs: Initializes all the ports by calling the necessary functions which are defined above
Logic: Call all necessary functions
Examples: port_init();

15. **Function**: void left_position_encoder_interrupt_init (void)
Inputs: void
Outputs: Interrupt 4 is enabled
Logic: INT4 is set to trigger with the falling edge by masking EICRB with 0x02 and enable interrupt INT4 for left position encoder.
Examples: left_position_encoder_interrupt_init ();

16. **Function**: void right_position_encoder_interrupt_init (void)
Inputs: void
Outputs: Interrupt 5 is enabled
Logic: INT5 is set to trigger with the falling edge by masking EICRB with 0x02 and enable interrupt INT5 for right position encoder.
Examples: right_position_encoder_interrupt_init ();

17. **Function**: ISR(INT4_vect)
Description: The Interrupt Service Routine will redirect the control here if an interrupt vector is generated by INT4 and increment the shaft count of the encoder on the left wheel

18. **Function**: ISR(INT5_vect)
Description: The Interrupt Service Routine will redirect the control here if an interrupt vector is generated by INT5 and increment the shaft count of the encoder on the right wheel

19. **Function**: angle_rotate(unsigned int Degrees);
Inputs: one variable of datatype unsigned int

Outputs: Makes the bot rotate by a specific angle which is passed as the parameter
Logic: The degrees to be rotated are mapped to a required shaft count and the reading of the position encoder is checked (i.e., shaft count) and the bot moves until the shaft count is less than the required value
Example: angle_rotate(30); It will rotate the bot by 30 degrees

20. **Function**: void linear_distance_mm(unsigned int DistanceInMM)
    Inputs: unsigned int DistanceInMM, i.e. distance to be moved.
    Outputs: Moves robot forward by specified distance entered in millimeters
    Logic: Compares ShaftCountRight value, which corresponds to the number of the number of interrupt calls, with a value ReqdShaftCount which is calibrated for DistanceInMM.
    Example: linear_distance_mm(100);

21. **Function**: print_values(unsigned int distance)
    Inputs: one variable of unsigned int datatype (distance)
    Outputs: It will print the parameter passed to function on the LCD
    Logic: The function uses the lcd_print() function in lcd.h to print the data passed on the specific coordinates
    Example: print_values(1,2,7,3); Prints 7 on the LCD at the first row and second column

22. **Function**: void forward_mm(unsigned int DistanceInMM)
    Inputs: one variable of datatype unsigned int (DistanceInMM)
    Outputs: Moves robot forward by specified distance entered in millimeters
    Logic: Calls functions forward() and linear_distance_mm(DistanceInMM) to move forward till ShaftCountRight reaches the required value corresponding to the given distance to be moved..
    Example: forward_mm(500); Moves the bot forward by half a metre

23. **Function**: left_degree(unsigned int Degrees);
    Inputs: one variable of datatype unsigned int viz the angle to be rotated
    Outputs: Makes the bot rotate by a specific angle to the left (+) which is passed as the parameter
    Logic: The degrees to be rotated are mapped to a required shaft count and the reading of the position encoder is checked (i.e., shaft count). Functions left() and angle_rotate(Degrees) are called to rotate the bot to left till Required shaft count is reached.
    Example: left_degree(30); It will rotate the bot by 30 degrees to left

24. **Function**: right_degree(unsigned int Degrees);
    Inputs: one variable of datatype unsigned int viz the angle to be rotated
    Outputs: Makes the bot rotate by a specific angle to the right (-) which is passed as the parameter
    Logic: The degrees to be rotated are mapped to a required shaft count and the reading of the position encoder is checked (i.e., shaft count). Functions right() and angle_rotate(Degrees) are called to rotate the bot to right till Required shaft count is reached.
    Example: right_degree(30); It will rotate the bot by 30 degrees to right

25. **Function**: void uart0_init(void)
    Inputs: void
    Outputs: Function To Initialize UART0
    Logic: Initialises various registers related to communication, and following are some specifications:
    Desired baud rate: 9600
    Actual baud rate: 9600 (error 0.0%)
    Char size: 8 bit
    Parity: Disabled
    Example: uart0_init();

26. **Function** : turntoCorrectAngle()
    Inputs : None

Outputs : None, just makes the bot align itself in the direction of the ball
Logic : To reduce the error in rotation, the bot is first rotated by the required angle in the main function and then the bot is rotated to the right by 10 degrees. One by one the bot turns to the left 5 degrees, each time and stores the maxima detected by the sharp sensor and from the left most position, the bot is rotated to this position, so that bot is always directed towards the next ball, to which it has to reach to.
Example : turntoCorrectAngle(); //called this way in main

27. **Function** : int convert_to_binary(int)
Inputs : int x , i.e. number whose binary equivalent is to be calculated
Outputs : binary equivalent of input - decimal number
Logic : binary equivalent is given by the remainders obtained after successive division by 2
Example : long int binaryDistance= convert_to_binary(data);

28. **Function** : int convert_to_int(long int)
Inputs : long int x , i.e. number whose decimal equivalent is to be calculated
Outputs : decimal equivalent of input - binary number
Logic : decimal equivalent is given by the series D= b0 + b1*2 + b2*4 +b3*8 + ⋯
Example : int distance= convert_to_int(data);

29. **Function**: SIGNAL(SIG_USART0_RECV)
Description: ISR(Interrupt Service Routine) for receiving the x-bee transmission from the laptop system

30. **Function**: init_devices();
Inputs: void
Outputs: Initializes all the ports and devices
Logic: By calling the functions defined above to initialize the specific ports
Example: init_devices();

MODULE 4:SERIAL COMMUNICATION VIA XBEE

Header Files:

1. iostream
2. fstream
3. cmath
4. windows.h

Structures:

1. struct InstructionVector

```
{
   double distance;
   double angle;
};
```

HANDLE: In windows programming, the windows maintains a table of all icons, ports, menus, etc, and each file has a unique identifier through which it can be accessed. Windows is given the handle and it returns the port we want.
HANDLE hPort = CreateFile("COM3", GENERIC_WRITE|GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

DCB: A structure that defines the control setting for a serial communications device.
e.g. DCB dcb;

DWORD: DWORD is datatype used to represent 32-bit quantities in Windows.

BYTE : BYTE is a datatype used to represent 8-bit data in Windows Programming.

FUNCTIONS :

Following is the list of functions used in the code-

- ❑ **Function** - writebyte();

  Input - unsigned char* , (pointer to an unsigned char data)

  Output - bool - TRUE or FALSE

  Logic - This function is used for writing a character (only 1 byte) on to the COM port (given by Handle hPort. First of all it checks whether the Port can be opened or not by checking GetCommState(),if FALSE, function returns, else the Port Baud rate, bytesize, parity , etc are specified and then data is written on to the port using writeFile(), which returns TRUE if data is written successfully, else returns FALSE.

  BaudRate = CBR_9600;

  ByteSize = 8;

  Parity = NOPARITY;

  StopBits = ONESTOPBIT;

  Example - char data = 0x05; bool x =writebyte(&data);

- ❑ **Function** - readByte()

  Input - None

  Output - ASCII value of the character(byte) read at the port

  Logic - This function reads a BYTE (8 bit) data from the port and then first checks whether the COM PORT is accessible or not. If it is, then SetCommMask() is called.

SetCommMask() - Specifies a set of events to be monitored for a communications device. Then the function waits until the COM event is finished, i.e. the entire COM file is received. Then this value is read and stored in BYTE byte, and finally an int variable of the type int with ASCII value of BYTE character is returned.

- ❑ **Function** : int convert_to_binary(int)
  Inputs : int x , i.e. number whose binary equivalent is to be calculated
  Outputs : binary equivalent of input - decimal number
  Logic : binary equivalent is given by the remainders obtained after successive division by 2
  Example : long int binaryDistance= convert_to_binary(data);

- ❑ **Function** : int convert_to_int(long int)
  Inputs : long int x , i.e. number whose decimal equivalent is to be calculated
  Outputs : decimal equivalent of input - binary number
  Logic : decimal equivalent is given by the series D= b0 + b1*2 + b2*4 +b3*8 + ...
  Example : int distance= convert_to_int(data);

MODULE 5: DEVELOPMENT OF USER INTERFACE

Header Files:

1. iostream
2. fstream
3. cstring
4. string
5. gtk/gtk.h

Global Variables:

file* data = new file;

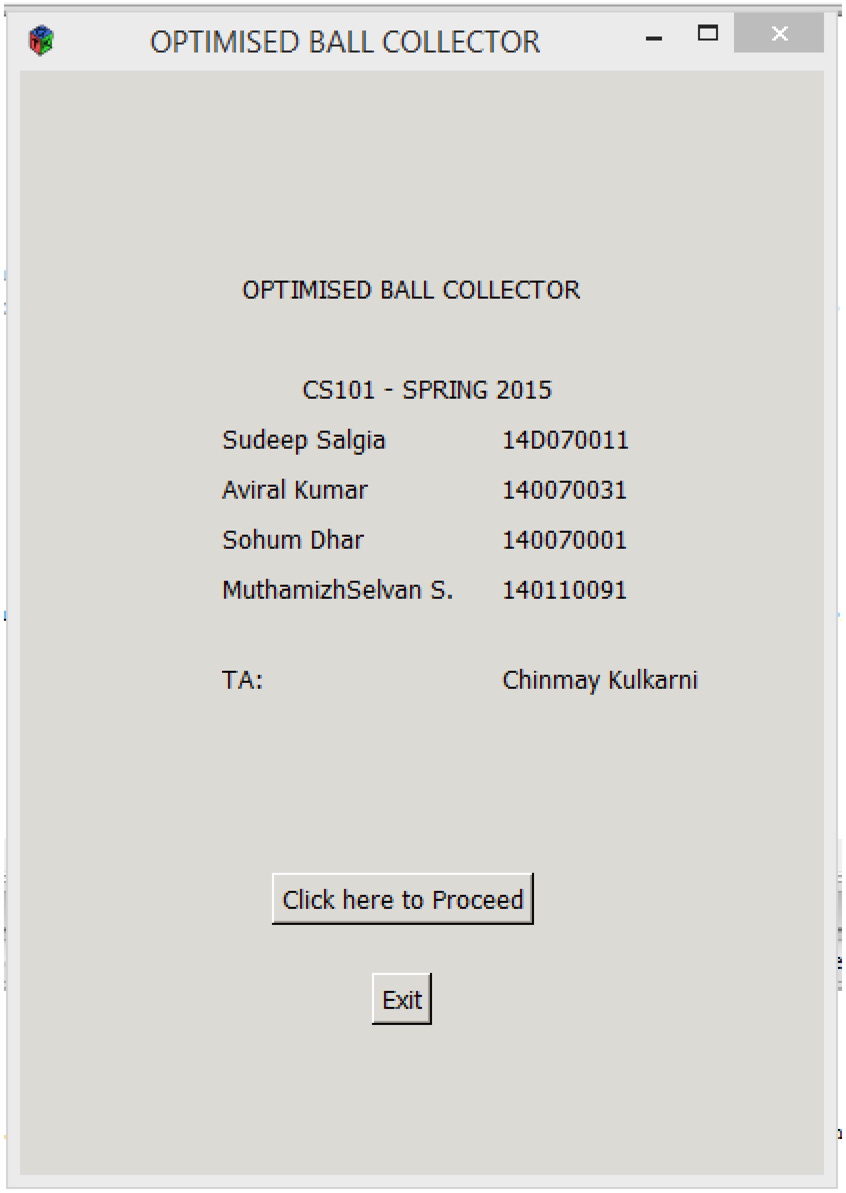
Functions:

1. **void image(GtkWidget *widget, GtkWidget *entry)**
   This function copies the address of the file that has been entered by the user in the entry boxes into the data->filepath variable and stores it.

2. **void dim(GtkWidget *widget, GtkWidget *entry)**
   This function copies the dimension of the image enteredby the user into the variable data->dimension and stores in it.

3. **void new1()**
   This function is used for creating a new window using a GtkWidget* and its attributes. After creating a window, it creates entry fields for entering the path and the dimension of the image. It also has a PROCEED button clicking on which causes the rest of the program, i.e. the image processing and shortest path algorithm to start**.**

Initial Window which opens when the program is executed:

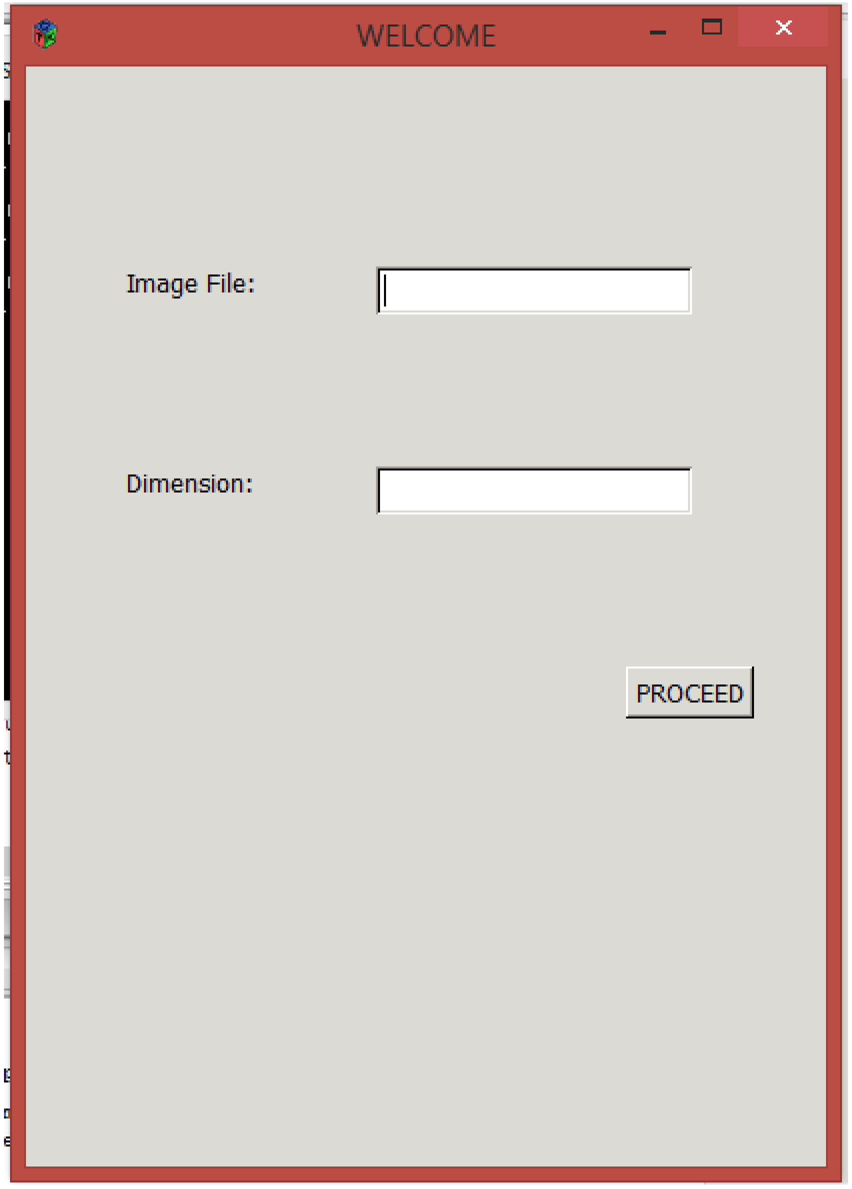


OPTIMISED BALL COLLECTOR

CS101 - SPRING 2015

Sudeep Salgia	14D070011
Aviral Kumar	140070031
Sohum Dhar	140070001
MuthamizhSelvan S.	140110091
TA:	Chinmay Kulkarni

Window created after the user clicks “PROCEED” in the previous window


WELCOME
Image File:
Dimension:
PROCEED