DOCUMENTATION

Group no. 140
Varunesh Goyal-140070006(Group Leader)
Saurabh Chavan-14D070036
Saurabh Pinjani-140070056
Devesh Khilwani-14D070045

SKETCH BOT

USER INTERFACE

The interface has been created in Java using Eclipse IDE and Java SDK 7.

Comprises of five classes

- MainPage
- UserFrame
- StringAnalyser
- CppCommunication
- CppComm1

Class MainPage

The execution of the inteface starts with the execution of this class. Used to generate the main page or the opening window of the GUI. Consists of two Radio buttons and two command buttons.

The two radio button direct to

- Upload Image
- Create Image

The two buttons are

- Enter
- Exit

Enter Button: Opens up the next window depending upon the selected radio button. If none of the options are selected the user is prompted to select one and then press Enter.

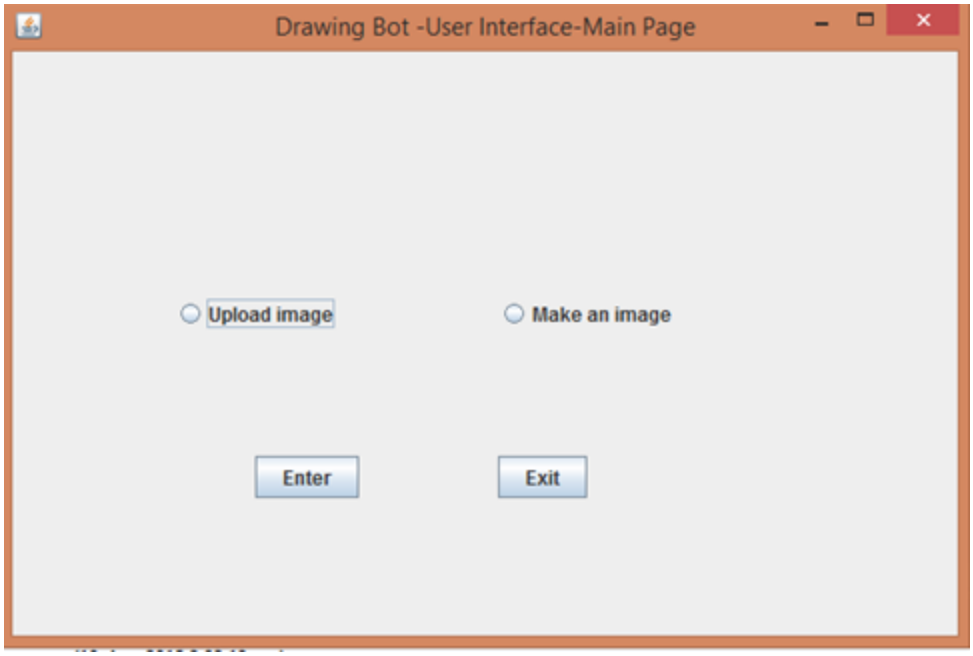
Exit Button : Closes the application

Constructor: MainPage()

It generate the main page.

Also call the constructor of class UserFrame.

Main Function : Explicitely calls the constructor of this class



-The main page of the GUI

Class StringAnalyser

This class is used while sending data to the C++ program from the GUI.

Consists of two functions

- analyse
- revAnalyse

Function : analyse(String)
Return type: String
This function receives a string as an argument. The string may contain spaces. This function replaces the space with ‘!’ (an exclamation mark). Then this modified string a returned.

Function : revAnalyse(String)
Return Type :String
This function receives a string which contains spaces replaced by ‘!’ (exclamation marks). This string replaces the ‘!’ with spaces. It essentially reverses the effect of the function analyse.

Class CppCommunication

This class is brought into use to communicate with the image processing code written in C++ using OpenCV libraries.
This class is used when the user chooses to upload an image file.

This class has only one function i. e. InitComm

Function : InitComm(String, String, Container, String, String)

This function accepts the path of the executable file of the image processing code , the path of the image that is uploaded and the resolution of the image. It also receives the contentpane of the calling page to display a message box.

Of these the address of the image, the resolution and a parity bit are sent to the c++ code by an output stream in the form of a single string

separated by commas. The single bit helped the executable determine that if the user chose to upload an image or draw an image. This function sends a ‘1’ to denote image upload.

Class CppComm1

This class is brought into use to communicate with the image processing code written in C++ using OpenCV libraries
This class is used when the user wishes to draw the image.

This class has only one function i. e. InitComm

Function : InitComm(String, Container, String, String)

This function accepts the path of the executable file of the image processing code and the resolution of the image. It also recieves the contentpane of the calling page to display a message box.

Of these resolution are sent to the c++ code by an output stream in the form of a single string separated by commas. A parity bit is also sent. this bit helps the C++ program determine whether the user chose to draw an image or upload an image
This function sends a ‘0’ to denote that the user chose to draw the image.

Class UserFrame

This is by far the most extensive class. It generates the form which allows the user to upload the image and resolution in X and Y direction.

It has four Buttons:

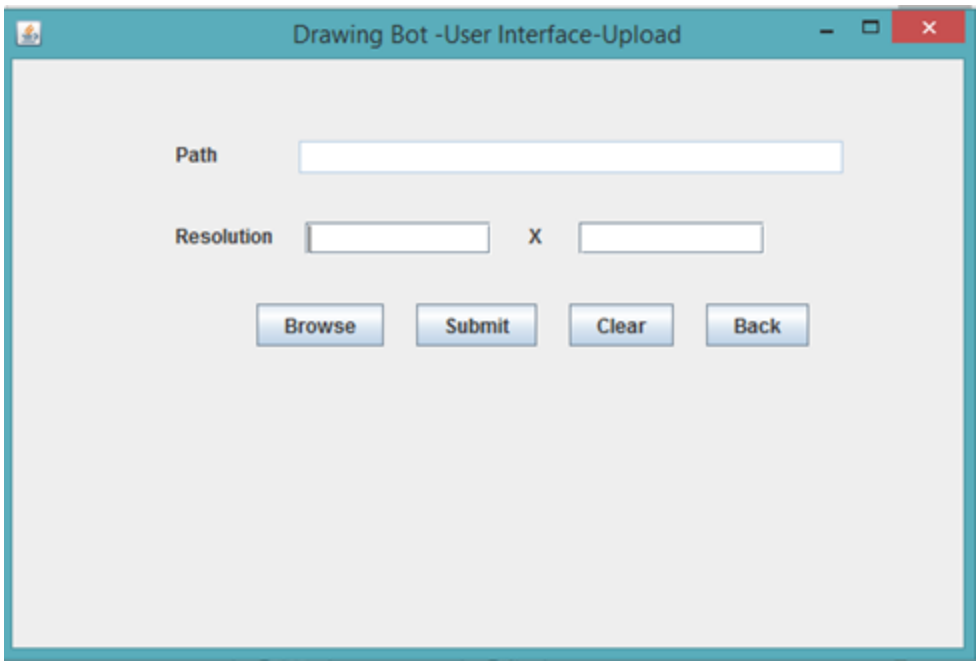
- Browse
- Enter
- Clear
- Back

Browse Button : It opens up the window which allows the user to browse and select the file.

Enter Button : It calls the function initComm of CppCommunication or CppComm1.

Clear Button : It clears the Text Fields

Back Button : It closes this window and takes the user back to the main page.



- The Uploading Page.

Function: getAddress(String)

Return Type: String

This function returns return the address of the executable file using the address of the image file. It uses the fact that the executable file and the image are in the same folder.

Constructor : UserFrame(int)

It generates the upload page by adding suitable GUI elements to the window.

It receives an integer as a parameter. this integer indicates whether the frame will be used for uploading the image or creating the image. The following values of the input parameter carry the specified meaning.

- 1 : Image Upload
- 2 : Image Draw

EMBEDDED C CODE

The entire embedded c code is written in Atmel Studio 6.0.
The additional softwares required are AVR Bootloader NEX Robotics, Microsoft Visual Studio 10, XCTU. The basic header files needed are <avr/io.h>, <avr/interrupt.h>, <util/interrupt.h>, <math.h>.
The Code contains additional header files servo.h, Motion(And its related PWM).h, XBee.h which are needed to be included.

The main functions contained in this header file are:

struct Point

Stores the x and y coordinates of a given point.

motion_pin_config (unsigned char)

return:void

Configures all the required pins related to motion control.

motion_set(unsigned char)

return:void

sets the port A of the bot to receive input in lower four pins and output in higher four pins.

servo1_pin_config (void)

return:void

Configures the servo related pins.

position_encoder_init()

return:void

Configures and initializes the position encoder interrupts.

Function:servo_1_free(void)

return type:void

Makes servo free rotating.

Function:servo_1(unsigned char)

return type:void

Rotates the servo arm to required number of degrees.

Function:penUp(void)

return type:void

Calls servo_1 function and rotates the motor arm to 100 degrees.

Function:penDown(void)

return type:void

Calls servo_1 function and rotates the motor arm to 0 degrees.

Function:move(Point, Point, float)

return:void

Moves from one point to another by calculating the distance and angle to rotate.

Function:update_current_angle(Point, Point, float&)

return:void

Updates the orientation of the bot.

calc_distance(Point, Point)

return:void

calculates the distance between two given points.

calc_rotate_angle(Point, Point, float)

return:float

returns the angle to be rotated.

Contain basic Functions-**left_encoder_pin_config(), right_encoder_pin_config(), left_position_encoder_interrupt_init(), right_position_encoder_interrupt_init()**

which do the given task.

Function:angle_rotate(unsigned int)

return:void

Tells the rotating bot when to stop rotating and stops it.

linear_distance_mm(unsigned int)

Tells the forward moving bot when to stop. Works with the help of position control interrupts and stops it.

Other motion functions include:

forward_mm(unsigned int)

return:void Moves forward by specified distance

back_mm(unsigned int)

return:void Moves backward by specified distance

left_degrees(unsigned int)

return:void turns left by specified angle

right_degrees(unsigned int)

return:void turns right by specified angle

soft_left_degrees(unsigned int)

return:void turns soft left by specified angle

soft_right_degrees(unsigned int)

return:void turns soft right by specified angle

soft_left_2_degrees(unsigned int)

return:void turns soft left 2 by specified angle

soft_right_2_degrees(unsigned int)

return:void turns soft right 2 by specified angle

uart0_init()

return:void

Configures the xbee pins and sets the baud rate.

buzzer_on() and buzzer_off()

return:void

Turns the buzzer on and off respectively by configuring the port C.

_delay_ms(unsigned int)

return:void

Makes delay in executing the next instruction.

Interrupts used:

1. Position encoder interrupt

ISR(INT4_vect) and **ISR(INT5_vect)**

interrupts for left and right position encoders respectively.

2. Signal receiving interrupt by the bot

SIGNAL(SIG_USART0_RECV)
Activates ISR whenever the bot receives an input through the xbee.
C++ CODE

This part of the program is written in Code::Blocks 13. 12 with mingw compiler. The developer need to download opencv libraries from net and include them in code::blocks. The link to install opencv is given in references.

The basic header files required are core. hpp, highgui. hpp, cv. h and windows. h. The additional header files required to be included are Points. h, image_functions. h, global_variables. h, path_algo_related_functions. h, Draw_image_on_windows. h, parsing. h

The functions used in the c++ code are:
Function:First Coordinate(string)
Return Type:int
It extracts the first coordinate of resolution into the cpp program from the data string received from User Interface.

Function:Second Coordinate(string)
Return Type:int
It extracts the second coordinate of resolution into the cpp program from the data string received from User Interface.

Function:Address(string)
Return Type:string
It extracts the substring corresponding to the address of the image into the cpp program from the data string received from User Interface.

class Points
Structure which stores the x and y coordinates of any point.

Function:analyser(string)
return:string
analyser for the purpose of making the '!' back to ' ' which we had deliberately done in java so that theres no problem in cin in cpp.

Function:CallBackFunc(int, int, int, int, void*)
return:void
Makes the pixels over which mouse has hovered while pressed black and puts them in a path

Function:Erase(int, int, int, int, void*)
return:void
Enables erasing unwanted part of the image from the input image taken.

Function:handleError(int, char, char*, char*, char*, int, void*)
return:int
This is to supress the error display if any related to inbuilt functioning of the opencv library.

Function:main()
return:void
The main function takes the image input or drawn image, processes it to convert it to Black and White, detects borders, removes noise from it, stores the border in a vector “path”, extracts specific points from it and then, sends the points along with the status of servomotor and sign of the points to the bot through xbee one by one after receiving previous data from xbee.

Function:draw_the_line(int, int, int, int, Mat &, vector<Points> &)
return:void
Draws a continuous line between 2 points and stores the points in path

Function:Draw_the_line_white(int, int, int , int, Mat &)
return:void
Makes a white line between the points i. e. erases that part of image.

Function:writebyte(char*)
return:bool
Writes a byte to the serial port.

Function:readByte()
return:int
reads a byte from serial port.

Function:sendAndRecieve(char &)
return:void
to write and then read the response

Function:Function:calc_rotate_angle(Points, Points, float)
Return:int
Calculates the angle to rotate given the starting coordinates, final coordinates and the current orientation of the bot

Function:calc_distance(Points, Points)
Return:float
calculates distance between 2 points

Function:update_current_angle(Points, Points, float &)
return:void
Updates the current orientation of the bot. To be executed after the bot completes the move.

Function:BlackNWhite(Mat &)
Return:void

First using 150 as the threshhold value. . . the input image gets changed. This one is used in the main function.

Function:BlackNWhite(Mat &, unsigned char)
Return:void
Here if user doesn't want the input image to change, he can provide an output image (threshhold = 150)

Function:BlackNWhite(Mat &, Mat &)
Return:void
Output image is different and converted to black and white image.

Function:BlackNWhite(Mat &, Mat &, unsigned char)
Return:void
Output image is different and option to set the threshhold value by user

Function:clear_traversed_path(Mat, vector<Points> &)
Return:void
Makes a 3x3 box around every pixel present in path white in the image entered. This is to remove excess pixels which might remain after a path is generated so one of the * has remained. It doesnt belong to a different path and so must be deleted

Function:clear_traversed_point(Mat, Points)
Return:void
Clears a point once crossed to avoid infinite recursion.

Function:detect_borders(Mat, Mat)
Return:void
Checks src_in_blackNwhite for black points which have white pixels surrounding them. These are boundary points and shown in the output image.

Function:make_a_path(Mat, vector<Points> &)
Return:bool
This function uses a vector to store the black pixels as encountered in a path in proper order.
returns 0 if no path present
returns 1 if a path is successfully determined and stored

Function:merge_paths(std::vector<Points> &, std::vector<Points> &, bool &, bool &)
Return:void
Checks if the path1 and path2 have any connection and if it exists, appends a path to another in the beginning or at the end in same order or reverse order as required
Changes the isPath value for the path which is added. So only the path to which addition is done stays.
If no connection, nothing happens.

Function:nextPoint(Mat, Points)
return:Points
Returns an adjacent black point(i. e. in a 3x3 box)

Function:nextPointToDraw(std::vector<Points>::iterator, std::vector<Points> &, float, float, float)
Return:Points
This is to search in a path for points which can be drawn by the bot (fits the least count of various movements of bot. Not used in main function as of now. There the code is reproduced.

Function:reduce_noise(Mat)
Return:void
Removes isolated pixels

We have also added an extra feature which enables the user to erase points (by RightMouse click + drag) which he doesnt want in final drawing

Function:resize_image(int, int, const Mat&, Mat&)
Return:void
Resizes image in case of uploaded image to a maximum height and maximum width in a proportional manner. Image isn't cropped or distorted

Function:startingPoint(Mat)
return:Points
If no black pixel left, returns (-1, -1). Input: image consisting of only borders.