Research Report

Navigation and Obstacle Avoidance for Blind People Using Kinect Vision System

by

Clemen Deng, Dina Zaslavsky, Shadman Abedin

March 31st, 2013
# Table of Contents

List of Figures ............................................................................................................. 4  
List of Tables ................................................................................................................ 5  
Abstract ...................................................................................................................... 7  
1. Question and Problem Being Addressed ............................................................... 8  
2. Research .................................................................................................................. 8  
   2.1 Overview of IVB System Design ....................................................................... 10  
   2.2 Microsoft Kinect Module ................................................................................. 11  
      2.2.1 Kinect Module ............................................................................................ 11  
      2.2.2 RGB Camera ............................................................................................ 11  
      2.2.3 3D Depth Sensor ....................................................................................... 12  
      2.2.4 Depth by Stereo ....................................................................................... 12  
      2.2.5 Depth by Blur ........................................................................................... 12  
   2.3 Hardware Design ............................................................................................... 12  
      2.3.1 Support Structure Design ......................................................................... 12  
   2.4 Path Finding Algorithms ................................................................................. 13  
      2.4.1 Background ............................................................................................... 13  
      2.4.2 Dijkstra Algorithm ................................................................................... 14  
      2.4.3 A* Algorithm ........................................................................................... 15  
   2.5 Canny Edge Detection ..................................................................................... 18  
      2.5.1 Background ............................................................................................... 18  
      2.5.2 Filtering .................................................................................................... 18  
      2.5.3 Image Gradient ......................................................................................... 18  
      2.5.4 Non-Maximum Suppression .................................................................... 19  
      2.5.5 Hysteresis Threshold ............................................................................... 19  
   2.6 Hough Transform ......................................................................................... 19  
      2.6.1 Background ............................................................................................... 19  
      2.6.2 Polar Coordinates ..................................................................................... 20  
      2.6.3 Process of Detecting a Line .................................................................... 20  
      2.6.4 Threshold ................................................................................................. 22  
   2.7 Object Detection ............................................................................................... 28
2.7.1 Background ........................................................................................................... 28
2.7.2 Door Detection ..................................................................................................... 28
2.7.3 Obstacle Avoidance ............................................................................................. 28
2.8 Path Following Detection ......................................................................................... 28
2.9 Viola Jones Algorithm ............................................................................................. 29
  2.9.1 Background .......................................................................................................... 29
  2.9.2 Haar Cascades ...................................................................................................... 29
  2.9.3 The Stages ............................................................................................................ 29
  2.9.4 Scanning ................................................................................................................ 30
  2.9.5 Adaboost ............................................................................................................... 31
2.10 Text Recognition ..................................................................................................... 32
  2.10.1 Background ......................................................................................................... 32
  2.10.2 Deskewing Lines ................................................................................................. 32
  2.10.3 Determining Each Character .............................................................................. 32
    2.10.3.1 Fixed Pitch ..................................................................................................... 32
    2.10.3.2 Non-Fixed Pitch ............................................................................................ 32
  2.10.4 Word Recognition ............................................................................................... 33
    2.10.4.1 Non-Fixed Pitch ............................................................................................ 33
    2.10.4.2 Classifying Words .......................................................................................... 33
  2.10.5 Location Prediction .............................................................................................. 33
2.11 Text to Speech ......................................................................................................... 33
  2.11.1 Background ......................................................................................................... 33
  2.11.2 Microsoft Speech API (SAPI) ............................................................................ 34
2.12 Shape-coded Keyboard Design .............................................................................. 34
2.13 Software Flowchart of IVB User Interface ............................................................. 36
3. Each Team Member’s Responsibility for This Project .............................................. 37
4. Hypothesis .................................................................................................................. 37
5. Variables .................................................................................................................... 38
6. Materials ..................................................................................................................... 38
7. Procedure * ................................................................................................................ 39
8. Data Analysis .............................................................................................................. 40
  8.1 Door Detection Data Table and Analysis ................................................................. 41
    8.1.1 Door Detection Data Analysis ............................................................................ 41
    8.1.2 Improved Door Detection Algorithm ................................................................. 41
3
List of Figures

Figure 1: IVB System Design Flowchart................................................................. 10
Figure 2 Kinect Sensor.......................................................................................... 11
Figure 3 Improved Wooden Frame Supporting Kinect Sensor ............................ 13
Figure 4 Map Example.......................................................................................... 13
Figure 5 Dijkstra Algorithm Tree Path Example .................................................. 14
Figure 6 Dijkstra Algorithm Smallest Path Example ............................................. 14
Figure 7 A* Algorithm Path Finding Example ..................................................... 15
Figure 8 A* Program in Action I ........................................................................... 16
Figure 9 A* Program in Action II.......................................................................... 16
Figure 10 A Line in Polar Coordinates ............................................................... 20
Figure 11 An Illustration Example of Hough Transform Line Detection .............. 21
Figure 12 Point in Parameter Space Where the Lines Intersect ............................. 22
Figure 13 Example of Edge Detection and Hough Transform – Building Corridor 1 (order: original, edge detection, Hough transform) ........................................................................ 23
Figure 14 Example of Edge Detection and Hough Transform – Door (order: original, edge detection, Hough transform) ........................................................................ 24
Figure 15 Example of Edge Detection and Hough Transform – Building Corridor 2 (order: original, edge detection, Hough transform) ........................................................................ 25
Figure 16 Example of Edge Detection and Hough Transform – Building Lounge (order: original, edge detection, Hough transform) ........................................................................ 26
Figure 17 Example of Edge Detection and Hough Transform – Building Corridor 3 (order: original, edge detection, Hough transform) ........................................................................ 27
Figure 18 An Example of a Haar Cascade ............................................................ 29
Figure 19 All the Cascades Checked in One Stage ............................................... 30
Figure 20 Near the End of the Entire Scanning Process .......................................... 31
Figure 21 Input and Output of Viola Jones Algorithm ........................................... 31
Figure 22 The process of determining if something is fixed pitch ......................... 32
Figure 23 Finding the concave vertices points ..................................................... 33
Figure 24 SAPI Interface ..................................................................................... 34
Figure 25 IVB System Distance Sensing Test Results ......................................... 42
Figure 26 Team Member 1 Wearing and Testing IVB System ............................ 45
Figure 27 Team Member 2 Wearing and Testing IVB System ............................ 45
Figure 28 Team Member 3 Wearing and Testing IVB System ............................ 46
Figure 29 IVB System Key Building Block Average Accuracy .............................. 47
List of Tables

Table 1  Door Detection Data Table ........................................................................................................... 41
Table 2  Depth Data Table ........................................................................................................................... 42
Table 3  Face Detection Data Table .......................................................................................................... 43
Table 4  A* Search-based Path Finding Data Table ..................................................................................... 44
Abstract

The purpose of this project is to design an easy to use navigation and object avoidance interactive system for blind people. Our hypothesis is we can design a navigation and obstacle avoidance system using the Kinect sensor to help blind people navigate a known building. These are the key features of our Interactive Vision for Blind (IVB) System. First, it incorporates a pre-programmed map of a known building by implementing the A* path finding algorithm. Second, it implements object recognition using edge detection and Hough transform image processing algorithms to detect objects such as doors. Third, it can detect human by implementing the Viola Jones face detection algorithm. Fourth, it integrates speech synthesis using the Text To Speech algorithm. Lastly, it has command recognition by using a wireless shape-coded keyboard.

The system runs on a laptop computer and a portable battery with the Kinect sensor connected to it. We blinded ourselves with a mask to test the system. Test results show our hypothesis was correct. The system can give the correct directions to a blind person on the destinations and give proper warnings on objects in front of the blind person such as doors and people. It has high precision distance detection with 97% accuracy. In summary, this is the first time a navigation and object avoidance interactive system was built using a Kinect sensor for the blind with the integration of a pre-programmed map, object and human detection, speech synthesis and command recognition using an innovative shape coded keyboard.
1. Question and Problem Being Addressed

Can we design a navigation and obstacle avoidance system using the Kinect sensor to help blind people to navigate the known building?

2. Research

The three of us that are working on this project attended a weekly image processing lecture with multiple students given by a professor at a local university. One day the professor told us about his friend and colleague who is a blind. As the three of us met and decided what kind of science fair project we wished to do, the blind professor stuck in our minds. We wanted to choose a science fair project that can actually solve practical everyday problems. So we decided to design a system that can help blind people navigate. The local university professor told us about the challenges this blind professor faced in navigating the corridors of a building. This project is aimed at making it easier to travel for blind people through hallways in not just their own university but in other buildings. We decided on using the new and innovative Kinect sensor to assist us. We researched similar projects such as the one called NAVI in Germany [2] and decided to improve on this one by including a pre-programmed map, object detection, human detection and communication to the blind through speech. We plan to build a navigation and obstacle avoidance interactive system for blind people using the Kinect sensor. What is new for this project from previous work is that this is the first time a building map, speech synthesis, human face recognition, object detection and obstacle recognition is integrated to create a navigation and obstacle avoidance interactive vision system for helping blind people in real time. This system tells a blind person what is in front of him or her. The user can also ask the system to take them somewhere and the system uses a pre-programmed map to navigate through the building to the desired location. The system detects people and objects such as doors. In the future, this system can be used not only by blind people but also can be adapted for robots to create robot vision.

Existing aids for blind people include Seeing Eye dogs, canes, and assistance by others. A Seeing Eye dog is a dog that has been specially trained to help blind people navigate. A blind person may also use a cane to navigate around his neighborhood. The user is taught to work with the cane and swing the cane as they walk to help them find objects and swerve around them. Both the Seeing Eye dogs and the cane have their limitations and aren’t ideal for a blind person. A blind person who is allergic to dogs or finds cane usage difficult has to work with a volunteer to help them walk around town, crossing busy streets and conducting such tasks as food shopping and banking [1]. This is not ideal since there may not be a volunteer at the blind person’s side all the time. Recent research using computer and robot vision to aid blind navigation include a project designed by a pair of graduate students at the University of Konstanz in Germany. This project uses a Kinect sensor to help blind people navigate their environment. This ‘NAVI’ (Navigational Aids for the Visually Impaired’) design consists of connecting a Kinect unit to a helmet which is connected to a laptop, linked with a Bluetooth headset and motor-controlled belt. The Kinect camera’s infrared sensors provide warnings and directions when a wall or door is coming up. [2] The design is bulky as the user needs to wear a laptop in a rucksack on their back and a helmet.
We plan to improve the NAVI design. The navigation and object avoidance interactive system we developed integrates the following new features that improve over prior designs. First, our design incorporates a pre-programmed map of a known building by implementing the A* search based path finding algorithm. Second, our design implements object recognition using the Canny edge detection and Hough transform image processing algorithms to detect objects such as doors. Third, our design implements human face detection by using the Viola Jones face detection algorithm. Fourth, our design integrates speech synthesis by using the Text To Speech algorithm. Fifth, the program implements an obstacle avoidance function that prevents the user from crashing into walls. Sixth, our method uses text recognition to help the user locate where he is in the building. Lastly, we plan to design the hardware to be easy to use by blind people by discussing and taking feedbacks from a blind professor on where to mount the Kinect sensor. We plan to attach the Kinect sensor to a laptop in a backpack worn by the blind person. The Kinect sensor is connected to the laptop and located above the head of a blind person. We also use a shape coded wireless keyboard to help the blind person communicate with the system. This is the first time a navigation and object avoidance interactive system has been designed and built using a Kinect sensor with the integration of a pre-programmed building map, object recognition, human detection, speech synthesis, text recognition and innovative easy to use hardware design with a wireless shape coded keyboard for command recognition. We call this system the Interactive Vision for Blind System, or IVB System.
2.1 Overview of IVB System Design

Figure 1 below is the flowchart of the overall IVB System design.

**Figure 1: IVB System Design Flowchart**

![IVB System Design Flowchart](image)

**Explanation**

After the Kinect sensor takes color and depth pictures, the pictures get put into both face detection and object recognition algorithms. In the face detection path, the image is inputted into the Viola Jones face detection program and searched for any human faces. In the object recognition path, convolution based Canny Edge Detection and Hough transform algorithms are applied to it to filter out any noise and detect edges and objects. Then, basic items are searched for, such as doors, windows, and any obstacles in the way. Both these algorithms print the information found by them to a text file, which is next used by the text-to-speech program. The text-to-speech program uses a speaker to convey the found image information to the blind person.

The system can also be controlled by the user using his or her command recognition to get directions for where he or she wants to go. The system processes the user’s command inputs by shape-coded keyboard, and tells the user where to go. It uses a pre-programmed building map to determine the user’s current location while telling him or her whether there is a door or person...
nearby. This information helps the blind person to make a decision about going forward or turning left or right.

When the user walks, the system goes in a loop of taking the input from the wireless keypad that the user controls and doing the function that corresponds to the button the user presses.

2.2  Microsoft Kinect Module

One of the main components of this project is a Microsoft Kinect Module. This is added in conjunction with a laptop.

2.2.1  Kinect Module

The Kinect Module contains four major pieces of hardware: the RGB color camera, the 3D Depth Sensor, the Multi-array Microphone, and the motorized tilt. In this project, only the RGB camera and the depth camera are needed.

![Kinect Sensor](image)

2.2.2  RGB Camera

The Kinect has an RGB Camera, which is used to store the data of an image digitally using the RGB color model – the way of filtering an image through the intensities of red, green, and blue at each pixel. The Kinect's RGB Camera returns a data stream of a 640 x 480 pixel image at 32 bits of color (meaning there are 232 different colors recognized) at a rate of 30 Hz (30 frames per second). The camera's resolution is 8 bit in VGA format (Video Graphics Array – which is the standard format for display in electronics).
2.2.3 3D Depth Sensor

The Kinect is also equipped with a 3D depth sensor. This sensor is made of two parts: the infrared projector, and the monochrome CMOS camera. The infrared projector projects a known infrared pattern of dots in the direction it is being pointed. The CMOS camera is then used to capture the location of the dot. Then, the previously known pattern and the detected dots are compared, and using the differences between the dots, the depth of the scene the Kinect is “seeing” can be determined. The Kinect calculates the depth using two methods: depth by stereo and depth by blur.

2.2.4 Depth by Stereo

Depth by stereo is based off of an idea known as parallax, which is the idea that objects that are closer will look like they move more than objects that are further away when the field of view is moved. For example, when a person is a field with a pole in front of him and a tree far off into the distance, and he moves a bit to the right, the pole will look like it moved more to the left than the tree. For this reason, the infrared projector and the infrared camera are located on different parts of the Kinect module.

2.2.5 Depth by Blur

Depth by blur works based off of the assumption that dots that are further away will be more blurry than dots that are close by. For example, when one puts a flashlight right up to his hand, the circle of light has a clearly defined perimeter, but when it is shined onto a far wall, the edges get blurry.

2.3 Hardware Design

We design the hardware with a backpack. The laptop computer is in the open backpack and is connected to the Kinect sensor. There’s a mobile battery pack for the Kinect sensor. Attached to the backpack’s two side pockets are two vertical wooden poles, which rise above the user. These poles converge in the center at a reasonable height (6-12 inches above the user) and support the Kinect. A shape coded keyboard is used by the blind person to communicate with the system interactively in real-time.

2.3.1 Support Structure Design

Initially we put measured and cut 3 wooden frames and used nails to put them together to support the Kinect sensor. But initial testing shows the wooden frame was not stable enough. We then use small wooden frames to support the original wooden frame at the corners using. The revised design is much more stable. The figure below shows the improved design.
2.4 Path Finding Algorithms

2.4.1 Background

The purpose of using a path finding algorithm is to help the user find his way from one room to another using a relatively short path. The user should be able to say where he wishes to go, the path finding algorithm will calculate a path to get from where the user stands at the moment to the desired location. For this project, we considered two different path finding algorithms: the Dijkstra algorithm and the A* algorithm. After comparing the two algorithms, we found that the A* program, though it does not always give the shortest path, gives an efficient path that does not require much computing power. For the descriptions of the path finding algorithms, the map example being referred to is the following. The numbers in blue represent the distance from the end of point and the numbers in red represent the cost to travel between those two nodes.
2.4.2 Dijkstra Algorithm

The Dijkstra Algorithm works by creating a tree of the possible paths to get from the starting point to the ending point. The tree made for the example map would look like:

Figure 5 Dijkstra Algorithm Tree Path Example

![Dijkstra Algorithm Tree Path Example](image)

The number next to each node is the total cost to reach that node, which is calculated by adding the cost to travel between the node and the node it branches from to the total cost of the node it branches from. The tree adds every node that is next to it that has not already been checked as a branch. If a node has been checked elsewhere as well, the tree only checks the one with the smallest cost, because the following paths would be the same, but only the one with the smallest total cost can be in the smallest path. This is why node P2, 9 continues when P2, 15 does not. If they have the same value, then both are recognized continued. When the end node is detected for all possible paths, it searches for the end node with the smallest total cost. It then traces up the branches to the start node to find the shortest path.

Figure 6 Dijkstra Algorithm Smallest Path Example

![Dijkstra Algorithm Smallest Path Example](image)
The shortest path is therefore: Start » P5 » P2 » P4 » End.

### 2.4.3 A* Algorithm

The A* algorithm works by comparing two values of each space (node) that can be crossed. The first value is the H or Heuristic value, which is how far away the current node is from the end node. The second value is the G or Total Cost value, which is how much energy needs to be spent in order to reach the current node. The sum of these values is then added into the F value of each node. From the starting node, every node that can be crossed that is adjacent is added to an open list, and makes the start node the parent to all of those nodes. The algorithm then adds the starting node to a closed list, and checks for the node with the smallest F value on the open list. After that is found, each of the node’s surrounding nodes is added to the open list, unless it was already in the open list or the closed list. This process continues checking the node with the smallest F value in the closed list until the ending node is added to the open list. When the end node is added to the open list the loop closes and the final path is returned by tracing the parents of each node, starting with the end node.

Figure 7 A* Algorithm Path Finding Example

<table>
<thead>
<tr>
<th>Open List</th>
<th>Closed List</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1, 20</td>
<td>Start</td>
</tr>
<tr>
<td>P5, 19</td>
<td></td>
</tr>
<tr>
<td><strong>Open List</strong></td>
<td><strong>Closed List</strong></td>
</tr>
<tr>
<td>P1, 20</td>
<td>Start</td>
</tr>
<tr>
<td>P2, 15</td>
<td>P5, 19</td>
</tr>
<tr>
<td><strong>Open List</strong></td>
<td><strong>Closed List</strong></td>
</tr>
<tr>
<td>P1, 20</td>
<td>Start</td>
</tr>
<tr>
<td>P4, 17</td>
<td>P5, 19</td>
</tr>
<tr>
<td></td>
<td>P2, 15</td>
</tr>
<tr>
<td></td>
<td>P4, 17</td>
</tr>
</tbody>
</table>

A short path is therefore: Start » P5 » P2 » P4 » End.
Figure 8 A* Program in Action I

This is an example map that is read by the A* program.

This text file is the file our program makes and the main program reads from.

This Command Prompt is what actually goes on in the A* Path Finding search algorithm.

Figure 9 A* Program in Action II

This is an example map that is read by the A* program.

This text file is the file our program makes and the main program reads from.

This Command Prompt is what actually goes on in the A* Path Finding search algorithm.
Above are examples of the A* program in action. The top left white box is the text file that contains the map to be inputted into the program. The map will be able to accept the symbols “S” to denote the starting space, “E” to denote the end goal, “D” to denote a door, “P” to denote a picture, and “-” to denote a wall. Anything else will be noted as an empty space in the room. The command prompt on the right shows the process of the A* Path Finding Algorithm. The first part of the program shows the program’s interpretation of the uploaded map. Behind the scenes, the map is converted into an array of nodes, each with seven values: the x value, the y value, the type value, the parent value, the G value, the H value, and the F value.

The x and y values tell the program where the node is located in the new array. The type value simply tells the program if the node is a wall, a door, a picture, or simply an empty space. The parent value tells the program the sequence of nodes in the final path list. The G value is the total cost value to get from the start node to that node; it is found by adding either a value of 14 if the next node is in the diagonal or a value of 10 if the next node is horizontal or vertical to the G value of the previous node. The H value is the heuristic value (distance from the end node); it is found using the Manhattan Method, which is simply the sum of the difference between the x value of the current node and the x value of the end node and the difference between the y value of the current node and the y value of the end node. The F value is the sum of the G value and the H value; this is the value that the program mainly uses.

After the map is converted into an array of nodes, three lists of nodes are created: the open list, the closed list, and the path list. The open list is a list of nodes that can possibly be crossed to get to the end goal. The closed list is a list of nodes that have been checked. The path list is a list of nodes that make up the final path for the map. The program then adds the open list and starts a loop that continues until the open list is found to contain the ending node.

In the loop, the program sets the node with the lowest F value as the current node. It then adds all nodes adjacent to it to the open list and sets the current node as the parent of those nodes. Then the current node is removed from the open list and added to the closed list. The node with the lowest F value in the open list becomes the new open list and the loop continues.

When the loop breaks, the actual making of the path begins. This path is found using recursion and adding the parent to the path list until the starting node is added. After the path is made, the next major thing to do is to make the commands. These are made by finding where the path changes direction, and recording how long the path has gone without changing direction. Next the program makes a file with the command lines, where the main program reads from it and speaks the commands out loud using text to speech.
2.5 Canny Edge Detection

2.5.1 Background

Edge detection is a widely used tool in feature detection and extraction. In an image, a change in brightness would most likely be one of the following: change in depth, change in surface orientation (i.e. a corner of a table), change in material, or lighting. Which one of these the change in brightness actually can be determined by other programs that need to use the output of edge detection as their input. In our project, we first apply edge detection to the image from the Kinect, and then use the output of that for Hough transform algorithm. There are also many types of edge detection methods, and the one we use is Canny edge detection.

2.5.2 Filtering

Canny edge detection first filters out any noise (pixels that aren’t supposed to be there, but are there because of various disturbances) that the image might have. This is done by using a convolution algorithm that uses a Gaussian filter:

\[
K = \frac{1}{159} \begin{bmatrix}
2 & 4 & 5 & 4 & 2 \\
4 & 9 & 12 & 9 & 4 \\
5 & 12 & 15 & 12 & 5 \\
4 & 9 & 12 & 9 & 4 \\
2 & 4 & 5 & 4 & 2
\end{bmatrix}
\]

Example of possible Gaussian – filters noise by blurring the image

This is applied to every pixel in the image. For example, if the filter is applied to one pixel, you multiply the brightness of that pixel by 15, and multiply the pixels directly horizontal and vertical to it by 12. This goes outward until it is a 5x5 square. Then all of the resulting brightness values are added up and divided by 159, which is the total sum of the numbers in the filter. This basically averages out the brightness level of the pixel and the ones around it, and makes that pixel’s value closer to its neighbors’.

2.5.3 Image Gradient

An image gradient is the direction of the change in intensity or color of an image. The image gradient is found for all of the possible edges. This direction is rounded to one of four possible angles/directions (0, 45, 90, or 135), and is used in the next step.

\[
G = \sqrt{G_x^2 + G_y^2}
\]

\[
\Theta = \arctan \left( \frac{G_y}{G_x} \right).
\]
2.5.4 Non-Maximum Suppression

Canny edge detection determines if the gradient magnitude of all the pixels is close to a certain gradient direction. This removes pixels that are not part of any edge, so that only thin lines (possible edges) remain.

2.5.5 Hysteresis Threshold

Canny uses Hysteresis thresholding, which has two thresholds: upper and lower. These are applied to every pixel from the result of the previous steps. If the pixel value is higher than the upper threshold, then it is automatically considered an edge. If it is under the lower threshold, then it isn’t considered an edge. If the value is between the two thresholds, then it is accepted if it is next to a pixel whose value is above the upper threshold. A good upper to lower ratio is between 2:1 and 3:1. These threshold values can be changed by the user. If the upper threshold is set too high, pixels that are edges could be rejected. If it is set too low, then almost everything would be considered an edge.

2.6 Hough Transform

2.6.1 Background

Hough transform is a method used to extract features of an image that has already had edge detection applied to it. It has many uses, including object recognition in robot vision. Originally, the main purpose was to detect lines, but now it can identify shapes such as circles and ellipses. It was invented by Hough and published in improved version in 1972 by Richard Duda and Peter Hart.
2.6.2 Polar Coordinates

A simpler version of Hough transform uses the $y = mx + b$ equation to find lines, but the one we are going to use, polar coordinates, is more complicated. It uses parameters $r$ and $\theta$ instead of $m$ and $b$. To represent a line in this format, you find a perpendicular line that intersects the origin. The length of that line is $r$, and the angle of it from the x axis is $\theta$. The line is represented as $(r, \theta)$. For example, the following figure shows a line in polar coordinates:

![A Line in Polar Coordinates](image)

$$r_1/x = \cos \theta$$

$$r_2/y = \sin \theta$$

Hence $r = r_1 + r_2 = x \cos \theta + y \sin \theta$

2.6.3 Process of Detecting a Line

If the program has three points in a line, the way Hough transform detects the line is by the following process:

1. Take the first point and draw lines from all angles that go through the point
2. For each of those lines, find their polar coordinates
3. Do steps 1 and 2 for the other two points
This figure above shows lines being drawn from each point. The graphs at the bottom are the parameters (in polar coordinates) of the corresponding color’s line.

4. A Hough space graph is created of these polar coordinates, which basically is a line graph with three lines – one for each of the points. The point where the lines intersect corresponds to the parameters of the equation of the line.
The figure above show the point in parameter space where the lines intersect. This point corresponds to the polar coordinates of the line – (81, 60) in the original image.

2.6.4 Threshold

In the previous example, the threshold would be set as three, because the Hough transform program only used three points on the line. If you want the program to detect more defined lines, then you would set the threshold higher. If you set the threshold too low, for example two, then it would detect certain areas in the image that aren't lines. A typical threshold setting is around 40 pixels.
Figure 13 Example of Edge Detection and Hough Transform – Building Corridor 1 (order: original, edge detection, Hough transform)
Figure 14 Example of Edge Detection and Hough Transform – Door (order: original, edge detection, Hough transform)
Figure 15 Example of Edge Detection and Hough Transform – Building Corridor 2 (order: original, edge detection, Hough transform)
Figure 16 Example of Edge Detection and Hough Transform – Building Lounge (order: original, edge detection, Hough transform)
Figure 17 Example of Edge Detection and Hough Transform – Building Corridor 3 (order: original, edge detection, Hough transform)
2.7 Object Detection

2.7.1 Background

The purpose of object detection is to let the user know where various objects, such as doors and pictures, are in the room or corridor. It also prevents the user from crashing into walls or other objects. The program relies heavily on the Kinect depth sensing capabilities to locate where a specific object is, but relies on the color images captured by the Kinect to know what object the user is observing.

2.7.2 Door Detection

Door detection works with both the Kinect’s depth data and its color data. The Kinect first takes a still shot of what is being observed of both the depth data and the color data. With the color picture, the program runs the Canny Edge Detection and the Hough Transform to get all of the straight lines in the image. The program then filters out all lines that are not vertical and are less than 75% the size of the longest vertical line detected. The program then compares the lines that are left and if two lines are the correct distance apart, it is declared as a door. Using the midpoint of the average of the two lines, the program finds the location of the door (front, left, right) and the distance of the door (2.34 meters).

2.7.3 Obstacle Avoidance

Obstacle avoidance is a fast acting part of the system that tells the user to stop when it detects there is something too close to him or her. The obstacle avoidance works by testing 48 evenly spaced points on the Kinect Depth feed. If the program samples the feed and more than ten of the 48 points are closer than .8 meters, the program will warn the user to stop to prevent him from crashing into walls. The program needs ten points that are too close to activate to prevent the program from unnecessary warnings if the Kinect happens to have an error at one of the tested points. Ten points is around one-fifth of the total screen capture, so if there are ten points, the user can be certain that there is something he is about to walk into.

2.8 Path Following Detection

To detect if the user is correctly following the path or not, the program uses both the A* path finding program and the door detection algorithm. The A* program runs another function parallel to itself to ascertain the number of doors on each side the user should pass if he or she follows the path correctly. A modified version of the door detection is then run once every second to count the number of doors currently observed. If the number of doors on either side is lower than it was before, the program notes that a door has been passed. If the number of doors passed on either side is significantly greater than the number of doors supposed to be passed on that stretch of the path, the program notifies the user that he has strayed off that path and should locate a door to try to find a new path to his destination.
2.9 Viola Jones Algorithm

2.9.1 Background

The Viola-Jones algorithm was created by two people. It is now considered to be the best algorithm for face detection and a good algorithm for arbitrary object detection. The Viola-Jones algorithm in this project is used only for face detection. The way it works is it puts Haar Cascades, explained in the next section, over different parts of an image and tests the difference between dark and light in the image. A human face has many shadows and since there are so many shadows, the face can be detected by locating where the shadows exist in the image and if there are any. If the shadows all match, then a face has been detected.

2.9.2 Haar Cascades

Haar Cascades check whether the shadows of a part of an image correspond with the shadows of an actual human face. A Haar Cascade is a rectangle that has a black section and a white section. There is a variety in where the sections appear, such as there could be two of each section. The rectangle is placed over a part of an image that has been converted to gray scale. On the image, the pixels’ color is summed up in each section. To be more precise, all the pixels on the image that are under the white section are added up to see where it falls on the grayscale (on the lighter or darker side) and then the same is done for the black section. Where it falls on the grayscale corresponds with another number. The number from the black section is next subtracted from the number from the white section. This new number is compared with certain threshold that is stored in an XML file. If it lies in a certain threshold, the section of that image has passed and if didn’t then it failed.

Figure 18 An Example of a Haar Cascade

2.9.3 The Stages

The algorithm has stages of different Haar Cascades. The first few stages are basic shadows in the face such as: is there a forehead? Or are there cheeks? It is just getting the outline of the face. If all the Haar Cascades are a pass in this stage then it moves on to the next stage. If any of the Haar Cascades have failed then the stage fails and the detection moves on to a different part of the image (this is explained in the next section). The stages get more complicated as it goes on, focusing on much more specific parts of a traditional human face. If all the stages have passed then the image is a human face. [16]
2.9.4 Scanning

All of the stages happen in a red square, which is placed over different parts of the image. The way it works is the red square starts in the top left at a certain size and moves from left to right, checking whether the stages have passed or failed each time. Once the square has finished a whole row, then it goes down a bit and checks the next row. When the square has gone through the whole image by going through each of the rows, it starts at the top left again, but this time the square is bigger. It does this whole process about 5 times. When all the stages match then a red square is dropped where it was scanning and stays on the image, as the algorithm continues to scan the rest of the entire image. Once the entire process has finished, if there are 3 overlapping red squares that were dropped onto a part of the image, then that part is detected as a human face.
2.9.5 Adaboost

Adaboost is an algorithm, which helps to speed up the Viola-Jones algorithm, essentially boost it. There are a million possibilities in an image for the red square in the Viola-Jones algorithm to scan over, and the Adaboost helps to limit these. If one side of the red square passes a majority of the stages then Adaboost will steer the red square to the left or right rather than following the usual scanning.

The following is an example of the input and output of the Viola Jones algorithm.
2.10 Text Recognition

2.10.1 Background

The purpose of using text recognition in this system is to help the user locate where in the building he is. By using text recognition, the user can scan the sign on any labeled door in the building and the system will be able to recognize where on the map of the building the person is.

2.10.2 Deskewing Lines

A regular OCR system finds the percentage by which the text is skewed by and then deskews it. Tesseract does not deskew text but instead filters out blobs of text using \(x\)-coordinates and can then assign a unique text line to each blob. It estimates the baseline and filters out the blobs more. When blobs horizontally overlap by at least half, it then assigns diacritical marks with the correct line.

2.10.3 Determining Each Character

2.10.3.1 Fixed Pitch

Tesseract then determines if a word is fixed-pitch, which it does by putting a circumscribed rectangle around each character and then a vertical line down the center of each rectangle. It finds the width between each of these lines and makes a histogram of it. If the histogram has a centralized tendency then the word has a fixed pitch font.

![Figure 22 The process of determining if something is fixed pitch](image)

2.10.3.2 Non-Fixed Pitch

If the word is not fixed pitch, it measures the gap between the baseline and mean line, which were determined through a quadratic spline. If the space is too close to the threshold, then it is made fuzzy so that a decision can be made after word recognition.
2.10.4 Word Recognition

2.10.4.1 Non-Fixed Pitch

If the word is non-fixed pitch, it chops each character by finding the concave vertices and placing points there. If a point has either another point opposite to it or a line segment then it chops the word to separate each character. If it has tried all possible chop points then associates possible characters with the word using the concave vertices points. It creates a hash table of all the characters that it tried.

Figure 23 Finding the concave vertices points

2.10.4.2 Classifying Words

All words, whether fixed pitch or not have each character separated now. Tesseract uses a class pruner, which makes a shortlist of possible characters that the unknown word may match. For each possible character, it finds the similarity between the unknown word and the possible character. It picks the characters, which have the most similarity with the unknown word.

2.10.5 Location Prediction

After finding out what the sign has written on it, the program compares that text to a file that contains the list of rooms and their locations. If the room number sensed exists, the program will place the user’s current location as right outside of the room based on what the file says.

2.11 Text to Speech

2.11.1 Background

The purpose of using text to speech is to convert the information from the Kinect sensor into something that the user can actually comprehend. Because this system is used to assist a user who is blind, the module cannot return information visually. The other options would be to return the information in audio or tactile format. We decided that using auditory feedback would be the most convenient for the user. We decided to put all of the information into text format so we could later use a text to speech program to return the auditory information.
2.11.2 Microsoft Speech API (SAPI)

Microsoft SAPI is a part of the Microsoft OS that allows for speech recognition and synthesis. The current version (SAPI v5) was made in 2000 and used the OS since then. This version uses a runtime component called sapi.dll to interface between the applications that call it and the drivers in the computer that actually run the speech recognition and synthesis.

Figure 24 SAPI Interface

Figure above explains the way SAPI interfaces between the application and the drivers.

2.12 Shape-coded Keyboard Design

To help blind people interact with the system, we decided to use a shape-coded wireless keyboard. We bought various shape buttons from a fabric store and chose the ones that are easy to recognize by touching the buttons. We pre-programmed the IVB system User Interface software to utilize these shape coded buttons. The following is the decoding table for the various shape coded button used in the system.
Figure 25 Shape-coded Keyboard and Decoding Table

<table>
<thead>
<tr>
<th>Key</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Star</td>
</tr>
<tr>
<td>2</td>
<td>Donut</td>
</tr>
<tr>
<td>3</td>
<td>Football</td>
</tr>
<tr>
<td>4</td>
<td>Moon</td>
</tr>
<tr>
<td>5</td>
<td>Square</td>
</tr>
<tr>
<td>6</td>
<td>Pine Tree</td>
</tr>
<tr>
<td>7</td>
<td>Snow Flake</td>
</tr>
<tr>
<td>8</td>
<td>Teddy Bear</td>
</tr>
<tr>
<td>9</td>
<td>Embedded Flower</td>
</tr>
<tr>
<td>0</td>
<td>Circle with Dents</td>
</tr>
<tr>
<td>Enter</td>
<td>Curved Square</td>
</tr>
</tbody>
</table>
2.13 Software Flowchart of IVB User Interface

The figure below shows the software flowchart of IVB User Interface using the shape-coded keyboard.

Figure 26 Software Flowchart of IVB User Interface

This flowchart represents what our program does. When you start the program, the user can choose different options with the wireless shape-coded keyboard. To let the user know what to do, there is also a help option, which is the teddy bear button. When you press it, the program tells the user what each shape-coded button does. The other buttons are: advance, which tells the user where to go next; scan, which takes a picture and scans the area for human faces and doors; and exit, which closes the program.

When user uses the IVB system, he or she will use the wireless shape-coded keyboard and follow instructions given by the system. He or she will hear from the IVB system, “Starting up the IVB System. Press the teddy bear for help.”

By pressing the key that has donut shaped button on it, the IVB system will use the Kinect sensor to take a depth and color picture of what’s in the front. The IVB system then process the picture and search for object such as doors and people in front and it will give out warning such as door in the front, 1.0 meters or face in front.

By pressing the key that has circle with dents shaped button on it, the system will use A* search based path finding algorithm to find the right path for user for his or her destination and tells the user where to go next.
By pressing the teddy bear, the system says, “To enter a command, press command, then curved square. To Advance, press the circle with dents. To scan, press the donut. To exit, press the embedded flower. To listen to these commands again, press the teddy bear”

User can exit the IVB system by pressing the key that has the embedded flower button on it.

**3. Each Team Member’s Responsibility for This Project**

Clemen Deng is in charge of researching and designing the Edge Detection and Hough Transforms algorithms and programming, as well as the programming for text recognition. The program developed by Clemen detects objects in the building such as doors. He is also in charge of design and integration of the Kinect Sensor program to capture the RGB and depth sensing pictures of the surrounding area inside a building and pass it through the Edge Detection and Hough Transform program for object recognition and distance sending. Clemen is in charge of designing the hardware including connecting and mounting all the hardware devices (laptop, Kinect sensor, battery pack, shape coded keyboards, backpack with wood supports). In his programming for text recognition, he implemented the Tesseract OCR software to find text above doors in order to tell the user of the program what room the user is in front of.

Dina Zaslavsky is in charge of researching and programming the Viola Jones face detection algorithm and the research of text recognition. The program developed by Dina uses the picture taken by the Kinect sensor and applies the Viola Jones algorithm to identify the human face. The output of the program becomes one of the inputs of the Text-To-Speech program that Shadman designed to tell the blind person if there is a human standing in front of him or her. Dina also organized and consolidated all team meeting notes.

Shadman Abedin is in charge of the research and design of the Text-To-Speech speech synthesis program and the A* search based path finding algorithm as well as the obstacle avoidance system. Shadman is also in charge of researching on the Kinect sensor hardware. The Text-To-Speech program developed by Shadman takes the outputs of the edge detection and Hough Transform program that Clemen developed and the Viola Jones face detection program that Dina developed and outputs a human voice warning the blind person what’s ahead and directions to follow. The A* program developed by Shadman takes a map of the building and the location of the user to find the efficient path to the user’s destination. The obstacle avoidance system program uses the Kinect’s depth sensor to warn the user when he or she is too close to an obstacle, e.g. a wall.

All three team members collaborated closely with each other in order to integrate their part of the design into the overall design and worked together on testing the overall system.
4. **Hypothesis**
   We will be able to design a navigation and obstacle avoidance system using Kinect sensor to help blind people to navigate the known building.

5. **Variables**
   
   *Independent Variables:* All the IVB system hardware including the Kinect sensor, laptop computer, keyboard, backpack, and battery set, and all of the software including the overall program for the design.

   *Dependent Variables:* The accuracy of the data received from the Kinect, the accuracy of door detection and face detection, the accuracy of the A* algorithm, and the overall efficiency of the design.

   *Constants:* The same Kinect sensor, same laptop, same keyboard, same backpack, same battery set, same wooden support fixture to mount the Kinect sensor on the backpack, and same programming language.

6. **Materials**
   
   *Hardware:*
   - Microsoft KINECT for Windows module
   - Toshiba laptop computer
   - 12V Universal battery
   - Keyboard
   - Different shape buttons
   - Glue
   - Wooden beam for supporting Kinect
   - Velcro
   - Screws and nails
   - Mounting tools
   - Tape measure
   - DC to AC power converter

   *Software:*
   - Microsoft Visual Studio 2010
   - OpenCV software Version 2.3
   - KINECT Developer Tool Kit Version 1.6
   - Microsoft Speech API
   - OpenNI
   - Tesseract OCR Version 3.01
   - EmguCV
7. **Procedure**

1. Install Microsoft Visual Studio 2010 on target laptop computer.
2. Install OpenCV 2.3 on target laptop computer.
3. Install KINECT Developer Tool Kit Version 1.6 on target laptop computer.
4. Install OpenNI on target laptop computer.
5. Connect the KINECT sensor to target laptop computer.
6. Integrate Kinect RGB capture software program to enable KINECT sensor to take various testing pictures and save on target laptop computer.
7. Develop KINECT sensor depth sensing program.
8. Learn, integrate, and test the Edge Detection and Hough Transform software program using pictures from Step 6.
9. Develop an object recognition algorithm using output of Step 8 to detect doors in the target testing building.
10. Learn, integrate, and test the Viola Jones face detection software program using pictures from Step 6.
11. Develop an A* search based path finding software program.
12. Develop an interactive system with the wireless shape-coded keyboard using the Text To Speech algorithm and program.
13. Integrate into one program and test all software programs from Steps 6-12.
14. Prepare fixtures and wood supports for mounting the KINECT sensor on the backpack.
15. Mount the KINECT sensor on the backpack higher than head-height with the wooden supports.
16. Glue shape coded buttons on the wireless keyboard to be used for target laptop.
17. Prepare a fixture to enable target laptop to be in the backpack with the lid open.
18. Prepare a fixture for a mobile battery set attached to the backpack.
19. Put the laptop and mobile battery set inside the backpack.
20. Blind one of the project team members by putting a mask on his or her eyes and have this project team member wear the IVB System (the test system with backpack and KINECT, laptop) and hold the wireless shape coded keyboard.
21. Test the IVB system by having the blinded team member walk through corridors of the target building using commands from the program.
22. Analyze and record the testing results (please see Data Analysis section below for procedures we used to analyze the data/results that answer research question or hypothesis).
23. Repeat the entire experiment (Steps 20-22) three times with each team members.
8. **Data Analysis**

This section describes the procedures we used to analyze the data and results that answer research questions and the hypothesis.

1. Test the IVB System (Interactive Vision for Blind System) by blinding one of your project team members with a mask.

2. The blinded team member wears the IVB system and walks through various locations of the corridor of the first floor of the target test building.

3. When the blinded team member walks in front of a door, check to see if the system gives out a warning with a voice stating a door is in front and how many meters in front. The other un-blinded team member records the data the IVB system reports and compares with actual data by measuring the door distance to see if the system reports correctly.

4. When the blinded team member walks in front of a person(s), check to see if the system gives out a warning with a voice stating that there are people in front. The other un-blinded team member records the data the IVB system reports and compares with actual data to see if the system reports correctly.

5. Let the blinded team member push the button that makes the IVB system tell him/her what to do next to get to his/her destination, and record what the system says. The other un-blinded team member records the data the IVB system reports and compare with actual direction to see if the system reports correctly.

6. **Repeat steps 1-5 at least three times with the same team member**

7. **Repeat steps 1-6 with another two team members**
8.1 Door Detection Data Table and Analysis

Table 1 Door Detection Data Table

<table>
<thead>
<tr>
<th>Trial</th>
<th>IVB</th>
<th>Actual</th>
<th>Comments</th>
<th>Trial</th>
<th>IVB</th>
<th>Actual</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>14</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>15</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>16</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>17</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Yes</td>
<td>No</td>
<td>Detected object similar to door</td>
<td>18</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>19</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Yes</td>
<td>Close door</td>
<td>20</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>21</td>
<td>Yes</td>
<td>Yes</td>
<td>Wide door</td>
</tr>
<tr>
<td>9</td>
<td>Yes</td>
<td>Yes</td>
<td>Close door</td>
<td>22</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>Tough angle</td>
<td>23</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>24</td>
<td>No</td>
<td>Yes</td>
<td>Close door</td>
</tr>
<tr>
<td>12</td>
<td>Yes</td>
<td>Yes</td>
<td>Glass door</td>
<td>25</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Total amount correct: 23/25  Success Rate: 92%</strong></td>
</tr>
</tbody>
</table>

8.1.1 Door Detection Data Analysis

Test results show the door detection accuracy is 92%, detecting 23 out of 25 doors. It takes the IVB system less than 1 second to detect a door.

Door detection failed on doors out of the Kinect range (0.8 - 4m). It also doesn’t detect doors if only part of door is in the image, or if lines of door are too far apart (this could be due to being too close to the door). The program detects doors in front within a certain distance, and it sometimes detects other objects as doors if they have enough long vertical lines.

8.1.2 Improved Door Detection Algorithm

Initially, the door detection program was taking around thirty seconds to 1 minute per image to find and declare a door. Although the algorithm was accurate, but it took too long to run and we thought this was not practical for blind people. The reason it was so slow is because it took all the lines in the image and searched for every group of 3 lines and compared the angles and the ratios of their length. There are hundreds of lines in an image, so there are more than one million combinations of three lines. Each combination was being tested to see if it was a door, but that took way too long.

We modified and improved the door detection algorithm. The program was changed so that it only took long, vertical lines, and it compared every two of them to find the correct distance in between. Now the door detection program only takes one second or less. It is more practical and efficient.
8.2 Depth Sensing Data and Analysis

Table 2 Depth Data Table

<table>
<thead>
<tr>
<th>Door</th>
<th>Trial</th>
<th>Distance (m)</th>
<th>Distance IVB Detects (m)</th>
<th>Direction IVB Detects</th>
<th>Distance Error (m)</th>
<th>Percentage Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3.01</td>
<td>2.93</td>
<td>Front</td>
<td>0.08</td>
<td>2.66</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2.89</td>
<td>2.799</td>
<td>Front</td>
<td>0.091</td>
<td>3.15</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2.835</td>
<td>2.711</td>
<td>Front</td>
<td>0.124</td>
<td>4.37</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1.365</td>
<td>1.337</td>
<td>Front</td>
<td>0.028</td>
<td>2.05</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1.365</td>
<td>1.377</td>
<td>Front</td>
<td>-0.012</td>
<td>0.88</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1.29</td>
<td>1.298</td>
<td>Front</td>
<td>-0.008</td>
<td>0.62</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2.84</td>
<td>2.743</td>
<td>Front</td>
<td>0.097</td>
<td>3.42</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2.84</td>
<td>2.69</td>
<td>Right</td>
<td>0.15</td>
<td>5.28</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2.84</td>
<td>2.799</td>
<td>Front</td>
<td>0.041</td>
<td>1.44</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1.74</td>
<td>1.719</td>
<td>Front</td>
<td>0.021</td>
<td>1.21</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1.74</td>
<td>1.649</td>
<td>Front</td>
<td>0.091</td>
<td>5.23</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1.79</td>
<td>1.76</td>
<td>Front</td>
<td>0.03</td>
<td>1.68</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2.5</td>
<td>Farther than 4</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0.5</td>
<td>Farther than 4</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.061</td>
<td>2.67</td>
</tr>
</tbody>
</table>

Figure 27 IVB System Distance Sensing Test Results
8.2.1 Depth Sensing Data Analysis

The average accuracy of depth sensing is 97% with an average test error of 2.67% ranging from 0.88% to 5.28%. It is harder for the Kinect to report an accurate distance when the object is farther away or in a mirror. There were some faulty reports of depth when the object was out of the Kinect depth sensing range (0.8 – 4 meters). We found that if the object is farther than 4 meters or less than 0.8 meters, IVB will report no doors are detected. When the door is within the Kinect sensor depth sensing range, the reported distance is 97% accurate. The average distance error is 0.06 meters (range: -0.012 ~ 0.124 meters). A possible source of testing errors could be attributed to inaccurate measuring of the actual distance.

8.3 Face Detection Data and Analysis

Table 3 below is the IVB system face detection test data.

Table 3 Face Detection Data Table

<table>
<thead>
<tr>
<th>Test #</th>
<th>Target Test Member</th>
<th>Detected?</th>
<th>Actual Direction</th>
<th>IVB Detected Direction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Test member 1: Dina</td>
<td>yes</td>
<td>Near Left</td>
<td>Near Left</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Test member 1: Dina</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Test member 1: Dina</td>
<td>yes</td>
<td>Near Left</td>
<td>Near Left</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Test member 1: Dina</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Test member 1: Dina</td>
<td>yes</td>
<td>Near Right</td>
<td>Near Right</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Test member 1: Dina</td>
<td>no</td>
<td>Front</td>
<td>N/A</td>
<td>It was blurry</td>
</tr>
<tr>
<td>7</td>
<td>Test member 1: Dina</td>
<td>yes</td>
<td>Front Right</td>
<td>Front Right</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Test member 2: Shadman</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Test member 2: Shadman</td>
<td>no</td>
<td>Front</td>
<td>N/A</td>
<td>It was a profile or side view</td>
</tr>
<tr>
<td>10</td>
<td>Test member 2: Shadman</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Test member 2: Shadman</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Test member 2: Shadman</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Test member 2: Shadman</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Test member 2: Shadman</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Test member 3: Clemen</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td>It was taken from the bottom of the face</td>
</tr>
<tr>
<td>16</td>
<td>Test member 3: Clemen</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Test member 3: Clemen</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Test member 3: Clemen</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Test member 3: Clemen</td>
<td>yes</td>
<td>Front</td>
<td>Front</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>3 test members together: Dina, Shadman, Clemen</td>
<td>Partially (Yes: Dina, Clemen; No: Shadman)</td>
<td>Front</td>
<td>Front</td>
<td>Reason of not detecting Shadman: no contrast in face because of skin color and shadows.</td>
</tr>
</tbody>
</table>

# of test total 20  
# of test detected 18  
% of test detected 90%

8.3.1 Face Detection Data Table Analysis

90% of the time the Viola Jones face detection test works. It detects faces well from the front, near left front, near right front, and under. For the 10% that doesn’t work, it’s due to the contrast in the picture of the face not being high enough or the picture being taken from side.

IVB System test results shows that the Viola Jones algorithm works well in detecting faces in the front, but not well with faces in the side or blurry faces. Results show the algorithm serves our purpose, because the important faces will be in front of user.
8.4 A* Search-based Path Finding Algorithm Test Data and Analysis

Table 4 A* Search-based Path Finding Data Table

<table>
<thead>
<tr>
<th>Trials</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>#7</th>
<th>#8</th>
<th>#9</th>
<th>#10</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step Size (m)</td>
<td>0.68</td>
<td>0.68</td>
<td>0.68</td>
<td>0.68</td>
<td>0.66</td>
<td>0.66</td>
<td>0.66</td>
<td>0.66</td>
<td>0.66</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>1 Forward 9</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>2 Turn Left</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>3 Forward 3</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>fail</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>4 Open Door</td>
<td>fail</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>5 Forward 1</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>6 Turn Right</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>fail</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>7 Forward 1</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>8 Turn Left</td>
<td>pass</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>9 Forward 5</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>10 Turn Right</td>
<td>pass</td>
<td>fail</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>fail</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>11 Forward 8</td>
<td>fail</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>fail</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>12 Turn Left</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>13 Forward 2</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>14 Turn Right</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>15 Forward 5</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>16 Turn Left</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>17 Arrive at Destination</td>
<td>pass</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>Total Instructions Followed</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>15</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>Success Rate</td>
<td>88%</td>
<td>94%</td>
<td>94%</td>
<td>88%</td>
<td>88%</td>
<td>94%</td>
<td>94%</td>
<td>94%</td>
<td>94%</td>
<td>100%</td>
<td>93%</td>
</tr>
</tbody>
</table>
Figure 28 Team Member 1 Wearing and Testing IVB System

Figure 29 Team Member 2 Wearing and Testing IVB System
8.4.1 A* Search-based Path Finding Algorithm Data Analysis

Test results show the average accuracy of the A* search based path finding algorithm is 93%. Most of the steps were done correctly, and the errors were possibly due to our steps being different sizes, or due to our steps being smaller because we were blinded and being more cautious. The actual A* program works well, and all three of us got very close to the destination. When user uses this program, after a few tries he/she will probably get used to it and errors will become rare.
8.5 IVB System Key Building Block Test Accuracy Summary

The figure below summarizes the IVB System key building component accuracy based on test results.

Figure 31 IVB System Key Building Block Average Accuracy

The average accuracy ranges from 90%–97%. The accuracy level for door detection is 92%, the accuracy level for distance sensing is 97%, the accuracy level for face detection is 90%, and the A* searched based path finding accuracy is 93%.

Based on the test results, we conclude that our hypothesis was correct. We can build a navigation and obstacle avoidance interactive system using Kinect sensor to help blind people to navigate the known building.
9. Areas of Future Work

If we have more time, we would like to improve the IVB system design. We can add voice recognition so the user can interact with the system without the need to hold something at all times. The A* search based path finding algorithm could be improved by adding an automatic calibration to the user’s step sizes and through the implementation of a pedometer. To improve object detection so that it could detect windows and chairs. To improve the system design, we could replace the wooden frames with an expandable tripod to make the IVB system easier to carry. We could look into contacting Microsoft to design a battery pack for the Kinect instead of using the current 12V battery, which is very heavy.

10. Conclusion

We were able to build the prototype of the IVB System (Interactive Vision for Blind System). The IVB system can help blind people navigate from the start location to the destination location of a known building. The IVB system can detect doors in front with a distance sensing accuracy of 97% for any door between 0.8 – 4.0 meters in front of the Kinect. It can also detect people in front and gave out a warning. This is the first time this kind of navigation and obstacle avoidance interactive system was built to help blind people. The IVB System uses a Kinect sensor, a laptop computer, integration of a pre-programmed building map using the A* search based path finding algorithm, object recognition by using the Canny Edge detection and Hough Transform image processing algorithm, obstacle avoidance using the Kinect depth feed, location prediction with text recognition using the Tesseract OCR, face detection using the Viola Jones algorithm, speech synthesis using the Text To Speech algorithm, and a shape coded wireless keyboard. Test results show the test results of the various programs range from 90-97% accuracy.

Based on the test results, we conclude that our hypothesis was correct. We can build a navigation and obstacle avoidance interactive system using Kinect sensor to help blind people to navigate the known building.

11. Acknowledgements

We would like to thank Professor Marek Perkowski for teaching us image processing and for his guidance. We would also like to thank Professor K.C. Smith for giving feedback to our questions on how to make this system the most efficient it can be. We would also like to thank Dr. Maher Hawash for giving us guidance and our parents for their support making the meetings and this entire project possible.
12. Bibliography


[3] Learning OpenCV, Computer Vision with the OpenCV Library, by Gary Bradski & Adrian Kaehler


[7] OpenCV v2.4.3 documentation, OpenCV Tutorials, Canny Edge Detector

[8] Robot Vision and Perception class material, Chapter 4. Introduction to edge detection feature extraction, by Professor Marek A. Perkowski, Department of Electrical Engineering, Portland State University http://web.cecs.pdx.edu/~mperkows/CLASS_479/perception.html


http://opencv.willowgarage.com/wiki/FaceDetection

http://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework

http://makematics.com/research/viola-jones/

http://szeliski.org/Book/


http://www.amazon.com/Artificial-Intelligence-Modern-Approach-3rd/dp/0136042597/ref=sr_1_1?s=books&ie=UTF8&qid=1359846232&sr=1-1&keywords=norvig+artificial+intelligence+a+modern+approach

[20] Assistive Technology for Blindness and Low Vision (Rehabilitation Science in Practice Series), by Roberto Manduchi, Sri Kurniawan
http://www.amazon.com/Assistive-Technology-Blindness-Rehabilitation-Practice/dp/1439871531/ref=sr_1_sc_10?s=books&ie=UTF8&qid=1359846332&sr=1-10-spell&keywords=visio+for+blind

http://www.sciencemag.org/content/312/5777/1124.full

[22] Robot Vision and Perception class material, Chapter 1. Introduction to Morphological Operators, by Professor Marek A. Perkowski, Department of Electrical Engineering, Portland State University
http://web.cecs.pdx.edu/~mperkows/CLASS_479/perception.html

[23] Robot Vision and Perception class material, Chapter 20, Canny Edge Detection, by Professor Marek A. Perkowski, Department of Electrical Engineering, Portland State University
http://web.cecs.pdx.edu/~mperkows/CLASS_479/perception.html

[24] Robot Vision and Perception class material, Chapter 22, Edge Enhancement, by Professor Marek A. Perkowski, Department of Electrical Engineering, Portland State University
http://web.cecs.pdx.edu/~mperkows/CLASS_479/perception.html

http://web.cecs.pdx.edu/~mperkows/CLASS_479/perception.html


http://www.google.com/patents/US20130034302

[27] An Overview of the Tesseract Engine, by Ray Smith, Google Inc.


[28] Some background information about the Tesseract engine

http://en.wikipedia.org/wiki/Tesseract_(software)

[29] The location of the downloads for the Tesseract engine

https://code.google.com/p/tesseract-ocr/
APPENDIX – SOURCE CODE

TEXTTOSPEECH.H:
/*texttospeech.h contains the code using Microsoft Speech API to make the *computer actually talk to the user. */
#endif TEXTTOSPEECH_H
#define TEXTTOSPEECH_H

#include <sapi.h> // Contains the actual speech synthesis functions
#include <string>
#include <fstream>
#include <vector>
#include <iostream>
using namespace std;

vector<string> ripFile(string filename); // This function rips text from the text file into a vector of strings that will be used inside of the program.
wstring s2ws(const string& s); // This function converts the string into a wstring, which can be converted to be read using the speakFile function.
void speakFile(vector<string> list); // This function actually speaks the vector of strings ripped from the text file.
void tts(string filename);

wstring s2ws(const string& s){
    int len;
    int slength = (int)s.length()+1;
    len = MultiByteToWideChar(CP_ACP, 0, s.c_str(), slength, 0, 0);
    wchar_t* buf = new wchar_t[len];
    MultiByteToWideChar(CP_ACP, 0, s.c_str(), slength, buf, len);
    wstring r(buf);
    delete[] buf;
    return r;
}

vector<string> ripFile(string filename){
    vector<string> list;
    ifstream infile;
infile.open(filename);
    string STRING;
    if(infile){
        cout << "opened
";
        while(getline(infile, STRING)){
            list.push_back(STRING);
        }
        infile.close();
    } else {
        cout << "Unable to open file :(
";
    }

    return list;
}

void speakFile(vector<string> list){
    bool automatic = true;
    for(int i = 0; i < list.size(); i++){
        string as = list.at(i);
        wstring stemp = s2ws(as);
    }
}
LPCWSTR a = stemp.c_str();

ISpVoice * pVoice = NULL;
if (FAILED(::CoInitialize(NULL))){
    return;
}

HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL, IID_ISpVoice, (void**)&pVoice);
if (SUCCEEDED( hr )){
    hr = pVoice->Speak(a, 0, NULL);
    pVoice->Release();
    pVoice = NULL;
}
::CoUninitialize();

if (automatic == true){
    int cont = 0;
    cin >> cont;
    if (cont == 1){
        automatic = false;
    } else if (cont != 0){
        i = list.size();
    }
} else{
    Sleep(1000);
}

void speakString(string type){
    wstring stemp = s2ws(type);
    LPCWSTR a = stemp.c_str();

    ISpVoice * pVoice = NULL;
    if (FAILED(::CoInitialize(NULL))){
        return;
    }

    HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL, IID_ISpVoice, (void**)&pVoice);
    if (SUCCEEDED( hr )){
        hr = pVoice->Speak(a, 0, NULL);
        pVoice->Release();
        pVoice = NULL;
    }
    ::CoUninitialize();
}

void tts(string filename){
    vector<string> list;
    list = ripFile(filename);
    speakFile(list);
}

#endif
NODE.H:

/*node.h contains the code for node and leg classes, both of which are used in "the astar.h code."*/

#ifndef NODE_H
#define NODE_H

class Node{ // Base class for every point that can be crossed
public:
    Node(); // Makes a node
    Node(int newx, int newy, int newtype); // Makes a node with specific parameters
        int x; // The X value of the node
t        int y; // The Y value of the node
t        int type; // The type of node (If it can be crossed or not)
t        int g; // The G (Total cost) value of the node
t        int h; // The H (Heuristic / Distance from the end) value of the node
t        int f; // The F value, is the sum of the G and H value
        Node* parent; // The node that comes before the current node
};

Node::Node(){
    x = 0; // Default Parameters
    y = 0;
    type = 0;
    g = 0;
    h = 0;
    f = 0;
    parent = NULL;
}

Node::Node(int newx, int newy, int newtype){
    x = newx;
    y = newy;
    g = 0;
    h = 0;
    f = 0;
    type = newtype;
    parent = NULL;
}

class Leg{
public:
    Leg(Node* startleg, Node* endleg);
    Leg (Node* startleg);
    void changeEnd(Node* newendleg);
    void setDoorsRight(int newrightdoors);
    void setDoorsLeft(int newleftdoors);
    int getDoorsRight();
    int getDoorsLeft();
    int getStartX();
    int getStartY();
    int getEndX();
    int getEndY();
}
Node* getStartLeg();
Node* getEndLeg();

private:
    int rightdoors;
    int leftdoors;
    Node* startleg;
    Node* endleg;
    int startx;
    int starty;
    int endx;
    int endy;

};

Leg::Leg(Node* newstartleg, Node* newendleg){
    rightdoors = 0;
    leftdoors = 0;
    startleg = newstartleg;
    endleg = newendleg;
    startx = startleg->x;
    starty = startleg->y;
    endx = endleg->x;
    endy = endleg->y;
    if(startleg->x > endleg->x)
    {
        startx = endleg->x;
        endx = startleg->x;
    }
    if(startleg->y > endleg->x)
    {
        starty = endleg->y;
        endy = startleg->y;
    }
}

Leg::Leg(Node* newstartleg){
    rightdoors = 0;
    leftdoors = 0;
    startleg = newstartleg;
    endleg = NULL;
    startx = startleg->x;
    starty = startleg->y;
    endx = NULL;
    endy = NULL;
}

void Leg::changeEnd(Node* newendleg){
    endleg = newendleg;
    endx = endleg->x;
    endy = endleg->y;
    if(startleg->x > endleg->x)
    {
        startx = endleg->x;
        endx = startleg->x;
    }
    if(startleg->y > endleg->y)
    {
        starty = endleg->y;
    }
eny = startleg->y;

void Leg::setDoorsRight(int newrightdoors){
    rightdoors = newrightdoors;
}

void Leg::setDoorsLeft(int newleftdoors){
    leftdoors = newleftdoors;
}

int Leg::getDoorsRight(){
    return rightdoors;
}

int Leg::getDoorsLeft(){
    return leftdoors;
}

int Leg::getStartX(){
    return startx;
}

int Leg::getStartY(){
    return starty;
}

int Leg::getEndX(){
    return enx;
}

int Leg::getEndY(){
    return endy;
}

Node* Leg::getStartLeg(){
    return startleg;
}

Node* Leg::getEndLeg(){
    return endleg;
}

#endif
DOORCOUNT.H:
/*doorcount.h contains the code that counts how many doors the user has passed on *the leg of the path he is currently on. */
#include <iostream>
#include "highgui.h"
using namespace std;

int pastdoorsleft;
int pastdoorsright;

int main(){

    return 0;
}

void linedetection(Mat dst, Mat depthMap, Mat extra)
{
    int depthindexSt=0;
    int depthindexEnd=0;
    vector<Vec4i> list;
    HoughLinesP(dst, list, 1, CV_PI/180, 25, 120, 20 );
    //FILTER OUT LINES
    vector<Vec4i> lines = filterLines(list);

    vector<Vec4i> vlines;
    for(size_t i = 0; i<lines.size();i++)
    {
        Vec4i l = lines[i];
        if( isVertical(l) )
        {
            vlines.push_back(l);
        }
    }
    if( vlines.size() == 0 )
    {
        speakString("no lines detected");
        return;
    }
    std::sort (vlines.begin(), vlines.end(), sortfunc_);
    int sumx=0;
    int sumy=0;
    int sumx1=0;
    int sumy1=0;
    float depthunit =0;
    Vec4i last = vlines[0];
    int count = 0;
    for( size_t i = 1; i < vlines.size(); i++ )
    {
        Vec4i l = vlines[i];
        if(abs(l[0] - last[0])<100 && abs(l[0]-last[0])>10)
        {
            depthindexSt= ((l[1]+last[1])/2)*640 +((l[0]+last[0])/2);
            depthunit = (depth[depthindexEnd]-depth[depthindexSt])/2;
        }
    }
}
if (depthunit == 0)
{ std::cout << "Lines farther than 4 meters" << std::endl;
  return;
}
else
{
  std::cout << "depth of lines" << ": " << depthunit/1000 << "m " << std::endl;
 line(extra, Point(last[0], last[1]), Point(last[2], last[3]), Scalar(0, 0, 255), 1, CV_AA);
  line(extra, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 0, 255), 1, CV_AA);
  cvLine(clean, Point(last[0], last[1]), Point(last[2], last[3]), CV_RGB(250, 0, 0), 1);
  cvLine(clean, Point(l[0], l[1]), Point(l[2], l[3]), CV_RGB(250, 0, 0), 1);
  string slope = "Vertical";
  std::cout << "line " << i+1 << " : " << Point(last[0], last[1])<<"" <<Point(last[2], last[3])
    "slope: "<< slope "length: "<< sqrt(sqrtval) "m " "Door is farther than 4 meters" << std::endl;
  abs(last[2] - last[0]);
}

if (count>1)
{
  depthindexSt= sumy/count*640 +sumx/count;
  depthindexEnd= sumy1/count*640 +sumx1/count;
  depthunit = (depthdepthindexEnd+depthdepthindexSt)/2;
  if(depthunit==0)
  { std::cout << "Door is farther than 4 meters" << std::endl;
    return;
  }
  else
  { std::cout << "depth of average" << ": " << depthunit/1000 << "m " << std::endl;
  }
  cvLine(clean, Point(sumx/count,sumy/count), Point(sumx1/count,sumy1/count),CV_RGB(0,0,255), 1);
  int averagex = sumx/count;
  float avgdepth = depthunit/1000;
  cvLine(clean, Point(sumx/count,sumy/count), Point(sumx1/count,sumy1/count),CV_RGB(0,0,255), 1);
  std::cout << "lines: " << count << std::endl;
  ISpVoice * pVoice = NULL;
  if(SUCCEEDED( hr ) )
  {
    if(avgdepth>0)
    {
      if(averagex<214)
      {
        output1 = "There is a door in left, in ";
        output1.append(ss.str());
      }
    }
  }
  HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL, IID_ISpVoice, (void**)&pVoice);
  wstringstream output;
  string output1;
  stringstream ss;
  ss << avgdepth;
  if( SUCCEEDED( hr ) )
  {
    if(avgdepth>0)
    {
      if(averagex<214)
      {
        output1 = "There is a door in left, in ";
        output1.append(ss.str());
      }
    }
  }
if (averagex < 417) {
    output1 = "There is a door in front, in ";
    output1.append(ss.str());
    output1.append("meters");
    output << "There is a door in front, In ";
    output << avgdepth;
    output << "meters";
}
else if (averagex < 214) {
    output1 = "There is a door in left, farther than four meters";
    output << "There is a door in left, farther than four meters";
}
else if (averagex < 417) {
    output1 = "There is a door in front, farther than four meters";
    output << "There is a door in front, farther than four meters";
}
else {
    output1 = "There is a door in right, farther than four meters";
    output << "There is a door in right, farther than four meters";
}
wstring stemp = s2ws2(output1);
LPCWSTR talk = stemp.c_str();
hr = pVoice->Speak(talk, 0, NULL);
pVoice->Release();
pVoice = NULL;
}
ASTAR.H:
/*astar.h contains the code for the astar algorithm, which reads an input map *file (map.txt) and reads out the path to the user.
*/
#ifndef ASTAR_H
#define ASTAR_H

#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>
#include <cmath>
#include <Windows.h>
#include "node.h"
using namespace std;

const int RIGHT = 0;
const int LEFT = 1;

bool vecContain(vector<Node*> list, Node* item);
int vecIndex(vector<Node*> list, Node* item);
void getManhattan();
void setF();
void getNext();
void getSurrounding(Node* p);
void searchDoors(Leg* l, int dir);

vector<vector<Node>> array;
vector<Node*> openlist;
vector<Node*> closelist;
vector<Node*> path;
vector<Node*> doorslist;
vector<Leg*> legs;
vector<int> directions;
vector<int> doorsdirect;
int maxx = 0;
int maxy = 0;
Node* start;
Node* position = start;
Node* endnode;
int cury = 0;
int curx = 0;
bool broken;

int astar()
{
    vector<string> list;
    string STRING;
    ifstream infile;
    infile.open("map.txt");
    if(infile == NULL)
    {
        cout << "Could not find file.\n"
        Sleep(1000);
        return 0;
    }

    while(infile.eof() == false)
    {
        getline(infile, STRING);
        list.push_back(STRING);
    }

    // rest of the code...
}
longest = 0;
for (int i = 0; i < list.size(); i++) {
    string current = list[i];
    if (current.size() > list[longest].size())
        longest = i;
}

maxx = list.size();
maxy = list[longest].size();

array.resize(maxx);
for (int i = 0; i < maxx; i++) {
    array[i].resize(maxy);
}

for (int x = 0; x < maxx; x++) {
    for (int y = 0; y < maxy; y++) {
        array[x][y] = Node(x, y, 0);
    }
}

int startcount = 0;
int endcount = 0;
for (int i = 0; i < maxx; i++) {
    string cur = list[i];
    for (int j = 0; j < maxy; j++) {
        if (cur[j] == '-' || cur[j] == '+') {
            array[i][j].type = 1;
        } else if (cur[j] == 'S' || cur[j] == 's') {
            start = &array[i][j];
            startcount++;
        } else if (cur[j] == 'E' || cur[j] == 'e') {
            endnode = &array[i][j];
            endcount++;
        } else if (cur[j] == 'D' || cur[j] == 'd') {
            array[i][j].type = 2;
        } else if (cur[j] == 'P' || cur[j] == 'p') {
            array[i][j].type = 3;
        } else {
            array[i][j].type = 0;
        }
    }
}

curx = start->x;
cury = start->y;

for (int i = 0; i < maxy; i++) {
    cout << "--";
}
cout << endl;
for (int x=0; x < maxx; x++) {
    for (int y=0; y < maxy; y++) {

Node* p = &(array[x][y]);
if(p == start){
    cout << "S";
} else if(p == endnode){
    cout << "E";
} else if(p->type == 1){
    cout << "|";
} else if(p->type == 2){
    cout << "D";
} else if(p->type == 3){
    cout << "P";
} else {
    cout << " ";
}
}
cout << "\n";
for (int i = 0; i < maxy; i++) {
    cout << "--";
} cout << endl;
Node* p = start;
while (!broken && !vecContains(openlist, endnode)){
    try {
        p = &(array[curx][cury]);
        getSurrounding(p);
        openlist.erase(openlist.begin() + vecIndex(openlist, p));
        closelist.push_back(p);
        setF();
        getNext();
    } catch (int e) {
        broken = true;
        cout << "It broke :( " << endl;
    }
}

// Add the correct nodes to the path list
if(!broken){
    Node* lister = endnode;
    path.push_back(lister);
    while (lister != start) {
        lister = lister->parent;
        path.push_back(lister);
    }
}
cout << "Commands:\n";
Node* listed = path[path.size() - 1];
cout << "hello\n";
int prev = 2;
int onPath = 0;
directions.push_back(prev);
vector<Node*> revPath;
for(int i = 1; i <= path.size(); i++){
    revpath.push_back(path[path.size() - i]);
}
legs.push_back(new Leg(path[path.size() - 1]));
string output = "";
for (int i = path.size() - 1; i > 0; i--) {
    int tx = path[i]->x;
    int ty = path[i]->y;
    int fx = path[i-1]->x;
    int fy = path[i-1]->y;

    int nextdir = 2;
    if (tx - fx == -1) {
        nextdir = 2;
        //cout << "down\n";
    } else if (tx - fx == 1) {
        nextdir = 0;
        //cout << "up\n";
    } else if (ty - fy == -1) {
        nextdir = 1;
        //cout << "right\n";
    } else if (ty - fy == 1) {
        nextdir = 3;
        //cout << "left\n";
    }

directions.push_back(nextdir);

onpath++;

if(nextdir != prev || path[i]->type != 0){
    legs.back()->changeEnd(path[i]);
    searchDoors(legs.back(), prev);
    legs.push_back(new Leg(path[i]));
    cout << "forward " << (int)(onpath * 1.5);
    stringstream ss;
    ss << (int)(onpath * 1.5);
    output = output + "forward " + ss.str();
    if(((prev + 1) % 4 == nextdir){
        cout << "; turn right\n";
        output = output + "; turn right\n";
    } else if ((prev + 3) % 4 == nextdir) {
        cout << "; turn left\n";
        output = output + "; turn left\n";
    } else if ((prev + 2) % 4 == nextdir) {
        cout << "; turn 180 degrees\n";
        output = output + "; turn 180 degrees\n";
    }
}

if (path[i]->type == 2) {
    cout << "; go through door\n";
    output = output + "; go through door\n";
}
onpath = 0;
/*bool paintcross = false;
   if (tx > 0) {
       if (array[tx - 1][ty].type == 3) {
           paintcross = true;
       }
   }
   if (tx < maxx - 1) {
       if (array[tx + 1][ty].type == 3) {
           paintcross = true;
       }
   }
   if (ty > 0) {
       if (array[tx][ty - 1].type == 3) {
           paintcross = true;
       }
   }
   if (ty < maxy - 1) {
       if (array[tx][ty + 1].type == 3) {
           paintcross = true;
       }
   }
   if (paintcross == true) {
       cout << "forward " << onpath << "; cross the picture\n";
       stringstream ss;
       ss << (int)(onpath * 2.25);
       cout << "forward " << (int)(onpath * 2.25) << "; You have arrived\n";
       output = output + "forward " + ss.str() + "; cross the picture\n";
       onpath = 0;
   }*/

   prev = nextdir;
}
legs.back()->changeEnd(path[0]);
onpath++;
stringstream ss;
ss << (int)(onpath * 2.25);
cout << "forward " << (int)(onpath * 2.25) << "; You have arrived\n";
output = output + "forward " + ss.str() + "; You have arrived\n";
cout << endl;

// Show the ending path
for (int i = 0; i < maxy; i++) {
    cout << "--";
}
cout << endl;
for (int x=0; x < maxx; x++) {
    for (int y=0; y < maxy; y++){
        Node* p = &array[x][y];
        if (vecContain(path, p)) {
            cout << "X";
        } else if(p->type == 1){
            cout << "|";
        } else if(p->type == 3){
            cout << "P";
        } else {

}}

}
cout << "\n";
}

for (int i = 0; i < maxy; i++) {
    cout << "--";
}

ofstream newoutput;
newoutput.open("output.txt");
newoutput << output;
newoutput.close();

for(int i = 0; i < legs.size(); i++)
{
    cout << legs[i].getStartX() << "," << legs[i].getStartY() << " -- " << legs[i].getEndX() << "," << legs[i].getEndY() << " L: " << legs[i].getDoorsLeft() << " R:" << legs[i].getDoorsRight() << endl;
    //cout << "Leg " << i << " starts at (" << legs[i].getStartX() << "," << legs[i].getStartY() << ") and ends at (" << legs[i].getEndX() << "," << legs[i].getEndY() << ").\n";
    //cout << " This leg has " << legs[i].getDoorsRight() << " doors to the right and " << legs[i].getDoorsLeft() << " doors to the left.\n";
}
return 0;

bool vecContain(vector<Node*> list, Node* item){
    bool contains = false;  // Go through each item on the list
    for (int i = 0; i < list.size(); i++) {
        if (list[i] == item) {
            contains = true;
        }
    }
    return contains;
}

int vecIndex(vector<Node*> list, Node* item){
    int index = 0;  // Go through each item on the list
    for (int i = 0; i < list.size(); i++) {
        if (list[i] == item) {
            index = i;
        }
    }
    return index;
}

void getManhattan(){
    for (int x=0; x < maxx; x++) {  // Add the distance between the x values and the distances between the y values
        for (int y=0; y < maxy; y++){
            Node* p = &array[x][y];
            int newh = abs(endnode->x - x) + abs(endnode->y - y);
            p->h = newh;
        }
    }
}
void setF(){
    for (int x=0; x < maxx; x++) {
        for (int y=0; y < maxy; y++) {
            array[x][y].f = array[x][y].g + array[x][y].h;
        }
    }
}

void getNext(){
    try {
        int index = 0;
        int smallf = openlist[0]->f;
        for (int x = 0; x < openlist.size(); x++) {
            if (openlist[x]->f < smallf) {
                index = x;
                smallf = openlist[x]->f;
            }
        }
        curx = openlist[index]->x;
        // Change the node being checked to the node with the smallest F value in the openlist
        cury = openlist[index]->y;
    }
    catch (int e) {
        // If it does not work, there is no path possible
        cout << "There is no path!" << endl;
        broken = true;
    }
}

void getSurrounding(Node* p){
    int cummug = p->g;
    if (p->y > 0) {
        // Checks if the node to the left is eligible to be added to the openlist
        Node* n = &(array[p->x][p->y - 1]);
        if (n->type != 1 && !vecContain(openlist, n) && !vecContain(closelist, n) && n->parent == NULL && n->type != 3) {
            n->parent = p;
            n->g = cummug + 10;
            openlist.push_back(n);
        }
    }
    if (p->x > 0) {
        // Checks if the node above is eligible to be added to the openlist
        Node* n = &(array[p->x - 1][p->y]);
        if (n->type != 1 && !vecContain(openlist, n) && !vecContain(closelist, n) && n->parent == NULL && n->type != 3) {
            n->parent = p;
            n->g = cummug + 10;
            openlist.push_back(n);
        }
    }
    if (p->x + 1 < maxx) {
        // Checks if the node below is eligible to be added to the openlist
        Node* n = &(array[p->x + 1][p->y]);
    }
    if (p->y + 1 < maxy) {
        // Checks if the node right is eligible to be added to the openlist
        Node* n = &(array[p->x][p->y + 1]);
    }
}
if (n->type != 1 && !vecContain(openlist, n) && !vecContain(closelist, n) && n->parent == NULL && n->type != 3) {
    n->parent = p;
    n->g = cummug + 10;
    openlist.push_back(n);
}

if (p->y + 1 < maxy) {
    // Checks if the node to the right is eligible to be added to the openlist
    Node* n = &(array[p->x][p->y + 1]);
    if (n->type != 1 && !vecContain(openlist, n) && !vecContain(closelist, n) && n->parent == NULL && n->type != 3) {
        n->parent = p;
        n->g = cummug + 10;
        openlist.push_back(n);
    }
}

void searchDoors(Leg* l, int dir){
    int rightdoor = 0;
    int leftdoor = 0;
    cout << "Start: (" << l->getStartX() << "," << l->getStartY() << ")    End: (" << l->getEndX() << "," << l->getEndY() << ")    Dir: " << dir << "n";
    if(dir == 0){
        int maxthisy = l->getStartY();
        int minthisy = maxy - 1 - l->getStartY();
        if(maxthisy > 5){
            maxthisy = 5;
        }
        if(minthisy > 5){
            minthisy = 5;
        }
        cout << maxthisy << "," << minthisy << endl;
        for(int j = l->getStartX(); j <= l->getEndX(); j++) // Go through each node unit of the leg
        {
            for(int i = 1; i <= maxthisy; i++){ // Look upwards
                if(array[j][l->getStartY() - i].type == 1)
                {
                    i = maxthisy + 1;
                }
                else if(array[j][l->getStartY() - i].type == 2)
                {
                    leftdoor++;
                }
            }
            for(int i = 1; i <= minthisy; i++)
            {
                if(array[j][l->getStartY() + i].type == 1)
                {
                    i = minthisy + 1;
                }
                else if (array[j][l->getStartY() + i].type == 2)
                {
                }
            }
        }
    }
}
rightdoor++;
}
}
}

if(dir == 1)
{
    int maxthisx = l->getStartX();
    int minthisx = maxx - 1 - l->getStartX();
    if(maxthisx > 5)
    {
        maxthisx = 5;
    }
    if(minthisx > 5)
    {
        minthisx = 5;
    }
    cout << maxthisx << ", " << minthisx << endl;
    for(int j = l->getStartY(); j <= l->getEndY(); j++) // Go through each node unit of the leg
    {
        for(int i = 1; i <= maxthisx; i++) // Look upwards
        {
            if(array[l->getStartX() - i][j].type == 1)
            {
                i = maxthisx + 1;
            }
            else if(array[l->getStartX() - i][j].type == 2)
            {
                leftdoor++;
            }
        }
        for(int i = 1; i <= minthisx; i++)
        {
            if(array[l->getStartX() + i][j].type == 1)
            {
                i = minthisx + 1;
            }
            else if(array[l->getStartX() + i][j].type == 2)
            {
                rightdoor++;
            }
        }
    }
}

if(dir == 2)
{
    int maxthisy = l->getStartY();
    int minthisy = maxy - 1 - l->getStartY();
    if(maxthisy > 5)
    {
        maxthisy = 5;
    }
    if(minthisy > 5)
    {
        minthisy = 5;
    }
    cout << maxthisy << ", " << minthisy << endl;
    for(int j = l->getStartX(); j <= l->getEndX(); j++) // Go through each node unit of the leg
    {
        for(int i = 1; i <= maxthisy; i++) // Look upwards
        {
            if(array[l->getStartY() - i][j].type == 1)
            {
                i = maxthisy + 1;
            }
            else if(array[l->getStartY() - i][j].type == 2)
            {
                leftdoor++;
            }
        }
        for(int i = 1; i <= minthisy; i++)
        {
            if(array[l->getStartY() + i][j].type == 1)
            {
                i = minthisy + 1;
            }
            else if(array[l->getStartY() + i][j].type == 2)
            {
                rightdoor++;
            }
        }
    }
}
for(int i = 1; i <= maxthisy; i++) { // Look upwards
    if(array[i][l->getStartY() - i].type == 1)
    {
        i = maxthisy + 1;
    }
    else if(array[i][l->getStartY() - i].type == 2)
    {
        rightdoor++;
    }
}
for(int i = 1; i <= minthisy; i++)
{
    if(array[i][l->getStartY() + i].type == 1)
    {
        i = minthisy + 1;
    }
    else if (array[i][l->getStartY() + i].type == 2)
    {
        leftdoor++;
    }
}
}
}

if(dir == 3)
{
    int maxthisx = l->getStartX();
    int minthisx = maxx - 1 - l->getStartX();
    if(maxthisx > 5)
    {
        maxthisx = 5;
    }
    if(minthisx > 5)
    {
        minthisx = 5;
    }
    cout << maxthisx << ", " << minthisx << endl;
    for(int j = l->getStartY(); j <= l->getEndY(); j++) // Go through each node unit of the leg
    {
        for(int i = 1; i <= maxthisx; i++) { // Look upwards
            if(array[l->getStartX() - i][j].type == 1)
            {
                i = maxthisx + 1;
            }
            else if(array[l->getStartX() - i][j].type == 2)
            {
                rightdoor++;
            }
        }
        for(int i = 1; i <= minthisx; i++)
        {
            if(array[l->getStartX() + i][j].type == 1)
            {
                i = minthisx + 1;
            }
            else if (array[l->getStartX() + i][j].type == 2)
            {
                leftdoor++;
            }
        }
    }
}
cout << "L: " << leftdoor << ", R: " << rightdoor << endl;

l->setDoorsLeft(leftdoor);
l->setDoorsRight(rightdoor);

int getDoorsRight(int i)
{
    return legs[i]->getDoorsRight();
}

int getDoorsLeft(int i)
{
    return legs[i]->getDoorsLeft();
}

#endif
CONSOLETEST.H:
/*consoletest.h runs the tesseract ocr code. */
#include "stdafx.h"
#include <Windows.h>
#include <iostream>

void console()
{
    system("C:\Users\cdeng_2\Documents\Visual Studio 2010\Projects\OCR\Debug\OCR.exe "
            "C:\Users\cdeng_2\Documents\Visual Studio 2010\Projects\composite_rev4\composite\numbers.jpg"
            "C:\Users\cdeng_2\Documents\Visual Studio 2010\Projects\composite_rev4\composite\pic_text.txt" Bridele);"
SENSING.H:
/*sensing.h contains the code that uses the Kinect depth and color streams and *uses them to run the door detection, face detection, and obstacle avoidance *programs. */
//#include "stdafx.h"
#include "highgui.h"
#include "cv.h"
#include <cmath>
#include <iostream>
#include <string>
#include <fstream>
#include "opencv2\core\core.hpp"
#include "opencv2\imgproc\imgproc.hpp"
#include <time.h>
#include <functional>
#include <numeric>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <ctype.h>
#include <windows.h>
#include <sapi.h>
#include <vector>
#include <Ole2.h>
#include <NuiApi.h>
#include <NuiImageCamera.h>
#include <NuiSensor.h>
#include "consoletest.h"
#ifdef _EIC
#define WIN32
#endif
using namespace std;
using namespace cv;
// Kinect variables
HANDLE rgbStream; // The identifier of the Kinect's RGB Camera
INuiSensor* sensor1; // The Kinect sensor
HANDLE depthStream;
INuiSensor* sensor2;
IplImage * clean = cvCreateImage(cvSize(640, 480), IPL_DEPTH_8U, 4);
int * depth = new int[640*480];

int high_switch_value = 0;
int highInt = 0;
int low_switch_value = 0;
int lowInt = 0;

int runningright = 0;
int runningleft = 0;
int maxrightpass = 0;
int maxleftpass = 0;

// Create memory for calculations
static CvMemStorage* storage = 0;
CvCapture* capture = 0;
const char* input_name;
// Create a new Haar classifier
static CvHaarClassifierCascade* cascade = 0;

// this is an approximation, for FOV = 45 degrees, 640x480 pixels
double verticalGradesPerPixel = 40.0/480.0;
double horizontalGradesPerPixel = 53.0/640.0;
double headWidth = 0.12; // supposing head's width is 12 cm

// Used to check if doors are passed
int prevdoorsright = 0;
int prevdoorsleft = 0;
int doorsright = 0;
int doorsleft = 0;

// Function prototype for detecting and drawing an object from an image
void detect_and_draw( IplImage* image );

// Create a string that contains the cascade name
const char* cascade_name = "haarcascade_frontalface_alt.xml"; /* "haarcascade_profileface.xml";*/

bool depthcapture = false;
bool KinectOpen = false;

bool initKinect()
{
    // Get a working kinect sensor
    int numSensors;
    if (NuiGetSensorCount(&numSensors) < 0 || numSensors < 1)
        return false;
    if (NuiCreateSensorByIndex(0, &sensor1) < 0)
        return false;
    HRESULT hr = NuiInitialize(NUI_INITIALIZE_FLAG_USES_DEPTH |
                                NUI_INITIALIZE_FLAG_USES_COLOR);

    return true;
}

IplImage* getKinectData() {
    const NUI_IMAGE_FRAME * imageFrame = NULL;
    NUI_LOCKED_RECT LockedRect;
    IplImage * Kinectimg = cvCreateImage(cvSize(640, 480), IPL_DEPTH_8U, 4);
    HRESULT hr = NuiImageStreamOpen(NUI_IMAGE_TYPE_COLOR, // Depth camera or rgb camera?
                                    NUI_IMAGE_RESOLUTION_640x480, // Image resolution
                                    0, // Image stream flags, e.g. near mode
                                    2, // Number of frames to buffer
                                    NULL, // Event handle
                                    &rgbStream);

    hr = NuiImageStreamGetNextFrame(rgbStream, 0, &imageFrame);
    if (FAILED(hr))
    {
        cout << "Get Image Frame Failed" << endl;
        return Kinectimg;
    }

    INuiFrameTexture* texture = imageFrame->pFrameTexture;
texture->LockRect(0, &LockedRect, NULL, 0);
if (LockedRect.Pitch != 0)
{
    BYTE * pBuffer = (BYTE*) LockedRect.pBits;
    cvSetData(Kinectimg,(BYTE*) pBuffer, CV_AUTOSTEP);
}

texture->UnlockRect(0);
NuiImageStreamReleaseFrame(rgbStream, imageFrame);

return Kinectimg;

void getDepthKinectData()
{
    const NUI_IMAGE_FRAME * imageDepthFrame=NULL;
    NUI_LOCKED_RECT LockedRect;

    IplImage * Kinectdepthimg = cvCreateImage(cvSize(640, 480), IPL_DEPTH_8U, 4);
    HRESULT hr=NuiImageStreamOpen(
        NUI_IMAGE_TYPE_DEPTH, // Depth camera or rgb camera?
        NUI_IMAGE_RESOLUTION_640x480, // Image resolution
        0, // Image stream flags, e.g. near mode
        2, // Number of frames to buffer
        NULL, // Event handle
        &depthStream);

    hr = NuiImageStreamGetNextFrame(depthStream, 0, &imageDepthFrame);
    for(int i = 0; i < 640 * 480; i++)
        depth[i] = 1000;

    if (FAILED(hr))
    {
        cout << "Get Image Frame Failed" << endl;
        return ;
    }

    INuiFrameTexture* texture = imageDepthFrame->pFrameTexture;
    texture->LockRect(0, &LockedRect, NULL, 0);
    if (LockedRect.Pitch != 0)
    {
        USHORT * pBuff = (USHORT*) LockedRect.pBits;
        for (int i = 0; i < 640 * 480; i++)
            {
            BYTE index = pBuff[i] & 0x07;
            USHORT realDepth = (pBuff[i] & 0xFFF8) >> 3;
            depth[i]=realDepth;
            }
    }
    texture->UnlockRect(0);
    NuiImageStreamReleaseFrame(depthStream, imageDepthFrame);

    //NuiImageStreamReleaseFrame(depthStream, imageDepthFrame);/*
    //cvShowImage("depth image", Kinectdepthimg);
    return ;
}

void switch_callback_h( int position )
{
highInt = position;
}

void switch_callback_l(int position)
{
    lowInt = position;
}

void help()
{
    std::cout << "This program demonstrates line finding with the Hough transform.
    "Usage:"
    
    "/houghlines <image_name>, Default is pic1.bmp" << std::endl;
}

bool vertical(Vec4i line)
{
    if (abs(line[0] - line[2]) < 20)
    {
        return true;
    }
    else
    {
        return false;
    }
}

vector<Vec4i> filterLines(vector<Vec4i> lines)
{
    vector<Vec4i> newlist;
    vector<bool> used;
    used.resize(lines.size());
    for (int m = 0; m < used.size(); m++)
    {
        used.at(m) = false;
    }
    int max = lines.size();
    for (int j = 0; j < max; j++)
    {
        Vec4i a = lines[j];
        for (int h = 0; h < max; h++)
        {
            Vec4i b = lines[h];
            double numerator = a[3] - a[1];
            double denominator = a[2] - a[0];
            double numeratorb = b[3] - b[1];
            double denominatorb = b[2] - b[0];
            if (a != b)
            {
                if (denominator == 0 || denominatorb == 0)
                {
                    if (denominator > -3 && denominatorb < 3)
                    {
                        {
                                b[3] < (a[0] + a[2]) / 2 + 10)
                        {
                            
                        }
                    }
                }
            }
        }
    }
}
used.at(h) = true;
used.at(j) = true;
int value[6];
value[0] = abs(a[1] - a[3]);
value[1] = abs(a[1] - b[1]);
int biggest = 0;
int avepoint = (a[0] + a[2] + b[0] + b[2]) / 4;
for(int q = 0; q < 6; q++)
{
    if(value[q] > value[biggest])
    {
        biggest = q;
    }
}
Vec4i newlin;
if(biggest == 0){
    newlin = Vec4i(avepoint, a[1], avepoint, a[3]);
} else if(biggest == 1){
    newlin = Vec4i(avepoint, a[1], avepoint, b[1]);
} else if(biggest == 2){
    newlin = Vec4i(avepoint, a[1], avepoint, b[3]);
} else if(biggest == 3){
    newlin = Vec4i(avepoint, a[3], avepoint, b[1]);
} else if(biggest == 4){
    newlin = Vec4i(avepoint, a[3], avepoint, b[3]);
} else if(biggest == 5){
    newlin = Vec4i(avepoint, b[1], avepoint, b[3]);
}
newlist.push_back(newlin);
}
else
{
    //Horizontal lines
    double slopea = -1* numeratora / denominatora;
    double slopeb = -1* numeratorb / denomina
    if( slopea < 0.05 && slopea > -0.05 )
    {
        if( (b[0]<a[2]+50 && b[0]>a[0]-50 && b[1]<((a[1]+a[3])/2)+10
        {
            used.at(h) = true;
            used.at(j) = true;
            int values[6];
            values[0] = abs(a[0] - a[2]);
            
            
        }
values[1] = abs(a[0] - b[0]);
values[2] = abs(a[0] - b[2]);
values[3] = abs(a[2] - b[0]);
values[5] = abs(b[0] - b[2]);

int greatest = 0;
for(int m = 0; m < 6; m++)
{
    if(values[m] > values[greatest])
    {
        greatest = m;
    }
}
Vec4i newline;
if(greatest == 0){
    newline = Vec4i(a[0], avpoint, a[2], avpoint);
}
else if(greatest == 1){
    newline = Vec4i(a[0], avpoint, b[0], avpoint);
}
else if(greatest == 2){
    newline = Vec4i(a[0], avpoint, b[2], avpoint);
}
else if(greatest == 3){
    newline = Vec4i(a[2], avpoint, b[0], avpoint);
}
else if(greatest == 4){
    newline = Vec4i(a[2], avpoint, b[2], avpoint);
}
else if(greatest == 5){
    newline = Vec4i(b[0], avpoint, b[2], avpoint);
}
newlist.push_back(newline);

for(int pi = 0; pi < lines.size(); pi++)
{
    if(used.at(pi) == false)
    {
        newlist.push_back(lines[pi]);
    }
}
return newlist;

struct sortfunc
{
    bool operator()(Vec4i l1, Vec4i l2) const
    {
        return l1[0] < l2[0];
    }
} sortfunc_;
// Function to detect and draw any faces that is present in an image
void detect_and_draw( IplImage* img )
{
    int scale = 1;
    // Create a new image based on the input image
    IplImage* temp = cvCreateImage( cvSize(img->width/scale, img->height/scale), 8, 3 );

    // Create two points to represent the face locations
    CvPoint pt1, pt2;
    int i;

    // Clear the memory storage which was used before
    static CvScalar colors[] =
    {
        {{0,0,255}},
        {{0,128,255}},
        {{0,255,255}},
        {{0,255,0}},
        {{255,128,0}},
        {{255,255,0}},
        {{255,0,0}},
        {{255,0,255}}
    };

    IplImage* gray = cvCreateImage( cvSize(img->width, img->height), 8, 1 );
    IplImage* small_img = cvCreateImage( cvSize( cvRound( img->width/scale ), cvRound( img->height/scale ) ), 8, 1 );
    cvCvtColor( img, gray, CV_BGR2GRAY );
    cvResize( gray, small_img, CV_INTER_LINEAR );
    cvEqualizeHist( small_img, small_img );

    // Find whether the cascade is loaded, to find the faces. If yes, then:
    if( cascade )
    {
        double t = (double)cvGetTickCount();
        // There can be more than one face in an image. So create a growable sequence of faces.
        // Detect the objects and store them in the sequence
        CvSeq* faces = cvHaarDetectObjects( img, cascade, storage, 1.1, 2,
        CV_HAAR_DO_CANNY_PRUNING, cvSize(40, 40) );
        t = (double)cvGetTickCount() - t;
        printf("detection time = %gms\n", t/((double)cvGetTickFrequency() * 1000.));
        // Loop the number of faces found.
        for( i = 0; i < (faces ? faces->total : 0); i++ )
        {
            // Create a new rectangle for drawing the face
            CvRect* r = (CvRect*)cvGetSeqElem( faces, i );

            // Find the dimensions of the face, and scale it if necessary
            pt1.x = r->x*scale;
            pt2.x = (r->x+r->width)*scale;
            pt1.y = r->y*scale;
            pt2.y = (r->y+r->height)*scale;

            // Draw the rectangle in the input image
            cvRectangle( clean, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
            double x1 = r->x*scale;
            double x2 = (r->x+r->width)*scale;
            double angle = (r->width)*scale * horizontalGradesPerPixel * 3.141592654/180;
double headDist = (headWidth/2) / (tan(angle/2)); //in meters

double xAngle = ((img->width)/2.0 - (r->x+r->width*0.5)*scale) * horizontalGradesPerPixel * 3.141592654/180;
double headX = tan(xAngle) * headDist;
double yAngle = ((img->height)/2.0 - (r->y+r->height*0.5)*scale) * verticalGradesPerPixel * 3.141592654/180;
double headY = tan(yAngle) * headDist;

std::wstring s2ws2(const std::string& s)
{
    int len,
    slength = (int)s.length()+1;
    len = MultiByteToWideChar(CP_ACP, 0, s.c_str(), slength, 0, 0);
}

ISpVoice * pVoice = NULL;
if(FAILED(::CoInitialize(NULL)))
{
    HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL,
    IID_ISpVoice, (void**)&pVoice);
    if(SUCCEEDED(hr))
    {
        if(headX > 0.2)
        {
            hr = pVoice->Speak(L"Face in far left", 0, NULL);
        }
        else if(headX < -0.2)
        {
            hr = pVoice->Speak(L"Face in far right", 0, NULL);
        }
        else if(headX > -2 && headX < -1)
        {
            hr = pVoice->Speak(L"Face in near right", 0, NULL);
        }
        else if(headX < 2 && headX > -1)
        {
            hr = pVoice->Speak(L"Face in near left", 0, NULL);
        }
        else
        {
            hr = pVoice->Speak(L"Face in front", 0, NULL);
        }
        pVoice->Release();
        pVoice = NULL;
    }
::CoUninitialize();
}

// Release the temp image created.
cvReleaseImage( &temp );
cvReleaseImage( &small_img );
cvReleaseImage( &gray );

std::wstring s2ws2(const std::string& s)
wchar_t* buf = new wchar_t[len];
MultiByteToWideChar(CP_ACP, 0, s.c_str(), slength, buf, len);
std::wstring r(buf);
delete[] buf;
return r;
}

void linedetection(Mat dst, Mat depthMap, Mat extra, IplImage *ipl)
{
  int depthindexSt=0;
  int depthindexEnd=0;
  vector<Vec4i> list;
  HoughLinesP(dst, list, 1, CV_PI/180, 25, 120, 20);
  //FILTER OUT LINES
  vector<Vec4i> lines = filterLines(list);
  cvFlip(ipl, ipl, 1);
  vector<Vec4i> vlines;
  for(size_t i = 0; i<lines.size();i++)
  {
    Vec4i l = lines[i];
    if( (vertical(l)) )
    {
      vlines.push_back(l);
    }  
  }
  if(vlines.size() == 0)
  {
    speakString("no lines detected");
    return;
  }

  std::sort(vlines.begin(), vlines.end(), sortfunc_);
  int sumx=0;
  int sumy=0;
  int sumx1=0;
  int sumy1=0;
  float depthunit =0;
  Vec4i last = vlines[0];
  int count = 0;
  int middlex=0;
  int middley=0;
  int middlexx=0;
  int middleyy=0;
  IplImage *tmp = ipl;
  for( size_t i = 1; i < vlines.size(); i++ )
  {
    Vec4i l = vlines[i];
    if(abs(l[0]-last[0])<100 && abs(l[0]-last[0])>10)
    {
      depthindexSt= ((l[1]+last[1])/2)*640 +((l[0]+last[0])/2);
      depthunit = (depth[depthindexEnd]+depth[depthindexSt])/2;
      if(depthunit==0)
      {std::cout << "Lines farther than 4 meters" << std::endl;}
      else{std::cout << "depth of lines" << ": " << depthunit/1000 <<"m" << std::endl;}
      line(extra, Point(last[0], last[1]), Point(last[2], last[3]), Scalar(0,0,255), 1, CV_AA);
      line(extra, Point(l[0], l[1]), Point(l[2], l[3]), CV_RGB(250,0,0), 1);
      cvLine(clean,Point(last[0], last[1]), Point(last[2], last[3]),CV_RGB(250,0,0), 1);
      cvLine(clean,Point(l[0], l[1]), Point(l[2], l[3]), CV_RGB(250,0,0), 1);
    }
  }
}
string slope = "Vertical";
std::cout << "\n" << std::endl;
}
{
}
else if((l[1]<l[3]&&last[1]<last[3]){middley=[l[1]+last[1])/2;middlex=[l[0]+last[0])/2;}

middlex = middlex + 55;
middlex = middlex - 55;
middley = middley + 25;
middley = middley - 25;
cvSetImageROI(ipl, cvRect(middlex, middley, 110, 50));
tmp = cvCreateImage(cvGetSize(ipl),ipl->depth,ipl->nChannels);
cvCopy(ipl, tmp, NULL);
cvFlip( tmp, tmp, 1 );
cvSaveImage( "numbers.jpg",tmp);
cvResetImageROI(ipl);
cvFlip( ipl, ipl, 1 );
cvSaveImage( "test.jpg",ipl);
}
last = vlines[i];
}
std::cout << "number of lines: " << count << std::endl;
if(count>1)
{

depthindexSt= sumy/count*640 +sumx/count;
depthindexEnd= sumy1/count*640 +sumx1/count;
depthunit = (depth[depthindexEnd]+depth[depthindexSt])/2;
if(depthunit==0)
{std::cout << "Door is farther than 4 meters" << std::endl;}
else{std::cout << "depth of average" << ": " << depthunit/1000 <<"m " << std::endl;}
cvLine(clean,Point(sumx/count,sumy/count),Point(sumx1/count,sumy1/count),CV_RGB(0,0,250), 1);
}

int averagex = sumx/count;
float avgdepth = depthunit/1000;
cvLine(clean,Point(sumx/count,sumy/count),Point(sumx1/count,sumy1/count),CV_RGB(0,0,250), 1);
std::cout << "lines: " << count << std::endl;
ISpVoice * pVoice = NULL;
if(FAILED(::CoInitialize(NULL)))
{
    HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL, IID_ISpVoice, (void**)&pVoice);
    stringstream output;
    string output1;
    stringstream ss;
    ss << avgdepth;
    if( SUCCEEDED( hr ) )
    {
        if(avgdepth>0)
        {
            if(averagex<214)
            {
                output1 = "There is a door in left, in ";
                output1.append(ss.str());
                output1.append("meters");
                output << "There is a door in left, In ";
                output << avgdepth;
                output << "meters";
            }
        }
        else if(averagex<417)
        {
            output1 = "There is a door in front, in ";
            output1.append(ss.str());
            output1.append("meters");
            output << "There is a door in front, In ";
            output << avgdepth;
            output << "meters";
        }
        else
        {
            output1 = "There is a door in right, in ";
            output1.append(ss.str());
            output1.append("meters");
            output << "There is a door in right, In" ;
            output << avgdepth;
            output << "meters " ;
        }
    }
    wstring stemp = s2ws2(output1);
    LPCWSTR talk = stemp.c_str();
    hr = pVoice->Speak(talk, 0, NULL);
    pVoice->Release();
    pVoice = NULL;
}
else
{
    if(averagex<214)
    {
        output1 = "There is a door in left, farther than four meters";
    }
    else if(averagex<417)
    {
        output1 = "There is a door in front, farther than four meters";
    }
}
output << "There is a door in front, farther than four meters";
else
{
    output1 = "There is a door in right, farther than four meters";
    output << "There is a door in right, farther than four meters";
}
wstring stemp = s2ws2(output1);
LPCWSTR talk = stemp.c_str();
hr = pVoice->Speak(talk, 0, NULL);
pVoice->Release();
pVoice = NULL;
}
::CoUninitialize();
}

int senseRoom( int argc, char** argv )
{
    // Images to capture the frame from video or camera or from file
    IplImage *frame, *frame_copy=0;
    IplImage *depth_frame= 0;
    // Input file name for avi or image file.
    cascade_name = "C:\OpenCV2.3\opencv\data\haarcascades\haarcascade_frontalface_alt.xml";

    // Load the HaarClassifierCascade
    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );
    // Check whether the cascade has loaded successfully. Else report and error and quit
    if ( !cascade )
    {
        speakString("Could not load classifier cascade" );
        return -1;
    }

    // Find if the capture is loaded successfully or not.
    Mat depthMap;
    // If loaded successfully, then:
    if( capture || KinectOpen )
    {
        // Capture the frame and load it in IplImage
        if(KinectOpen)
        {
            frame = getKinectData();
            getDepthKinectData();
        }
        else
        {
            if( !cvGrabFrame( capture )
                return 0;
            frame = cvRetrieveFrame( capture );
        }
// If the frame does not exist, quit the loop
if (!frame)
    return 0;

// Allocate framecopy as the same size of the frame
if (!frame_copy)
    frame_copy = cvCreateImage( cvSize(frame->width, frame->height),
                                IPL_DEPTH_8U, frame->nChannels );

    // Check the origin of image. If top left, copy the image frame to frame_copy.
if( frame->origin == IPL_ORIGIN_TL )
    cvCopy( frame, frame_copy, 0 );
    // Else flip and copy the image
else
    cvFlip( frame, frame_copy, 0 );
    Mat extra(frame);
    Mat src(frame);

    Mat dst, cdst, final, src_gray;
    IplImage *ipl = frame;

    cvtColor( src, src_gray, CV_BGR2GRAY );
    blur( src_gray, dst, Size(3,3) );
    Canny(dst, dst, 20, 50, 3);

    linedetection(dst, depthMap, extra, ipl);
    // Call the function to detect and draw the face
    detect_and_draw( frame );

} return 0;

bool checkCollision()
{
    int total = 0;
    for(int i = 40; i < 640; i += 80){
        for(int j = 40; j < 480; j += 80){
            if(depth[i*j] < 850 && depth[i*j] > 100){
                total++;
            }
        }
    }
    if(total > 10)
    {
        return true;
    }
}

int senseCollision( int argc, char ** argv )
{
    // Images to capture the frame from video or camera or from file
    IplImage *frame, *frame_copy=0;
    IplImage *depth_frame= 0;

    // Input file name for avi or image file.
    cascade_name = "C:\OpenCV2.3\opencv\data\haarcascades\haarcascade_frontalface_alt.xml";

    // Load the HaarClassifierCascade
    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

    // Check whether the cascade has loaded successfully. Else report and error and quit
    if( !cascade )
{ 
    speakString("Could not load classifier cascade");
    return -1;
}

// Find if the capture is loaded successfully or not.
Mat depthMap;
// If loaded successfully, then:
if( capture || KinectOpen ) {
    // Capture the frame and load it in IplImage
    if(KinectOpen)
    {
        frame = getKinectData();
        getDepthKinectData();
    }
    else
    {
        if( !cvGrabFrame( capture )
            return 0;
        frame = cvRetrieveFrame( capture );
    }
    // If the frame does not exist, quit the loop
    if( !frame )
        return 0;
    // Allocate framecopy as the same size of the frame
    if( !frame_copy )
        //frame_copy = cvCreateImage( cvSize(frame->width,frame->height),
        //                            IPL_DEPTH_8U, frame->nChannels );
        // Check the origin of image. If top left, copy the image frame to frame_copy.
        //if( frame->origin == IPL_ORIGIN_TL )
        //cvCopy( frame, frame_copy, 0 );
        // Else flip and copy the image
        //else
        //cvFlip( frame, frame_copy, 0 );
        Mat extra(frame);
        Mat src(frame);
        Mat dst, cdst, final, src_gray;
        cvtColor( src, src_gray, CV_BGR2GRAY );
        blur( src_gray, dst, Size(3,3) );
        if( checkCollision() )
            speakString("Watch Out. There is something close in front of you.");
}
return 0;
}

int senseText(int argc, char** argv)
{
    // Images to capture the frame from video or camera or from file
    IplImage *frame, *frame_copy=0;
    IplImage *depth_frame=0;

    // Input file name for avi or image file.
    cascade_name = "C:\OpenCV2.3\opencv\data\haarcascades\haarcascade_frontalface_alt.xml";
}
// Load the HaarClassifierCascade
cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

// Check whether the cascade has loaded successfully. Else report and error and quit
if( !cascade )
{
    speakString("Could not load classifier cascade");
    return -1;
}

Mat depthMap;
// If loaded successfully, then:
if( capture || KinectOpen )
{
    // Capture the frame and load it in IplImage
    if(KinectOpen)
    {
        frame = getKinectData();
        getDepthKinectData();
    }
    else
    {
        if( !cvGrabFrame( capture )
            return 0;
        frame = cvRetrieveFrame( capture );
    }

    // If the frame does not exist, quit the loop
    if( !frame )
        return 0;

    // Allocate framecopy as the same size of the frame
    if( !frame_copy )
        frame_copy = cvCreateImage( cvSize(frame->width,frame->height),
                                    IPL_DEPTH_8U, frame->nChannels );

    /// Check the origin of image. If top left, copy the image frame to frame_copy.
    if( frame->origin == IPL_ORIGIN_TL )
        cvCopy( frame, frame_copy, 0 );
/// Else flip and copy the image
//else
    //cvFlip( frame, frame_copy, 1 );
    Mat Tesserac(frame);
    Mat extra(frame);
    Mat src(frame);
    //std::ofstream newfile;
    //newfile.open("textpic.jpg");
//    newfile << frame;
//    newfile.close();
    Mat dst, cdst, final, src_gray;
    cvtColor( src, src, CV_BGR2GRAY );
    //blur( src, src, Size(3,3) );
cv::flip(src, src, 1);
imwrite("test.jpg", src);
}

console();
// return 0 to indicate successful execution of the program
return 0;
void doorCount(Mat dst, Mat depthMap, Mat extra)
{
    doorsleft = doorsright = 0;
    vector<Vec4i> list;
    HoughLinesP(dst, list, 1, CV_PI/180, 25, 120, 20);
    //FILTER OUT LINES AND CHECK WHICH LINES ARE VERTICAL
    vector<Vec4i> lines = filterLines(list);
    vector<Vec4i> vlines;
    for(size_t i = 0; i<lines.size();i++)
    {
        Vec4i l = lines[i];
        if(vertical(l))
        {
            vlines.push_back(l);
        }
    }
    if(vlines.size() == 0)
    {
        return;
    }
    std::sort (vlines.begin(), vlines.end(), sortfunc_);
    int sumx=0;
    int sumy=0;
    int sumx1=0;
    int sumy1=0;
    float depthunit =0;
    Vec4i last = vlines[0];
    int count = 0;
    for( size_t i = 1; i < vlines.size(); i++ )
    {
        Vec4i l = vlines[i];
        if(abs(l[0] - last[0]<100 && abs(l[0]-last[0])>10)
        {
            line( extra, Point(last[0], last[1]), Point(last[2], last[3]), Scalar(0,0,255), 1,
            CV_AA);
            line( extra, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 1, CV_AA);
            cvLine(clean,Point(last[0], last[1]), Point(last[2], last[3]),CV_RGB(250,0,0),
            1);
            cvLine(clean,Point(l[0], l[1]), Point(l[2], l[3]), CV_RGB(250,0,0), 1);
            string slope = "Vertical";
            * abs(last[2] - last[0]);
            std::cout << "line " << i+1 << ": " << Point(last[0], last[1])<<" ": " << Point(last[2],
            last[3]) << " slope: " << slope << " length: " << sqrt(sqrtval) << std::endl << "n" << std::endl;
            std::cout << "line " << i+1 << ": " << Point(l[0], l[1])<<" ": " << Point(l[2],
            l[3]) << " slope: " << slope << " length: " << sqrt(sqrtv) << std::endl << "n" << std::endl;
            sumx=sumx+l[0]+last[0];
            sumy=sumy+l[1]+last[1];
            sumx1=sumx1+l[2]+last[2];
            sumy1=sumy1+l[3]+last[3];
            count++;
            count++;
        }
        last = vlines[i];
    }
    std::cout << "number of lines: " " count << std::endl;
    if(count>2)
    {
        return;
    
    }
}
countDoors( int argc, char** argv )
{
    // Images to capture the frame from video or camera or from file
    IplImage *frame, *frame_copy=0;
    // Load the HaarClassifierCascade
    cascade_name = "C:\OpenCV2.3\opencv\data\haarcascades\haarcascade_frontalface_alt.xml";
    // Check whether the cascade has loaded successfully. Else report and error and quit
    if( !cascade )
    {
        speakString("Could not load classifier cascade");
        return -1;
    }
    // Find if the capture is loaded successfully or not.
    Mat depthMap;
    // If loaded succesfully, then:
    if( capture || KinectOpen )
    {
        // Capture the frame and load it in IplImage
        if(KinectOpen)
        {
            frame = getKinectData();
        }
        else
        {
            frame = cvCaptureFromCAM(0);
        }
        // Images from file, use IplImage!
    }
    // Images to capture the frame from video or camera or from file
    // Images to capture the frame from video or camera or from file
    IplImage *frame, *frame_copy=0;
    // Input file name for avi or image file.
    cascade_name = "C:\OpenCV2.3\opencv\data\haarcascades\haarcascade_frontalface_alt.xml";
    // Load the HaarClassifierCascade
    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );
    // Check whether the cascade has loaded successfully. Else report and error and quit
    if( !cascade )
    {
        speakString("Could not load classifier cascade");
        return -1;
    }
    // Find if the capture is loaded successfully or not.
    Mat depthMap;
    // If loaded succesfully, then:
    if( capture || KinectOpen )
    {
        // Capture the frame and load it in IplImage
        if(KinectOpen)
        {
            frame = getKinectData();
        }
        else
        {
            frame = cvCaptureFromCAM(0);
        }
        // Images from file, use IplImage!
    }
getDepthKinectData();

else
{
    if( !cvGrabFrame( capture ))
        return 0;
    frame = cvRetrieveFrame( capture );
}

// If the frame does not exist, quit the loop
if( !frame )
    return 0;

// Allocate framecopy as the same size of the frame
if( !frame_copy )
    frame_copy = cvCreateImage( cvSize(frame->width,frame->height),
            IPL_DEPTH_8U, frame->nChannels );

///// Check the origin of image. If top left, copy the image frame to frame_copy.
if( frame->origin == IPL_ORIGIN_TL )
    cvCopy( frame, frame_copy, 0 );
///// Else flip and copy the image
else
    cvFlip( frame, frame_copy, 0 );

Mat extra(frame);
Mat src(frame);
Mat dst, cdst, final, src_gray;
cvtColor( src, src_gray, CV_BGR2GRAY );
blur( src_gray, dst, Size(3,3) );
Canny(dst, dst, 20, 50, 3);
doorCount(dst, depthMap, extra);

if(prevdoorsleft > doorsleft)
{
    std::cout << "Passed " << prevdoorsleft - doorsleft << " doors to the left";
}
if(prevdoorsright > doorsright)
{
    std::cout << "Passed " << prevdoorsright - doorsright << " doors to the left";
}

prevdoorsleft = doorsleft;
prevdoorsright = doorsright;

return 0;
}

void startNewLeg(int newmaxright, int newmaxleft){
    maxrightpass = newmaxright;
    maxleftpass = newmaxleft;
    runningright = 0;
    runningleft = 0;
}
COMPOSITE.CPP:
/*composite.cpp is the main code of the program, running the entire system on a *loop and allowing use of the wireless keyboard. */
#include "stdafx.h"
#include "astar.h"
#include "texttospeech.h"
#include "sensing.h"
#include <Windows.h>
#include <WinUser.h>
#include <tchar.h>
#include <strsafe.h>
#define PRESSED 0x8000

void speakHelp();

int main( int argc, char** argv )
{
    // Allocate the memory storage
    storage = cvCreateMemStorage(0);
    // Find whether to detect the object from file or from camera.
    if( !input_name || (isdigit(input_name[0]) && input_name[1] == '\0') )
    {
        KinectOpen = initKinect();
    }
    else
    {capture = cvCaptureFromAVI( input_name );
        vector<string> list;
        speakString("Starting up IVB system.");
        speakString("Press the teddy bear for help");
        astar();
        speakString("Calculating route");
        list = ripFile("output.txt");
        int i = 0;
        bool running = true;
        int prevTickCount = getTickCount();
        int timePassed = getTickCount() - prevTickCount;
        while(running){
            if(GetKeyState(0x60) & PRESSED || GetKeyState(0x62) & PRESSED || GetKeyState(0x68) & PRESSED || GetKeyState(0x69) & PRESSED || GetKeyState(0x67) & PRESSED || GetKeyState(0x66) & PRESSED){
                if(GetKeyState(0x60) & PRESSED){
                    if(i < list.size())
                        speakString(list[i]);
                    i++;
                    if(i < list.size())
                        startNewLeg(getDoorsRight(i), getDoorsLeft(i));
                } else if(GetKeyState(0x69) & PRESSED){
                    speakString("Do you want to quit? Hold down to confirm.");
                    Sleep(2000);
                    if(GetKeyState(0x69) & PRESSED){
                        speakString("Quitting route.");
                        running = false;
                    } else{
                        speakString("Canceled exit.");
                    }
                }
            }
        }
    }
}
else if(GetKeyState(0x62) & PRESSED){
    speakString("Scanning");
    if(! KinectOpen )
    {
        capture = cvCaptureFromCAM( !input_name ? 0 : input_name[0] - '1');
    }
    senseRoom(argc, argv);
} else if(GetKeyState(0x68) & PRESSED){
    speakHelp();
} else if(GetKeyState(0x66) & PRESSED){
    senseText(argc, argv);
} else if(GetKeyState(0x67) & PRESSED){
    countDoors( argc, argv );
}
Sleep(1000);  
else {
    senseCollision(argc, argv);
    int tempTick = getTickCount();
    int timepassed = tempTick - prevTickCount;  
    if(timepassed >= 10000000)
    {
        countDoors(argc, argv);
        prevTickCount = getTickCount();
    }
}
}  

cvReleaseCapture( &capture );
    cvClearMemStorage( storage );
    return 0;
}

void speakHelp(){
    speakString("To enter a command, press the command button and then the curved square. To advance to the next step, press the sercle with dents. To exit the current route, press the embedded flower pattern. To scan the room, press the donut. To listen to these commands again, press the teddy bear.");
}
OCR.CPP:

/*OCR.cpp is the program that runs the text detection using the Tesseract OCR *system. */
#include "stdafx.h"
#include <baseapi.h>
#include <allheaders.h>
#include <iostream>
#include <fstream>
using namespace std;

int main(int argc, char** argv)
{
    tesseract::TessBaseAPI api;
    api.Init("", "eng", tesseract::OEM_DEFAULT);
    api.SetPageSegMode(static_cast<tesseract::PageSegMode>(7));
    api.SetOutputName("out");
    cout<<"File name:";
    char * image;
    if(argc < 1){ return -1; }
    image = argv[1];
    PIX *pixs = pixRead(image);
    STRING text_out;
    api.ProcessPages(image, NULL, 0, &text_out);
    char * output_file;
    output_file = argv[2];
    std::cout<<text_out.string();

    Boxa* bounds = api.GetWords(NULL);
l_int32 count = bounds->n;
    for(int i=0; i<count; i++)
    {
        Box* b = bounds->box[i];
        int x = b->x;
        int y = b->y;
        int w = b->w;
        int h = b->h;
        cout<<x<<" "<<y<<" "<<w<<" "<<h<<endl;
    }
    ofstream newfile;
    newfile.open(output_file);
    newfile << text_out.string();
    newfile.close();
    return 0;
}