Obstacle Detection Using The Kinect Sensor

Amardeep Singh Chawla
Shikher Sitoke
{amchawla, sitoke} @cs.umn.edu
Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55414

Abstract - The aim of this paper is to use Microsoft's Kinect™ sensor to detect obstacles. The data from its color sensor and the infrared sensor is collectively analyzed to get a good picture of what the surrounding is with respect to the present position of the Kinect and thus detect the obstacle.

I. INTRODUCTION

In a dynamic environment, obstacle detection is of prime importance for a robot to accomplish its tasks. This can be achieved using computer vision methods and camera systems play a vital role in this. Various techniques are implemented using stereo or multiple cameras that aid in a 3D surface reconstruction; however they introduce some level of noise.

Obstacle detection presents a complex problem due to the randomness of the shape of the objects in the environment and the perception of depth. A recent advancement in the field of computer vision has been made by Microsoft in the form of the Kinect – a device which has two types of cameras which helps it understand the picture in front and gets an idea about the depth at the same time.

The idea is to collect data from the two Kinect cameras and make a decision of whether or not there is an obstacle in front. This could then be mounted on an autonomous robot and the Kinect informs the robot when it reaches a critical distance to an obstacle in its path. Using this feedback, the robot can impede its on-going activity of moving towards the obstacle and can either change its path or come to a halt.

The autonomous robot can thus be used for various applications. It can be used in locations where there is risk to human life and we do not have the expertise to control a human driven robot in an emergency situation. It can also be used in the household for vacuuming a carpet where it can detect and avoid objects along its path.

The paper has been described as follows: Section II describes the problem, explains the technical capabilities of the Kinect and how we implemented this project. Section III deals with the implementation work involving Depth calculation and Depth Interpretation to detect obstacles using the Kinect. Section IV deals with ongoing work in the field of computer vision and object detection using depth. Section V deals with the issues faced during the implementation of the Kinect.

II. PROBLEM DESCRIPTION

The Kinect sensor has an Infrared sensor that helps it to perceive depth by using infrared vision. The other camera is a mono vision color sensing camera, the data of which can be processed by an image processing system.

The project deals with reading data from the Kinect when it faces an obstacle. Manipulation of the data from both sensors is required in order to keep a track of the obstacle. The basic concept is that as an obstacle gets closer, the value of the depth data from the infrared camera decreases. This can then be used on a robot and thresholding the depth parameter, a decision whether we should trigger commands to move the robot around the obstacle or continue on its own path can be taken.

Certain open source software packages are available, from "LibFreenect" for Linux machines, Microsoft's Kinect SDK using the Kinect namespace on Visual Studio 2010 Express (C++, C# or Visual Basic), and Google's released Robotic Operating System (ROS). We shall be using the Microsoft's Kinect SDK using the Kinect namespace on Visual Studio 2010 primarily for the native support we would get of the Microsoft-built device on the Windows platform.

III. IMPLEMENTATION

1. Calculating Depth

The IR camera of the Kinect is responsible for returning a continuous set of frames at the rate of over 20fps. The method adopted involves processing each of these image frames and captures the gray scale value of each of the pixel.

Each pixel is represented by 8 bits resulting in a possible set of 2^8 = 256 grayscale values. If a pixel is completely saturated, i.e. the pixel is completely white in color, the Kinect returns the value 0. If a pixel is completely black, it returns the value 255. Each of the pixel returns a value between 0 and 255 based on the

distance of that particular point in the image from the camera. The frame size is set to 480x320 pixels so as to have a total of 480*320 = 153,600 pixels that give the resultant depth information.

2. Interpreting Depth Information

If a point on the image is close to the camera, the pixel value returned is closer to 0 than to 255. Likewise, if the point on the image is far, it is likely to be gray in color with a value closer to 255.

Suppose there is only a single obstacle in the entire frame, the distance of the obstacle can be calculated by calculating the average gray value of the pixels. This average can be calculated as:

$$\boldsymbol{Avg} = \frac{\sum_{p=0}^{153600} \boldsymbol{G(p)}}{\mathbf{153600}}$$

Below are the calculations of the average gray values for a few scenarios. When there is no obstacle nearby, the value computed was expectedly high, as shown in Figure 1.

Figure 1: Depth image with no object in the vicinity.

Introducing an object in between, the value reduces as the image becomes darker overall as shown in Figure 2.

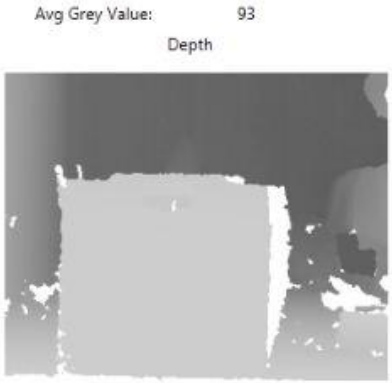

Figure 2: Depth image with object closer shows reduced gray value.

Bringing the object closer further decreases the value as shown in Figure 3. The gray value reduces drastically in this case.

Figure 3: Depth image with object even closer.

As the obstacle in the frame becomes closer, its average depth value reduces and as the value decreases it is clear that the obstacle is being approached. The technique of calculating thresholds can be modified by calibrating the instrument and scaling up the range of gray values so as to get a greater resolution.

C. Elevation angle of the camera

By changing the elevation angle of the Kinect camera, user can get an adjustable field of view of the concerned object whose depth images are captured and sent for further processing. This is an important step in the process as this can help us in distinguishing near objects from far.

The basic principle behind the Kinect depth sensor is emission of an IR pattern and the simultaneous image capturing with a CMOS camera, fitted with an IR-pass filter. The depth sensor has an infrared projector which captures 3D data under ambient light condition. The range of sensing data can be adjusted using the libraries provided by Microsoft. There is however a threshold level for the vertical and Horizontal view of the Kinect. The Kinect sensor can also be adjusted to physical tilt range but care needs to be taken regarding its threshold values which lie between -27 and +27 degrees.

These values, through the Microsoft open source library, can be altered by using an inbuilt property called the "Elevation Angle". The UI allows user to alter the value of this property. The code written for this paper makes use of a constant increase and decrease in camera's elevation angle with each click. Care needs to be taken regarding the upper and lower bounds of these values.

Besides, the angle of the Kinect should not be frequently changed else it could damage the Kinect. In Figure 4 below, changing elevation angle of the camera results in a shift in the field-of-view of the concerned object.

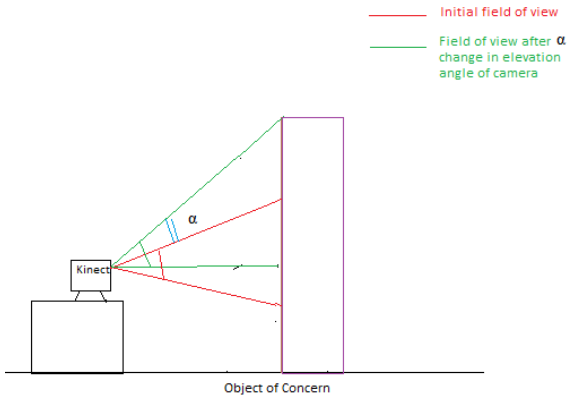

Figure 4: Adjusting the field of view of the Kinect camera.

D. Smoothing of an object

Irregularities at the object's edges and presence of noise in image frames captured by the Kinect result in an aberration in captured image. This is illustrated in the Figure 5 below:

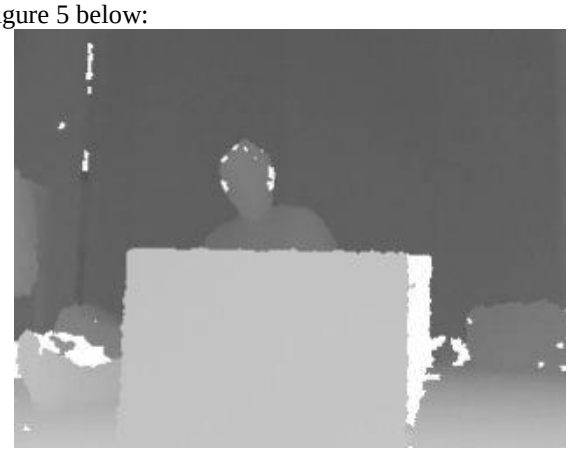

Figure 5: Raw Depth Image.

The smoothening of edges of the object is achieved through the application of a function that modifies the pixels of the image based on some function of a local neighborhood of the pixels. To achieve this, the depth value is calculated from the *byte[ ]* of an *ImageFrame* type, (the type sent from the Kinect sensor), which is then converted into an array of bits. A filter is then applied on this array, by scanning the entire array, for zero values that correspond to a pixel which the Kinect processed incorrectly. The aim is to change the bit value of such a pixel to either 1 or 0, without affecting the quality of image. The bit position is considered for filtering if its value is 0. The filtering algorithm applies two neighboring bands around the candidate pixel, and looks for non-zero values in other pixels. A frequency distribution of these values is calculated, and a count of the number of pixels in each band is maintained. Comparing these values to the threshold value for each band determines if the candidate should be filtered. If the calculated value is more than threshold value, then bit value of candidate's is flipped else it is left alone. The result of the smoothening is shown in the Figure 6 below:

Figure 6: Smoothened Depth Image.

E. Detecting an object's edges

Having adjusted the camera position and smoothened the view, the video stream is processed to detect the position of the object. The Aforge.NET C# framework is used for this. The video stream is broken down into individual image frames which are converted to a Bitmap image format. The bitmap image format received from the Kinect sensor is of type "*Format32bppArgb*". This means that there is 32 bits of information per pixel - 8 bits for each of the Alpha, Red, Blue and Green components. However before applying any filtering, conversion of the image to GrayScale format is necessary. This converts the pixel format to "*Format8bppIndexed*" so there is only 8 bit of gray shade information in the pixel. Applying the *Sobel* algorithm or the *Canny* algorithm for edge detection yields best results as shown below.

Figure 7: *Canny* filter applied onto the smoothened image.

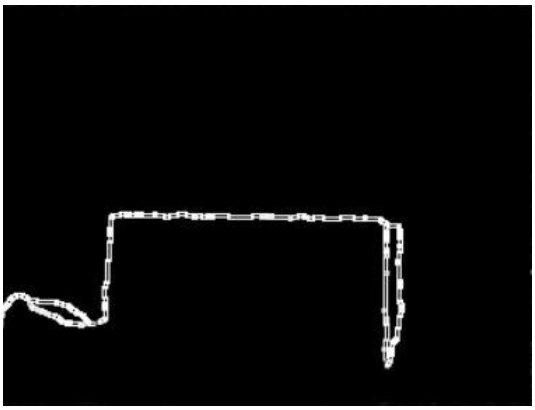

Figure 8: *Sobel* filter applied onto the smoothened image.

Directing the camera to the lower half of the field of view or processing only the lower third of the Bitmap image stream ensures that the algorithm detects objects which are placed on the ground and thus avoid false positives.

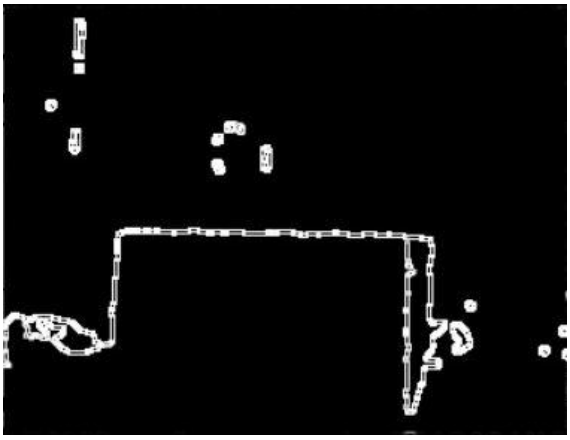

Figure 9: *Sobel* filter applied onto the Raw Depth Image.

Metallic objects are often seen as white by the infrared sensors. This can give a false impression to the camera that it is an object which is situated close to the Kinect. If the white blob in the image is touching the base of the field of view of the sensor, the object is indeed an obstacle. In case of a metallic almirah at a distance however, since the blob would not be touching the base, it is not detected as an object.

It is interesting to note that if any of the filters is applied onto the raw depth image, the resultant image as shown in Figure 9, has several aberrations.

IV. RELATED WORK

Huang *et. al* proposed [1] various algorithms are available to detect object and to decide the path of the autonomous robot. Their decision to choose the algorithm for image processing is based on the results of robustness and performance of processing speed (as in the frame rate per second processed). Some of the algorithms mentioned include Region of Interest, Gaussian Image and Kalman Filter Framework.

Sciotti *et. al* used an indoor map of a topological environment [2] to get a picture of the environment and proceed with obstacle detection with some prior knowledge of the system. Our system however does not plan on scanning the environment and obstacle detection is solely based on the camera's current interpretation of what is front of it.

Lu xia *et. al* proposed [3] algorithm to detect object using only depth information and use gradient approach to accurately detect the object. The proposed algorithm detects all poses and appearances of human from the depth array, and provides an accurate estimation of the whole body outline of the person. Besides, this proposed algorithm can be applied in multiple tasks such as object segmentation, human detection, tracking, and activity analysis. But algorithm fails to detect an object with a random shape. This is different from our proposed model which uses the threshold distance from the obstacle.

Ryo Tanabe *et.al* proposed [4] an algorithm using marker approach where using detection function object detected in the given area and thereafter slope information of an object along vertical and horizontal axis is used to estimate object's posture accurately.

Motion of hand Human-Computer-Interaction (HCI) is a primitive way to communicate with machines, since it closely resembles how humans interact with each other. Zhou *et. al* in [5] proposed a hand gesture recognition system with the Kinect sensor, which operates robustly in uncurbed environments and remains unaffected to hand variations and distortions. A proposed system model comprises of two important components, hand detection and gesture recognition. It makes use of the depth and color information from the Kinect sensor to detect the hand shape, which ensures the robustness in unregulated conditions.

V. ISSUES FACED

Some of the issues faced while implementing this project were:

1. *Aberration in depth real frame*

The depth map formed from the Kinect sensor is sometimes ill-formed at the boundaries of the image. Some spots are also seen at a distance reflected from small metallic objects or edges of larger objects in the background like doors or windows. This noise is possibly

due to the scattering of the infrared light at the boundaries of objects. This is illustrated in Figure 5.

Solution: By passing the image stream through a smoothening filter as described earlier we resolve this issue to an extent. It is important to note here that computing the exact boundary of the image is not important. A mere estimate of the area where the object is located is good enough for a navigational robot to move around and our algorithm is successfully able to give us that estimate. The processed image after the smoothening is applied is shown in Figure 6.

2. Distinguishing near objects against far object

Differentiating between an object and a background served as a major challenge in the implementation. Metallic objects like an almirah at a distance were sometimes misrepresented with a lower grayscale value, thus affecting the average depth calculations.

Possible Solution: Focusing the camera on the lower third of the view ensures that we only detect things that are situated in the lower view. A metallic almirah, which would appear as a blob in the upper part of the image can thus be differentiated from an object which appears on the lower third.

3. Certain objects are more difficult to detect

Objects like glass and plastic bottles were undetected during our analysis. The reason behind this is the infrared light projected by IR projector is observed by such objects as a result IR ray fail to get reflected and captured by IR sensor.

4. Memory limitations

If the software was kept running on a system for greater than say eight minutes or so, it gets stuck and throws an out of memory exception. For a 32 bit windows application, the process can only have 2GB of addressable memory[6] out of which only 1.5 GB is accessible to the application.

Possible Solution: Running the software on a 64bit system and enabling the “LargeAddressAware” flag can raise the addressable memory to upto 12 GB. We can also have and efficient garbage collection mechanism to overcome this issue on a system with limited memory.

REFERENCES

[1] El-laithy, R.A.; Jidong Huang; Yeh, M.; "Study on the use of Microsoft Kinect for robotics applications," Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION, pp.1280-1288, 23-26,April 2012

[2] Sciotti, D.F.; Prado, M.G.; Sales, D.O.; Wolf, D.F.; Osorio, F.S.; "Mobile Robots Navigation in Indoor Environments Using Kinect Sensor," Critical Embedded Systems (CBSEC), 2012 Second Brazilian Conference on , pp.36-41, 20-25 May 2012

[3] Lu Xia; Chia-Chih Chen; Aggarwal, J.K.; "Human detection using depth information by Kinect," Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on , pp.15-22, 20-25 June 2011

[4] Tanabe, Ryo; Cao, Meifen; Murao, Toshiyuki; Hashimoto, Hiroshi; "Vision based object recognition of mobile robot with Kinect 3D sensor in indoor environment," SICE Annual Conference (SICE), 2012 Proceedings of , pp.2203-2206, 20-23 Aug. 2012

[5] Zhou Ren, Jingjing Meng, Junsong Yuan Zhengyou Zhang; "Robust hand gesture recognition with Kinect sensor", MM '11 Proceedings of the 19th ACM international conference on Multimedia, pp 759-760

[6] Calvin Hsia’s Weblog, 2010. Available in: http://blogs.msdn.com/b/calvin_hsia/archive/2010/09/27/10068359.aspx, Accessed in Nov 2012.

[7] Karl Sanford 2012. Available in: http://www.codeproject.com/Articles/317974/KinectDepthSmoothing, Accessed in Nov 2012.

[8] Microsoft Visual Studio Forums, 2007. Available in: http://social.msdn.microsoft.com/Forums/en-US/wpf/thread/aad606b1-3f64-4314-a552-370afd361926/, Accessed in Dec 2012.

[9] Microsoft Developer Network Platforms documentation on Elevation Angle. Available in: http://msdn.microsoft.com/en-us/library/microsoft.kinect.kinectsensor.elevationangle.aspx, Accessed in Nov 2012.

[10] AForge.NET documentation on Edge Detectors. Available in: http://www.aforgenet.com/framework/features/edge_detectors_filters.html, Accessed in Nov 2012.