Kinect Module

Depth Based Room Mapping Using Kinect

Aniket Patel
Mukesh P

IIT Bombay

July 15, 2015

Introduction to Kinect Sensor

- Kinect is a line of motion sensing input devices by Microsoft for Xbox 360
- The device features an RGB camera, depth sensor and multi-array microphone which provide full-body 3D motion capture, facial recognition and voice recognition capabilities.
- The depth sensor consists of an infrared laser projector combined with a monochrome CMOS sensor, which captures video data in 3D under any ambient light conditions.
- The sensor has an angular field of view of 57 horizontally and 43 vertically, while the motorized pivot is capable of tilting the sensor up to 27 either up or down.
- The default RGB video stream uses 8-bit VGA resolution (640 x 480 pixels)

Installation of Kinect API

- The python wrapper for Kinect API is freenect.
- The prerequisites for using freenect module is to have python version 2.7.6 or higher (For Ubuntu users, Ubuntu 14.04 or higher version of Ubuntu is recommendable)
- To install freenect library from Terminal, enter the following command:
  sudo apt-get install python-freenect
- To use Kinect functions given in this module you need to install the following libraries:
  opencv, numpy, time, serial, matplotlib, math
- To use the functions of this module just **import Kinect** in your code.

Depth Frame I

get_depth(void) :

- It takes no input arguments.
- The depth frame is retrieved in mm and it is divided by a constant so that it falls in 0 to 255 range.
- The frame is passed through a filter and noise is eliminated.
- The return value is a 640 * 480 matrix of depth frame.
- **Example Call:** depth_array = get_depth()

Depth Frame II

filter_smooth(depth_array) :

- It takes the original noisy depth frame
- It masks the noise pixels
- It is than passed through a filter to remove the noise.
- It is then passed through a bilateral filter for better results
- It returns the smoothened image of the frame
- **Example Call:** depth_array = filter_smooth(depth_array)

Depth Frame III

filter_noise(depth_array, noise mask, original image with inverted noise mask, rows, columns) :

- It takes the original frame as input for processing. It takes noise mask and original frame with noise pixels made white. It takes the number of tiles in a row or a column
- The algorithm takes the mean of the frame in a particular tile and replaces the noise pixels with those values
- It returns the noise filtered depth frame
- **Example Call:** blur = filter_noise(depth_array, mask_1, masked_array, 3, 4)

Robot Traversing I

regular_movement(void) :

- It takes no input arguments.
- The depth frame is divided in 8 regions.
- The for loop starts with the middlemost frames and goes sideways in both directions at a time.
- Whenever an object is detected in the range the function sends the number of the section in which it is detected
- If no object is detected in any of the divisions than the robot searches for the wall
- The robot follows the wall and maps the objects
- **Example Call:** regular_movement()

Robot Traversing II

door_movement(void) :

- It takes no input arguments
- The frame is divided into 3 regions: left area, middlearea, rightarea.
- The mean of all the regions is taken.
- If the middleregions goes below the threshold than left and right area means of pixel values are checked
- The region with higher mean is selected and robot turns in that direction as higher value indicates more depth
- So the robot traverses on the intuition of the pixel mean value of the area after door is detected
- **Example Call:** door_movement()

Robot Traversing III

doorway_movement(mid_point) :

- It takes the midpoint of the width of the door edges as input
- If the point is left of the frame than robot turns left so it comes in between and vice versa
- The robot therefore orients themselves so that the door is in front of the robot
- **Example Call:** doorway_movement(200)

Robot Traversing IV

back_movement(depth_array) :

- It takes depth map as input
- The depth frame is divided in three regions
- If the object comes in the blind spot of the Kinect than the robot will move back
- Here we have used the disadvantage of the blind spot of Kinect as an advantage because the Kinect cannot determine depth of object close to it and gives 255 value to those pixels
- If the number of white pixels crosses a threshold than the robot will move back
- **Example Call:** back_movement(depth_array)

Robot Traversing V

stuck_pos_movement(void) :

- It takes no input arguments
- It calculates the mean of the left and right pixels of the frame
- The direction with higher mean is selected and the robot takes a left or right turn accordingly
- This function helps in navigating the robot out of a corner position
- **Example Call:** stuck_pos_movement()

Robot Traversing VI

search_wall(Direction) :

- It takes only the direction of the wall as input. 0 is left and 1 is right
- The robot takes a left or right until the left part of the frame or the right part falls in range of the Kinect
- **Example Call:** search_wall(0)

Dimension Calculation I

actual_height_in_mm(left_bottom,left_top,right_bottom,right_top co-ordinates) :

- The input arguments are the co-ordinates of the left top, left bottom, right top, right bottom points of the door
- The function creates a small 10x30 area and finds the minimum pixel value as that will be the depth value of that point
- So we get the depth distance of the four points.
- Now 640 pixels corresponds to 57 degrees and 480 pixels to 430 degrees. So we find the pixels height to find the angle formed by the line joining the top and bottom points
- Using Cosine rule we find the height of the third side of the triangle formed by joining all the points.
- Rescaling the height we get the actual height of the left and right edges of the door
- **Example Call:** left_height, right_height = actual_height_in_mm((100,200), (100,100), (200,200), (200,100))

Dimension Calculation II

actual_width_in_mm(left_bottom,left_top,right_bottom,right_top co-ordinates, x-coordinate of centroid right and left edges) :

- The input arguments are the co-ordinates of the top and bottom of the left edge and the right edge, the midpoints of the contours of the left and right edges
- Using the co-ordinates we choose the area around the edges and using warp perspective we make it a rectangle
- We find the minimum of the matrix and find the depth of the left and right edges
- 640 pixels corresponds to 57 degrees so we find the pixel width using the centroid of the left and right edges
- Using the pixel width we find the angle between the two edges and using cosine rule we find the actual width of the door
- **Example Call:** width = actual_width_in_mm((100,200), (100,100), (200,200), (200,100), 200, 100)

Dimension Calculation III

return_height_in_mm(left_bottom,left_top,right_bottom,right_top co-ordinates) :

- The input arguments are the co-ordinates of the left top, left bottom, right top, right bottom points of the door
- The function creates a small 10x20 area and finds the minimum pixel value as that will be the depth value of that point
- So we get the depth distance of the four points.
- So we find the pixels height to find the angle formed by the line joining the top and bottom points
- Using Pythagoras theorem we find the actual height of the edge of the door
- The assumption is that the depth of the bottom points is in line with the robot and the triangle formed is a right angled triangle
- **Example Call:** leftheight, rightheight = return_height_in_mm((100,200), (100,100), (200,200), (200,100), 200, 100)

Dimension Calculation IV

probability(mean value, variance, data) :

- The input arguments are the standard value, the variance and the data for which we will find the probability of closeness to the standard value
- First we form a gaussian curve from the mean and the standard value
- The numbers are mapped to 0 to 100 for percentage probability.
- If the number falls in the range than a probability value is returned and if the value falls out of range it returns a zero probability
- **Example Call:** probtop = probability(0, 200, 150)

Left - Right Movements I

take_right(void) :

- It takes no input arguments
- The function finds the depth frame in a while loop and calculates the mean of the pixel values in the middle frame
- If the mean goes above threshold the robot moves forward and resumes it normal functioning
- The robot takes a right until the depth is more in the middle frame
- **Example Call:** take_right()

Left - Right Movements II

take_left(void) :

- It takes no input arguments
- The function finds the depth frame in a while loop and calculates the mean of the pixel values in the middle frame
- If the mean goes above threshold the robot moves forward and resumes it normal functioning
- The robot takes a left until the depth is more in the middle frame
- **Example Call:** take_left()

Left - Right Movements III

take_right_near(void) :

- It takes no input arguments
- The function finds the depth frame in a while loop and counts the number of pixels of the objects that fall in its range
- If the count goes below a threshold than the robot will move forward and resume its movement
- The robot takes a right until the objects go out of the Kinect sensor range which we specify
- **Example Call:** take_right_near()

Left - Right Movements IV

take_left_near(void) :

- It takes no input arguments
- The function finds the depth frame in a while loop and counts the number of pixels of the objects that fall in its range
- If the count goes below a threshold than the robot will move forward and resume its movement
- The robot takes a left until the objects go out of the Kinect sensor range which we specify
- **Example Call:** take_left_near()

Door tests I

actual_height_test(left edge height, right edge height) :

- It takes two arguments: The left edge height and the right edge height
- It returns the probability percentage of the edges to be door edge. We have sent 1.5m as standard value and 1500mm is the variance
- The average of the two probabilities is returned from this function
- **Example Call:** prob_2 = actual_height_test(1000, 1000)

Door tests II

actual_width_test(width) :

- It takes only one argument: the width of the edges
- It returns the probability of the width to be a door width. We have set 1m as the standard width and the variance is 1m
- The function returns the actual width between the two edges
- **Example Call:** prob_3 = actual_width_test(1000)

Door tests III

rectangle_door_test(left_bottom,left_top,right_bottom,right_top co-ordinates,left centroid,x-coordinate of centroid of right and left edges, co-ordinates of left and right points and centroid of the edge) :

- We send the co-ordinates of the top and bottom edges of the left and right edges, left and right edges centroid, the co-ordinates of the left and right points of horizontal edge and the centroid of the edge
- We find the pixel width of the door using the three points of the left and right edges and find the error with the horizontal edge pixel length
- The error is probabilistic in three cases and we find the average of the three values
- The probability of horizontal edge being a door edge is returned
- **Example Call:** max_prob = rectangle_door_test((100,200), (100,100), (200,200), (200,100), 200, 100, (100,100), (200,100), 150)

Door tests IV

is_door(left_bottom,left_top,right_bottom,right_top co-ordinates,x-cordinate centroids of left and right edges) :

- The arguments needed are the co-ordinates of the top and bottom points of the left and right edges and the centroids of those edges
- The pixel difference is calculated between the two pixel heights of the edges.
- The pixel difference between the centroids should be within range and the difference in door pixel heights should be in range
- The function returns a true or a false value based on the intuition of pixel measurements
- **Example Call:** dec = is_door((100,200), (100,100), (200,200), (200,100), 200, 100)

Edges and lines I

contours_return(depth_array, pixel shift) :

- The arguments are the depth array and the shift for detecting the edges
- The frame pixels are shifted by the number specified from the input argument
- The shifted frame is subtracted with the original so the difference in the pixels are stored as a frame
- The more difference is masked in the frame. The masked regions form a contour and thus contours are formed
- The function returns all the contours where depth difference is more
- **Example Call:** contour = contours_return(GLOBAL_DEPTH_MAP, 6400)

Edges and lines II

horizontal_lines(void) :

- The horizontal edge contours are detected by shifting the pixels by 6400 so that the frame is shifted 10 pixels down
- The horizontal edges are drawn for the contours and all the horizontal edges data are saved in corresponding lists
- the function returns the list of left and right point edge co-ordinates the centroid of the edge and the number of egdes
- **Example Call:** hll, hrl, cxhl, temph = horizontal_lines()

Edges and lines III

left_right_lines(left contours, right contours) :

- The arguments are the left and right contours
- The function creates a list of co-ordinates of left and right edges in a list
- The data is stored only if the centroid value is not zero
- The function returns the list of all the values of co-ordinates of the left and right edges
- **Example Call:** ltl, lbl, cxll, rtl, rbl, cxrl, templ, tempr = left_right_lines(contours_right, contours_left)

Edges and lines IV

horizontal_edge(contour) :

- Argument is a contour
- If the contour area crosses a threshold than its extreme points are deduced
- To ensure almost straight lines are selected as edges we find the pixel difference between the y co-ordinates of the top and bottom points
- If the difference is more than threshold than it is a potential edge.
- The pixel width is also calculated and if it crosses a threshold than the edge is a potential door edge
- The function returns the horizontal edges in the frame
- **Example Call:** height_left, height_right, cxh = horizontal_edge(contours)

Edges and lines V

potential_leftedge(contour) :

- Argument is a contour
- If the contour area crosses a threshold than its extreme points are deduced
- To ensure almost straight lines are selected as edges we find the pixel difference between the x co-ordinates of the top and bottom points
- If the difference is more than threshold than it is a potential edge.
- The pixel height is also calculated and if it crosses a threshold than the edge is a potential door edge
- the right area of the edge is taken and using warping the mean of the area is found. If the mean is above threshold than it is potential left edge
- The function returns the co-ordinates of left edges in the frame
- **Example Call:** left_top, left_bottom, cxl = potential_leftedge(contours)

Edges and lines VI

potential_rightedge(contour) :

- Argument is a contour
- If the contour area crosses a threshold than its extreme points are deduced
- To ensure almost straight lines are selected as edges we find the pixel difference between the x co-ordinates of the top and bottom points
- If the difference is more than threshold than it is a potential edge.
- The pixel height is also calculated and if it crosses a threshold than the edge is a potential door edge
- the left area of the edge is taken and using warping the mean of the area is found. If the mean is above threshold than it is potential right edge
- The function returns the co-ordinates of right edges in the frame
- **Example Call:** right_top, right_bottom, cxr = potential_rightedge(contours)

Communication and Pixel Count I

count_near_pixels(area, distance) :

- Specifies the area and the distance threshold under which object should be detected
- The function masks the frame based on the depth distance given. for eg. If we give 1 m than it will display values only within that distance
- The visible pixel count is returned from the function
- **Example Call:** count_near_pixels(area, 900)

Communication and Pixel Count II

data_send(left motor mode, right motor mode) :

- The arguments are the modes of the left and right motor
- The mode selection is based on the division in which the object is detected
- Based on the mode selected a particular char value is send to the robot which sets the velocity of the two motors of the robot
- If the mode is 4,4 than the robot is stuck in a position and a function is called for that movement
- **Example Call:** data_send(2,0)

Door Detection I

door_detection(left contours, right contours) :

- Left, right and horizontal lines are formed by the respective functions
- Probability of all the three tests are calculated and their weighted average is taken
- 50 percent to height 40 percent to width and 10 percent to the horizontal edge tests
- If the weighted sum crosses a certain threshold than the frame contains a door.
- The graph is plotted simultaneously with the data being given as input
- If the door is detected the flag is set to true and the buzzer gives a beeping sound
- **Example Call:** door_detection(contoursrightnear, contoursleftnear)

Door Detection II

start(void) :

- This function runs the actual code for door detection exit and algorithm
- The kinect is set to 20 degrees and initial frames are skipped
- The program runs after these steps and robot traverses the room and exits room from the door
- **Example Call:** start()