Assignment 1

EECS 4422 Computer Vision

Date: October 17, 2016

Andrew Lau 212905253

Michael Williams 211087798

**Question 1 Part B**

The main objective of this question was to demo the gabor filters.  In doing so, we were required to make a function that takes in the wavenumber and the change in the angle.  The result is the continuous projection of images after the gabor filters have been applied, with each subsequent image being the resultant of rotating the filter angles by the given amount.

For our program, k refers to the wavenumber and theta refers to the change in angle.  The program works as follows:

By default, the angle applied to the gabor filters starts at 0.  From there, we capture an image and store it inside of a variable.  Then, while the angle has not yet made a full rotation (ie, while angle <= 360 degrees of rotation), we apply the cos gabor and sin gabor functions to the image separately and store them as separate images.  These new images, named c\_image and s\_image are then squared and added to produce the final image, which is displayed subsequently.

The main problem when implementing this program was to make the angles increment every time a new image was taken, so that each new image had the new angle applied to it.  This was capable when using the regular web camera, but when using the grey point camera, this proved to be much more difficult.  Otherwise, the program didn’t offer much more challenge in terms of implementation or design.

**Question 1 Part C**

The main goal of this program was to design an algorithm that computed the line of best fit, given an image from the grey point camera and selected points on 2 edges of the image.   These points are manually selected by a user.

The program is supposed to work as follows, after obtaining an image:

1) First, we select 2 edge points at random

2) Afterwards we fit a line going through them

3) Afterwards, we count the number of edge points that are within 5 pixels of the line created

4) We when repeat these steps 200 times.  Upon finishing this, we select the line that has the highest number of edge points counted in accordance with step 3.

5) Using the rule of least squares, we then fit a line that goes through all the points from the selected line in step 4

6) Then, we overlay the line on the image

7) After doing so, we remove the set points so that the algorithm can start detecting the next line

8) Last, we repeat to detect and obtain the next line

We came across a lot of trouble when attempting to implement this program.  For the most part, understanding it turned out to be a challenge all by itself.  If more time were available to us we most likely would have finished, but due to time constraints, which resulted in a lack of understanding, this was not plausible.

**Question 2**

The main focus of this program was to measure and calculate camera noise over a set number of images.  The result of the program is a returned list, with the first image containing the mean or average image calculated and the second image containing the average variance of the noise as an image.  The average and variance values are computed as an average of the images taken.

For this program, we begin by taking 30 images from the point grey camera and storing them all inside of an array.  It is important to try to keep the camera steady so that the exact images can be created.  After doing so, and with the use of for loops, we calculate the mean or average of all the images stored inside the array.  Then, we compute the variance by finding the summation of each image, subtracted from the mean image, who's resultant is then squared (ie   variance += (array[i] - mean)\*(array[i] - mean)).  After obtaining this value, we simply divide the resulting variance by the number of images (30 in this case), to obtain the average variance.  Then, we simply display the 2 images using the gshow function and return them as a list.

Overall the implementation of this program was fairly simple.  The only part that proved to be a little taxing was returning the values form the function as a list, but this was easily corrected upon the help of the professor.  Otherwise, the implementation was fairly simple and self-explanatory.

**Question 3: Robot Arm**

This function takes in 2 standard parameters and 1 optional parameter. The 2 standard parameters include an x and a y value that represent the position the user wants the robot arm to move to, on the world coordinate system. The 1 optional parameter includes the option to set the state of the gripper after the movement is completed.

The function is written in mediamath. In order to write this function one must get a full understanding of how to get every angle on each joint of the robot arm and be able to convert it to the world coordinate system. Below are diagrams drawn to help with deriving the formulas to find the angles for the arm. The parameters of the function are in centimetres, with the rotational pivot point of the base as the reference point (as [0, 0]). If an unreachable value is entered, the arm does not move due to the fact that the function “rob\_move\_abs” does not move the arm unless an acceptable value is entered.

Some things worth noting when working on this function was that all the trigonometric functions (sin(), cos() and tan()) were set to take in and return radian values. However, the functions that move the robot arm take values in degrees, so a conversion must be done in order for the arm to move the correct distance to the user defined position.