**read\_pgm\_image(char \*infilename, unsigned char \*\*image, int \*rows,**

**int \*cols)** -> reads image in PGM format. The image can be

read in from either a file or from standard input. Memory to store the image is allocated in this function.

On failing, this function returns 0, upon sucess it returns 1.

**write\_pgm\_image(char \*outfilename, unsigned char \*image, int rows,**

**int cols, char \*comment, int maxval)** -> writes image in PGM format. The file is

written to file specified by outfilename or to standard output otherwise.

**read\_ppm\_image(char \*infilename, unsigned char \*\*image\_red,**

**unsigned char \*\*image\_grn, unsigned char \*\*image\_blu, int \*rows,**

**int \*cols)** -> reads image in PPM format. The image can be

read from file or from standard input. Memory to store the image is allocated in this function.

On failing, this function returns 0, upon sucess it returns 1.

**write\_ppm\_image(char \*outfilename, unsigned char \*image\_red,**

**unsigned char \*image\_grn, unsigned char \*image\_blu, int rows,**

**int cols, char \*comment, int maxval)**-> writes image in PPM format. File is either

written to file specified by outfilename or to standard output otherwise.

**fprintf(fp, "P6\n%d %d\n", cols, rows)** -> Writes header information to the PGM file.

**canny(unsigned char \*image, int rows, int cols, float sigma,**

**float tlow, float thigh, unsigned char \*\*edge, char \*fname)**-> perform canny edge detection

**gaussian\_smooth(unsigned char \*image, int rows, int cols, float sigma,**

**short int \*\*smoothedim**)-> Blur an image with a gaussian filter.

**make\_gaussian\_kernel(float sigma, float \*\*kernel, int \*windowsize)** -> Create a one dimensional gaussian kernel

**derrivative\_x\_y(short int \*smoothedim, int rows, int cols,**

**short int \*\*delta\_x, short int \*\*delta\_y)**-> Compute the first derivative of the image in both the x any y

\* directions.

**magnitude\_x\_y(short int \*delta\_x, short int \*delta\_y, int rows, int cols,**

**short int \*\*magnitude)**-> Compute the magnitude of the gradient. This is the square root of

\* the sum of the squared derivative values.

**apply\_hysteresis(short int \*mag, unsigned char \*nms, int rows, int cols,**

**float tlow, float thigh, unsigned char \*edge)**-> finds edges that are above some high threshhold or

\* are connected to a high pixel by a path of pixels greater than a low

\* threshold.

**radian\_direction(short int \*delta\_x, short int \*delta\_y, int rows,**

**int cols, float \*\*dir\_radians, int xdirtag, int ydirtag);** - > To compute a direction of the gradient image from component dx and

\* dy images. Because not all derriviatives are computed in the same way, this

\* code allows for dx or dy to have been calculated in different ways.

**rea** -> computes the angle of a vector with components x and

\* y. It returns this angle in radians with the answer being in the range

\* 0 <= angle <2\*PI.