Tools used for this course

Lecture notes, images, lab assignments, papers, code snippets and other materials are available at the course website: http://www.cecas.clemson.edu/~ahoover/ece431/.

You will need a gcc environment. For windows I recommend msys32 https://en.wikipedia.org/wiki/Mingw-w64#MSYS2. For Mac you need Xcode https://en.wikipedia.org/wiki/Xcode.

You will need Microsoft Visual Studio 2010 or later version. https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#Visual_Studio_2010

The best way to understand computer vision algorithms is to implement them. You will do this by coding several algorithms in C. To start, you will create command line programs that have no GUI. These programs will read images, perform one or more computer vision algorithms, and output results. For this part of the work, you can use either the GCC compiler or Microsoft Visual Studio. Later, you will learn how to build a GUI using the Win32 API. As part of the course we will study event-driven programming. You will learn how to overlay graphics that help you visualize and understand the computer vision algorithms. For this part of the work, you will need Microsoft Visual Studio, although the motivated student may implement the same concepts using the X window system in any version of Linux.

You will need an image viewing program. There are many (see http://en.wikipedia.org/wiki/Comparison_of_image_viewers). The program needs the ability to view PPM formatted images. The PPM format is the simplest image file format to read and write, but is not supported by all image viewers, so choose carefully. For MS Windows users, I recommend the free program IrfanView: http://www.irfanview.com/. Later, once we study GUI event-driven programming, I will help you develop your own code to display images.

There are several libraries that incorporate standard computer vision algorithms. The most popular is OpenCV (see http://opencv.org/). This is a great resource, but will not be used in this course. I recommend only using computer vision libraries after you have become proficient in the basics, which includes the ability to implement them.

The following is a short tutorial on C-code to read a PPM formatted image, do some simple processing, and write a result. The code will be demonstrated and discussed in class.

```
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])

{
FILE        *fpt;
unsigned char     *image;
char        header[80];
int         ROWS,COLS,BYTES;
int         i,j,b;

      /* tell user how to use program if incorrect arguments */
if (argc != 3)
```

```
  {
  printf("Usage:  brighten [filename] [amount]\n");
  exit(0);
  }
b=atoi(argv[2]);
if (b < 0  ||  b > 255)
  {
  printf("amount must be 0...255\n");
  exit(0);
  }
      /* open image for reading */
fpt=fopen(argv[1],"r");
if (fpt == NULL)
  {
  printf("Unable to open %s for reading\n",argv[1]);
  exit(0);
  }
      /* read image header (simple 8-bit greyscale PPM only) */
i=fscanf(fpt,"%s %d %d %d",header,&COLS,&ROWS,&BYTES);
if (i != 4  ||  strcmp(header,"P5") != 0  ||  BYTES != 255)
  {
  printf("%s is not an 8-bit PPM greyscale (P5) image\n",argv[1]);
  fclose(fpt);
  exit(0);
  }
      /* allocate dynamic memory for image */
image=(unsigned char *)calloc(ROWS*COLS,sizeof(unsigned char));
if (image == NULL)
  {
  printf("Unable to allocate %d x %d memory\n",COLS,ROWS);
  exit(0);
  }
      /* read image data from file */
fread(image,1,ROWS*COLS,fpt);
fclose(fpt);

      /* simple CV operation to add b to every pixel */
for (i=0; i<ROWS*COLS; i++)
  {
  j=(int)(image[i])+b;
  if (j > 255)
    j=255;
  image[i]=(unsigned char)j;
  }

      /* write out brighter.ppm, the output result */
fpt=fopen("brighter.ppm","w");
if (fpt == NULL)
  {
  printf("Unable to open brighter.ppm for writing\n");
  exit(0);
  }
fprintf(fpt,"P5 %d %d 255\n",COLS,ROWS);
fwrite(image,1,ROWS*COLS,fpt);
fclose(fpt);
}
```