

GE423 Laboratory Assignment 8

Robot Vision

Recommended Due Date: By 4:00PM April 17th

Possible Points: If checked off before your lab time the week of Apr. 20th ... 14 points
If checked off after your lab time the week of Apr. 20th and before 4:00PM Apr. 24th ... 11 points
If work not finished before 4:00PM Apr. 24th ... 0 points

Goals for this Lab Assignment:

1. Learn low level vision processing algorithms in order to use the robot's color camera.

DSP/BIOS Objects Used:

SWI, PRD, HWI, TSK

New Library Functions: Init_colorvision, SendImagetoUART2every25calls, SendBWImagetoUART2every25calls, userProcessColorImageFunc_laser, processvisiondata, MEM_alloc.

Prelab: Read the final contest write up and think about how to solve the contest. You may want to spend some time in lab testing out different approaches to solving the maze. The contest maze will be built and ready in lab by the start of Lab 8.

Continue updating your VB application. If you haven't already, you should start modifying the VB program so that you can download new values to variables in your DSP program. This will be very useful when you are trying to tune your algorithm and controllers for the final contest.

Laboratory Exercise

Introduction

The CMOS color camera on the DSP returns an image every 1/25 of a second. The image size coming from the camera is 288 rows by 352 columns of 8 bit pixel data, 101376 bytes of data. You may recall that the DSP has 192,000 bytes of internal RAM and 16 Mbytes of external SDRAM. Up to this point, our programs have only used the internal memory of the DSP because this RAM is much faster. The internal RAM runs at the CPU's clock speed of 225 MHz while the external SDRAM runs at 90MHz. The DSP does have cache memory that speeds up most accesses to the external SDRAM but the access is still quite a bit slower than the internal RAM. Due to the size of our color image, we have to store the image in external SDRAM and take the hit of slower memory access time.

It takes roughly 10 ms. to loop through all the pixels of the 288 by 352 image. We receive an image every 40ms. If we were to process this large image we could only choose algorithms that loop through the image at the most three times. This could limit your ability to write the algorithms you wish and would starve the DSP of processor time needed for other processes. For this reason the image is given to you compressed by a factor of four. The given code loops once through the image creating a 72 row by 88 column image for you to process. You still want to minimize the number of passes your code loops through this smaller image, but it is now possible to do more than three passes.

Each pixel of this 72 by 88 image has a red, green and blue 8-bit intensity value associated with it. 0 indicates that there is no color intensity and 255 indicates full color intensity. In the file `user_ColorVisionFuncs.c`, you will be adding code to the function `userProcessColorImageFunc_laser(volatile bgr **ptrImage, volatile bgr **ptrLaser)`. This function is called each time a new image is received from the camera. The parameter `ptrImage` is a pointer to a multi-dimensional array of the type defined structure `bgr`. `bgr` is defined in the include file `dspvisioncolor50Hz_cLCD.h` to be:

```
typedef struct bgr {
```

```

        unsigned char blue;
        unsigned char green;
        unsigned char red;
    } bgr;

```

For example, a C statement to access the green component of the 40th row and 30th column could be: `value = ptrImage[39][29].green.`

For this lab discussion we will ignore the second parameter `ptrLaser`. `ptrLaser` points to a higher resolution sliver of the image that could be used to find the distance to an object with a laser diode shining on it. This is something you may want to explore in your final project.

Exercise 1: Find the X Y pixel location of the center of a bright light

For this first exercise, you will assume that there is only one bright object seen by the camera. You are to find the center of this one bright object and display its X and Y pixel coordinates to the text LCD screen. The robot can see the room lights when it is on the floor so you will need to ignore the top section of the image so the bright pixels of the lights are not added to your centroid calculation. In Exercise 3 you will be asked to have the robot follow this bright light so the centroid values need to be communicated to your robot control process. Set up global variables to communicate the new centroid and the number of image pixels to your control process. Then in the control process, print the centroid and the number of pixels to the LCD screen every 100 ms.

In this exercise only use the red pixels of the image. Your algorithm then will find the centroid of any bright red or white light. Create a dual loop to loop through the image once, thresholding the image and finding the center of all pixels greater than this certain threshold value. You will need to experiment to find a good threshold value. To display your thresholded image to the robot's LCD screen set the blue and green pixels to 0 and set the red pixels to 0 for pixels less than the threshold, 255 for pixels greater than threshold. **Remember not to process the upper row pixels so the overhead lights do not cause you problems.** Two `#defines` `IMAGE_ROWS 72` and `IMAGE_COLUMNS 88` will be helpful in writing this code.

Exercise 2: Watching the Image on the PC

It can also be useful to display your image on the PC. The main problem with displaying the image on the PC is that it is slow. To send one color component of the image to the PC takes one second; this means that it takes three seconds to upload the complete image to the computer. This method is really slow compared to the color LCD screen, which displays 12.5 images every second. However, sometimes it is useful to display a blown up image on the PC's monitor. This way you can see details hard to recognize on the small LCD screen. The image data is transmitted to the PC through UART channel two. Your TA will show you how to connect the PC to the robot. You will need to uncomment the code line `SendImagettoUART2every25calls(ptrImage)` in your function `userProcessColorImageFunc_laser`. `SendImagettoUART2every25calls` does pretty much what the name says. You call it each time a new image has been processed but it only sends every 25th image to the PC. Exercise 5 will demonstrate how to send only one color to the PC. On the PC's end you need to start the program `V:\colorcamera\vbcolorvision\vbcolorvision.exe`.

Exercise 3: Go toward the light

Using a laser shining on a nearby wall (or a flashlight pointed at the robot), design a P controller to make the robot drive toward the light. Use the ultrasonic sensors and IR sensors to detect when you get close to the wall, and slow the robot down to a stop when it gets to within 1 tile of a wall. A good starting K_p for vision error measured in pixels is 0.05.

Exercise 4: Following an object of a certain color

To follow an object of a certain color is obviously very similar to following a bright light; the only difference with exercises 1-3 is that we assumed that there was only one bright light in the robot's field of view. If there were two light sources, our light following code would not have worked as well. So for the remaining exercises we are going to introduce some new code that adds a segmentation algorithm to the vision processing. This new code is generated for you when you use a new project creator located at `V:\c6713dsk\project_creator\ge423\segmentation\segmentation_project_creator.exe`. Open up this program and generate a new project. Then open this new project inside CCS. If you look at the file `user_ColorVisionFuncs.c` you will find that much more code is given to you. The code in `userProcessColorImageFunc_laser` implements the algorithm discussed in Spong, Hutchinson, M. Vidyasagar *Robot Modeling and Control* <http://coecsl.ece.uiuc.edu/ge423/datasheets/vision.pdf>. This code will be discussed some in class but part of this exercise is for you to become familiar with it. Read through the code and explain to your TA the "big picture" of what this code is doing.

Looking at the other "user_" source files you will see that quite a bit of other code is given to you.

- a. In your main "user_" file we have given you a starter shell for the control loop. The ADC is started every 1ms by a PRD function and then on conversion of the ADC, INT7's interrupt function is called. Also notice that the state of the dip switches changes what is output to the text LCD screen.
- b. In `user_IR_UltraFuncs.c` we give you tasks for reading the IR, Ultrasonic and Atmel IR sensors and a task for writing to the Atmel RC servos.
- c. In `user_PIFuncs.c` we give you a tested version of the `PIVelControl` function.
- d. In `user_UARTFuncs.c` we give you two tasks for reading characters from UART1 and UART2.

You may not want to use all the code given here and instead use the code you developed over the past labs. This is the main reason we had you divide your code up into these "user_" source files. This way you can now just copy your source files that you would like to use over the files created by project creator.

For this exercise using this new created project, program the robot to follow a light blue golf ball or any other (non-orange) object that you choose. To help you find the HSV colors of your object use the dipswitch setting of 8. If you look at the robot's color LCD screen you will see a blue cross on the screen. The value of the HSV color at the center of the cross is displayed to the text LCD screen in this mode. Your TA will show you how to attach a modified optical mouse to your robot so you can change the location of the blue cross and find your desired HSV values.

Exercise 5: Watching the Threshold Image on the PC

Now, display the thresholded image on the PC. For this use the program

`V:\colorcamera\vbcolorvision_threshold\vbcolorvision_thres.exe`. Set the DSP dipswitches to 9 and the image should start uploading to the PC. The function call `SendBWImageToUART2every25calls(Thres_Image)` in `userProcessColorImageFunc_laser` performs this upload.

Exercise 6: Find the distance between a golf ball and the center of the robot.

Using centroid information of the light blue golf ball, calibrate a distance measurement to the golf ball. The golf ball is always assumed to be lying on the floor. Approximate the error at different distance measurements. Make sure to have the robot center the golf ball in its field of view before calculating the distance.

Exercise 7: Use Landmarks to Update your Dead-Reckoned position.

In this exercise you are going to use color recognition to correct the drift errors of your robot position calculations. The course is set up with two green colored corners. Your final goal will be to determine when you are at a certain colored corner and update your robot's X,Y position to that corner's coordinates.

As a first step add your wall following code to this new project. Set the robot's speed at about 1.0 tiles/second. You can work on speeding up the robot when you get to the final contest. Once you have the wall-following working add code to determine when the robot is in any corner. Then use the compass to determine which corner you are in.

The above step works well if the robot is only performing the wall following of our square course. If the robot finds itself in a different corner (say in the center of the course) it will be tricked to think it is at one of the outside corners. For that reason we need to use the camera to determine when we are definitely at an outside edge corner. To this point your vision algorithm only looks for one color when processing the image. You will need to modify your code so that multiple colors can be recognized in order to search for both green landmarks and light blue golf balls. It may be possible to search for the two colors each image sample but that will add a lot of processing load on the DSP. (Have your TA show you how to use a DAC output and the oscilloscope to check your time loading.) For this exercise, alternate which color is being searched for each time you come in the vision processing function.

When you determine that you are at a landmark, update your robot's X,Y coordinates to the approximate coordinates of that corner. Also increment an integer variable indicating the number of times you have been at this landmark. Display this number to the text LCD screen. Now your VB application should be able to display the robot always staying inside the square of the course.

Lab Check Off:

1. **For three (2,3 and 5) of these check-off items, your VB application needs to be working and displaying the location of your robot on the screen.**
2. Demo your robot following a bright light. If the robot gets close to a wall it should stop.
3. Demo your robot following a colored object.
4. Demo your robot displaying the distance to a light blue golf ball lying on the floor.
5. Demo your robot performing wall-following and correcting its dead-reckoned position when it is at a colored corner landmark.