University of Florida
EEL 5666
Intelligent Machine Design Lab

Anthony Huereca
Spring 2004

CMUcam

Introduction

The special sensor I'm using for my robot Sbob is the CMUcam, developed by Carnegie-Mellon University as a low cost and low power vision solution for robots. It consists of a SX28 microcontroller paired with an OV6620 Omnivision CMOS camera, which communicates with the microprocessor via a RS232 serial port. The camera includes an instruction set to find the mean color of an image, or the center of an object, or even track an object as it moves around the viewing area. It also supports a screen dump of the raw image data from further processing. It is a very versatile piece of hardware, and allows for many image possibilities.

Operation

The operation of the camera is fairly simple. It reads in ASCII commands from an instruction set that is listed in the CMU manual. Each instruction is then entered by sending a \r (ASCII 13) at the end of the string. The camera then sends back either ACK\r or NCK\r depending on if the instruction was received correctly or not. Then, depending on the command sent to the camera, as well as the mode that it is in, one of six different types of output packets is returned. These are also all in standard ASCII. The exception for this is the DF command, which outputs the raw camera data column by column.

Once a command has been executed and the camera is ready for another command, it outputs a ":".

After receiving the output packet, it is up to the program on the microprocessor to parse the string and get the important values out of it.

The receive and transmit lines on the microprocessor board are backwards of what they should be. If you connect the camera to a laptop, or the board to a laptop serial port, then it will work correctly. However the transmit and receive lines need to be crossed so that the transmit line from the camera goes to the receive line on the board, and vice versa.

Software

The job of the software is to write out the commands to the UART port on the board, and then read in the replies that it gets. The STK500 comes with a standard 9 pin serial header, which is then connected, to UART0 in the microprocessor. Using AVR GCC's built in string library, I can then parse the strings that get sent back from the camera, and use that to read the values received.

For my robot I configured the setting on the camera to run in poll and raw modes. Raw mode makes it so the data that the camera sends back is the numerical values instead of ASCII values. It defaults to ASCII since it makes it human readable on a terminal screen for use on a laptop. However for use on the microcontroller, it is much faster and easier to read the straight numerical values so that no conversion is necessary. I also suppressed the ACK/NCK reply since I had never had it not correctly receive a command, and it is just one more thing to read in that isn't really needed. Poll mode is also very useful when using it with a microprocessor since you only get one reply per command sent. Otherwise after sending it a "GM" or "TC" command, it will continue to send back data packets until you send another command. This causes extra unneeded interrupts to fire, and it is much easier to debug and

control this way.

I also set the baud rate to 19200, which was the fastest available using the ATMega128 that had a low percent error. The camera supports several different rates set by placing jumpers, created via two female headers, over certain pins on the camera, with the drawback being that the lower baud rates results in decreased number of frames per second of image data available. For the applications I was using however, I didn’t need an incredibly fast FPS, and 19200 was more than adequate.

I originally tried to get my robot to detect red via the “GM” command which calculates the statistical mean and deviation of the Red, Green, and Blue values of the current image. This however didn’t work very well since it created a lot of false positives, and it required the red object to completely saturate the camera viewing area to get a large gain the red mean value. I’ve included the experiments I did on this part in the interest of completeness, even though I did not end up using it.

I then played around with the “TC” command which Tracks a Color based on the min and max values for the Red, Green, and Blue channels that you provide which range from 16 to 240. It then returns a packet that has the bounding coordinates of pixels that it finds that fall within those color bounds, as well as a confidence value is then calculated from the ratio of the number of pixels falling in the color range and the area of that bounding box. This value ranges from 0 to 255. This produced far more accurate results and it detects a red object from much further away. It “detects” if the object is red or not based on its confidence reading. The manual says anything below a reading of 8 should be ignored, but I found that it can get as high as 15 without there being an object in view, so I set my threshold at 20. When a red object is placed within a foot of the camera using the default color tracking parameters in my code, it receives a confidence value of over 200. The camera also sends back the middle x and y values of the tracked object, which can be used to track the object

fairly successfully. There is also the option to output a PWM signal to control a servo to try and track a moving object.

The main limitation to the CMUcam is something that is prevalent to most cameras, and that is that it doesn't work well under low lighting conditions. The response time to a red object being placed in front of the camera was much longer in my dorm room, up to 3 seconds, versus less than half a second in the brightly lit IMDL lab. The distance that an object could be from the camera to detect red was much higher in the IMDL lab than in the dorm as well. This required certain threshold distances and wait times to have to be adjusted when working in different areas.

It was with this in mind that I tried to extend my original goal of a red charging robot into a robot that would charge any color held in front of it, as well as calculating the time lag and readable distance. I had my robot working with hard coded values, and the curious engineer in me wanted to find out if I could do it dynamically at start up. I also thought it would help in testing the robot since I could then easily adjust for the light differences between working on it in my dorm and in lab.

If an object is held within 6" of the camera at start up, then it will first take a screen dump of the object using a very small window, and calculate the color tracking thresholds via a simple algorithm that looks for the minimum and maximum values of the red, green, and blue channels, while also making sure that any abnormally high values are discarded. The CMUcam comes with a command that does this automatically, the TW command by which the TC parameters are calculated by taking the mean values for the RGB image and adding +/- 30 to each of them to get the tracking parameters. I thought I could maybe improve on its performance since it doesn't do as good of a job as the hardcoded values and sometimes worked very poorly. After that is accomplished, it asks the user to lift the object out of the way, and

then reinsert it. It then starts a counter which determines the lag delay in detecting the color. Then the user is to pull the object straight back until the camera no longer gives a good reading of the color tracking confidence, and it sets its distance variable that way. In theory this scheme will enable Sbob to work in any initial lighting condition, and dynamically calculate the required “wait” time after detecting an object in front of it, as well as the maximum distance that it can correctly identify a colored object.

However I was unable to get it to work consistently or very well. I found that blue gave a lot of false positives because the RGB tracking values were very wide, and other colors often didn’t do much better. It did work fairly well on red though. I also had trouble getting the timing down, as it did not match up to what I calculated by placing it back in front of the camera. It would start a counter based on when it was reinserted which was determined via the sonar distances, and then count until the confidence value raised above 150. The distance value was then calculated by keeping a record of the greatest sonar values until the confidence fell below 100.

A better color analysis algorithm and more time to try and tweak the code more would help I’m sure. A future robot could maybe use these ideas to implement something that dynamically tracks different colors based on certain input, or can calibrate its thresholds on the fly.

Experiments

Imaging

Using the java program that came with the camera, it was simple to get a screen dump of what the camera is seeing. The first thing done was to focus the camera, which is done by twisting the lens in and out.

I've read a lot about how different lighting conditions can affect the quality of the image produced, so I tried taking images under various conditions. To illustrate the differences, I used the camera to take a picture of a set scene. There is a dark

green folder, a red folder, and a blue folder resting against a black toolbox with a yellow latch. Using sunlight, the test image looked like this:

In my dorm room, which is lighted by florescent lighting, I got an image that looked like this with white balance off:

And then when white balance was turned on, the image looked like this:

The sunlight image is dark because it was a cloudy day and dorm windows aren't very good at letting in sunlight. With white balance turned on, the colors show through a little more and the image doesn't look quite as dark.

I also tested the camera in the IMDL laboratory, since that is where the official demo will take place. This time I used a yellow multi-meter, blue pliers, and red wire to show the different colors. Without white balance on, the image came out like below:

With white balance on, it looked like the image below:

Again the image with white balance on tends to give truer color and looks less red overall, which is will be very important in detecting red then. The camera also uses a new special coating to decrease the effect of IR light, and that seems to help out with the image quality as well.

Mean Value

I initially tried detecting red by calling the "GM" command from the camera which returns the median color value of the current image as well as the deviation as

calculated by the camera. Being that I want it to find red objects, the color I was most concerned with was the Red value and its deviation.

The camera can update this value at up to 17 times per second. The first thing that is noticeable is that the median color value and its deviation are not very stable, and can vary wildly between tests. Also just moving the camera around can affect how it sees its surroundings.

As expected, turning the white balance on and off can greatly affect the mean value achieved, as well as the deviation. By trying both methods, I found that turning the white balance on achieves the greater disparity between seeing “red” and not seeing red. This was tested by having the camera pointed at a neutral scene, and then putting a bright red folder in front of it to see how the mean color changed. After a large initial jump, the red channel would settle down to a value usually about 20 points higher than a neutral scene. Also the deviation would go drastically down, up to 30 points.

The distance that the red object was held also greatly affected the mean color value. With the folder less than a foot away, the red mean value jumped up dramatically, up to 150 and 160, and the deviation dropped into the teens. However at about 3 feet away, there was a very small increase in the red mean value, often only going up maybe 5 or 10 points, with the deviation dropping only 5.

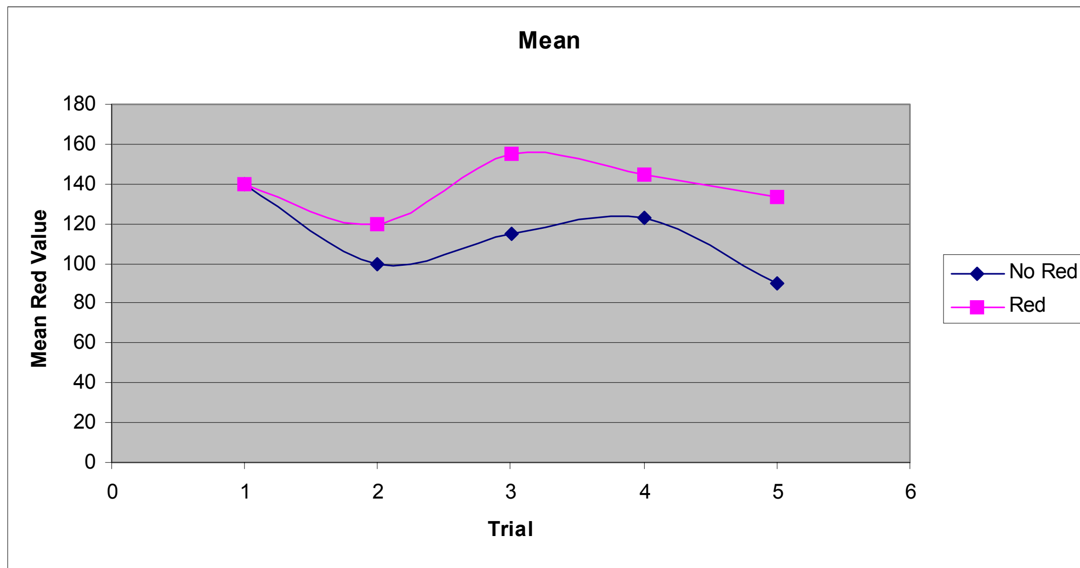
Below is a table of a few readings I made with white balance on at about a foot away. This should show the fluctuation in readings, but also that there is a mean gain

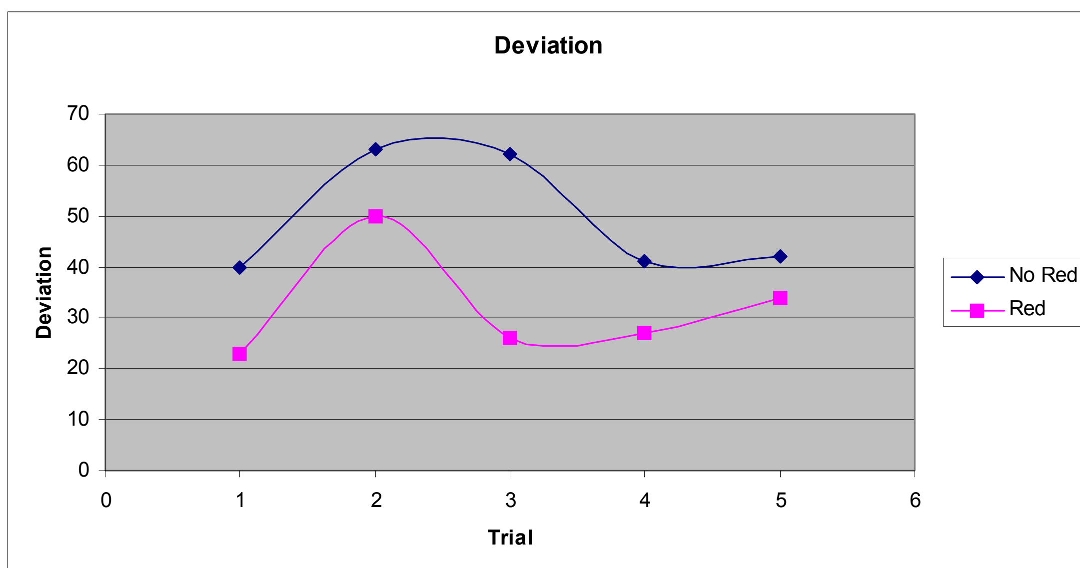
Neutral Scene

Mean	Deviation
140	40
100	63
115	62
123	41
90	42

Red Scene

Mean	Deviation
140	23
120	50
155	26
145	27
133	34





From these results, I decided the best course of action would be to take in

account both the mean and deviation. An initialization step after each turn, because that would represent a new “scene”, would be needed so that the camera knows when a red object has been placed in front of it. The actual values may not be very constant, but there is always a jump in the red mean value, as well as a drop in deviation, when a red object is placed in front.

However as I stated above, this scheme gave many false positives, and required the object to be very close. It has worked for other robots when detecting objects in a static settings or a pre-determined background, but for what I wanted to accomplish it worked poorly. Thus I abandoned this method in favor of the Track Color method.

Track Color

I then experimented with the TC command, which proved to work far better for my robot. Many of the CMUcam reports from previous semesters had already used it to track red, so I used their parameters of 100-240 for Red, and 16-50 for Green and Blue. I found these worked in most lighting conditions and gave very few false positives while always giving correct values when red was held in front of the camera. Also having white balance on or off did not affect results much, and so I left it off. The two main variables then became how long it took the camera to recognize it was red, and how far away the camera could detect red. The time issue was important because it is not good to have a robot just sitting around and I wanted to minimize that delay time as much as possible.

The results from testing it in my dorm room are as follows:

Inches Away	Confidence Value
3	251
6	255
9	230
12	225
15	220
18	190
21	150
24	2

As can be seen, the camera almost acts like a digital switch on its confidence value. It floated between lower to mid 100's at about 22-23 inches, and just a little bit past that would drop to close to zero. Also the time lag from placing the object in front of the camera grew as the object was placed closer to the lenses. It ranged from approximately 2 and a half seconds when at about 21 inches, versus over 4 seconds when held 3 inches away. Also the camera is fairly finicky at the longest distances, and a slight shift in position or angle can cause a low reading to go rise to a confidence value of over 100.

In lab it performed far better. The results are as follows:

Inches Away	Confidence Value
3	255
6	255
9	255
12	254
24	230
36	170

It continued to track a red object over 15 feet away until I reached the wall. The response time was also much faster, and there was no noticeable delay in putting a red object in front of the camera and the confidence value rising.

Conclusion

The CMUcam is a very versatile piece of equipment, and it is nice to have a lot of image analysis functions built into the camera. Many of the camera problems that previous reports commented on such as red saturation, misaligned lenses, and narrow viewing field seem to have been fixed. I did find it to not be very consistent and it still has a lot of trouble in low light. However it performed well for red color detection, and overall I was happy with the camera.

References

CMUcam Website
http://www-2.cs.cmu.edu/~cmucam/home.html
CMUcam Manual
http://www.seattlerobotics.com/CMUcamManualv15A.pdf
CMUcam Lens Information
http://www.seattlerobotics.com/lensinfo.htm
Brian Ruck's Nigel Report
http://mil.ufl.edu/imdl/papers/IMDL_Report_Spring_03/ruck_brian/cmu.pdf
Hubert Ho's Brutus Report
http://www.mil.ufl.edu/imdl/papers/IMDL_Report_Fall_02/ho_hubert/brutus.pdf