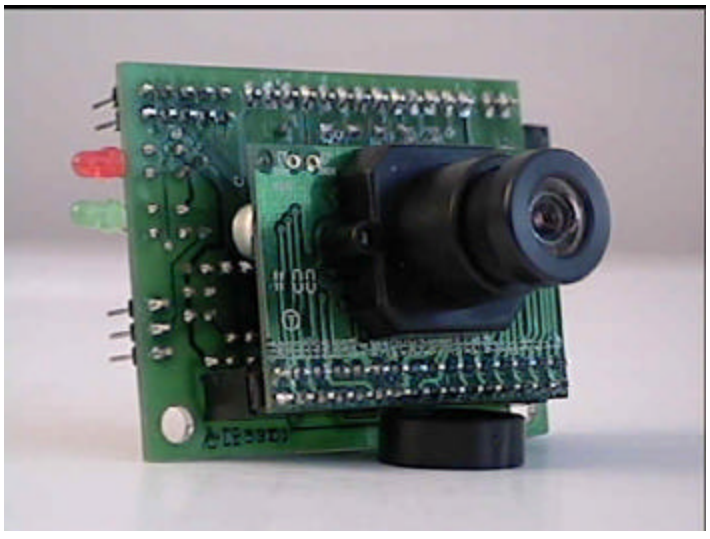
Special Sensor Report:
CMUcam

David Winkler
12/10/02
Intelligent Machines Design Lab
Dr. A. A. Arroyo
TAs: Uriel Rodriguez
Jason Plew



Introduction

This report covers the CMUcam and how I was able to use its color tracking features in my project. My robot, Chester, used the CMUcam to find a bright red ball placed at the feet of a human requesting service from the robot. I purchased my CMUcam from Acroname. The total cost of the kit was about 115 after shipping and it came unassembled. There was one discrepancy between Acroname's part lists and the diagram of the built system in the CMUcam Manual. To build it correctly, just follow the diagram in the manual. Seattle Robotics also sells the camera for the same price and it comes preassembled. The figure below shows the assembled board. One thing to note though is that the camera from Acroname comes with an IR filter already install over the lens. I believe the IR filter is available from Seattle Robotics also, but you must mention it when you order. I had success with the camera from Acroname, so I would recommend it over Seattle Robotics' version.

Camera Features and Settings

CMUcam is a low-cost, low-power sensor for mobile robots. The camera has several real-time image processing features and is easily compatible with many microcontrollers via the serial port.
At 17 frames per second, CMUcam can:

- track the position and size of a colorful or bright object
- measure the RGB or YUV statistics of an image region
- automatically acquire and track the first object it sees
- physically track using a directly connected servo
- dump a complete image over the serial port
- dump a bitmap showing the shape of the tracked object
- supports multiple cpu boards on one camera for parallel processing

Some things to note about the CMUcam is that it is a special purpose camera whose primary function is basic image processing. Some people have tried to use it as a regular camera to acquire images and have been greatly disappointed. If you want to take pictures, consider another camera. Some additional features from the CMUcam manual include:

- Arbitrary image windowing
- Adjust the camera's image properties
- Dump a raw image
- 80x143 Resolution
- 115,200 / 38,400 / 19,200 / 9600 baud serial communication
- Slave parallel image processing mode off a single camera bus
- Automatically detect a color and drive a servo to track an object upon startup
- Ability to control 1 servo or have 1 digital I/O pin

The only settings of the camera I changed were the Polling mode and Raw Data mode settings. I tried messing with the auto gain and auto while balance settings but I found that these worked best in their default state. Polling mode is useful because it reduces the

overhead on a processor. When a camera is sent to the camera with polling mode off the camera will continuously send data packets at 17fps if the camera is operating at 115,200 baud. This can exhaust most of the processor resources. Instead, in polling mode, the camera only sends back one packet of information. Thus, the processor has time to do other operations and can control exactly when it wants to receive a packet of information from the camera. Another nice setting is raw data mode. Raw data mode configures the camera to return packets in raw data bytes rather than in printable ASCII characters. This again saves time because you don't have to go to the trouble of converting the ASCII to a raw data byte because in raw data mode this has already been done for you.

Camera Command Set and Commands Used

The camera has several different serial commands which are well documented in the CMUcam manual. For my robot, the camera is fixed onto the front of the robot so there is no need for servo movement. The only commands I ended up using besides setting polling mode and raw data mode were the track window and track color commands. I used the track window command with no arguments. Track window with no arguments calls the get mean command for the center of the currently set window. The get mean then extrapolates the color information and statistics and passes this to the track color command. The next time you call the track color command, it has memory of the parameters used in a previous call. So calling a track window command with no parameters followed by a track color command with no parameters allows you to easily track a defined colored blob at the center of the window.

For Chester, I put the object I want to track 5 inches in front of the camera. Then, I initiate the track window command with a back bumper press. This essentially trains the robot to see the object which is being held in front of the camera. Now subsequent calls to track color with no parameter will return a data packet of the following format:

Type M packet
M mx my x1 y1 x2 y2 pixels confidence\r

The M indicates that the camera is returning a middle mass TC packet. Mx is the middle mass x coordinate of the tracked region and My is the middle mass y coordinate. The next four bytes give the coordinates of the tracked rectangle. Pixels is the # of pixels in the tracked region capped at 255. Confidence indicates how well the camera has a lock on the desired color. A confidence > 50 indicates a good lock, while a confidence < 10 indicates a poor lock.

Chester uses an interrupt service routine to periodically check the confidence level for color tracking data and is able to determine if the colored object is nearby. For proximity measurements, # of pixes will be used to determine if the object is close or far away to Chester. Mx will be used to align the robot with the object and will keep the robot from veering of to one side.

The following pseudo code is essentially what I used to track and follow the bright red ball.

Reset CMUcam
Put camera into poll mode and turn on raw data mode for returned packets from camera
Call Track Window
LOOP_START
Call Track Color
If # of Pixels > 250 And Confidence > 50 Then Go Backward
If Middle Mass X > 55 And Confidence> 25 Then Go Right
If Middle Mass X< 35 And Confidence > 25 Then Go Left
If # of Pixels < 120 And Confidence > 25 Then Go Fwd
GOTO LOOP START

To test this algorithm I used the setup shown in the image below:

Results

One thing I learned from this project was the importance of lighting in image processing applications. Chester's performance varied greatly depending on the lighting where the experiments were performed. Standard incandescent lighbulbs worked ok, but florescent lighting such as that in the lab works much better. In the lab environment, Chester recognizes objects from 7-9 feet away while in my apartment Chester will recognize objects from only 3-5 feet away. Also, the color red seemed to work the best. I tried bright yellow objects and orange objects, but the robot would get confused and would sometimes recognize the wrong object.

The following code written in CodevisionAVR may be of some help please reference the Seattle Robotics website also as they have written a very similar example.

EXAMPLE:

```
printf("RS\r");     // Reset the camera
delay_ms(100);   // must delay …can delay less than 100ms
printf("\r");
delay_ms(100);
printf("PM 1\r"); // turn poll mode on
delay_ms(100);
printf("RM 3\r"); // turn on raw data mode for packets received from camera
delay_ms(100);
printf("SW 1 1 80 143\r");  // not necessary this is default window
while(adc_data[2]!=47 || adc_data[2]!=46){};          // wait for front bumper press
printf("TW\r");        // track color of object directly in front of robot
while(adc_data[2]!=88); // wait for back bumper press to begin
while (UCSRA.7) temp=UDR; //flush the receive buffer
while (cpacket[9]<15) {
   count=0;
         drive(1,-1);
         delay_ms(100);
         drive(0,0);
         printf("TC\r"); // track color obtained from TW command
         // see manual for packet description..11 bits per data packet
         while (count<11) {
          cpacket[count]=getchar();
          count++;
         }
         PORTC=~cpacket[9];
}
while (UCSRA.7) temp=UDR; //flush the receive buffer… very important!
delay_ms(100);
while (1) {
         count=0;
         printf("TC\r"); // track color obtained from TW command
         // see manual for packet description..11 bits per data packet
         while (count<11) {
          cpacket[count]=getchar();
          count++;
         }
         PORTC=~cpacket[9]; //show confidence value on portC

         // 45 is H center
         if (cpacket[8] > 250 && cpacket[9] > 50) {
         drive(-1,-1);
```

```
            drive(0,0); //Bwd
            }
//          if (cpacket[2] > 55 && cpacket[9] > 15) drive(0,1); //Right
//          if (cpacket[2] < 35 && cpacket[9] > 15) drive(1,0); //Left
//          if (cpacket[8] < 120 && cpacket[9] > 15) drive(1,1);  //Fwd
//  else drive(0,0);
            delay_ms(100);

}
```