# Omni-Directional Vision System for Mobile Robots

# Critical Design Review

Presented to Professor Ricardo Gutierrez-Osuna
on
March 5, 2003

by
The A Team

**Denise Fancher – Kyle Hoelscher – Michael Layton – Eric Miller**

## Introduction

The purpose of the Critical Design Review (CDR) is to review and summarize our current position on the omni-directional vision robot project. In the review, we cover the selection of the "mirror," BOE-Bot assembly, tracking color hard code, obstacle avoidance options, the pseudocode and camera commands, and risk status and scheduling issues.

## Omni-directional mirror selection

Our original proposal included an omni-directional mirror manufactured by Neovision with a hefty price of $290.00. After consultation with our advisors, we concluded that we really do not need the accuracy that this advanced mirror would offer, thus we should be able to find something much less expensive to fulfill our requirements.

After much brainstorming, one of our group members found a spherical lightbulb with a silver chrome finish on the bottom half at Lowe's Home Improvement store. Actually, the lightbulbs are sold as pairs – two for $2.97. Now if our tiny BOE-Bot tips over and breaks our "mirror," the consequences will not be nearly as severe as they would have been with a $300 mirror.

## Fastening the camera and "mirror" to the BOE-Bot

We considered two designs for fastening the camera to the BOE-Bot (Board Of Education robot). The camera will require power source and ground connection wires to connect the camera to the BASIC stamp. The wires will serve as a serial interface for the

input and output ports. The two wires will be connected to the specified pins on the BASIC stamp.

Our next hardware assembly issue is fastening the lightbulb to the BOE-Bot. We have to be very careful that the bulb is the desired distance away from the camera lens to get the optimal image while not creating a top-heavy contraption. We initially had two different designs for this assembly.

The first design consists of attaching a light bulb fixture to the top of the camera. We can then screw the bulb into the fixture. We can fasten the light fixture above the BOE-Bot at the specified distance using three rods to hold the fixture above the robot. Once in place, we can screw the bulb into the fixture.

The second design for this issue is to first fasten a circular ring above the robot using three rods to secure the ring. The ring will have a radius that lets the bulb sit in the ring meanwhile providing the largest viewing area possible on the bulb for the camera. We will then place the bulb into the ring above the camera lens.

The solution we chose for mounting the bulb onto the robot is the ring configuration. We chose this solution for several different reasons. Placing the bulb into the ring will allow more movement for the bulb. We may create the mount so that the ring height can be adjusted. The distance from the camera lens will be very important for obtaining an optimal image on the bulb. We also concluded that the height will be easier to calculate

with the ring design. We will not have to take into consideration the thread length when figuring the height.

**Hard coding the tracking color**

One issue we also need to address is if we can change the tracking color after the robot initially locks onto the target upon power up. When the robot initially powers up, it will lock onto a specified target that we place in front of the lens. In order to change the color of the target, we will have to power down and turn the robot back on. Then once again during the initial lock on phase, we can place the desired color in front of the lens.

We can change the color of the target after the robot has locked onto another target. This will require us to hard code the color into the BASIC stamp using the CMU camera commands. We can do this by using the trace color command (TC). This command will take the minimum and maximum RGB value to distinguish the desired color. In order to implement this into the hardware, we will need to configure a switch that will activate the corresponding command for that particular color.

**Obstacle avoidance algorithm options**

One issue we will have with our object tracking project is obstacle avoidance. We have two different approaches to this problem. The first solution involves using potential fields to avoid obstacles while staying locked onto the target. The second solution is to implement an algorithm that will avoid the obstacle using basic forward and backward commands to the servos.

The first algorithm involving potential fields is a fairly common algorithm used for obstacle avoidance. The main advantage to this algorithm is that it will give the BOE-Bot a smooth path to the desired destination. This algorithm will take the sum of the two force vectors and travel in that direction. When the avoidance hardware detects an obstacle, it will generate a force vector to tell the robot to avoid the obstacle. Meanwhile, another force vector is always present which tell the robot where to go to get to its destination. The potential field algorithm will take these two force vectors and sum them up. The sum will be the vector path the robot will take.

The second algorithm we considered is simple and less expensive. The algorithm will use the sonar in front of the robot to prevent it from bumping into obstacles directly ahead of the robot. The robot will redirect itself when the sonar picks up the obstacle. When the robot comes into contact with obstacles on the side, its whiskers will touch the obstacle. The robot will maneuver away from the obstacle, and then go back to chasing the target.

The main reason we chose this second algorithm is because we are restricted to budget and complexity. The potential fields algorithm requires more sonar or infrared devices to detect obstacles surrounding the robot. Our whiskers would not supply us with the necessary data to implement a potential field obstacle avoidance algorithm.

**Pseudocode**

*Obstacle avoidance*

Obstacle avoidance will be achieved by integrating the output from a sonar unit pointed in the forward direction and the two whiskers attached on the left and right sides of the BOE-Bot. In our implementation, these devices will be active low: 0 means that an obstacle is detected while 1 means that nothing has been detected. From the sonar unit's output, we can calculate the distance between the BOE-Bot and the object detected. If this distance is less than 4 inches, we will set a flag to 0, indicating that the sonar has detected an obstacle. If the distance is greater than 4 inches, we will set the flag to 1, indicating the sonar has not detected an obstacle.

By polling the left whisker, right whisker, and sonar unit, we can set a binary variable that will tell us in which of the eight possible cases the BOE-Bot currently is. We are using the format described in the following table. Based on the case in which the BOE-Bot finds itself, an action or series of actions may be taken. The basic actions are stop, back up, turn left 90°, turn right 90°, turn left gradually, turn right gradually.

| Obstacle Avoidance Algorithm | | | | | |
|---|---|---|---|---|---|
| **Case** | **Left Whisker** | **Right Whisker** | **Sonar** | **Interpretation** | **Action** |
| 7 | 1 | 1 | 1 | Blocked left, right, front | Stop, back up |
| 6 | 1 | 1 | 0 | Blocked left, right | Stop, back up |
| 5 | 1 | 0 | 1 | Blocked left, front | Stop, turn right 90° |
| 4 | 1 | 0 | 0 | Blocked left only | Turn right gradually |
| 3 | 0 | 1 | 1 | Blocked right, front | Stop, turn left 90° |
| 2 | 0 | 1 | 0 | Blocked right only | Turn left gradually |
| 1 | 0 | 0 | 1 | Blocked front only | Stop, turn right/left 90° |
| 0 | 0 | 0 | 0 | Clear | Continue in current direction |

*Tracking algorithm*

The tracking algorithm will use the object coordinates received from the CMU camera. The coordinates will be separated into X-Y coordinates. The image the CMU camera will see is a 144 by 80 pixel image. With the position (0,0) located in the top left corner, the robot will be located at the center of the image at position (72,40). We will also implement a circle around the robot, which will define our comfort zone. The comfort zone will be the ideal distance the robot will attempt to stay away from the object. If the object is inside this comfort zone, then the robot will back away from the object until the object is outside the comfort zone.

The object coordinates will be used to control the servos speed and direction, clockwise or counterclockwise. We will find the radius distance from the object by taking the square root of the sum of the X-Y coordinate position. We will use the X coordinate to find the direction the robot should travel. If the X coordinate is less than 72, then the

robot will move the right servo faster than the left servo to turn left. If the X coordinate is greater than 72, the servos will move in the opposite direction. As the robot moves in the direction of the object, the radius and Y values will become smaller. The speed of the robot will be controlled by the radius and Y values. If the values are large, the robot will move faster to the target. As the values get smaller, the robot will slow its approach until the object's distance from the robot is equal to the distance of the comfort zone.

*CMU commands*

In order to have the CMUcam communicate with the BOE-Bot, we must be able to issue commands to poll the information off the camera and back to the onboard memory so that the appropriate actions can be taken. There is an initialization period that must occur on the camera that allows it to set the tracking window, initial tracking color, and sets the appropriate commands in the camera registers. In order to poll the information from the camera, the Stamp will issue a `SEROUT` command with the appropriate camera command followed by a `SERIN` command to accurately deal with the return information. When the camera receives a command, it will return the appropriate data values and `ACK` (for successful acknowledgement) or `NCK` (for command failure).

Initialization will consist of `SEROUT` commands in a particular order as specified by the manufacturer to set up the camera while the `SERIN` will simply listen for an acknowledgement from the camera to insure the command was taken. After the initialization process has occurred, there is an option to redefine the color to be tracked or leave it as the color that was captured during the initial set up. During the actual running of the robot, assuming there is no color change, the only commands necessary to issue in

order to poll information from the camera are `SEROUT outpin, 240, ["TC",CR]` where outpin is the output pin connected from the stamp to the camera and `SERIN inpin, 240, [STR CamData\10]` where inpin is the input pin connected from the camera to the stamp. By calling `SERIN inpin, 240, [STR CamData\10]`, this will take the returned values from the track color command and place them into a string array of size ten for easy access.

There will be actually only a few commands that will need to be issued to the camera. These commands are `L1, PM, TC, TW, RS, RM,` and `CR.` More detail about these particular commands can be found in the CMU cam documentation but a brief synopsis is given below:

`L1` – This command controls the green LED on the camera board. While it serves no real purpose, it can be useful in testing and error checking. The command has three arguments that can be issued to it: 0, 1, and 2(default). 0 disables the LED, 1 turns the LED on, and 2 enables auto mode so that the LED is on when tracking and off when the desired color is not present.

`PM` – PM controls the poll mode status of the camera board and limits information coming back across the serial line. It has two arguments that can be issued: 0 (default) or 1. 0 disengages poll mode and the camera returns multiple packets of info to the camera, and 1 enables poll mode to allow only one packet of data to be returned. Enabling poll mode is used for

rapidly changing data or a microcontroller that runs to slowly to keep up with the frame rates on the camera.

`TC` – Track Color is issued as the $R_{min}$, $R_{max}$, $G_{min}$, $G_{max}$, $B_{min}$, and $B_{max}$ where each indicates an intensity value in the range of 0 – 255. When the track color command is issued without any arguments, the last color values used will be utilized again. Track color returns information about the object being tracked and its location in the screen.

`TW` – Track Window has no arguments and will simply look to the center of the active window on the camera to obtain a new color and then issue the track color command on its own to track that new color. Since it calls track color, similar return values will be produced.

`RS` – This command resets the camera board to its original state. It returns all the registers to default values and wipes information from the camera board.

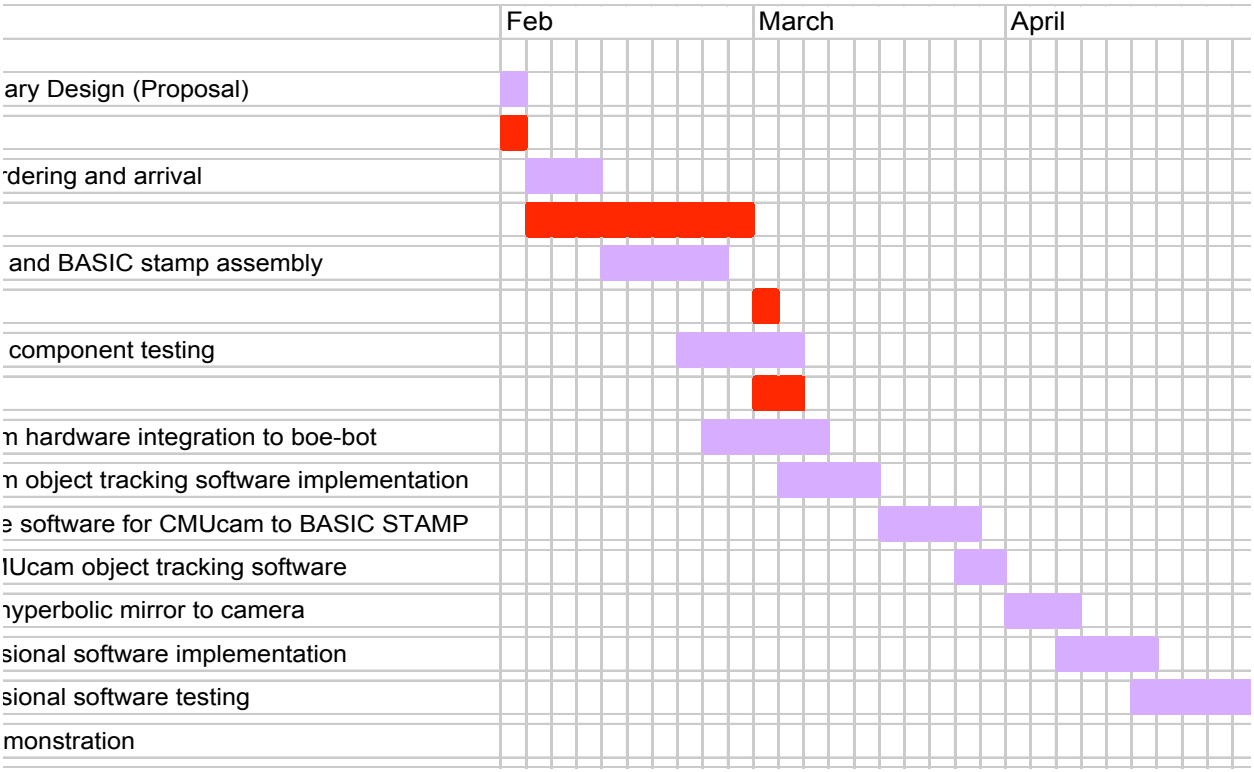
`RM` – This command causes data to be transferred in raw serial transfer mode. It reads a three-bit number and sets the transfer according to how those bits are activated. If the LSB is enabled, then the output to the camera is in raw bytes. If the middle bit is enabled, then the `ACK` and `NCK` returns from the camera are suppressed. If the MSB is enabled, then the input to the

camera is in raw bytes. For this project, only the LSB and middle bit will be enabled.

`CR` – Altering the camera's internal registers allows for more flexibility and control over the system. A multitude of registers can be modified to control the saturation, brightness, contrast, clock speed, color mode, band filter for fluorescent lights, and auto adjusting for white balance and gain. Essentially this command will be used only to activate band filter and ensure that the auto white-balance is activated.

**Risk status and scheduling issues**

At this point, we have experienced only minor delays in our proposed schedule. Since the parts arrived later than expected, we were slightly behind schedule. We thought there was a chance this might happen, and allotted extra time for construction. At this point in time we are still on schedule. The next major completion deadline will be to have the basic tracking software functioning by the end of March. This seems entirely feasible. We have already begun pseudocoding the omni-visional software, placing us ahead of schedule in this area.




**Conclusion**

We have refined the design of our omni-directional vision system for the BOE-Bot in many ways since the submission of our original proposal. We found an inexpensive substitution for a very expensive omni-directional mirror. We conceived the idea of having a comfort radius to guide the speed of the robot when tracking its target. We explored obstacle avoidance options and determined that a simple algorithm will meet our needs, and that it can be implemented with the single sonar unit that we requested originally. We have a clearer understanding of our project and the specific steps we need to take to reach our goals.