Omni-Directional Vision System for Mobile Robots

Final Documentation

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

by
The A Team

Denise Fancher – Kyle Hoelscher – Michael Layton – Eric Miller

Table of Contents

Omni-Directional Vision System for Mobile Robots

Proposal for Research

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

by

The A Team

Denise Fancher – Kyle Hoelscher – Michael Layton – Eric Miller

Table of Contents

Introduction

The A Team proposes the development of an omni-directional vision system for mobile robot navigation. We will deliver a mobile robot equipped with an imaging system consisting of an omni-directional mirror and a small video camera. The robot will also be outfitted with an obstacle detection and avoidance system to prevent collisions.

The primary goal of our project (Phase I) will be the implementation of a video tracking system to induce our robot to follow a desired target object. The robot will "lock on" to a brightly colored object which it will then pursue. The robot will move towards the target, constantly monitoring the target's position, and if the robot "catches" the target, the robot will stop a specified distance from the target. The robot should move without colliding with walls or other obstacles in its path.

We have two options for an expansion phase for our project (Phase II) should time, budget, and expertise allow us to pursue additional complexity in our system.

1. *Interaction of two robot systems in a gaming manner such as tag*

   This option would require the purchase and fabrication of a second robot and omni-directional imaging system, after which we could program one robot to "chase" the other one. The original Phase I robot would "lock on" the new robot and follow it. The Phase II robot would be programmed to move in the opposite direction from its given target, and thus the robots would engage in a game of tag. Constraints would be enforced so that the robots would not actually collide with one another, but instead the game would be over when the chasing robot gets within a specified distance of its target robot.

2. *Remote robotic control system utilizing a live video feed to allow an individual to steer the robot in any desired direction*

   This option would allow for a second operational mode to implemented on the original robot system. A switch or other method for choosing between operating modes would be added. The original Phase I tracking would be one of the operating modes, and we would develop an alternative operating mode which would allow a computer user to control the movements of the robot remotely. The omni-directional image acquired by the robot would be transmitted back to the computer and unfolded for panoramic viewing. The user could then see the exact location of the robot and its surroundings, and he could simply click on a region of the robot's view to tell the robot to go in that direction. We would implement an algorithm to convert the coordinates on the user's view into a format where we could tell the robot to move in the specified direction.

Background Information

Omni-directional vision is the ability to see in all directions from a given center. Rather than using a traditional camera to scan 360 degrees or using several cameras to gain a 360 degree view, many researchers are now using a standard video camera pointed upwards at a convex mirror. The resulting view can be used in its raw form, or it may be unwarped or reprojected onto a cylindrical surface by mapping its spherical coordinates to cylindrical coordinates (see Figure 1).

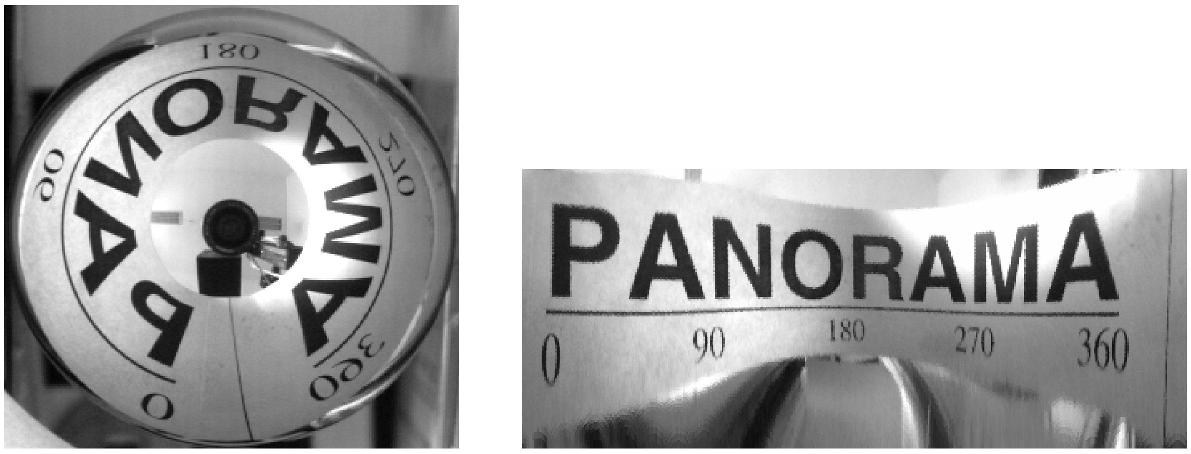


Figure 1. Omni-directional image captured by camera viewing convex mirror (left); same image reprojected onto cylindrical surface (right).

Applying omni-directional vision to the field of mobile robotics could yield more efficient and dexterous robots. A robot with standard single-direction vision may lose a target it is tracking if the target moves out of the robot's field of view. The robot would then be forced to stop and scan for the target before resuming the tracking operation. Omni-directional vision eliminates this problem by expanding the robot's field of view to 360 degrees.

Design Objectives

- Gain knowledge of omni-directional and computer imaging and mobile robotics
- Design and construct the omni-directional imaging apparatus for our mobile robot
- Implement software to control the movement of the robot based on omni-directional images acquired
- Gain experience in the engineering design process including planning, ordering materials, hardware and software implementation, testing and validation, and delivery
- Practice good project management and planning techniques to ensure delivery of our project by the end of the semester

Method of Solution

Survey of Literature, Market, and Existing Components

The project advisors provided us with a website that has links to many projects dealing with omni-directional vision. It was from this site, The Page of Omnidirectional Vision, that we obtained most of our information.

The current market for omni-directional vision applications includes not only mobile robotics, but also video surveillance, security, and virtual reality systems.

We know that the Boe-Bot and the CMUcam are compatible because Parallax sells them together, and we have seen several video clips of these systems in action. Because the sonar and radio frequency transceivers are all sold as accessories to the Boe-Bot, we know that they work individually. Other Boe-Bot experiments with cameras, but not omni-directional

Design Constraints

There several limiting constraints that will affect the design and implementation of the omni-directional vision system for our mobile robot.

- Time – Our project must be completed by the end of the semester.
- Budget – The amount of money we can spend on our project is limited by the professor's discretion.

- Component Availability & Shipping Delays – We must use the components that we will be able to obtain in a timely manner.
- Knowledge – Considering the short time we have studies omni-directional vision and mobile robotics, our design reflects our basic understanding of these subjects.
- Object Avoidance – The robot must avoid collisions with walls, people, and other obstacles.

Feasibility Study

We are confident that our components will work together because the hardware has previously been tested on projects including the CMUcam and the Boe-Bot. The Boe-Bot, coupled with the BASIC stamp, is a fairly inexpensive combination of a robot and microcontroller. The CMUcam user manual offers free software for tracking brightly colored objects which will allow us to have a solid foundation for our new omni-directional software.

Proposed Design Implementation

The omni-visional robot will contain several main hardware components. The following components are the most feasible components found for constructing an omni-directional motion-tracking robot. The Board of Education Robot (Boe-Bot) is the robot chosen for the project. This parallax made robot will offer a solid platform for fastening the camera while also providing a moderately simple interface between the BASIC stamp, robot, and the camera. The BASIC stamp, also developed by parallax, will function as the micro controller. The camera is a

CMU-camera developed by Seattle Robotics. In order to create the omni-directional image, a hyperbolic mirror manufactured by Neovision will be fastened to the lens of the camera. A combination of whiskers and sonar devices will be used to prevent the robot from colliding with other objects.

Basic Stamp

The BASIC stamp (Figure 2) will serve as the micro controller for the project. The stamp will provide I/O pins, which can be fully programmed to interface with other components such as LED's, speakers, and other devices. The stamp will include a 5-volt regulator, resonator, serial EEPROM, and PBASIC interpreter.

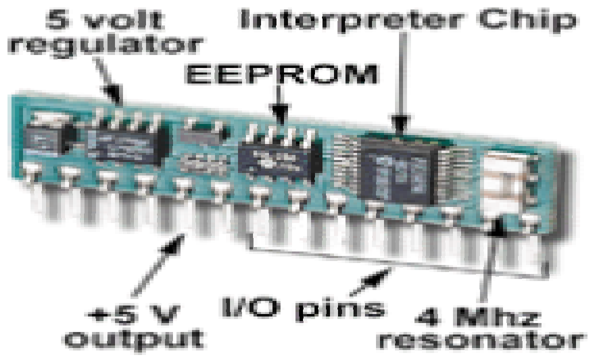


Figure 2. The BASIC stamp.

The EEPROM is a non-volatile memory, which will store the BASIC program received. The BASIC program will be written using the editor supplied by the parallax vendor and can be transferred directly from the PC to the EEPROM through the serial connection. Since it is non-volatile, it will store the program even when power is shut off. The user can erase or write over the program by reloading a new program into the EEPROM. The PBASIC interpreter will fetch and write the instructions to the EEPROM.

The BASIC stamp preferred is the BS2p40, which is manufactured by parallax. The BS2P40 provides several advantages over the other Parallax BASIC stamps. This particular stamp offers 16 extra I/O pins. The stamp offers several new commands for interfacing with a larger variety of components along with a polled interrupt capability feature. The starter kit for the stamp also offers a serial cable, BASIC stamp manual, thermometer, 4K EEPROM, and a 4.7K resistor, which can all be useful for the project. The recommended power supply for the stamp is a 7.5 V DC 1 amp source. The programming language used by the BASIC stamp is the BASIC programming language. The software loaded into the stamp will be used to control the Boe-Bot. The stamp will be plugged into the Boe-Bot. The BASIC program software will set up the serial interface connection between the robot and the CMUcam's own SX28 micro controller. The software will instruct the camera to lock onto and track a brightly colored object. The stamp will command the Boe-Bot to follow the object being tracked based on imaging data sent from the CMUcam and stop when it comes within a specified distance.

CMU Camera

The CMUcam is a fairly common camera used as for robot vision. The Seattle Robotics Company manufactures the CMUcam. The CMUcam camera was chosen because it does not require much power and also can be purchased relatively cheap. A SX28 micro controller is used for handling all the processing of the camera data. It is connected directly to an OV6620 Omni vision C-MOS camera chip. Firmware to handle these tasks must be installed to the micro controller using either a RS-232 or TTL serial port. The serial communication parameters are specified in Table 1.

Table 1. SX28 Microcontroller Serial Communication Parameters

115,200 Baud
8 Data bits
1 stop bit
No parity, and No flow Control.

The camera can be used to output information such as the position of a brightly colored object. The information will be sent to the stamp, which will use the image coordinates to control the robot and advance to the object. The stamp will send instruction commands to the SX28. Each command will be sent using visible ASCII characters. When the transmission is successful, an ACK string will be returned. If the transmission is unsuccessful, a NCK string will be returned. The commands, located in Table 2, are a sample of common examples used to communicate with the CMUcam. They will be integrated into the BASIC program downloaded onto the stamp. All commands will be followed by either an ACK or a NCK reply from the CMUcam notifying whether or not the transmission was successful.

Table 2. Sample CMUcam Commands

Command	Description
\r	Sets the camera into an idle state. The camera should return an ACK acknowledgement
CR [reg1 value1]\r	This command will set the internal registers of the camera. The user can send up to 16 register-value combinations. To reset the camera and restore the camera registers to their default state, simply call this command with no arguments.
DF\r	This command will dump a frame out to the micro controller via the serial port. This dumped data will consist of raw video data column by column and also consisting of a frame synchronize byte and column synchronize byte.
L1 value\r	This command is used to control the green LED tracking light. The three possible values are 0, 1, and 2. 2 is the default auto mode. 0 disables the tracking light and 1 turns the tracking light on.
DM value\r	This command sets the delay before packets that are transmitted over the serial port. The possible delay values range from 0-255.

The CMUcam operates at 17 frames per second, which will be sufficient for the project. The main purpose of the CMUcam will be to lock on a brightly colored object and track it. The camera will supply the robot with an image and the robot will use the image to gather coordinates and move to the target. The camera will be pointed vertically straight up with the hyperbolic mirrors fastened to the lens. This will enable the robot to see a full 360-degree picture. When tracking a brightly colored object using the CMUcam, it is important to follow the following calibration procedure:

1. Make sure the camera currently has no objects in front of it.
2. Hold and release the Boe-Bot reset button.
3. After a 1 second delay, the green LED on the CMUcam will turn on for about 5 seconds as the camera adjusts to the lighting.

4. Once the green light turns off, the user will have 5 seconds to place the object in front of the robot and allow the robot to lock onto the object.
5. The green light will illuminate again once the robot has locked onto the object.
6. The camera will now track the objects of that particular color.

Boe-Bot (Board of Education robot)

The Boe-Bot will require assembly upon arrival from the vendor. The Boe-Bot is built on an aluminum chassis which will include a breadboard, battery pack, two servos, wheels, along with other smaller components. The breadboard will offer a platform for connecting other devices and adding circuitry. The Boe-Bot can be programmed to perform many functions. Once the assembly has been made, testing must be done to ensure all components work properly. The Boe-Bot includes two servos or wheel motors which will be used to control the path of the robot. An electric engine also known as a pulse train will control the servos. The pre-modified servos will be pulsed in such a way as to control the wheel to turn a full 360 degrees continuously in either a clockwise or counterclockwise direction. The pulse widths will range between 1 ms and 2 ms. At 1.5 ms pulses, the servo will stay still. At 2 ms pulses, the servo will rotate at full speed in the counter clockwise direction. Likewise, when the pulses are less than 1.5 ms, the servo will rotate at a clockwise direction with 1 ms being the fastest speed. The software in the BASIC stamp will control how much each wheel should turn in order to advance the robot towards the object that the camera is tracking.

Hyperbolic mirror

A hyperbolic mirror will be needed to accomplish the 360-degree omni visional image. The mirror chosen will be a half spherical shaped mirror developed by Neovision. The mirror will be fastened to the lens of the vertically mounted upward CMUcam camera. The lens of the camera will be located at the focal point of the mirror to ensure that the image will be correct. The mirror will be made of glass.

Collision prevention

Wires (whiskers) and a sonar device will be fastened to the sides of the robot to help prevent it from running into walls or other obstacles. The whiskers will allow the robot to avoid obstacles by touch. The robot will then change direction and continue without colliding into the object. The whiskers will be wired to the Boe-Bot through the supplied Boe-Bot breadboard. Each switch circuit I/O pins will use the 10 KΩ pull-up resistors to monitor their individual voltages. When the whiskers are free, the voltage at the I/O pin should be around 5 volts. When the whisker is touched, the line is shorted to ground returning a voltage value of 0 V. The I/O line will see logic 0 rather than logic 1. The BASIC stamp will be programmed to detect when a whisker has physical contact with another object. A sonar device will also be used. The sonar device will send out sonar waves, which will bounce back to the receiver. If an obstacle is in the sonar path and too close, the receiver will receive the waves quicker than expected and adjust its path to avoid the obstacle.

Alternative Solutions

There are several alternative solutions to the design problem. One solution would be to use different types of mirrors. There are three main types of mirrors: conical, parabolic and spherical (Figure 3). Parabolic is our choice due to the limited distortion of the images. Spherical tends to warp the image increasingly more, the further away from the camera the image gets. Conical mirrors tend to reflect the image more downward than straight out from the camera.

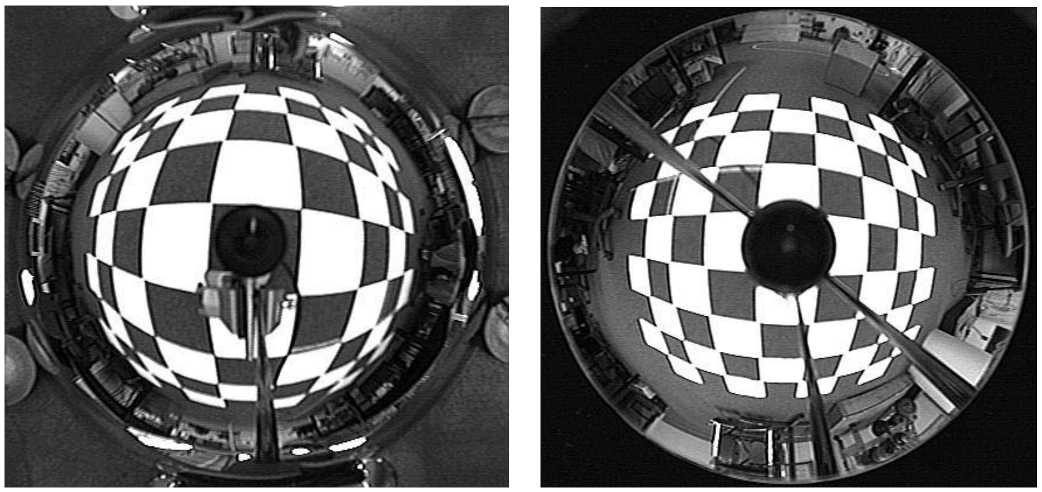


Figure 3. Spherical mirror (left) and parabolic mirror(right).

Another alternative solution is to use multiple cameras. Cameras can point in different directions providing a view of the varying directions. There are several problems with this solution. One such downfall is the lack of one continuous image. The arrangement produces several different images instead of one single image. One could stitch the image back together using software, but this adds a great deal of complexity to the design. Simply having multiple cameras increases the complexity and also increase the hardware needs of the micro controller. The greatest downfall of this solution is the escalation in cost. Even inexpensive cameras will cost over $100. This would quickly put us well over budget.

A third solution is also available. Instead of using mirrors, we would use a super-wide angle "fish eye" lens. The lens distorts the light entering the lens and provides a large field of view.

This is also an economical solution. The down fall of this alternative, is the relative limited field of view provided compared to that of the parabolic mirror.

Design Validation Approach

We must fully test the end product in order to insure that the robot and vision system design match all the appropriate design criteria. For the basic project, we will have to test the entire vision field to validate that the robot senses and tracks the desired object over the full 360-degree field. This can be accomplished by moving the desired target completely around the robot faster than the robot can respond so that it verifies that the target object is never lost.

Moving an object and thus the robot down corridors with trashcans or other objects that may impede its path can test obstacle avoidance. The sonar and whiskers will have to detect and navigate the robot away from said object and the vision system will maintain visual contact with the target at all times. Obstacle avoidance is critical to prevent damage to the robot, camera system, and external environment.

A standardized checklist may be as follows:

Robot successfully tracks the object in front of robot:	☐
Robot successfully tracks the object on left side:	☐
Robot successfully tracks the object on the right side:	☐
Robot successfully tracks the object from behind:	☐
Rapidly move object from front to back to insure visual tracking system:	☐
Tag a human with the tracking object and move around the building:	☐

Additional Considerations and Analyses

Economic Analysis

Part	Part Number	Vendor	Cost	Quantity	Total Cost
Boe-Bot	28132	Parallax	$229.00	1	$229.00
CMUcam System	30051	Parallax	$139.00	1	$139.00
Devantech SRF04 Ultrasonic Ranger	28015	Parallax	$30.00	1	$30.00
Hyperbolic Mirror for Omni-directional Vision	H3G	Neovision	$290.00	1	$290.00
BASIC STAMP upgrade	BS2P24-IC	Parallax	$79.00	1	$79.00
Miscellaneous Parts	varied	Radio Shack / Lowe's	$60.00	1	$60.00

Possible budget additions if expansion projects are taken on

Phase II a					
Boe-Bot	28132	Parallax	$229.00	1	$229.00
CMUcam System	30051	Parallax	$139.00	1	$139.00
Devantech SRF04 Ultrasonic Ranger	28015	Parallax	$30.00	1	$30.00
Hyperbolic Mirror for Omni-directional Vision	H3G	Neovision	$290.00	1	$290.00
BASIC STAMP upgrade	BS2P24-IC	Parallax	$79.00	1	$79.00
Phase II b					
433.92 MHz Transceiver (SIP/solid/raw)	27997	Parallax	$95.00	3	$285.00
		Total Cost before expansion =			$827.00
		Total Proposed Cost for Phase IIa =			$1,594.00
		Total Proposed Cost for Phase IIb =			$1,112.00

Boe-Bot by Parallax, Inc.

Parallax, Inc provides a lightweight, general robotics and student version of a pre-fabricated robot known as the Boe-Bot. This robot meets the general design requirements for the specified project because it is relatively cheap, comes with on-board electronics, and programming software. The main modifications that will be made to the robot will be in the form of adding addition hardware and vision components to allow the omni-directional vision system. Compared to other pre-fabrication robots, the Boe-Bot is the cheapest and most versatile with a multitude of available accessories from Parallax, Inc.

CMUcam System:

The CMUcam system was originally designed by Carnegie Melon University and is specially modified by Parallax to seamlessly integrate with the Boe-Bot platform. The connection method to the robot only requires a couple of screws and plugging in the serial components so that the stamp onboard the robot can interpret the data sent to it. The CMUcam already contains the hardware necessary for image tracking, pattern locking onto a specific target, and the interface necessary for control the wheel servos for movement. With a few modifications, the CMUcam may be able to lock onto a target of a specified color rather than having to use the pattern lock feature that is already implemented in the hardware.

Devantech SRF04 Ultrasonic Ranger:

In order to complete the task of obstacle avoidance, the Boe-Bot will have to be outfitted with electronic whiskers and sonar devices. The whiskers come as part of the Boe-Bot full kit but the sonar devices have to be purchased separate. Once again, Parallax, Inc. already has sonar devices on a printed circuit board and ready for integration into the Boe-Bot platform. These sonar devices have a range of 3cm to 3m, which should allow for plenty of control and collision detection.

Hyperbolic Mirror for Omni-directional Vision:

In order to achieve the 360-degree field of vision required for the project, we must use some sort of conical or hyperbolic mirror to reflect a complete image into the camera lens. Through this reflection, the camera and thus the robot will have a 360-degree field of view in a single frame shot without the need for camera rotation. Currently, the cheapest vendor for said mirror is Neovision in the Czech Republic. They can build a 25mm diameter hyperbolic lens, which would reflect the appropriate image for $290. Compared to other vendors, Neovision presents the cheapest option as Accowle, Inc. in Japan sells similar mirrors for $430 to $800.

BASIC STAMP upgrade:

Viewing the project description and to allow for the most flexibility and expandability for design, we have opted to upgrade our stamp package to the BS2P24-IC. This particular stamp has 19 more BASIC commands, processes approximately 3 times the instructions per second, has more on-board memory, faster serial port communication speeds, and faster processor speed, which may be necessary if we take on the expansion phase projects. Essentially, the BS2P24-IC combines the most functionality and most flexibility for the robot for modest price increase.

Miscellaneous Parts:

The miscellaneous parts heading consists of the numerous pieces necessary to make the Boe-Bot functional. This budgetary calculation is based on the cost of batteries, protoboards, wires, solder, mirror mount hardware, and other small pieces necessary to make everything work together. The reason there is no itemized list of parts and costs is because the mirror itself has yet to arrive, and thus it is impossible to determine exactly what is needed to mount it properly.

Also, the number of protoboards and wires is completely dependent on how complex the project becomes as more hardware is custom crafted rather than purchased as plug and play components.

433.92 MHz Transceiver (SIP/solid/raw):

Parallax, Inc. offers transceivers ready built and all that is necessary is to connect the serial port and power them. If the computer controlled extension is taken, then transceivers for both the computer and robot(s) would be needed causing the specification for 3 transceivers. These transceivers have a 2-inch antenna and transmission range of 150 feet. It is noted that in a budget crunch, a self-built transceiver could be constructed for approximately $28 each but would require the obtainment of new printed circuit boards or protoboards in order to connect the transmitter, receiver, and antennae. These materials would be purchased from Reynolds Electronics in model numbers TWS-434, RWS-434, and 434 MHz for the transmitter, receiver, and antenna respectively. It must be noted, however, that by custom building this component it would be much more complex and require more testing time to ensure the construction was done properly.

Environmental, Societal and Safety Analysis

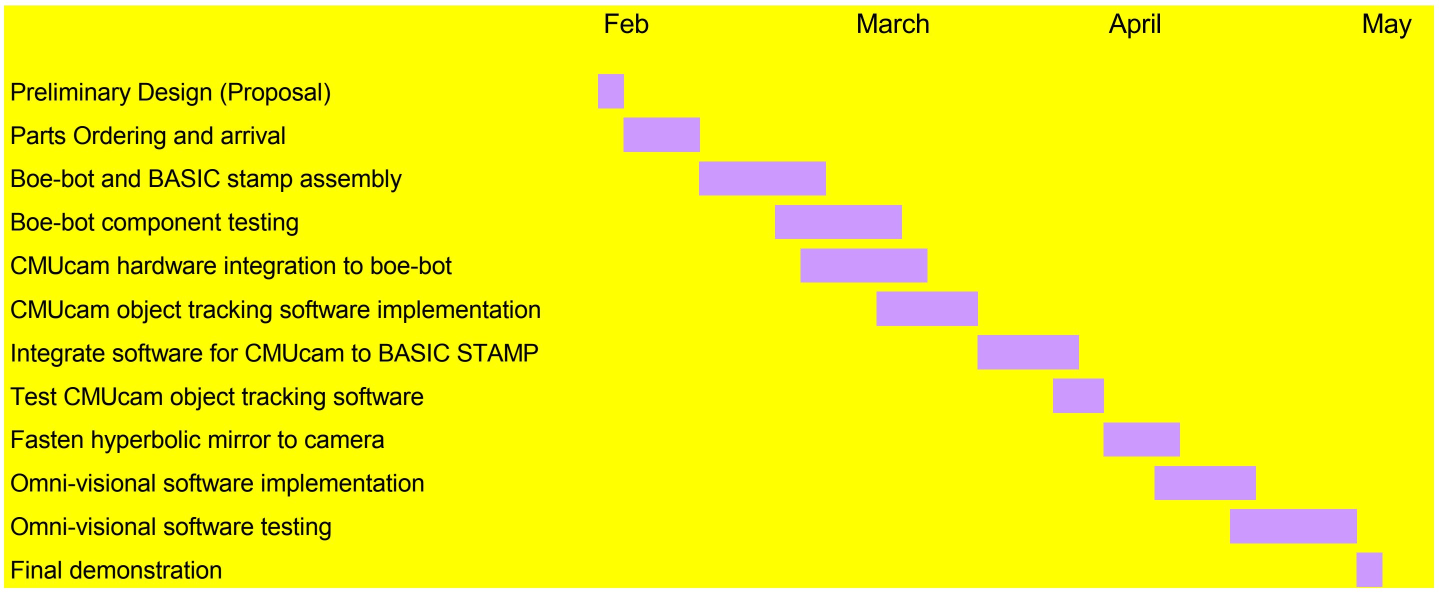
The environmental concerns associated with our robot and omni-directional vision system are minimal. The robot itself is small, 127 millimeters in its largest dimension, weighs about three pounds, and is relatively slow-moving; thus, the danger from impact is negligible.

Power is supplied to the robot in the form of four AAA batteries. Proper disposal of used batteries is required to ensure environmental safety. The robot and imaging devices include no other hazardous materials.

The robot will include two sonar units for collision avoidance. The sonar units emit ultrasonic pulses which will allow the robot to determine its distance from obstacles. While there are no environmental or safety concerns associated with sonar, it does carry the possibility of interfering with other sonar devices and delicate equipment, such as in a research environment.

Proposed Schedule and Work Breakdown

We have developed the following Gantt and PERT charts to assist in planning our design.



Conclusion

The A Team feels that the requirements proposed in this document for the omni-directional vision system for the Parallax Boe-Bot can be completed satisfactorily within the next three months. After careful consideration and deliberation, we have selected all the major components and are ready to begin purchasing. The Gantt and PERT charts will be valuable tools in assisting with project scheduling and ensuring we meet our final deadline.

Considering the rarity of omni-visional robot projects, we felt that our project was both challenging and innovative. The project touched a variety of engineering areas including writing and testing software, integrating hardware, and understanding optics. The base design allows for expandability and future experimentation within the omni-directional vision control system fields. Hopefully with future innovations and research, this particular design can become the backbone for other researchers' fields of study.

References

The Page of Omnidirectional Vision, http://www.cis.upenn.edu/%7Ekostas/omni.html viewed on February 2, 2003.

Seattle Robotics CMUcam Vision System for Boe-Bot User Manual, Carnegie-Mellon University, 2002.

CMUcam Vision Board User Manual, Carnegie-Mellon University, 2002.

Robotics! Student Workbook Version 1.5, Parallax, Incorporated, 2001.

Omni-Directional Vision System for Mobile Robots

Implementation Notes

May 5, 2003

Table of Contents

Overview

The omni-directional vision system for the Board of Education robot (BOE-Bot) meets all of the requirements for image tracking and mobile robot navigation as originally specified in our proposal. The vision system is housed on the BOE-Bot, tracks bright objects with a 360° field of view, and responds to movement while avoiding obstacles.

Our research and development for this project included the following activities:

- Researching omni-directional vision, computer imaging, and mobile robotics
- Developing a prototype design
- Researching products and weighing benefits and costs
- Planning the timetable for completing the project by the end of the semester
- Constructing the omni-directional imaging apparatus for our mobile robot
- Implementing software to control the movement of the robot based on omni-directional images acquired
- Troubleshooting unexpected problems
- Testing and verification of vision system functionality

We gained experience in the entire engineering design process and practiced exceptional project management and planning techniques to ensure completion of our project by the deadline. The following discussion presents the details of our hardware and software implementation procedures.

Parts List

Part	**Qty**	**Vendor**	**Part No.**
Board of Education robot (BOE-Bot)	1	Parallax	28132
BASIC stamp upgrade	1	Parallax	BS2P24-IC
Devantech SRF 04 Ultrasonic Ranger	1	Parallax	28015
CMUcam system	1	Parallax	30051
1 mf/50V radial electrolytic capacitor	4	Mid-State Electronic Supply	1M50-RDE
16-pin high reliability IC socket	1	Mid-State Electronic Supply	HST16B
DB9 male to IDC 10 socket with bracket	1	Mid-State Electronic Supply	F-BRK9M10
MAX232 chip	1	DigiKey Electronics	MAX232cpe
5-by-2 .100 header	1	various	none
2-pin high reliability socket	4	various	none
Jumper header with shroud	2	various	none
DecorLite 1/2 Chrome 2 inch lightbulb	1	Lowe's	03758
17 x 12 inch clear acrylic sheet	1	Lowe's	none

Hardware Implementation

After the order from Parallax arrived, we assembled the BOE-Bot and CMUcam system according to the instructions provided by the manufacturer. We tested the communication between the robot and camera using the example software available from the Parallax website: http://www.parallax.com.

To construct the apparatus to suspend the mirror above the camera at approximately the right height, we heated and molded a clear acrylic sheet in a U-shape and cut a hole in it for the light bulb base. Later, when we were sure of the correct height and angle at which the light bulb should be positioned, we secured it with a hot glue gun.

The omni-vision robot included obstacle avoidance features. The robot scanned its vision for obstacles before locking onto and approaching the target. We implemented this in the BASIC software. The obstacle avoidance hardware is mounted on the front and two sides of the robot. We used sonar to scan the front of the robot to prevent collisions. The sonar was connected to the breadboard. We used the pins from the breadboard to connect to the BASIC stamp. The sonar transmits sound forward in front of the robot. If an obstacle is in the path of the robot, then the sound waves will bounce off the obstacles and be received by the sonar. The sonar can predict how far the obstacle is from the robot by how long the sound waves take to return to the robot after transmission.

We also used this concept to set up obstacle avoidance on the sides of the robot. Instead of using sonar, we attached infrared sensors to the breadboard and faced them to the sides of the robot. The infrared also includes a transmitter and receiver. The transmitter emits the infrared light, which bounces off the obstacles and is detected by the receiver.

The CMUcam vision system that we ordered from Parallax was modified to work with the BOE-Bot particularly. It comes from the manufacturer strapped to 9600 baud, and the only communication is through TTL ports directly to the BASIC stamp. However, in order to correctly position the mirror and focus the camera, one must first be able to see an image from the camera. Unfortunately, the modified version does not support image dumps, and thus modifications must be made to incorporate the image dump capability.

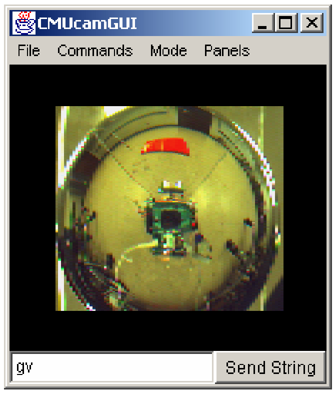
In order to make the camera dump images, you must modify the board that came from the manufacturer. Please note that when you do this, the warranty is void and no support will be available for improper modifications. The necessary components for modification that must be obtained are four 1uf 50 mV electrolytic capacitors, a MAX232 chip, 5-by-2 .100 header, 2 jumper headers with shrouds, 10 pin to female DB9 serial cable, 16 pin high reliability socket, and four 2 pin high reliability sockets. All of these components will be placed onto the board in order to allow it to communicate directly with the computer because the BOE-Bot will not be able to handle the information at this point.

Start the modifications by soldering the sockets into their respective positions as indicated by the camera manual found at http://www-2.cs.cmu.edu/~cmucam/Downloads/CMUcamManual.pdf. These sockets will allow the removal of the components at a later time without the need of having to break any solders on the board. After the sockets have been soldered into place, solder the 5-by-2-pin header to the serial connection port at the corner of the board. This will allow you to attach the serial cable from the camera board directly to the computer.

Before installing the MAX232 chip or capacitors, first break the soldered jumpers on JP1 and insert the new pin jumpers with the shrouds off. With both the shrouds off, the baud is now set to 115,200 – which is the communication level of the computer program that views the images. With the last of the solders done, insert the MAX232 chip and capacitors onto the board and plug in the serial cable. Note that the image of the serial cable connection in the manual is of the aspect from the cable entering the DB9 connector and thus if you look at the side of the connector that plugs into the male end of the serial cable, the connections are mirrored.

With the modifications complete, you can reinstall the camera onto the base and insert the red wire into V_{dd} and black into V_{ss} (or ground). You will need to install the Java software available from Carnegie Melon University at http://www-2.cs.cmu.edu/~cmucam/downloads.html. Run this software from computer you are at and allow it to communicate with the camera. If everything is configured properly, you should be able to click on `Commands->Dump Image`. This will pull an image from the camera that when the mirror is centered and focused will look something like Figure 1.

Figure 1: Omni Picture



When all of this is completed, remove the camera from the robot base by taking off the screws that hold it on the standoffs. This will allow you to maintain the exact position of the camera relative to its components while you remove components. Once the camera is off, carefully remove the MAX232 chip, the serial cable, and the four capacitors as they can interfere with the TTL logic on the board. At this time also put the shroud on both JP1 jumpers so that the communication is strapped to 9600 Baud, the max for the BASIC stamp. With those quick changes made, simply place the camera back on the standoffs and reconnect the wires to the robot base. Insert the blue and white wires into whatever input pin locations you want to use and the red and black into the V_{ss} and V_{dd} respectively.

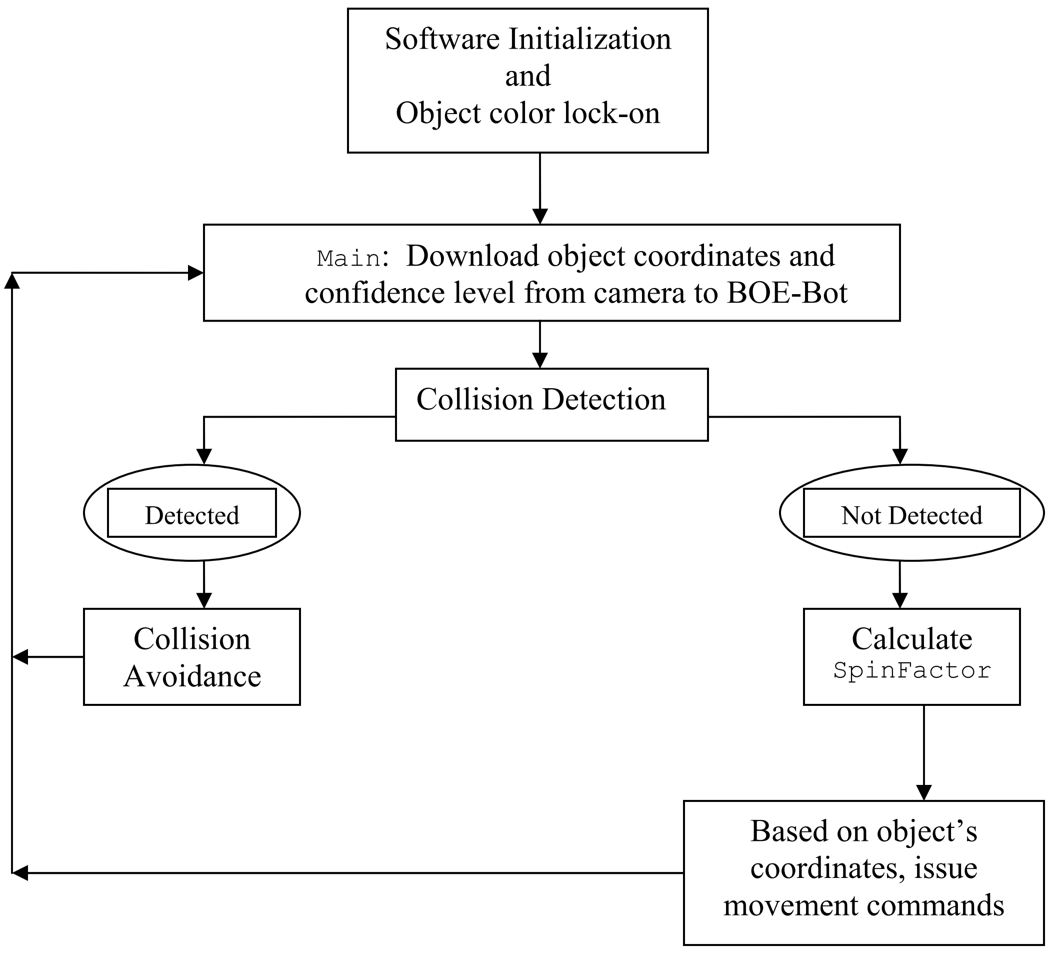
With that done, all of the camera hardware manipulation is complete. If you want to lock onto a color by a means other than holding the robot upside down and placing an object between the camera and mirror, then you need to issue some Set Window (SW) commands in the code. From experimentation, if an object is place next to the right wheel, then the code is `SEROUT 10,240,["SW 10 60 20 80",CR]` followed immediately by a track window command and then issue another set window command but with no coordinates so that the full window is used again. This will allow the camera to look to the middle of the window specified and lock onto the color at that location.

Software Implementation

The software written for the project consists of several main steps:

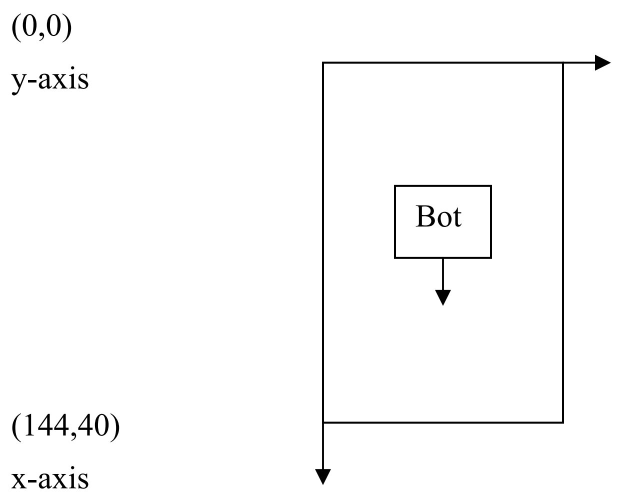
- Software Initialization and object color lock-on procedure
- Download object coordinates and confidence level from camera to BOE-Bot
- Perform collision detection
    - If collision imminent, issue appropriate movement to avoid collision then return to `Main`
- If no sensors detect obstacles, calculate `SpinFactor` based on object coordinates
- Based on object coordinates: spin to face object, do nothing, or move forward
- Return to `Main`

The following diagram illustrates proper flow of the software.


Software Initialization and Object color lock-on
Main: Download object coordinates and confidence level from camera to BOE-Bot
Collision Detection
Detected
Not Detected
Collision Avoidance
Calculate SpinFactor
Based on object's coordinates, issue movement commands

The first module of software is executed only once because it is the software initialization and object lock-on portion of the code. After waiting one second for CMUcam to start up, a reset is sent to synchronize the CMUcam and the BASIC stamp. Next the auto adjust is turned on for five seconds to take into account the current lighting environment. After setting the window for the proper location to look for object, the Track Window command looks at the center of CMUcam image and grabs the color information and sends to the Track Color function. Finally, the command, `freqout 2, 2000, 1000` makes an audible sound so that operator knows the lock-on procedure has completed.

The second module is the `MAIN` procedure. First, the command `serout 10, Baud, [”TC”,CR]` is sent. This command will track the last color grabbed by the track window (TW) command. Next, the command `serin 9, Baud, [STR RcvData\10]` is sent. It returns certain information such as x-coordinate of middle mass, y-coordinate of middle mass, confidence level, and other info to the stamp in the form of an array of length 10, called `RcvData`. The next line of code, `Confid = RcvData(9)`, sets the variable `Confid` equal to the confidence level returned by the previous command. The array `RcvData` contains several other important variables including the x-coordinate located at `RcvData(3)`, and the y-coordinate located at `RcvData(2)`. The range of the x-coordinate is 0 to 144. However, the range of the y-coordinate is 0 to 80. The x-y axis is arranged according to figure below. Now that this important information has been transferred from the camera to the stamp, it can be used for calculations in the remaining modules.



The third module is used strictly for obstacle detection. The variable `State` keeps track of detected obstacles. `State.bit0` is used for the sonar, while `State.bit1` and `State.bit2` are used for the right and left IR sensors respectively. A value of one represents no obstacle. If the value drops to zero, then an obstacle is present at the location the `State` bit represents and must be avoided. The last portion of code in this module is an important decision in the software flow. If the `State` bits are all ones, then there are no obstacles present and the program moves to the fifth module – calculating `SpinFactor`. Otherwise, an obstacle is present, and the last line of code in the module sends the program flow to the fourth module – object avoidance.

Module four is only taken when an obstacle has been detected and must be avoided. Based on the `State` variable set in module three, the robot may find itself in any one of the eight cases shown in the following table. In the obstacle avoidance algorithm each case is interpreted into an appropriate evasive action; then the program returns to the second module to reaquire the tracking color.

Obstacle Avoidance Algorithm					
Case	`State` Variable			Interpretation	Action
	Left IR	Right IR	Sonar		
0	0	0	0	Blocked left, right, front	Backup
1	0	0	1	Blocked left, right	Backup
2	0	1	0	Blocked left, front	Turn right 90°
3	0	1	1	Blocked left only	Turn right gradually
4	1	0	0	Blocked right, front	Turn left 90°
5	1	0	1	Blocked right only	Turn left gradually
6	1	1	0	Blocked front only	Backup to the left
7	1	1	1	Clear	No change in direction

The fifth module calculates the value for the variable `SpinFactor`. The concept behind this variable is very important for smooth and speedy robot response. To avoid short jerky spin movements, `SpinFactor` is used as an estimate for how much the robot should spin in order to face the object it is tracking. The further it must spin, the larger `SpinFactor` will be and the longer the greater number of iterations the for-loop that spins the servos will execute. `SpinFactor` is calculated in the following manner. If the x coordinate is less than 70, then `SpinFactor` is calculated using the `BigSpin` method. Otherwise, it uses the second method, which has two variations that are similar except that one is used when the object is to the right of the robot (y-coordinate < 40) or to the left of the robot (y-coordinate < 40). The `BigSpin` method only takes into account the x-coordinate and consists of a single line of code, `SpinFactor =(76-RCVData(3))`. The farther towards the back of the field of view, the smaller the x-coordinate `RcvData(3)` becomes, and the larger the value calculated for `SpinFactor`. If the object is not very far behind the robot, one of the other method of computing `SpinFactor` is used. This method only uses the y-coordinate and also consists of a single line of code, `SpinFactor = 40-RCVData(2)`. This particular line of code is used when the object is to the right of the robot. The farther towards the side of the field of view and the smaller the y-coordinate becomes, the larger value `SpinFactor` will be assigned. When the object is on the left of the robot, `SpinFactor` is calculated in a similar fashion using the line of code `SpinFactor = RcvVData(2)-40`. After `SpinFactor` is calculated, the software flow progresses to the sixth module of code.

The sixth and final module of code issues the movement commands to the servos based on the coordinates of the tracked object. If the object is in the ideal location, the robot issues a distinctive sequence of tones to the speaker. If it is to the right or left, it turns right or left a variable amount based on the calculated value of `SpinFactor`. If none of the above if statements are taken, then the object is centered in the front of the robot or the confidence level is to low. If the level is not too low, then it is definitely centered. If it is also far away from the robot, the x-coordinate will be greater than 120, and the robot moves closer to the object. Despite what type of movement is executed, or even if none is executed at all, the software flow returns to the top of the main module, and coordinates are reacquired from the camera starting the sequence over again.

Sources

Parallax
Website: http://www.parallax.com
Technical Support: (916) 624-8333 or support@parallax.com

Carnegie Mellon University
Website: http://www-2.cs.cmu.edu/~cmucam/
Technical Support: cmucam@cs.cmu.edu

Mid-State Electronic Supply, Inc.
811 South Texas Avenue
Bryan, TX 77803-3987
(979) 779-1226

Omni-Directional Vision System for Mobile Robots

User Manual

May 5, 2003

Table of Contents

Introduction

The Omni-Directional Vision System for Mobile Robots was developed with user-friendliness in mind. Using a camera and mirror system, the Board of Education robot (BOE-Bot) can track an object up to three feet away in any direction. The color of the target object is set during the power-up routine and can be reset at any time by pressing the reset button. The system uses sonar and infrared (IR) sensors to detect and avoid obstacles as the robot approaches its target.

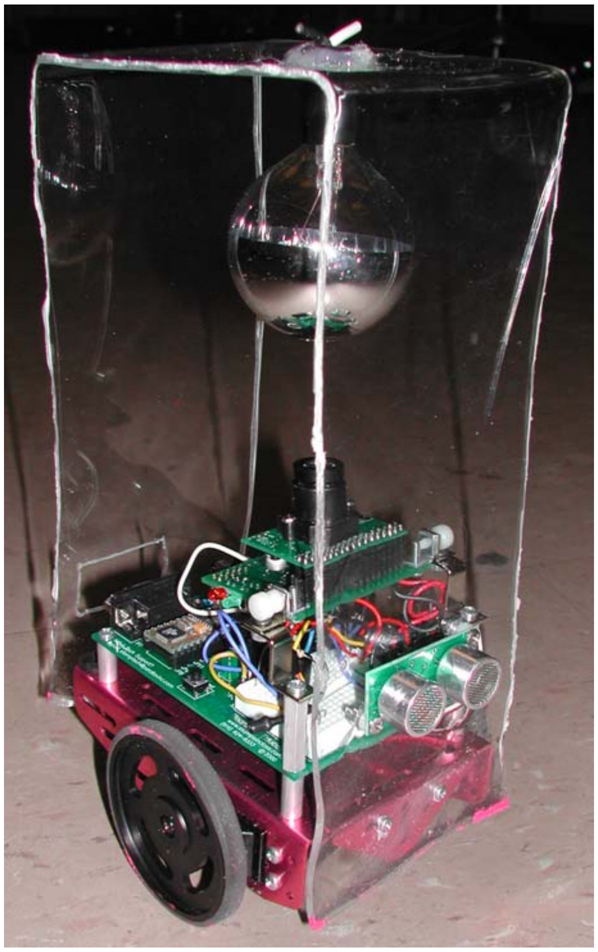


Figure 1 – BOE-Bot with Omni-Directional Vision System

Using the System

Selecting an object to track

Before turning on the robot system, obtain a brightly colored object that you would like for the robot to track. For best results, select an object whose color is in sharp contrast to the surrounding environment, such as red or magenta. The object that will be tracked should be at least four inches square and its surface should be non-reflective. Shiny surfaces that reflect light inconsistently cause the camera to lose sight of its target in changing light conditions.

Preparing to track

Place the robot on the ground in the environment in which it will be running. Be sure to choose a location that has good lighting. In order for the camera to lock on to the object which it is going to track, the object must be placed on the ground directly beside the right wheel of the robot (see the example in Figure 2).

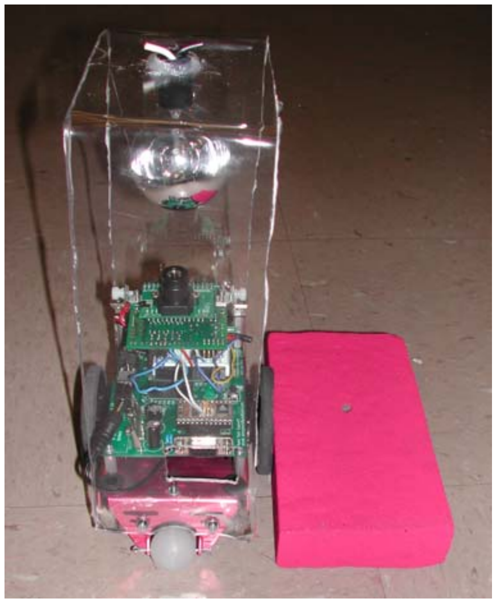


Figure 2 – Object to be tracked is placed on the right side of the robot.

Starting the system

Power on the BOE-Bot by plugging in the cord from the battery source located on the bottom of the robot. Alternatively, you may choose to use an AC power supply to conserve battery power.

Locking on to the tracking color

There is a delay of five seconds after starting the system to allow the user to position the object to be tracked beside the right wheel of the robot. During this delay, the green LED light on the upper circuit board of the robot will be illuminated, as shown in Figure 3. When the camera begins the color lock-on process, the green light will turn off; when the camera is finished locking-on to the color, the green light will turn back on. From this point on, the green light will be illuminated when the camera detects the target color in its field of view.

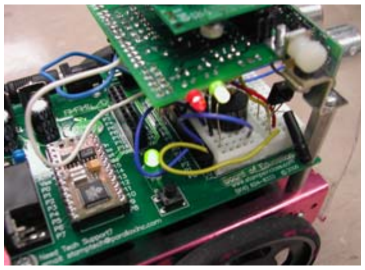


Figure 3 – Green LED picture

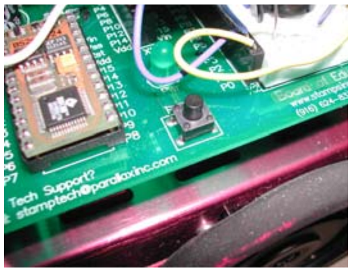


Figure 4 – Reset button

Resetting the tracking color

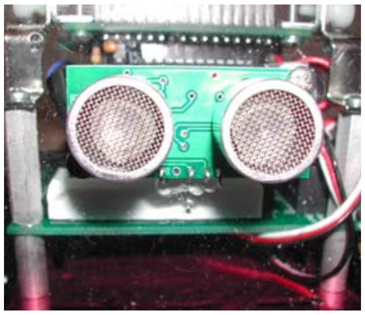
In some instances, the user may need to reset the tracking color. Simply press the black button (see Figure 4 above) and the system will stop and reset itself. The steps detailed in the previous section will be repeated including the initial five second delay.

Tracking the object

Move the target object all around the robot, and watch as it turns to face the object and then approaches it. In good lighting conditions, the robot will be able to detect the object anywhere within a radius of approximately three feet. If the robot is not tracking the target object consistently, please read the troubleshooting suggestions in the next section.

Obstacle avoidance

The robot is equipped with sonar and infrared (IR) sensors to allow it to detect obstacles and avoid collisions. The sonar unit is mounted directly on the front of the robot and is shown below in Figure 5. When the sonar unit detects an obstacle, the robot backs away from the obstacle and attempts to approach the target object by a different path. The infrared sensors are mounted on the left and right sides of the robot (see Figure 6 below), and they also signal the robot to move away from obstacles.



Figures 5 – Sonar unit

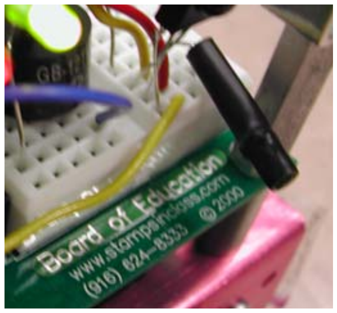


Figure 6 – Infrared sensor

Turning off the system

To turn off the system, simply unplug the battery source or the AC power adapter.

Troubleshooting

Problem	*Possible Solutions*
Robot does not consistently track target object.	1. Lock onto the color again. Press the reset button and be sure to place the color as close to the right wheel as possible. See Figure 2 to be sure you have the robot oriented correctly for locking onto the tracking color. 2. Provide consistent lighting. The camera looks for a very specific color match to track an object. If the object reflects too much light or is under shadows, the very sensitive camera sees a color that is different from the one onto which it locked initially. 3. Use a brightly colored object that contrasts with its surroundings. If the camera sees the color it is looking for in several places, the robot may be confused and will not track consistently.

Problem	*Possible Solutions*
Robot stops while tracking target object.	1. The object has traveled out of the camera's field of view. Remember that the camera can only see about three feet away in any given direction. 2. The batteries are exhausted. Replace the batteries or use the AC adapter to supply the robot with power.

Problem	*Possible Solutions*
Robot travels in irregular path.	1. The AC power cord may be triggering the sonar and/or infrared (IR) sensors, which causes the robot to behave as if it is avoiding obstacles it has detected.

Contact Information

Development Team Members

Denise Fancher	denise2002@tamu.edu
Kyle Hoelscher	satisfaction@neo.tamu.edu
Michael Layton	cml6387@neo.tamu.edu
Eric Miller	emiller@tamu.edu

Professor & Advisors

Dr. Ricardo Gutierrez-Osuna	rgutier@cs.tamu.edu
Siddharth Choudhuri	choudhri@cs.tamu.edu
Steve Ortiz	steve-ortiz@tamu.edu
Marco Zavala	marco.zavala@tamu.edu

Texas A&M University
Department of Computer Science
301 Harvey R. Bright Bldg.
College Station, TX 77843-3112
(979) 845-5534

Parallax – BOE-Bot
Website: http://www.parallax.com
Technical Support: (916) 624-8333 or support@parallax.com

Carnegie Mellon University – CMUcam
Website: http://www-2.cs.cmu.edu/~cmucam/
Technical Support: cmucam@cs.cmu.edu

Omni-Directional Vision System for Mobile Robots

Course Debriefing

May 5, 2003

Group Management

Our group management style worked very well. We were blessed to have team members that respected each other and cared a great deal about pulling their own weight and being fair to the other group members. Each team member volunteered to work on the part of the project that he found most interesting or best suited to his strengths and abilities. We had outstanding attendance at group meetings; not once was any group member absent when he had not informed his peers that he would be unable to meet. We corresponded intensively by email, keeping all team members aware of the latest developments in the project and requesting assistance as needed. We cannot think of anything that would have made our team run smoother than it did.

Safety and Ethical Concerns

There were no significant safety or ethical concerns involved in our project.

Verification and Testing

We tested our robot and vision system extensively in the environment in which we planned to demonstrate its abilities, and it works "as advertised." There are a number of different lighting situations and target colors that would affect the vision system's ability to track colors and navigate successfully. We feel that the robot and camera will operate successfully in these other conditions, and if we had more time, we would explore these possibilities with further testing. If we were to do this project again, we would test and verify the correctness of our code more consistently throughout the project. When we first began working on the project, we tested each algorithm and snippet of code before integrating it into the main program. Later in the semester, we were less rigid in our testing and added multiple subroutines to the code all at once; and as a result we spent much more time debugging a complicated mess than if we had cautiously added small chunks of code little by little.

Challenges and Issues

Throughout our project, we had to overcome many problems that were unforeseen at the beginning of the semester. The challenges included both hardware and software issues.

We were unfamiliar with the CMUcam, and we could not really familiarize ourselves with the camera until the part actually arrived. This made it challenging to speculate what kind of problems the camera would present to us.

Camera image dump

The first problem we had to overcome was transferring an image dump to a PC so that we could see what the camera was seeing. This was a very important step in our project because we were using a reflective surface to supply a 360-degree field of vision. We did not have existing documentation regarding the CMUcam being used for this purpose. We also desperately needed the image dump so that we would know precisely where to place the reflective bulb. We wanted to find the best distance in order to give us the maximum and clearest field of view. The original CMUcam came with a MAX232 chip and four capacitors which are needed to dump the image serially to the PC. Unfortunately, we ordered the CMUcam that was modified for assembly onto the BOE-Bot. This camera had the chip and capacitors removed and instead had brackets installed for mounting to the robot. To enable the CMUcam to dump the image via a serial connection to the PC, we had to order the chip and capacitors. Next, we had to attach the components onto the CMUcam board. We also added a serial connection directly to the CMUcam. Our initial tests did not give us the data we had expected. After hours of testing, we discovered that the serial connector had crossed its wires with the adaptor on the robot. The off-the-shelf connector would have to be modified. We crossed the wires back to the correct pins and a connection was made. The image could finally be dumped to the PC. Being able to see the image allowed us to adjust the distance the bulb sat above the lens.

Lock on issues

The next issue we ran into was the locking onto target issue. The CMUcam was very sensitive to the lighting. Unfortunately, the lighting that we have in the lab is fluorescent light. This light caused problems for the camera by making the environment very bright. The high level of brightness caused the camera to have difficulty picking out the different shades of colors. The camera will lock on to an object that contrasts the bright surroundings. The color may not exactly match between the lock on target and the object in some cases. These cases were very minimal however. The other problem with lock on was when the target was in a shadow or darker shade; the tracking robot could not detect the object. This was a little surprising. We did not expect the camera to be so sensitive. This caused some problems during a couple of demonstrations when the object being tracked accidentally slipped into these shaded areas, and the camera lost the target. We concluded that a solution to this issue is to have the demonstration in a well-lit room and use a very bright or neon colored object made of material that minimizes reflection.

Coding the BASIC stamp

One last issue that also proved to be a somewhat difficult challenge for us was programming the BASIC stamp. The PBASIC language that we used to write the software was very limited. We had to use a series of subroutines; goto statements, and returns to write code that would enter the necessary subroutines. This type of programming would sometimes prove problematic when we nested too many goto statements together without returning out of subroutines. We also could not thread the BASIC stamp microprocessor. We had to move from subroutine to subroutine one at a time. We could not write a multithreaded program where each thread implemented a specific task. Instead, the BASIC stamp moves from one line of code to the next. This slowed down the robot reaction time. We managed to remove most of the jitter the robot gave us when performing the object tracking and obstacle avoidance in the final code. It may have been simpler and more efficient however if we had a more powerful microprocessor that could in fact implement multithreading.

Omni-Directional Vision System for Mobile Robots

Code Listing

May 5, 2003

```
'{$STAMP BS2p}



'--------initialization beep---------'
output 2
freqout 2, 2000, 1000
low rtwheel
low lftwheel

'-----Constants-----

LeftIRInputPin CON 7   'input pin from left IR
LeftIROutputPin CON 6   'output pin to left IR
RightIRInputPin CON 0   'input pin from right IR
RightIROutputPin CON 1   'output pin to right IR

SonarTrigger CON 14   'output pin to sonar
SonarEcho CON 15    'input pin from sonar
SonarComfortDistance CON 111  'sonar will signal an obstacle within
     '4 inches based on tested values


StopValue CON 1825   'servo period is from 1250 to 2500
     'stop value is 1825
Baud CON 240    'Baud rate for BS2P stamp



'-----Variables-----

pulse_timer var byte    'pulse value used for timer
PulseWidth  VAR WORD   'pulse width from sonar sensor (echo time)
Distance    VAR WORD  'converted value
    'Distance (tenths of an inch) = echo time / 7.3746
Blips       VAR NIB  'loop counter for measurement
Temp        VAR WORD   'value for comparing to sonar comfort zone
SpinFactor  VAR BYTE    'a multiple that Determines how fast BOT should
spin
'SquareRt    VAR WORD 'square root will tell the how far object is away


State       VAR NIB  'variable used to determine state of 'obstacles
      'bit2 = left IR
      'bit1 = right IR
      'bit0 = sonar
    '0=000=blocked left, right, front
    '1=001=blocked left, right
    '2=010=blocked left, front
    '3=011=blocked left only
    '4=100=blocked right, front
    '5=101=blocked right only
    '6=110=blocked front only
    '7=111=clear


'-----Initialization-----
```

```
rtwheel con 12   'right servos is pin 12
lftwheel con 13   'left servos is pin 13

input  LeftIRInputPin  'Set LeftIRInputPin to input
output LeftIROutputPin  'Set LeftIRInputPin to output
input  RightIRInputPin  'Set RightIRInputPin to input
output RightIROutputPin  'Set RightIRInputPin to output

output SonarTrigger  'Set SonarTrigger to output.
input  SonarEcho  'Set SonarEcho to input.


RcvData Var Byte(10)

n var byte
Confid var byte

' Pause 1 second for CMUcam startup
pause 1000

' Send "reset" to sync CMUcam and Stamp
serout 10, Baud, ["RS",CR]
 GOSUB WaitForACK
pause 1000

' Green LED on
serout 10, Baud, ["L1 1",CR]
 GOSUB WaitForACK
pause 100

' Turn on auto adjust for 5 seconds
serout 10, Baud, ["CR 18 44",CR]
 GOSUB WaitForACK
pause 100

' Pause 5 seconds for CMUcam to auto adjust to lighting conditions
pause 5000

' Turn off auto adjust
serout 10, Baud, ["CR 18 40 19 32",CR]
 GOSUB WaitForACK
pause 100

' Green LED auto mode
serout 10, Baud, ["L1 2",CR]
 GOSUB WaitForACK
pause 100

' Give user time to place color target close in front of camera
pause 5000

' Send command - Set poll mode - only sends one return packet -
' of data after each command - reduces data flow
serout 10, Baud, ["PM 1",CR]
 GOSUB WaitForACK
pause 100
```

```
' Send command - Set raw data mode - also suppress Ack:/Nak: to -
' further reduce serial data
serout 10, Baud, ["RM 3",CR]
pause 100

'Send command - Set Window - sets the target window on the right ride
serout 10,Baud, ["SW 20 60 30 80",CR]
pause 10

' Track Window command looks at the center of CMUcam image -
' grabs the color information and sends to the Track Color function

' Send command - Track window
serout 10, Baud, ["TW",CR]
freqout 2, 2000, 1000

' Ignore the S packet and M packet from TW
pause 2000

'Send command - Set Window, reset to the entire window
serout 10, Baud, ["SW 1 1 80 143", CR]
pause 10


'----------------------------------------------------------
'---------------------   Main   -------------------------
'----------------------------------------------------------

main:

' Send command - Track color (with no arguments) -
' will track last color grabbed by TW command


serout 10, Baud, ["TC",CR]
serin 9, Baud, [STR RcvData\10]

Confid = RcvData(9)


'SquareRt = SQR ((RCVData(2)-40)*(RCVData(2)-40) + (RCVData(3)-
75)*(RCVData(3)-75))


goto collisionDETECT


CalcSpinFactor:
  'Calculate spinFactor
  If ((RCVData(3) < 70) and (Confid > 20)) Then BigSpin
  If (RCVData(2)<40 AND (Confid > 20)) Then CalcFactorLow
  If (RCVData(2)>40 AND (Confid > 20)) Then CalcFactorHigh

IfStatements:
TrackMovements:
```

```
If ((RCVData(3) > 75) And (RCVData(3) < 119) And (RCVData(2) < 45) And
(RCVData(2) > 35) And (Confid > 20)) Then NoSpin
If ((RCVData(2) <= 35) And (Confid > 20)) Then RightSpin
If ((RCVData(2) >= 45) And (Confid > 20)) Then LeftSpin

'if object is centered in front and far way from BOT then move forward
IF ((RCVData(3)  > 120) And (Confid > 20)) Then Forward

goto main
end



BigSpin:
  SpinFactor = (76 - RCVData(3))
goto IfStatements

CalcFactorLow:
  SpinFactor = 40-RCVData(2)
goto IfStatements

CalcFactorHigh:
  SpinFactor = RCVData(2)-40
goto IfStatements


collisionDETECT:
  'check sonar
  PulseWidth=0
  PULSOUT SonarTrigger, 9                 '10 uS trigger pulse
  PULSIN SonarEcho, 1, PulseWidth         'measure distance to target

  Distance = PulseWidth ** 8886           'divide by 7.3746

  IF Distance <= SOnarComfortDistance THEN BitZero
  IF Distance > SOnarComfortDistance THEN BitOne

  BitOne:
    'set state.bit0 to 1
    State.bit0=1
  goto afterSonar

  BitZero:
    'set state.bit0 to 0
    State.bit0=0
  goto afterSonar

afterSonar:
  'check_IR
  FREQOUT LeftIROutputPin, 1, 10212       'Check left IR
  State.bit2 = in7
  FREQOUT RightIROutputPin, 1, 10212            'Check right IR
  State.bit1 = in0


  IF ( (State.bit2=1) AND (State.bit1=1) AND (State.bit0=1)) THEN
CalcSpinFactor
```

```
  'if previous staement not taken, then must avoid obstacle
  goto collisionAVOID

collisionAVOID:
  'Signal collision detected
  '-----State Cases-----
  '0=000=blocked left, right, front
  IF ( (State.bit2=0) AND (State.bit1=0) AND (State.bit0=0)) THEN
Backup
  '1=001=blocked left, right
  IF ( (State.bit2=0) AND (State.bit1=0) AND (State.bit0=1)) THEN
Backup
  '2=010=blocked left, front
  IF ( (State.bit2=0) AND (State.bit1=1) AND (State.bit0=0)) THEN
Right90
  '3=011=blocked left only
  IF ( (State.bit2=0) AND (State.bit1=1) AND (State.bit0=1)) THEN
GradualRight
  '4=100=blocked right, front
  IF ( (State.bit2=1) AND (State.bit1=0) AND (State.bit0=0)) THEN
Left90
  '5=101=blocked right only
  IF ( (State.bit2=1) AND (State.bit1=0) AND (State.bit0=1)) THEN
GradualLeft
  '6=110=blocked front only
  IF ( (State.bit2=1) AND (State.bit1=1) AND (State.bit0=0)) THEN
BackLeftAvoid
  '7=111=clear -- no change in direction
GOTO main


'-----------------------------------------------------------
'---------------- Navigation Routines ---------------------
'-----------------------------------------------------------
BackLeftAvoid:
'back up then move forward and left
    freqout 2,600,1000
    freqout 2,600,1000
    freqout 2,600,1000
    freqout 2,600,1000
    for pulse_timer = 0 to 60
      pulsout rtwheel, 2456
      pulsout lftwheel, 1650
      pause 20
    next
   for pulse_timer=0 to 80
      pulsout rtwheel, 1650
      pulsout lftwheel, 2456
      pause 20
   next
goto forward

BackRightAvoid:
'back up then move forward and right
    freqout 2,600,1000
    freqout 2,600,1000
    freqout 2,600,1000
    freqout 2,600,1000
    for pulse_timer = 0 to 40
      pulsout rtwheel, 2456
      pulsout lftwheel, 1250
      pause 20
    next
   for pulse_timer=0 to 40
      pulsout rtwheel, 1650
      pulsout lftwheel, 2456
      pause 20
   next
goto main

NoSpin:
  'tell servos to stop
    pulsout rtwheel, 1850
    pulsout lftwheel, 1856
    pause 20
  freqout 2,1000,1000
  freqout 2,1000,750
  freqout 2,1000,500
goto main

RightSpin:
  for pulse_timer = 0 to SpinFactor
    pulsout rtwheel, 2250
    pulsout lftwheel, 2256
    pause 20
  next
goto main


LeftSpin:
  for pulse_timer = 0 to SpinFactor
    pulsout rtwheel, 1450
    pulsout lftwheel, 1456
    pause 20
  next
goto main


  NoMove:                            'code to Stop servos
    PULSOUT rtwheel, 1850
    PULSOUT lftwheel, 1856
    pause 20
  GOTO main

Forward:
   for pulse_timer=0 to 20
      pulsout rtwheel, 1250
      pulsout lftwheel, 2456
      pause 20
   next
GOTO main

  Backup:
    freqout 2,600,1000
    freqout 2,600,1000
    freqout 2,600,1000
    freqout 2,600,1000
    for pulse_timer = 0 to 20
      pulsout rtwheel, 2456
      pulsout lftwheel, 1250
      pause 20
    next
  GOTO main


  Right90:
    freqout 2,600,2000
    freqout 2,600,2000
    for pulse_timer = 0 to 35
      pulsout rtwheel, 2500
      pulsout lftwheel, 2400
      pause 20
    next
  goto Forward

  Left90:
    freqout 2,600,2000
    freqout 2,600,2000
    for pulse_timer = 0 to 35
      pulsout rtwheel, 1250
      pulsout lftwheel, 1250
      pause 20
    next
  goto Forward

  GradualLeft:
    freqout 2,600,500
    for pulse_timer=0 to 20
      pulsout rtwheel, 1250
      pulsout lftwheel, 2256
      pause 20
    next
  GOTO main


  GradualRight:
    freqout 2,600,500
    for pulse_timer=0 to 20
      pulsout rtwheel, 1450
      pulsout lftwheel, 2456
      pause 20
    next
  GOTO main

WaitForACK:
 serin 9,Baud, 1, Timeout, [Wait (":")]
Timeout:
RETURN
```