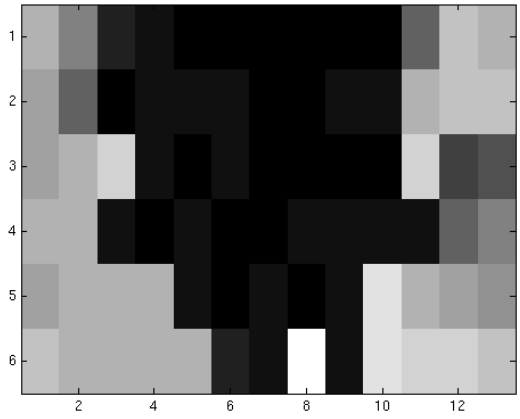
Sensor Report
for Lunar-cy

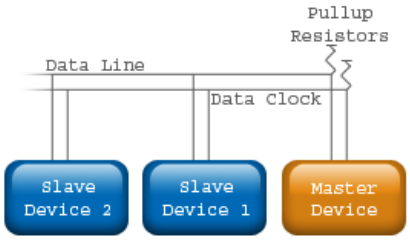
Brad Mouring
Lunar-cy
EEL 5666
Intelligent Machine Design Lab
Dr. Arroyo
Dr. Schwartz
Adam Barnett
Kevin Claycomb

Sensors are the robot's window into the real, analog world (unless you believe we are all in the Matrix, in which case it's a window into a simulation of the real, analog world). The controllers in the robot gather data from the sensors, review the data, and modify its behavior based on that information. This simply means that the sensors play a crucial role in any project of this scope.

In order to detect and avoid collisions, my project utilizes two ultrasonic range finders. I used a pair of Devantech SRF10s. The SRF10 allows me to chain sensors on the TWI bus, simplifying wiring and minimizing used I/O on the controller board. These sensors receive a request to ping, and when ready, can be read from to get the perceived distance to an object. This frees the controller board to continue with other tasks while the sensor is gathering the reading. By determining the distance to a perceived object, coupled with the readings from the camera, the robot can appropriately react to that object. Below is a figure depicting the results of testing the accuracy of the sonar's perceived distance to an object and the actual distance (blacks are close to the actual distance, whites indicate a larger discrepancy between actual distance and perceived distance). This reveals the sensing cone as well as a noticeable dead spot near the receiving transceiver within 1 inch, something that should not affect my robot but is noteworthy nonetheless.



The wiring of the sensors is made quite easy with the use of the I^2C (or TWI) bus. This bus uses two signal wires, a serial data line and a data clock line. All communication is initiated by the bus master, in this case the Mavric board. Depending on the request type sent by the master to initiate communication, either sender or receiver may control the data clock. Multiple devices may reside on the bus as each device has an address to which it alone will answer. Below is a simplified diagram of the wiring of the TWI bus.



A bump sensor is used in the gripper mechanism to determine when the arm has the block properly positioned for pickup. This is a fine example of a simple solution to a simple problem.

The most “visible” sensor (pardon the pun) is the camera. This, when used in conjunction with detection code detailed in the final report, locates blocks and returns the heading and distance of the block. This

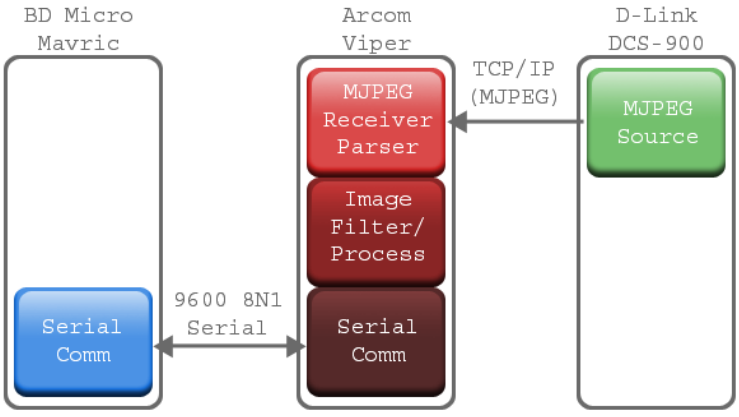
complexity is added in order to overcome some of the shortcomings of the various pre-made vision solutions such as the CMUcam. I've decided to term the system I have made the BradCam.

The system consists of a network-capable camera, the D-Link DCS-900, and an Intel XScale single-board computer (SBC), the Arcom Viper, running a spartan Linux based around uClib (optimized GCC library for smaller embedded architectures) and BusyBox (a system of packaging many common Unix tools that drastically reduces space requirements at the cost of feature completeness).

The DCS-900 can provide either a single JPEG current image from the camera or a stream of current images in MJPEG stream format. The single image would be simpler to implement, however through experimentation it is discovered that the camera takes approximately 0.25-0.5 seconds to respond to a request for an image, for each image. This automatically limits the framerate available to the vision algorithms to only 2-4 frames a second. The MJPEG stream, on the other hand, while still having the same initial delay of approximately ¼ to ½ of a second, will thereafter provide a near-constant 30 frames a second, more than sufficient for my needs and allows for future expansion. It was after this experimentation that I decided to implement the more difficult but usable MJPEG backend.

The standard JPEG decompression library available to the GNU Compiler Collection (GCC) provides a framework for other developers to design specialized "source" and "destination" management code for decompression reading and compression writing, respectively. I did some cursory searching for online sources of examples, finding only a few examples of how the included source and destination managers were written (the provided managers worked on Files, so they were useful as a reference but not much else).

Ultimately, the source manager I created would spawn a thread that would only serve to continuously retrieve the MJPEG stream, parse out a JPEG from the stream, and when requested, load a compressed JPEG into a mutex-protected buffer. This allows the system to have the latest image available on request up to 30 frames per second. As concurrent programming can be a source of incredibly difficult-to-debug issues, this code was carefully examined and audited for concurrency issues as well as any memory leaks. After decompression, simple "distance from the desired color" calculations are performed on each pixel, with both horizontal and vertical histograms kept during calculation. These histograms are used to determine the center of the block and are sent back to the Mavric board via a serial link. The X coordinate can be used for heading information while the Y coordinate can be used to determine approximate distance to the block. Further experimentation with new image procession techniques will be discussed in the final report's future work section. Below is a figure depicting the interaction between the various components to achieve the described vision system.



Another sensor that was created and successfully tested but not currently used was a pair of motor encoders that are coupled to a stand-alone encoder counter board. These will be used to determine how far the robot has turned when trying to reacquire the next block to pick up. Also, this can be used as a

sanity check if the sensors indicate that the robot has stopped moving. The encoder counter board has been designed, created, and tested successfully and will be used in future work.

The camera and sonars work together to give the robot's intelligence a picture of the surrounding environment so that the software can make intelligent decisions regarding the next action. The bump sensor and encoder board (and with some interpretation the other two sensors) give the software feedback on its actions in the real world.

Some of the lessons learned from working with these sensors are that you can never do too much testing, especially when concurrent programming you wrote is an integral part of one. In the end my undergrad software engineering background served me well as there were no issues with the vision system code, however it would have been very easy to make a small mistake that would work *most* of the time and fail at the worst possible time.

Another difficult set of lessons learned resulted from one of my sonars dying suddenly less than a week before demo day. If you have a sensor you are relying on, if you can afford it you should buy at least one spare. Secondly, if you failed to observe the previous warning, if you need a part quickly, do some cursory checking of the shop you are buying from to see if they are timely about their shipping. I failed to do this as well and ended up having to order the part again a few days later from another vendor and start a credit reversal for the first order since the company (Junun.org) had been missing shipments for the past few months.