A Low Resolution Vision System

E155 Final Project Report

Charles Matlack and Andrew Mattheisen

January 2, 2003

Abstract

This project uses an array of 24 CdS photocells to form a crude image of its field of view, and from this image detect and seek sources of light and/or moving objects. The sensor array is mounted on a 2 DOF platform, allowing it to pan and tilt in the direction of interest. An HC11EVB reads sensor values via its A/D port (with the help of analog multiplexers), controls two servos, and drives a 24 LED array via a serial shift register implemented on a Xilinx Spartan XCS10 FPGA. Using a light-seeking algorithm, the system can track a flashlight or a high-contrast moving object. Due to the characteristics of the sensors, a successful pure motion-seeking algorithm could not be implemented.

1 Introduction

Robotic vision systems traditionally fall into either the extremely primitive and limited hobbyist category, or the graduate/industrial research level category, employing such computing power that they could just as easily raytrace the scene they are attempting to view. This project seeks to accomplish a reasonably advanced visual task- tracking a specific and easily identified target- while retaining the hobbyist's elegance of minimimal and cheap hardware. Thus, we find ourselves using an HC11EVB in conjunction with an FPGA to control a simple low-resolution vision system consisting of an array of 24 CdS cells mounted on a platform that can be panned and tilted by a pair of hobby servos. To build the system efficiently, we broke it down into several subsystems with the goal that each could be constructed and tested independently, minimizing the possible sources of error in integrating all the subsystems at the end.

The movement subsystem consists of two servos and a lego platform, as well as the HC11 code needed to drive the servos. The display subsystem is an array of LEDs driven by a serial shift register implemented on the FPGA, which in turn is driven by HC11 code using the SPI port. The A/D subsystem is the circuit composed of CdS cells in series with matching resistors (forming voltage dividers) and analog multiplexers that allows the HC11 to scan the array of 24 CdS cells efficiently. The control subsystem will interpret sensor data and implement light and motion tracking algorithms.

2 New Hardware

2.1 Hitec HS-300 Hobby Servos

To tilt and pan our sensor array, we needed some kind of actuators with minimal response time that are also easy to interface to the HC11. Hobby servos in general make for a good solution to this problem. We had two Hitec Servos on hand and decided to use them after checking their performance specifications on the internet. The Hitec HS-300's we used have 180 degrees of rotation and are specified to deliver 49 oz/in torque and to rotate 60 degrees in 0.17 seconds with a 6.0v power supply. Interfacing them to the HC11 was extremely simple.

All hobby servos, with the exception of newer digital types, are controlled by sending them positive pulses between 1.0 and 2.0ms in length. The length of the pulse determines the servo's desired position, i.e. a 1.5ms pulse will send the servo to it's center of rotation. The servo only tries to reach/hold this position while it is recieving pulses. This means that with power connected but no control signal the servo only offers the resistance of its gear box to external torques; conversely, the servo will oppose external torques to hold its position while recieving pulses.

To control the servos, we provided them with 5VDC from the system power supply and connected their control inputs to PA4 and PA6, which were controlled by the HC11's output compare function (see Microcontroller Design section for control code algorithm details). The servos worked well to move our sensor array; we had built it to be as symmetrical as possible and with minimal moments of intertia about the servo rotation axes. Oscillation of the structure when it stopped moving posed a potential problem but was avoided by moving the servos slowly and continuously, minimizing sudden movements.

2.2 CdS Photocells

There were several major reasons why we used CdS cells as the sensing elements in our vision system, and a single more important reason why we shouldn't have. Photocells have the highest sensing to package area ratio of small and inexpensive light sensors, and they have a frequency response very similar to that of the human eye. Furthermore, they vary their resistance with changes in light intensity, minimizing the interface circuit needed to connect them to an A/D converter. While photocells simply require a matching resistor to form a voltage divider, sensors such as photodiodes require more complex op-amp circuits to produce a voltage that can be read by an A/D converter. Photocells have a relatively slow response time, but not slow enough to affect their performance in our application; testing in the lab revealed that they are able to pick up the 60Hz flickering of fluorescent lights.

The differences among the responses of photocells and their nonlinearity made them unsuitable. Photocells are generally used alone in commercial applications, which is reflected in their extremely loose manufacturing tolerances. Further elaboration on this problem and

how we worked around it are in the Results section. In summary, photocells are poorly suited to comparative sensing applications; they should only be used when a single rough value for light intensity is needed.

2.3 Toshiba TC4052B CMOS Dual Quad Input Analog Multiplexers

To read all 24 photocells with the HC11's 8 A/D converters, we needed a way to multiplex the voltages generated by the photocell voltage divider circuits. As it turned out analog multiplexers are available that can lower their resistance to the selected signal to only a few hundred ohms, low enough to pass our voltage essentially unchanged. With 24 signal sources for 8 inputs, we needed a three-input multiplexer for each channel. Also relevant was that groups of 4 A/D inputs could be switched to the same multiplexer input simultaneously. Thus, the Toshiba chip, which contains a pair of 4-input multiplexers sharing the same control lines, fulfilled our requirements nicely- see the datasheet in the appendicies for more details. To operate the multiplexers, the control lines were connected to four Port B outputs on the HC11 and the inputs and outputs were connected to A/D inputs and CdS circuits respectively. See the overall schematic in the appendicies.

2.4 CdS Photocells

3 Schematics

Each CdS cell is in a voltage divider with a 330k resistor. As incident light increases, more voltage is dropped across the resistor because the CdS's resistance decreases. The voltage across all 330k resistors is sent to the analog multiplexers. The HC11 reads these values through it's A/D port 4 at a time. Control signals from Port B determine which CdS signals are read. Data is stored as two byte numbers ranging from 00 hex to FF hex. High and low voltage references connected to the 5VDC supply and ground insure that data spans the entire numerical range. The HC11 converts this data into servo control and SPI output signals through various algorithms discussed later. The servos are controlled through Output Compare ports. By varying signal duty cycle, the HC11 changes the servos' positions. The

HC11 sends data related to the CdS cells' resistances to the FPGA. The FPGA contains a 24 bit serial shift register that outputs this data to 24 LEDs on the protoboard. This data is used to debug CdS irregularities. There is a current limiting resistor between the FPGA and each LED, with the other side of the LEDs tied to ground.

4 Microcontroller Design

The HC11EVB had several peripheral interfacing tasks to perform as well as implementing our light tracking algorithm, including driving the LED array via a serial shift register, controlling the servos, and reading the photocell array.

Control of the servos was trivial using the output compare system. Output compare 1 sets both servo controls high on each clock counter overflow; output compares 2 and 4 set the two lines low a set period of time later. Thus, by writing to output compare 2 and 4 registers the duration of the generated pulses can be changed, instructing the servos to rotate to a new position. See the program appendix for the code that initializes this control scheme.

Reading the voltages generated by the photocell circuits is straightforward but somewhat complicated by the desire to minimize the time needed to scan all 24 sensors. The algorithm we arrived at initializes an A/D conversion, changes the multiplexer control lines corresponding to the 4 channels not being sampled, waits for the conversion to complete, and copies the results to an array before repeating the process. Port B is written to directly to control the multiplexers, and the A/D converter is operated in single-conversion multiple-channel mode.

Interfacing to the LED array on the serial shift register is trivial: three bytes are sent to the SPI to update the array. The more difficult problem is condensing 24 bytes of data from the sensors into 3 bytes, with a bit corresponding to each byte. To accomplish this, we compare each byte to a threshold and set the corresponding bit based on the result of this comparison. This is done in a loop with two synchronized index variables- one to iterate through the SPI bits 3 times and one to iterate through the 24 byte data array.

The top-level control algorithm we used and the process by which we developed it is discussed in detail in the Results section. Most of the building block algorithms used to

implement it that deal with an array of data are coded such that the size of the array is an easily changed parameter, making it easy to switch from 24 sensor items to 4 averaged sensor data items. Most of these algorithms also include code to apply offsets to sensor data.

One of the first and most basic operation performed on incoming sensor readings is averaging data corresponding to groups of 6 sensors; since these groups consist of non-contiguous portions of the array, this algorithm had to be hardcoded for each sensor. The data for each sensor is divided by 6 before being added to an accumulated average. This code includes commands to add an offset and multiply by a constant; we add a different constant to each average to normalize them.

To decide how to move the servos, we find the greatest element in the array representing the average light level in each of the 4 quadrants. With this information we set a variable with the direction to move encoded as two flags; code follows that reads this variable and adjusts the contents of the output compare registers by a constant to reflect the desired movement.

5 FPGA Design

The FPGA is programmed with a 24-bit shift register. It grabs a bit of data from the HC11 SPI on the rising clock edge of the SPI clock (SCK). It displays information about each of the CdS cells in our array (see overall schematic appendix). This allows us to compare cells relative responses to light levels and changes between light levels. The LEDs are spatially arranged to display the world from the robots point of view. Construction of the status display subsystem was broken into three parts: breadboarding the LED array and implementing the 24-bit shift register in Verilog, and then writing the HC11 code to drive the display over the SPI. The Verilog code is fairly simple, as was breadboarding the array. Proper operation of the shift register and array was verified by connecting the SCK pin to the MCLK pin of the FPGA board, and connecting the MOSI pin to either Vdd or Gnd. Then the expected performance could be observed while flipping the toggle switch with the clock set to manual mode; positive edge-triggered clocking and proper ordering of the LEDs were noted. Initially the FPGA was to be used to debug movement algorithms by

displaying the LED to move towards. Instead, the FPGA was used to expose and explore the irregularities between the 24 CdS cells.

6 Results

Construction and testing of the project was broken into modules, which were completed as independently as possible before integrating them and completing the high level algorithm that depended on all the others. High-level algorithm development had two main thrusts to accomplish what we wanted to do. As stated in the proposal, we wanted to be able to track a light source as well as a specific object. To do this, we wanted to write a simple brightness-seeking algorithm as well as a motion-seeking algorithm. The first algorithm was the only one that performed successfully due to differences in the responses of the sensors; fortunately we were able to use this algorithm to perform both of the tasks we needed to. The remainder of this section will focus on the development process for these two algorithms and how we tried to work around the limitations of our sensors.

The first piece of code to integrate multiple modules of the project turned on LEDs in the array when the response of the corresponding photocell was above a threshold. To our utter horror, we found that the LEDs turned on and off in a relatively conistent and arbitrary order regardless of the way we illuminated the sensor array. Hoping that responses were at least linear, we put together an algorithm that accumulated differences between recent sensor readings until one of the accumulators reached a threshold and then applied a different threshold to these values to drive the LED display. Again we were disappointed- there was evidence of correlation to incident light in the way the LEDs behaved, but it was obvious that some sensors were much more sensitive than others, enough so that no simple threshold-based processing of the data could produce useful information.

6.1 Blood From a Turnip: Arriving at a Functional Light Seeking Algorithm

Having been thwarted in our attempts to use the straightforward light- and motion-seeking algorithms we had planned to use, we investigated alternatives. We briefly examined individual sensor values under varying light levels to see if a certain number of them had

similar behavior. This failed, so we moved on to try using different matching resistors to compensate for the differences. This method brought two sensors to the same A/D value for a particular light level, but the values became uncomparable if the light level significantly changed. Finally, we hit upon the idea that averaging enough sensors should result in similar performance among groups- a 'law of large numbers' line of reasoning.

To test this idea, we simply added code to take the averages of incoming A/D values before they were processed in any other way and applied the new dataset to our previous algorithm. We chose quadrants of 6 as the largest group size that would still provide us with reasonable information about which way to move. Still we found significant differences in response among the groups.

To improve the performance of the light-seeking algorithm, we added offsets to the average values, essentially doing numerically what we had attempted before electrically with matching resistors. This gave us an algorithm that could seek light sources, but the offsets only worked near a specific light level that we could calibrate for. We were now able to track a flashlight and a high-contrast object using different offset values.

It occured to us that we should take the offsets a step farther and attempt some kind of linear approximations to increase the working range of light levels. While the responses of the photocells were incredibly different and nonlinear, it seemed that they should still be similar in some vague way. Hopefully, they would be similar enough that the response of one photocell would be a nearly linear function of another. To test this hypothesis, we recorded group-averaged but otherwise raw sensor values at several different light levels and plotted them as a function of the group average that showed the greatest and most consistent variation. The data turned out to be surprisingly linear, with a flattening of the slope at the top and bottom due to the behavior of the voltage divider. Truncating the data at the extremes, we did linear fits which resulted in R values slightly above .99; see appendicies. While this seemed promising, our hopes were crushed as the implemented linear approximation algorithm performed as poorly as if not worse than the one that only used offsets.

6.2 Kicking a Dead Horse: Further Modifications to the Motion Sensing Algorithm

The motion sensing algorithm was still in bad shape after averaging sensor values, so we tried a few more modifications. The sensor values we saw reflected gross differences in response but still roughly correlated to movement in the field of view, however, when there was no movement the sensor values became more random and varied over similar ranges. Based on this observation, we made a slight change to the algorithm: instead of waiting for a difference accumulator to reach a threshold, we ran the accumulation loop a set number of iterations before analyzing the data and starting over. This eliminated the random values in the absence of movement and improved the correlation of the values to movement.

At this point we also realized that our current averaging algorithm was poorly suited to detecting motion- if, for example, the light upon one sensor decreased as the light upon another sensor in the same group increased, there would be a cancelling effect. This method essentially turned the whole quadrant into one pixel for analysis. A better way of processing sensor input is to keep all 24 values and accumulate 24 differences, averaging the differences at the end of the loop. This method takes advantage of the available resolution, it just doesn't 'trust' any individual pixels.

Having made these two changes to the motion algorithm, we saw much improved correlation of the quadrant values to movement, but we were still unable to distill meaningful information by applying offsets and noise thresholds to these values due to the nonlinear response. It turned out that sensor fluctuation is a function of ambient light, so changes in ambient light would result in an increase in false movement detection.

Despite the apparent failure of this idea, we now had another movement detection possibility- if the average accumulated change in sensor readings varies with ambient light, the recent change in *these* values should be an indicator of the change in ambient light, i.e. movement! So we implemented this and tried a new way of normalizing the values- dividing a value by it's previous value to get a relative change would hopefully eliminate most of the remaining response differences. It was troublesome to do with only integer division, and it didn't work. Between normalizing this way and applying offsets to the recent change in average accumu-

lated differences, we saw numbers that correlated well enough to movement in the field of view to where we could almost deduce the direction of movement by watching them, but the values were still short of being useful to a servo control algorithm.