Summary of Circuits Team's Activities—Fall 2007

Steven Gilson, Jay Dev Mahadevan, Abraham Cantwell

PREPARED BY: Steve Gilson, Abraham Cantwell, Jaydev Mahadevan

OBJECTIVES:

1. Design and implement an optical encoder system to measure the angular velocity of each wheel for SpongeBob
2. Design and implement sonar sensors on SpongeBob and transmit data to ITX
3. Interface the Gladiator Technologies inertial measurement unit with Matlab for use on SpongeBob or Gladiator

DETAILED DESCRIPTION

OPTICAL ENCODER SYSTEM

OBJECTIVES:

1. Construct four optical encoder sensors for implementation on SpongeBob's wheels
2. Use a cost-effective design for the encoder circuits
3. Deliver the completed encoders to the Frame Team for implementation

An optical encoder sensor was to be configured on each wheel to provide feedback to the main computer of each wheel's angular velocity. This allowed the computer to verify that the robot is moving at the correct speed and in the correct direction. Each encoder consisted of an encoder wheel, two IR sensors (IR LED and photodiode in one chip) and external resistors and capacitors. The circuit biased the IR LED at the correct operating point and included a pull-up resistor for the circuit output. The output was driven low by the IR sensor when the white segments of the encoder wheel were detected. The circuit design was provided by Vikas Reddy from M&AE 378: Mechatronics.

At first a custom printed circuit board (PCB) was designed for these circuits. This PCB was to fit inside each wheel of the robot and implement the encoder circuit. Unfortunately, after completing the design and parts list for this PCB, it was determined to be too expensive to have the boards fabricated.

The low cost solution was to construct each encoder circuit on a small breadboard as was done in the Mechatronics lab. Four fully-functional encoder boards were assembled and tested. They were then provided to Vikas Reddy for implementation on SpongeBob.

In the future, for the Gladiator robot, a commercial off-the-shelf (COTS) encoder system will most likely be pursued. This would provide more accuracy and easier integration into the robot than a custom built solution. However, the PCB that was originally designed for SpongeBob can also be adapted to fit into Gladiator.

SONAR SYSTEM

OBJECTIVES:

1. Determine how the MaxBotix sensors work and how to integrate them with a computer
2. Integrate an array of MaxBotix sonar sensors with the ATmega32
3. Collect and process analog data
4. Pass complete sensor data to the computer via a serial connection
5. Install the sensors and processing circuit on SpongeBob

An ATmega32, placed within on the "Mega32" board provided by Vikas Reddy, formed the core of the sonar system. Sensors were powered and grounded through individual connections to the Vcc and ground on the "Mega32" board. The analog data pin on each sensor was then connected directly to the PORTA pins on the ATmega32. This setup allowed for a total of eight sensors to be run at a time. If necessary, additional sensors could be added through an external multiplexer. Some accommodations were made for this operation within the ATmega32's code but it was never executed due to the low number of available, operational sensors (six). However, the system is prepped for this type of operation if it proves necessary in the future. Two specific methods of operation were created, to do a tradeoff between speed and sensor accuracy.

In the first mode, the internal multiplexer sequentially stepped through the 8 sensors. These values were then fed to the internal analog to digital converter (ADC) which performed the conversion and temporarily stored the value in the ATmega32's memory. The ATmega32 then stepped to the next pin and repeated the process. Once a complete read of all 8 sensors was complete, the data was concatenated into a string and passed out through an RS-232 serial connection at a baud rate of 9600bps. The formatting of this string was determined by the CS team and was designed for easy connectivity between the ATmega32 and the main computer. The specific formatting was as follows:

&Sensor1;Sensor2;Sensor3;Sensor4;Sensor5;
Sensor6;Sensor7;Sensor8#

This process was free running and continued to output data approximately every 400 ns when provided power.

The second mode, while very similar, attempted to deal with the problem of interference between the sensors. Because the pings sent by the MaxBotix sensors were not coded, potential for interference between sensors was possible. Initial tests of the system within an enclosed space suggested that this interference could substantially degrade the quality of the data. Accordingly, a system was designed in which each sensor was powered on only for the period in which it was being sampled. This was achieved by powering the sensors directly from the output pins of the ATmega32 and providing the necessary code to power and shutdown the sensors as necessary. Unfortunately, while increasing accuracy, this system increased the amount of time finish reading the sensors as it was necessary to wait until the power to the sensor stabilized before taking a reading. This time, multiplied across the eight sensors yielded a full read time of about 1.5 seconds per cycle. Conversion and decoding operated the same as for the first system.

IMU INTERFACE
OBJECTIVES:

1. Read and parse data from the IMU into Matlab, and store the data for later processing
2. Ensure parser meets requirements of Computing Team

The Inertial Measurement Unit is a device that can calculate its own rotational velocity and translational acceleration with very high accuracy and precision. It will be used by the Minesweeper Computing Team to aid in GPS tracking and route navigation algorithms for the robot.

The initial task was to connect the IMU to a computer and then read the measurements that the IMU provides using a Matlab script. At a later date, the Computing Team plans to convert the Matlab code into C. The IMU connects to a computer via serial port (RS-232). Basic design requirements necessitate reliable communication between the IMU and processer, so these concerns were addressed first. As shown in Figure 1, the IMU transmits 115,200 bits/sec, and for each group of 8 bits transmitted there is 1 stop bit but no parity bit. These conditions needed to be set as property values of the variable that controls serial port communication in Matlab. Also, the serial input buffer size was set to 50,000 bytes to ensure that there would be no incoming data memory overflow. Each time it runs, the script extracts 10 samples worth of data for each velocity/acceleration field in about 16 ms – the Computing Team only requires 1 sample every second, so this should be more than enough.

Arriving data is read in as char (8 bit) values and then converted into hexadecimal. A message packet consists of a start byte, followed by the processed values from the IMU, and finally a checksum to make certain there were no errors during transmission. The Matlab script searches the data sequentially until it finds the start byte (0x3E for every message) and then parses the data into its fields, which are received in the same order each time. Validating the checksum would have been a good verification step, but requires binary computation which is difficult in Matlab but easy to code in C. Thus, it was decided to leave this to the Computing Team. Then the data is converted from hexadecimal to

(signed) decimal. The IMU is capable of detecting rotational velocity and translational acceleration in the x, y, and z directions, but the Computing Team only requires acceleration in the x and y direction and velocity in the z direction, so those were the only measurements that were saved. (The Matlab function that performs all the serial parsing is called readIMU() and is included in the appendix.)

Some caveats: all acceleration and velocity fields are 16 bits, but they are received in Little Endian format (i.e. LSB first). Also, the least significant bit for the data is weighted as 0.01 deg/sec for rotational velocity and 0.001 g for translational acceleration. Furthermore, it was discovered with much frustration that Matlab cannot detect signed hexadecimal numbers, so the Circuits Team had to write its own Matlab function for converting from signed hexadecimal to signed decimal format. (This function is called shex2dec() and is included in the appendix.)

CONTACTS

Gladiator Technologies, Inc.
Technical Support Hotline: 425-391-0229 x223
techsupport@gladiatortechnologies.com

IMAGES

Parameter	Value
Bits/second:	115200
Data bits:	8
Parity:	N
Stop bits:	1

Figure 1: IMU Serial Communication Settings

Description	Number of bytes	Value	LSB weight
Start of message	1	0x3E	N/A
Message counter	1	Mod 256 counter	N/A
Gyro - X axis	2	Signed 16-bit integer	0.01 deg/sec
Gyro - Y axis	2	Signed 16-bit integer	0.01 deg/sec
Gyro - Z axis	2	Signed 16-bit integer	0.01 deg/sec
Accel - X axis	2	Signed 16-bit integer	0.001 g
Accel - Y axis	2	Signed 16-bit integer	0.001 g
Accel - Z axis	2	Signed 16-bit integer	0.001 g
Temperature	2	Signed 16-bit integer	0.01 deg C
Test indicator	1	Normal=0x00, test=0xFF	N/A
Checksum	1	Two's complement sum	N/A
Total size	**18**		

Figure 2: IMU Message Packet Format

APPENDIX

MATLAB FUNCTION readIMU()

```
function [accelx_num,accely_num,gyroz_num] = readIMU()

s = serial('COM1');
```

```
s.InputBufferSize = 50000;  %set buffer to 50000 bytes
s.BaudRate = 115200;
s.DataBits = 8;
s.StopBits = 1;
fopen(s)

[indata,count] = fread(s,1800,'char');  %read data values as char -
1800 bytes

%turn char data into hex
indata_hex = dec2hex(indata);
[rows, cols] = size(indata_hex);

%matrices for storing extracted data
gyro_x = [];
gyro_y = [];
gyro_z = [];
accel_x = [];
accel_y = [];
accel_z = [];
temp = [];

startval = find(indata == 62);   %find start byte (3E in hex, 62 in
dec)

%parse data stream
for i = startval(1):18:rows
    gyro_x = [gyro_x; indata_hex(i+3,:),indata_hex(i+2,:)];
    gyro_y = [gyro_y; indata_hex(i+5,:),indata_hex(i+4,:)];
    gyro_z = [gyro_z; indata_hex(i+7,:),indata_hex(i+6,:)];
    accel_x = [accel_x; indata_hex(i+9,:),indata_hex(i+8,:)];
    accel_y = [accel_y; indata_hex(i+11,:),indata_hex(i+10,:)];
    accel_z = [accel_z; indata_hex(i+13,:),indata_hex(i+12,:)];
    temp = [temp; indata_hex(i+15,:),indata_hex(i+14,:)];
end

gyrox_num = cast([],'double');
gyroy_num = cast([],'double');
gyroz_num = cast([],'double');
accelx_num = cast([],'double');
accely_num = cast([],'double');
accelz_num = cast([],'double');
temp_num = cast([],'double');

[datarows datacols] = size(gyro_x);
%convert data from hex to decimal
for i = 1:datarows
    gyrox_num(i,:) = shex2dec(gyro_x(i,:))/100;
    gyroy_num(i,:) = shex2dec(gyro_y(i,:))/100;
    gyroz_num(i,:) = shex2dec(gyro_z(i,:))/100;
    accelx_num(i,:) = shex2dec(accel_x(i,:))/1000;
    accely_num(i,:) = shex2dec(accel_y(i,:))/1000;
    accelz_num(i,:) = shex2dec(accel_z(i,:))/1000;
    temp_num(i,:) = shex2dec(temp(i,:))/100;
```

```
end

%convert from g's to m/s^2
accelx_num = accelx_num * 9.80665;
accely_num = accely_num * 9.80665;

%close serial port
fclose(s)
delete(s)
clear s
```

MATLAB FUNCTION shex2dec()

```
%convert signed hex number to decimal
%hexnum should be a string
function decnum = shex2dec(hexnum)

binnum = dec2bin(hex2dec(hexnum));  %get binary number from hex
l = length(binnum);
zerochar = '0';
zerostr = [];
for j = 1:16-l
    zerostr = [zerostr,zerochar];
end
binnum = [zerostr,binnum];   %pad w/ zeros to length 16
decnum = 0;
for i = 1:16    %convert to decimal from binary
    if (i == 1)
        decnum = decnum - 2^(16-1)*str2num(binnum(1));
    else
        decnum = decnum + 2^(16-i)*str2num(binnum(i));
    end
end
```