AUTOMATED GUIDED VEHICLE:
SPECIAL SENSOR REPORT

Prepared for: A. A Arroyo, Instructor
Eric M. Schwartz, Instructor
William Dubel, TA
Steven Pickles, TA

Prepared by: Trevor Skipp, Student

March 17, 2005

University of Florida
Department of Electrical and Computer Engineering
EEL 5666: Intelligent Machines Design Laboratory

CONTENTS

ABSTRACT

"Automated Guided Vehicle"

By Trevor Skipp

The objective of this project is to develop a robot that autonomously moves pallets around a model warehouse floor. The Automated Guided Vehicle, AGV, uses several sensor suites to characterize its behaviors. An infrared remote control allows the robot to dynamically receive part orders. Its capabilities are enhanced by the ability to send and receive tasks to an Automated Storage and Retrieval System through RF data communication. While operating, infrared and bump sensors allow the vehicle to avoid collisions.

SENSOR SUITES

1. INFRARED DETECTOR

The goal of this project is to develop a robot that streamlines the warehousing process. An infrared remote control will allow the user to dynamically communicate with the vehicle on the factory floor. Buttons on the remote control correspond to requests to store or retrieve a specific pallet.

A Sony television remote control is used to send infrared data (see Appendix A). Each button on the remote has a unique bit pattern. When a button is pressed, the remote forms a packet of data including start bits, data elements, and stop bits. Digital Signal Encoding is used, and the packet is sent serially through a 40kHz modulator. Modulating the signal decreases the probability of ambient infrared crosstalk. After modulation, the signal is sent to the infrared generator.

The AGV uses a Fairchild Semiconductor infrared detector to receive the signal. A bandpass filter is incorporated into the detector so that only 40kHz signals are accepted. When the sensor detects infrared heat, it demodulates the signal and sets the output pin low. Internally, the sensor uses a Schmitt trigger to reduce switching noise on the output pin. This is highly desirable because false pulses could be mistaken as part of the incoming bit stream.

The output is connected to the microprocessor's low priority external interrupt, which was configured to a falling edge. In the interrupt subroutine, the width of the first pulse is calculated and authenticated as a valid start bit. The first pulse transmitted by the remote is longer than all subsequent pulses and is accordingly considered unique. If it is valid, the data is serially converted into bits by analyzing the length of the high pulses.

After seven samples are calculated, the data is compared against that in memory to determine which remote control key is pressed. To ensure an accurate reading, the entire process is repeated. For error checking, the results from both executions are compared.

2. PROXIMITY

It is desirable for the AGV to be capable of safely traversing a warehouse without colliding with obstacles in its path. Two forward facing Sharp GPD2D12 infrared range finders are placed approximately two inches apart. Both are pointed 30 degrees toward the center of the robot. Obstacles are detected when something passes into the sensor's line of sight.

Because the robot will be operating in a model warehouse, close range sensors where chosen to stay consistent with scale. The farthest distance the GPD2D12 can measure is 80 centimeters. With a body length of 13 centimeters, the AGV is considered to be a 1:14 scale. Applying the scale, a life size AGV could detect obstacles 35 feet in front of it.

3. LINE FOLLOWER

Navigation will be achieved by following black lines on the warehouse floor. A four-pair line-tracking module is constructed with Optek OPB745 Reflective Object Sensors. They are constructed with infrared light emitting diodes coupled with phototransistors. Because the reflective properties of black and white surfaces are different, the sensor will return varying analog values relative to the surface they are above. The microprocessor polls these analog values and converts them to digital data: black is 230_{16} and white is 135_{16}.

Two sensors are offset one half of a centimeter from the center of the module. This allows the robot to center itself on a two centimeter wide strip of electrical tape. Both of the other sensors are three centimeters from the middle. They serve to detect intersecting black lines. Combining two center sensors with an outside sensor allows the AGV to distinguish intersections from curves.

4. COLLISION

Two active low bump sensors are located on the back of the robot. They are wired in parallel, and the output is tied to a low priority interrupt. The interrupt is configured to a falling edge. In the event that the interrupt is fired, the AGV permanently stops.

5. RF LINK

The AGV is designed to work hand to hand with another vehicle on the warehouse floor. Communication is achieved with a Laipac TRF-2.4G RF transceiver. The development of this system was completed by Albert Chung, and it will be inserted into the AGV as a “plug and play” device.

Laipac merged several devices into one convenient package: a bidirectional transmitter, Cyclic Redundancy Check generator, and an antenna. The transceiver uses an external clock to serially input data from a microprocessor. Once the internal data buffer is full, the chip uses ShockBurst technology to assemble a packet: including an internally calculated preamble and CRC. Data is transmitted with a signaling rate as high as 1Mbps.

RESEARCH METHODS

A Sony television remote control (universal remote control code 202) generated an IR signal and was detected using a Fairchild Semiconductor QSE159. It was desirable to know the precise signal outputted from the infrared detector. Initial tests were conducted on the PIC18F8720 microprocessor, which operates on a 5MHz internal clock. Random data values were collected, and the researcher was unsure whether they were the result of the infrared transmitter, infrared detector, or the software running on the microprocessor. To remedy this solution, the detector was connected to an oscilloscope, and an algorithm was created that characterize the signal. The oscilloscope was also used to measure the lengths of start and data bits. These lengths were used in software, and the LCD screen displayed the pattern when a remote button was pressed. All tests were performed in a small room with the window blinds closed. The room was lit with an incandescent light bulb.

RESULTS

In its initial state, the output of the detector is high. Once a button on the remote control is pressed, the signal immediately goes low for a period of 0.25ms. The signal then returns high for a period of 1.75ms, and this is considered the start bit. Each data bit following this is 1ms. If the bit is a zero, the signal goes low for 0.25ms and then high for 0.75ms. If the bit is a one, the signal remains high for 1ms. High bits do not appear back to back. Once the data is finished transmitting, the signal goes high for a period much greater than 2ms. The remote control automatically repeats the signal while the user holds the button down.

Sony Television Remote Control Pulse Widths

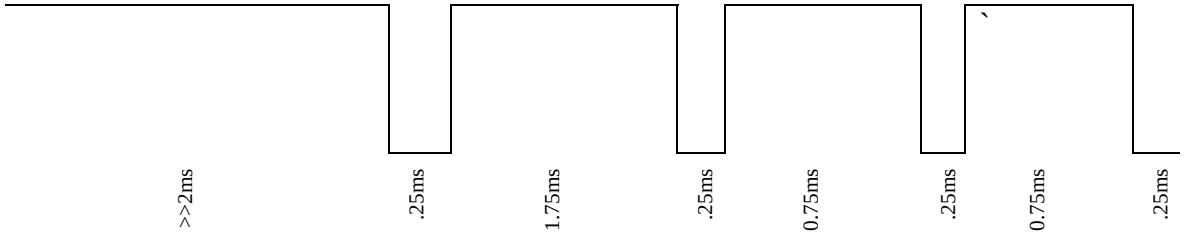

Figure 1.

Several steps in software are required to differentiate between buttons on the remote. Following the validation of the start bit, the length of subsequent high data periods are read in from an internal timer. The data bit is considered to be a one bit if the period is 2ms. Otherwise, the period is 1ms and the data bit is a zero.

Sony Television Remote Control Bit Pattern

Figure 2.

Ten data samples are required to achieve unique combinations.

Long Pulses in Data Sequence	
Button 1	1, 6
Button 2	1, 7
Button 3	1, 6, 7
Button 4	1, 8
Button 5	1, 6, 8
Button 6	1, 7, 8
Button 7	1, 6, 7, 8
Button 8	1, 9, 16
Button 9	1, 6, 9
Button 0	1, 7
Ch Up	1, 6, 10
Ch Down	1, 7, 10
Vol Up	1, 8, 10
Vol Down	1, 6, 8, 10
PWR	1, 7, 8, 10

Table 1.

Infrared Detector

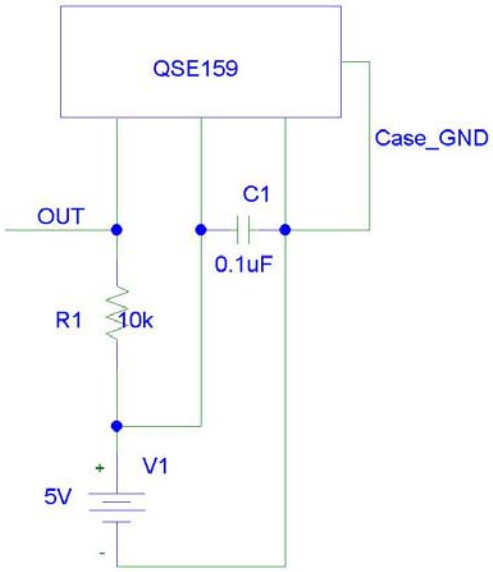

Figure 3.

REFERENCES

Fairchild Semiconductor Datasheet:

http://rocky.digikey.com/scripts/ProductInfo.dll?Site=US&V=46&M=QSE159

Laipac TRF-2.4G Datasheet:

http://www.sparkfun.com/datasheets/RF/RF-24G_datasheet.pdf

Nordic Semiconductor nRF2401 Datasheet:

http://www.sparkfun.com/datasheets/RF/nRF2401rev1_1.pdf

William Dubel's Reliable Line Tracking Report:

http://www.mil.ufl.edu/imdl/handouts/lt.doc

APENDIX A: CHOOSING A REMOTE CONTROL

Special consideration should be applied when choosing a remote control. Although any remote control will work, some produce better bit patterns than others. For example, different remotes handle start bits differently. More importantly, some produce a bit sequence that is not obviously unique. The output of a programmable remote should be viewed on an oscilloscope. The remote can simply be reprogrammed until a desirable pattern is produced.

There are desirable features in a bit pattern. First, the pulse width should be constant. With a constant pulse width, the signal can be looked at logically: true or false. Conversely, if the pulse width varies, the software will have to determine its value by calculating the length of the pulse and comparing it to memory. Assuming the microprocessor has a reasonably fast clock, the timer will overflow while waiting for the signal to change. Calculating the time and accounting for overflows adds a tremendous amount of overhead to an interrupt that needs to operate very quickly.

Another advantage to constant pulse widths is that sequences can easily be pulled off of an oscilloscope. Memorizing the sequence on a microprocessor should be avoided. Remote control orientation and ambient surroundings affect the IR detector readings. If a signal is being compared against a memorized one, it is necessary to account for error. For example, a signal within plus or minus ten percent of the memorized one is accepted.

Many remotes do not require a lot samples to obtain a unique pattern. This is good because it saves space in memory. However, the IR will continue to send the rest of the bit pattern. A remote with a blatant start bit should be chosen. Thus, the software can check the incoming signal to see if it is the start of a new sequence.

APENDIX B: VENDOR INFORMATION

Description:	Bump Switch
Supplier:	IMDL Lab
Price:	Free

Description:	Fairchild Semiconductor Inverted Photosensor QSE159
Part #:	QSE159-ND
Supplier:	Digi-Key
Website:	www.digikey.com
Price:	$0.75

Description:	Laipac TRF-2.4G RF Wireless Transceivers
Part #:	RF-24G
Supplier:	Spark Fun Electronics
Website:	www.sparkfun.com
Price:	$19.95

Description:	Optek OPB745 IR Emitter/Detector Pairs
Part #:	828-OPB745
Supplier:	Mouser Electronics
Website:	www.mouser.com
Price:	$3.60

Description:	Sharp GPD2D12
Supplier:	Mark III
Website:	www.junun.org
Price:	$8.25

APENDIX C: SOURCE CODE

```
/*****************************************************
 * AUTOMATED GUIDED VEHICLE
        *
 *
        *
 * EEL5666 Intelligent Machines Design Laboratory        *
 * University of Florida                                                *
 * Written for the PIC18F8720 @ 20 MHz                        *
 * Copyright (2005) Trevor Skipp 2.12.2005                *
 *****************************************************/

#include <p18f8720.h>
#include <delays.h>
#include <stdio.h>
#include "adconv.h"
#include "bios.h"
#include "motors.h"
#include "rf.h"
#include "interrupts.h"

#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = ON
#pragma config MODE = MC

void main(void)
{
        int before = 0;
        char ch = 'A';

        DDRD = 0x00;
        DDRB = 0xFF;

        LCD_Init();

        T1CON = 0b00000001;

        Delay1KTCYx(0);

        Init_IR_IRQ();
        Init_Global_IRQ();

        // Update LCD screen
```

```
	for(;;)
	{
		if(before != display)
		{
			if( display < 10 )
			{
				sprintf(message, "Button %d", display);
			}
			else if( display == 10 )
			{
				sprintf(message, "Ch Up");
			}
			else if( display == 11 )
			{
				sprintf(message, "Ch Down");
			}
			else if( display == 12 )
			{
				sprintf(message, "Volume up");
			}
			else if( display == 13 )
			{
				sprintf(message, "Volume down");
			}
			else if( display == 14 )
			{
				sprintf(message, "Power");
			}
			else if( display == 15 )
			{
				sprintf(message, "waiting");
			}

			Update_LCD();
			before = display;
		}

	}

	return;
}
```

```
/*********************************************************
 * Interrupt Polling                                                    *
 *                                                                      *
 * Written for the PIC18F8720                                           *
 * Interfaces:                                                          *
 * High Priority interrupts: INT0 -> NJM2670 Alarm                      *
 *                                                                      *
 * Low Priority interrupts:      INT1 -> TRF-24G RD1                    *
 *                                       INT2 -> Rear Bump switches     *
 *                                                                      *
 *                                                      (falling edge)  *
 *                                                                      *
 * Credits: Microchip                                                   *
 * Modified by Trevor Skipp & Albert Chung                              *
 *********************************************************/

#include <p18f8720.h>
#include <delays.h>
#include "rf.h"
#include "arbitor.h"

void low_isr(void);
void high_isr(void);
void Init_Int(void);

#define samples 11
#define BITTIME 20

unsigned char current[samples];
unsigned char count = 0;
int display = 15;

/*************************************************************
* For PIC18 devices the low interrupt vector is found at      *
* 00000018h. The following code will branch to the            *
* low_interrupt_service_routine function to handle            *
* interrupts that occur at the low vector.                    *
*************************************************************/

#pragma code low_vector=0x18
void interrupt_at_low_vector(void)
```

```
{
_asm GOTO low_isr _endasm
}
#pragma code /* return to the default code section */

#pragma interruptlow low_isr
void low_isr (void)
{
	if ( INTCON3bits.INT2IF == 1)			// Infrared triggered RB2 pin
	{
		// Grab the length of the high pulses

		for(; count < samples; count++)
		{
			TMR1H = 0x00;
			TMR1L = 0x00;

			while (PORTBbits.RB2 == 0)		//wait for rising edge
			{}

			while (PORTBbits.RB2 == 1)		//wait for falling edge
			{}

			if( TMR1H > BITTIME)
			{
				current[count] = 1;
			}
			else
			{
				current[count] = 0;
			}
		}

		// Compare sample with buttons in memory

		if(current[5] == 1 )
		{
			if(current[6] == 1 && current[7] == 1)
			{
				display = 7;
			}
			else if(current[9] == 1 && current[7] == 1)
			{
				display = 13;
			}
			else if(current[6] == 1)
```

```
		{
			display = 3;
		}
		else if(current[7] == 1)
		{
			display = 5;
		}
		else if(current[8] == 1)
		{
			display = 9;
		}
		else if(current[9])
		{
			display = 10;
		}
		else
		{
			display = 1;
		}
	}
	else if(current[6] == 1)
	{
		if(current[7] == 1 && current[9] == 1)
		{
			display = 14;
		}
		else if(current[7] == 1)
		{
			display = 6;
		}
		else if(current[8] == 1)
		{
			display = 0;
		}
		else if(current[9] == 1)
		{
			display = 11;
		}
		else
		{
			display = 2;
		}
	}
	else if(current[7] == 1)
	{
		if(current[9] == 1)
```

```
                {
                        display = 12;
                }
                else
                {
                        display = 4;
                }
        }
        else if(current[8] == 1)
        {
                display = 8;
        }

        Delay1KTCYx(0);
        count = 0;
        INTCON3bits.INT2IF = 0;                         // Clear INT2 flag
    }

    if ( INTCON3bits.INT1IF == 1)       // RF Rx data ready
    {
        Receive();
        INTCON3bits.INT1IF = 0;             // Clear the INT1 Flag
    }
}

/*****************************************************************
* For PIC18 devices the high interrupt vector is found at      *
* 00000008h. The following code will branch to the                *
* high_interrupt_service_routine function to handle               *
* interrupts that occur at the high vector.                           *
*****************************************************************/

#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
_asm GOTO high_isr _endasm
}
#pragma code /* return to the default code section */

#pragma interrupt high_isr
void high_isr (void)
{
    if ( INTCONbits.INT0IF == 1)
    {
        //MOTOR_ALARM = 1;
        INTCONbits.INT0IF = 0;              // Clear the INT0 Flag
```

```
                return;
        }
}


/*************************************************************
* Initialize interrupts
        *
*************************************************************/



void Init_Global_IRQ(void)
{
        RCONbits.IPEN = 1;              // Enable interrupt priority
        INTCONbits.GIEH = 1;            // Enable all high priority interrupts
        INTCONbits.GIEL     = 1;        // Enable all low priority interrupts
}


void Init_IR_IRQ(void)
{
        INTCON2bits.INTEDG2 = 0;        // INT2 = falling edge interrupt
        INTCON3bits.INT2IP = 0;         // INT2 = low priority interrupt
        INTCON3bits.INT2IF = 0;         // Clear the INT2 Flag
        INTCON3bits.INT2IE = 1;         // Enable INT2 XIRQ
}
```