AUTOMATED GUIDED VEHICLE:
FINAL REPORT

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

Prepared by: Trevor Skipp, Student

January 25, 2005

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

Contents

Abstract

"Automated Guided Vehicle"

By Trevor Skipp

High demands on manufacturers have left their shipping warehouses in havoc. Human error has a negative effect on safety, efficiency, and quality. These expenses are reduced with the introduction of an Automated Guided Vehicle, AGV. A driverless, intelligent forklift uses an optical path to quickly and safely traverse a warehouse. Its capabilities are enhanced by the ability to send and receive tasks through RF data communication.

Executive Summary

Introducing worker robots into warehouses has the potential to increase the standards currently held by the industry. The autonomous machines have the capability of dynamically storing pallets, lowering labor and insurance rates, and reducing the risk of personal injury. To broach these goals, the Automated Guided Vehicle, AGV, was designed to collaborate with another autonomous vehicle, the Automated Storage and Retrieval System, ASRS.

Although they appear similar, the AGV and ASRS serve two separate purposes. Catered to industries whose products can expire, the ASRS keeps track of when each pallet enters the warehouse and its current location. This ensures that the oldest product gets shipped out first. The ASRS is a tall vehicle designed to navigate in the tight aisles of a warehouse floor. Although its tall mast gives it the capability of lifting pallets onto a third tier shelf, it restricts the vehicle travels to slow speeds to maintain a safe environment. With all considerations in mind, the ASRS can not efficiently traverse long distances in a warehouse. The problem is easily solved with the introduction of a small, light vehicle. The AGV can quickly travel across a warehouse while carrying a pallet. Its simple pallet jack can lift up and set down pallets at one height. Because of its small body and simple forklift mechanism, the AGV is much less expensive than the ASRS.

The AGV receives tasks from the ASRS, which can vary from picking up a pallet at the incoming dock to delivering one a pallet to the outgoing dock. Accurate navigation is obtained by mapping high contrast lines on the warehouse floor in software. While navigating and backing up, the vehicle senses its surrounding to predict an imminent collision.

Introduction

Advancements in manufacturing technology allow companies to rapidly produce products. This has provoked a trend to reduce bulk inventory in favor of short term supplies. Although this allows corporations more financial freedom, it requires warehouses to accommodate temporary, selective storage. Improved product handling and speed can be achieved with the implementation of an Automated Guided Vehicle, AGV.

In a traditional warehouse, human safety governs the productivity. With the help of intelligent computers, the AGV can safely achieve higher speeds. Precision turning allows it to accurately navigate in tight spaces.

The AGV is highly flexible as a result of remote communication. Its ability to communicate with other autonomous vehicles provides a seamless operation. Continuous coordination between vehicles delivers money saving efficiency.

The introduction of unmanned vehicles onto a warehouse floor has favorable effects on safety. With the aid of environmental sensors, the AGV can detect objects in its collision path. Automation eliminates vehicle traffic jams and their potential for accidents.

For companies building new warehouses, there are many monetary benefits to investing in intelligent machinery. The workforce required to run the warehouse and the additional overhead (e.g., insurance) required to support that overhead will be drastically reduced. Increased product turn-around and faster shipping will result in more satisfied customers. Also, automation reduces the risks of personal injury.

Integrated System

The AGV operated in a model warehouse, built to scale. Its primary task was to relocate pallets within the warehouse. A external input generated by an infrared remote control notified the AGV whether a pallet was entering or exiting the warehouse. To get to its destination, the vehicle traversed the warehouse by following high contrast lines. When the four pair line follower module detected an intersection, the AGV determined whether to turn or go straight by using an algorithm that incorporated the vehicles current location and direction.

In the first situation, the remote control signaled that a pallet was entering the warehouse. The AGV picked up the pallet off of the incoming shipping dock and dropped it off at one of several docks at the other end of the warehouse. Through RF communication, the AGV told another autonomous vehicle, the ASRS, the new location of the pallet. The second situation allowed for a pallet to be shipped out of the warehouse. The AGV waited for the ASRS to confirm that it dropped off a pallet at one of the transistion docks before it picked it up and moved the pallet to the outgoing shipping dock.

While traveling, the AGV polled two forward facing infrared range finding sensors to determine if an object was in its forward collision path. If and object was detected, the vehicle would stop and wait for the obstruction to be cleared. While reversing, bump sensors detected the occurrence of a rear collision, which would permanently disable the vehicle.

Mobile Platform

The designer intended the AGV to be a small vehicle that could quickly traverse a warehouse and move pallets around. Creating the smallest vehicle possible required creative thinking and experimentation. A concept drawing was entered into AutoCAD 2005 and rendered in three dimensions. After the blue print was verified, a T-Tech prototyping machine cut the platform out of eighth inch plywood. The top of the vehicle was constructed from thin balsa wood, which was glued to a wood skeleton that matched the curve of the AGV. Although the fork protruded through a rectangular cutout, the lid could still be easily taken on and off. A second cutout was made for the infrared distance sensors, and the display lens off of a V.C.R. was modified to conceal the sensors. The platform was primed and painted with black and silver lacquer spray paint, and a high gloss was created by applying polyurethane.

The body was five and a half inches wide, five and a half inches long, and six inches tall. A rectangle was cut out of the back and bottom to provide for the housing of the LCD and line follower module, respectively. Arches were cut out to provide clearance for two 2.2 inch diameter tires, which were mounted on the rear end of the robot, while two casters supported the front of the robot. Three tier shelving was used to optimize space inside of the AGV. The center of gravity was intended to be above and forward of the motors. However, an error on the chasis design was quickly discovered once it was assembled. A third caster was added to the extreme rear of the vehicle so the robot wouldn't tip back while driving forward. The height of the new caster was precisely measured so the vehicle would rock as little as possible.

Actuation

The AGV required the capability to move around a model warehouse built out of a sheet of plywood and to pick up pallets once it reached it's destination. Two types of actuation where needed to meet these objectives. Drive motors and tires added the function of movement, while a third motor supported a pallet jack.

1. Wheels and Movement

The AGV was propelled by two 200 R.P.M. D.C. gear head motors, which where attached to the rear tires. The tires were 2.2 inches in diameter and one inch wide. With the addition of the third caster, the majority of the vehicle's weight rested on the rear caster, and the tires were slipping. Thick coats of rubber cement were painted onto the tires and the vehicle regained traction. The AGV had an excellent turning radius as a result of the platform layout. The tires resided one inch from the middle of the robot, which almost allowed it to turn in place.

Forklift Mechanism

The forklift was created out of wood and mounted to the platform with hinges. Serendipity struck when the hinges could not be installed perfectly straight. The hinges had extra friction, and the fork did not flop around even though they were still moveable. An arm was rigidly attached to the forks and mounted so that it was parallel to the forks and in the opposite direction. A slot in the arm was created with a T-Tech prototyping machine and connected to a servo arm with a pin. When the servo was in its neutral position (zero degrees), the forks were down and parallel to the ground. Altering the servo's pulse width moved the servo arm back to negative thirty degrees and the forks up to positive thirty degrees. Wooden stops where mounted on the fork to prevent pallets from sliding back and crashing into the robot.

Sensors

1. Infrared Detector

The goal of this project was to develop a robot that streamlines the warehousing process. An infrared remote control allowed the user to dynamically communicate with the vehicle on the factory floor. Buttons on the remote control corresponded to requests to store or retrieve a pallet. A Sony television remote control (Sony code # 202) was used to send infrared data. Each button on the remote had a unique bit pattern. When a button was pressed, the remote formed a packet of data including start bits, data elements, and stop bits. Digital Signal Encoding was used, and the packet was sent serially through a 40kHz modulator. Modulating the signal decreased the probability of ambient infrared crosstalk. After modulation, the signal was sent to the infrared generator.

The AGV used a Fairchild Semiconductor infrared detector to receive the signal (Figure 1). A bandpass filter was incorporated into the detector so that only 40kHz signals were accepted. When the sensor detects infrared heat, it demodulates the signal and sets the output pin low. Internally, the sensor used a Schmitt trigger to reduce switching noise on the output pin. This was highly desirable because false pulses could be mistaken as part of the incoming bit stream. The output was connected to the microprocessor's low priority external interrupt, which was configured to a falling edge. In the interrupt subroutine the data was serially converted into bits by analyzing the length of the high pulses. The signal was first analyzed on an oscilloscope. The researcher noticed that all of the pulses (excluding the start pulse) had one of two qualities. There was always a 0.25ms high or low pulse which was fallowed by a 0.75ms high period (Figure 2.). This made it possible to have either a 1ms high period or a 0.75ms high period. The back end of one 0.75ms pulse plus

the 1ms pulse provided a total pulse width of 1.75ms. This long pulse width never appeared back to back.

Figure 1. IR Schematic

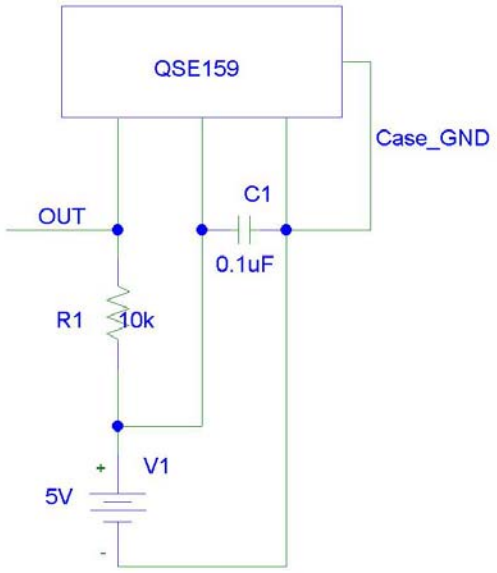
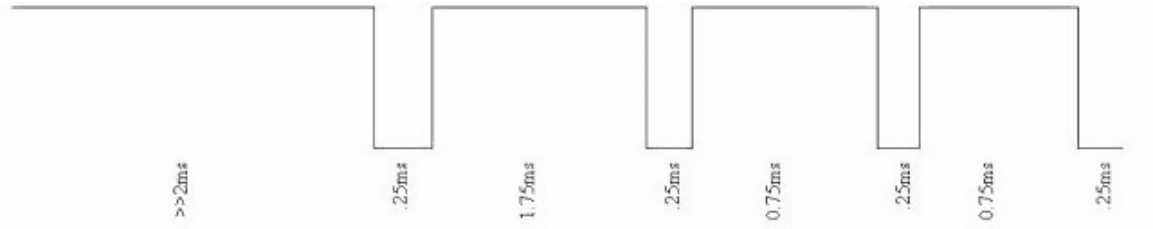


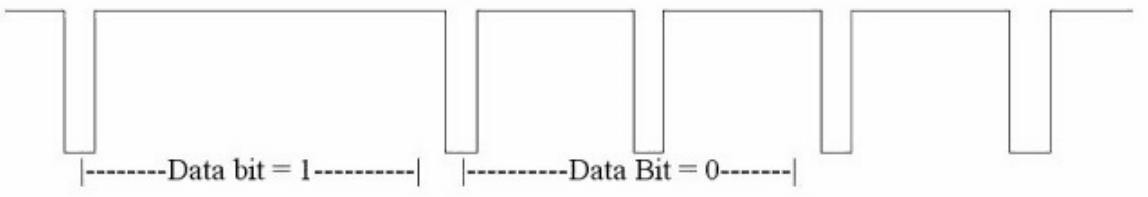
Figure 2. Sony Code #202 Pulse Widths



In all of the bit patterns, there were several more short pulses that long pulses. The researcher guessed that the data was riding on the long pulses, so he decided to call the short pulses '0' and the long pulses '1' (Figure 3.). The bit patterns were collected for the entire signal. A pattern was discovered upon analyzing the data. After the first five samples were ignored, the subsequent five samples where stored into an array. All other samples where ignored. The array was reversed so that first sample collected became the least significant bit in binary form. The array was converted into a hexadecimal number. With this procedure,

button "one" on the remote gave the hexadecimal number of 1, and button "two" on the remote gives the hexadecimal number 2. This was the Sony code!

Figure 3. Sony Code #202 Bit Definition



2. Proximity

It was 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 were placed approximately two inches apart. Both where pointed 30 degrees toward the center of the robot. Obstacles were detected when something passed into the sensor's line of sight. The sensors were preassembled and powered up with 3.3 volts. The analog output was connected to the microprocessor.

Because the robot was 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 (Figures 4 and 5). 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.

Figure 4. Line Follower Schematic

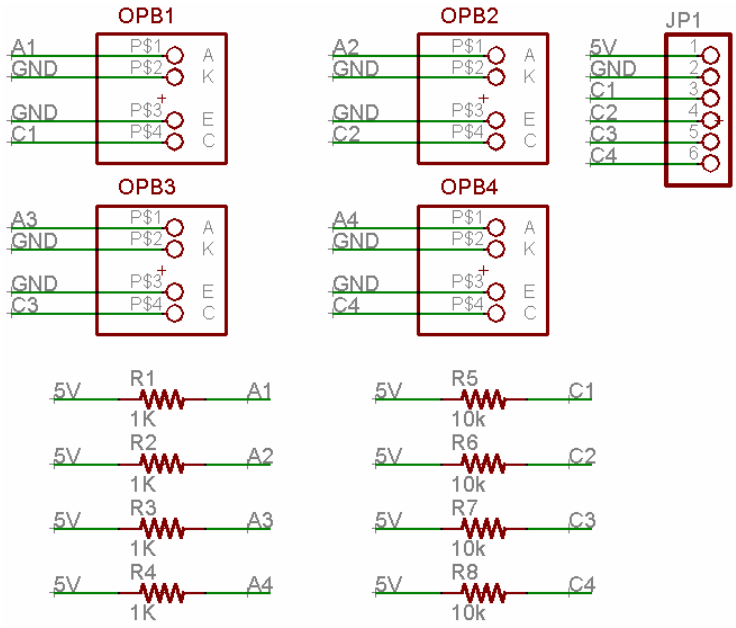
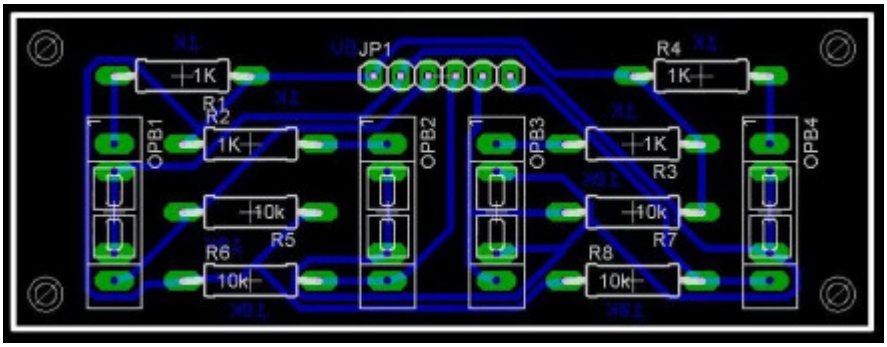


Figure 5. Line Follower Module



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. A complete Special Sensor Report on the RF data link can be found at http://plaza.ufl.edu/tskipp/agv_asrs/RF.htm .

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.

To address the possibilities of errors, the Stop and Wait Automatic Request protocol was used. This had several advantages over direct communication, including an alternating frame number and positive acknowledgment. If frames were received out of order, the receiver would NAK the sender and wait for the correct frame. However, things were not perfect and the two robots went quickly out of synchronization. To account for this, software allowed for the dynamic resynchronization of frame numbers. The biggest source of error was ambient noise that crosstalked to our system. We implemented header error control to help counter this. By inserting a standard header in the unused bits, the receiver could test the incoming message. Another possibility of error comes from both devices

transmitting at the same time. Both robots were programmed with separate timeout lengths. Thus, if one robot sent something and never received and ACK, it would resend its packet before the other would.

Behaviors

The AGV uses several behaviors to complete its objectives. These behaviors where programmed in separate modules, and an arbitrating function coordinated the events. Priority interrupts were used to address the precise timing requirements of some devices.

1. Communication

The AGV originally utilized two forms of communications in the forms of an RF transceiver and an infrared remote control. The ASRS required more data because it needed to keep track of the location of pallets, different shelf heights, and pallet ages. With the remote control on the AGV, the AGV simply passed all of the information to the ASRS and then discarded everything it did not need. This created a lot of overhead, so the remote control was moved to the ASRS, which in turn passed the AGV the little information it needed.

The ASRS communicated to the AGV under one of two circumstances. If a pallet was being shipped into the warehouse, the ASRS would immediately tell the AGV that a pallet was incoming and to what dock the AGV needed to deliver it to. Once the AGV picked up the pallet and dropped it off at the correct dock, it was transmit the exact packet back to the ASRS. This would notify the ASRS that the pallet has arrived on the bottom shelf and needs to be moved to a higher shelf, which would clear out the dock for more incoming pallets.

The second scenario was the outgoing pallet routine. The ASRS moved a pallet to the bottom shelf and told the AGV two pieces of information: the dock location and that the pallet was outgoing. Immediately after the AGV picked up the pallet, it notified the ASRS that it completed the task. The ASRS was now free to reuse the dock.

2. Queue

The AGV was much faster than the ASRS, so the delivery and storage of pallets was not one to one. The implementation of a queue insured that the neither of the robots would ever be sitting idly by. Upon the successful reception of an RF packet, the AGV pushed it onto the queue. When there were no jobs being processed, the robot continually polled the queue for a new job. The jobs where handled in a "First In First Out" order.

3. Tracking

The four pair line tracking module was used to navigate the AGV on a dark brown, glossy floor with white strips of electrical tape. Three motors speeds were defined: medium fast, medium, and slow. Normal navigation was done with the medium fast speed. If the vehicle started to stray off of the line, the software would notice a difference in the values from one of the two center sensors and decrease the appropriate motor's speed to medium. If the vehicle completely left the line, the robot turns in the opposite direction of the last sensor read. For example, if the robot last saw "white" on the right-center sensor, it would turn left. The software detected an intersection when the output from either of the outer line following sensors read "white."

The detection of intersections allowed for mapping system on a Cartesian coordinate system. The warehouse was laid out so that the vertical segments of line had Y values of negative one, zero, and one (where negative one was closest to the shelves). The X segments had values ranging from zero to four, which were coordinated with the dock numbers. There was an obvious need for direction when the robot was turning; the robot needed to turn right if it were traveling one away and left if it were traveling another. A cardinal direction system was implemented: zero represented East, one represented North,

two represented West, and three represented South. Each time the vehicle turned, the software would adjust the direction.

4. Lifting

For all jobs, the AGV first picked up a pallet and then dropped it off at a new location. Two white lines were placed in front of each dock and allowed the arbitrating function to make different actions. At the first line, the AGV would lower its fork before driving in. It would raise its fork at the second line before turning around and heading toward its destination. Similar procedures were followed when the AGV went to drop the pallet off.

5. Obstacle Detection

Although there was no need for humans in the automated warehouse, people could be unpredictable (unlike robots). Distance sensors determined if an object was in the forward path of the vehicle. If they detected something, the robot would pause before checking to see if the obstruction was still present. For obvious reasons, the sensors were temporarily disabled when the AGV was approaching the shelves. While backing up, the rear bump sensors were activated so a rear collision could be detected. If this event occurred, the vehicle would be permanently disabled.

Experimental Layout and Results

Experiments were performed on all of the sensors individually before they were compiled into one integrated system. Tests on the infrared distance sensors yielded analog values that corresponded to their distance from objects. The value was read for six inches and hard coded into the program. The sensor outputs from the infrared line following module varied from each other; typical readings from left to right were 165, 145, 99, and 148. The values were highly conditional of ambient infrared sunlight, so the module was recalibrated to white every time the robot powered up.

Experiments were performed on a Sony television remote control (code number 202) which 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 a fluorescent light bulb.

Conclusion

All deliverables set forth in the project proposal were successfully met. The designer initially intended the AGV to be a super fast vehicle. Even though it could follow lines at high speeds, it occasionally missed intersections. Although the AGV had to be slowed down, it was still much faster than the ASRS, which met specifications. The forklift mechanism was not built when the platform was designed and assembled, and the original concept for the forklift failed. The limited amount of space hindered subsequent ideas for the forklift, and several different designs were prototyped before the final version was built. However, the forklift turned out very well and the wait was well worth it.

The biggest area for improvement was in the warehouse. Although it took forty hours to build and cost seventy dollars, there were inherent flaws. First, the ground was not perfectly smooth. Paneling was used on the AGV side while plywood was used on the ASRS side. The plywood surface would not sand down, and was quite bumpy. This jostled the ASRS while it was driving and gave inaccurate depictions to the line follower sensors. Another problem with the warehouse was the warping shelves. The ASRS had a maximum fork height of thirteen inches, so there was not much room left for pallets. Thick wooden shelves would have greatly reduced the clearance for each shelf, so half inch plywood was used instead. Although this problem was overcome in software, metal shelves would have simplified things.

The AGV design project went fairly smooth. The most frustrating problems stemmed from Microchip's MPLAB compiler. The project designer was highly skilled in the C language and had to make adjustments to his coding techniques to adjust for inadequacies in the compiler. Although none of the problems were detrimental or prevented something from eventually working, many weekends were wasted on "something stupid." I would

recommend future students who enroll in the Intelligent Machine Design Laboratory to talk to someone who has used their compiler. They might be able to save people an ample amount of time by passing words of wisdom.

There are many areas for improvement on this project. A Sliding Window Automatic Request protocol could be implemented to further reduce RF transmission errors. Both robots could continually send a null packet of data to each other, which would allow the robots to know if they went out of range. More importantly, it would allow them to see how many errors they were receiving, and they could dynamically adjust for it. A larger warehouse with more shelves would allow the demonstration to be more meaningful. A swarm approach could be implemented by building several AGVs for every ASRS. Conveyor belts could lead in and out of the warehouse and notify the vehicles when a part is present. The easiest improvement would be to add an RF link to a personal computer and replace the remote control with graphical software.

Documentation

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

Appendix 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.

Appendix B: Photographs

Figure 6.Automated Guided Vehicle

Figure 7. Automated Guided Vehicle Front

Figure 8. Warehouse Layout (AGV Side)

Figure 9. Warehouse Layout (ASRS Side)

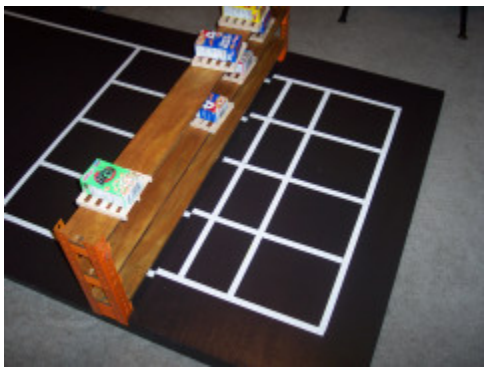


Figure 10. Pallet

Figure 11. AGV Bottom Tier

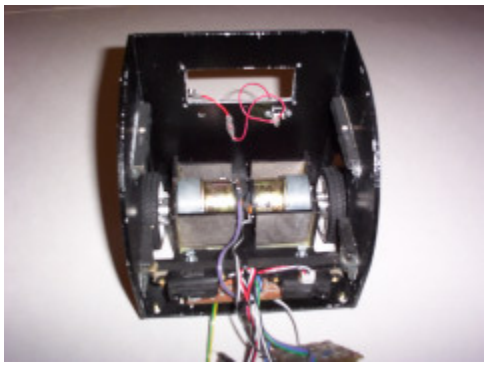


Figure 12. AGV Middle Tier

Figure 13. AGV Top Tier

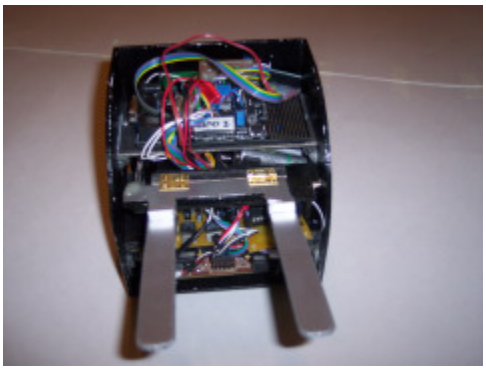


Figure 14. AGV Lid

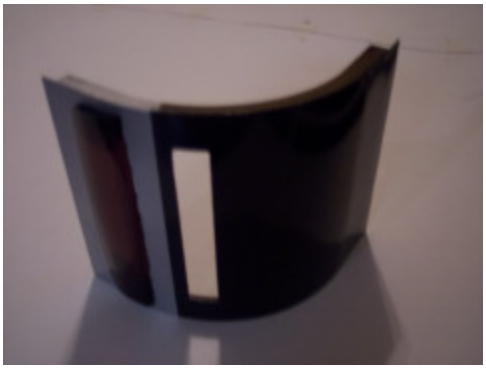


Figure 15. Power Board Footprint

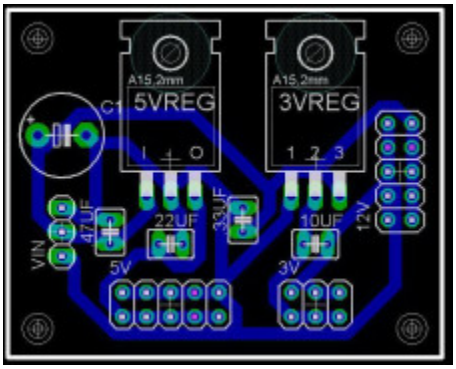


Figure 16. Line Follower Module Footprint

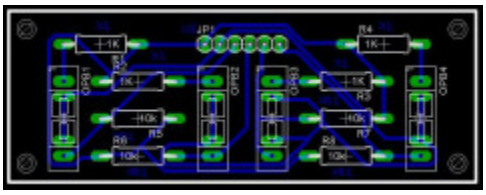


Figure 17. Motor Driver Footprint

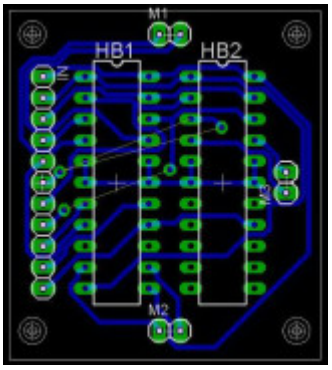


Figure 18. LCD Footprint

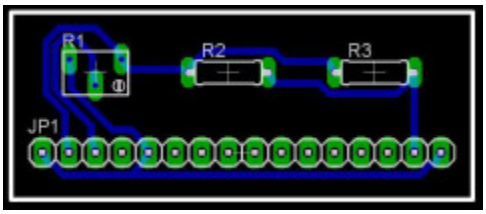
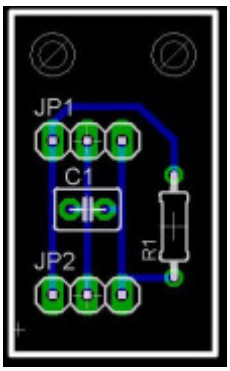


Figure 19. IR Footprint



Appendix C: 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

Appendix D: Source Code Commentary

- Poll_AD_Done: Poll until the conversion is complete.
- Read_ADx: Select "x" analog channel and sample it. Returns the 8 bit value of the analog channel.
- Move_Pallet: Runs through the steps necessary move a pallet across the warehouse floor. Acts as an arbitrator for the motors and servo.
- Queue_Push: Push a request onto the queue.
- Queue_Pull: Pull a request off of the queue.
- Update_LCD: Sends a string one character at a time to LCD_Write.
- LCD_En: Enables the LCD. This tells the LCD to read in data.
- LCD_Init: Initializes the LCD for 4 bit mode, 2 lines, 5x11 dot matrix, display on, cursor off, blink off, clear screen, return cursor home, increment cursor to the right, and don’t shift the screen.
- LCD_Command: Modify data to set RS (Register Select) to command mode. Sends data to the LCD.
- LCD_Write: Reads in the argument and displays it on the LCD.
- PWM_Init: Initialize the PWM module for 4 DC motors using Timer 2 and 1 servo using Timer 1.
- Fork: Raise or lower the fork.
- Motors: Adjust the DC motor speed. Implement a smoothing function to improve driving.
- Calibrate_LF: IR sensors in line follower can be affected by ambient light. This function allows for the dynamic calibration of the line follower module.

- Navigate: Reads the IR line following sensors and converts the array into an integer.
- Turn_Left: Sets motors and delays processor so the vehicle can get off of the current line and begin polling for the next line.
- Turn_Right: Sets motors and delays processor so the can get off of the current line and begin polling for the next line.
- Turn_Around: Sets motors and delays processor so the vehicle can get off of the current line and begin polling for the next line. Do to the operating environment, special considerations where taken into the direction of the turn.
- Back_Up: Reverses the motors and delays for a set amount of time. No consideration for line tracking is needed. Polls rear bumper to determine if a collision has occurred.
- Navigate: Use value from line following sensors to determine what action to take. Stops robot if something is in its forward collision path.
- Map_Intersection: Uses current location and destination location to determine whether to go forward, left, right, or to turn around.
- Receive: Clock in data from the RF chip and error control such as parity to determine if data is valid.
- Transmit: Adds header error control and clocks out data to the RF chip.
- Init_RF: Configure the RF module.

Appendix E: Source Code

```
/*****************************************************
 * AUTOMATED GUIDED VEHICLE                          *
 *                                                   *
 *                                                   *
 * EEL5666 Intelligent Machines Design Laboratory    *
 * University of Florida                             *
 * Written for the PIC18F8720 @ 20 MHz               *
 * Copyright (2005) Trevor Skipp 4-15-05             *
 *****************************************************/

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

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

#define rear_bump PORTBbits.RB4

void main(void)
{
    unsigned char i, temp;

    DDRD = 0x00;
    DDRB = 0xFF;
    DDRH = 0x00;
   DDRC = DDRC & 0b11111011;
   DDRG = DDRG & 0b11100110;
    DDRE = DDRE | 0b01110100;
    DDREbits.RE3 = 0;

    PWM_Init();
    Motors(0,0);

    LCD_Init();
```

```
    PORTEbits.RE3 = 0;                          // Servo always low
    Init_Servo_Comp();

    Fork(UP);

    Init_RF();

    GLOBAL_IRQ_ON;

    sprintf(message, " ...AGV v.1.1...");
    UPDATE_LCD;

    Delay10KTCYx(0);

    Calibrate_LF();

    sprintf(message, "...Waiting For  Command...");
    UPDATE_LCD;

    while (1)
    {
        if (first != last)
        {
                Move_Pallet();
        }
    }

    return;
}
```

```
#include <p18f8720.h>
#include <delays.h>
#include "motors.h"
#include "nav.h"
#include "agv.h"
#include "bios.h"
#include <stdio.h>

// Shelf Location Variables
unsigned char IO;

// Queue variables
#define QUEUE_LENGTH 8
unsigned char Queue[QUEUE_LENGTH];
unsigned char first = 0;
unsigned char last = 0;

/*****************************************************
 * Move_Pallet: Runs through the steps necessary     *
 * move a pallet across the warehouse floor.  Acts   *
 * as an arbitrator for the motors and servo.        *
 *****************************************************/
void Move_Pallet(void)
{
    unsigned char packet, IO, temp;

    packet = Queue_Pull();

    if( (packet & 0b00001000) == 0b00001000 )          // Pallet entering the system
    {
sprintf(message, "Incoming pallet");
UPDATE_LCD;
        IO = 1;
        x_dest = 0;
        y_dest = -1;
    }
    else                                                // Pallet
    leaving the system
    {
sprintf(message, "Outgoing pallet");
UPDATE_LCD;
        IO = 0;
        x_dest = packet & 0b00000111;                   // Get the location of the first
    pallet to go
        y_dest = 1;
    }

    if ( direction == 0 )
```

```
    {
        Navigate();                         // Follow center line until destination lane is
    reached
    }
    else
    {
        Turn_Around();
        Navigate();                         // Follow center line until destination lane is
    reached
    }

    Fork(DOWN);

    sprintf(message, "Picking Pallet  Up ");
    UPDATE_LCD;

    Motors(85, 85);                         // Get off the line
    Delay10KTCYx(125);

    Navigate();

    Delay10KTCYx(50);

    Fork(UP);                           // Lift pallet above shelf

    sprintf(message, "Pallet Retrieved");
    UPDATE_LCD;

// Tell ASRS the pallet has been picked up
    if (IO == 0)                            // Pallet leaving the system
    {
        sprintf(message, "Picked Up Packet");
        UPDATE_LCD;

        Transmit(packet);               // send shelf number where the output resides to the
    AGV
    }

    Delay10KTCYx(50);

    Back_Up();                              // Drive out

    Fork(DISABLE);

// go to second set of (x,y) coordinates

    if( IO == 1 )                           // Pallet entering the system
    {                                               // Go to the shelves second
```

```
        x_dest = packet & 0b00000111;                   // Go to the docks
        y_dest = 1;
    }
    else                                            // Pallet leaving the system
    {                                                       // Go to dock second
        x_dest = 4;                                 // Go to output dock
        y_dest = -1;
    }

    Turn_Around();

    Navigate();                         // Follow center line until destination lane is reached

// if incoming, tell ASRS to pick up the pallet
    if (IO == 1)                                    // Pallet entering the system
    {
        sprintf(message, "Requesting ASRS Pickup");
        UPDATE_LCD;

        Transmit(packet);               // send shelf number where the output resides to the
    AGV
    }

// Raise pallet above shelf
    Fork(UP);

    Delay10KTCYx(50);

    Motors(85, 85);                                 // Get off the line
    Delay10KTCYx(125);

    Navigate();

    Delay10KTCYx(50);

// Lower pallet onto shelf
    Fork(DOWN);

    Delay10KTCYx(50);

    Back_Up();

    Fork(DISABLE);

    sprintf(message, "Mission Complete");
    UPDATE_LCD;

    Delay10KTCYx(0);
```

```
    sprintf(message, "...Waiting For  Command...");
    UPDATE_LCD;

    return;
}

/*****************************************************
 * Queue_Push: Push a request onto the queue.                    *
 *****************************************************/
void Queue_Push(unsigned char packet)
{
    Queue[last] = packet;

    if (last == QUEUE_LENGTH - 1)
    {
        last = 0;
    }
    else
    {
        last++;
    }
}

/*****************************************************
 * Queue_Pull: Pull a request off of the queue.            *
 *****************************************************/
unsigned char Queue_Pull(void)
{
    unsigned char temp;

    temp = Queue[first];

    if (first == QUEUE_LENGTH - 1)
    {
        first = 0;
    }
    else
    {
        first++;
    }

    return temp;
}
```

```
/*****************************************************
 * HITACHI LCD CONTROLLER                                          *
 *                                                                 *
 * Written for the PIC18F8720 @ 20 MHz                             *
 * Interfaces: Enable on RD0, RS on RD1, DB4:7 on  *
 *   RD4:7                                                         *
 * Copyright 2005 Trevor Skipp & Albert Chung                      *
 *****************************************************/

#include <p18f8720.h>
#include <delays.h>
#include "bios.h"

#define lcdport PORTD
#define EN PORTDbits.RD0
#define RS PORTDbits.RD1

void LCD_Init(void);
void LCD_En(void);
void LCD_Write(char data);
void LCD_Command(int);
void Update_LCD(void);

char message[32];

/*****************************************************
 * Update_LCD: Sends a string one character at a       *
 *   time to LCD_Write.                                            *
 *****************************************************/
void Update_LCD(void)
{
    int i, j;

    LCD_Command(0x01);                          // Clear display & return
    cursor home
    Delay10KTCYx(5);                            // 5 msec

    for(i = 0; i < 32; i++)
    {
        if ( message[i] == '\0' )
        {
                break;
        }
                if (i == 16)
                {
                        for(j = 0; j < 24; j++)
```

```
                        {
                                LCD_Write(' ');
                        }

                        LCD_Write(message[i]);
                }

                else
                {
                        LCD_Write(message[i]);
                }
        }
}

/*****************************************************
 * LCD_En: Enables the LCD.  This tells the LCD to *
 *   read in data.                                                    *
 *****************************************************/
void LCD_En(void)
{
    Delay10TCYx(100);                                   // 50 usec
    EN = 1;                                                     // PORTDbits.RD0 =
    1
    Delay10TCYx(100);                                   // 50 usec
    EN = 0;                                                     // PORTDbits.RD0 =
    0
    Delay10TCYx(100);                                   // 50 usec

}

/*****************************************************
 * LCD_Init: Initializes the LCD for 4 bit mode,        *
 *   2 lines, 5x11 dot matrix, display on, cursor*
 *   off, blink off, clear screen, return cursor    *
 *   home, increment cursor to the right, and don't      *
 *   shift the screen.                                                *
 *****************************************************/
void LCD_Init(void)
{
    int i;
    int setup[] = {0x33, 0x32, 0x2C, 0x0C, 0x01, 0x06};

    Delay10KTCYx(40);                                   // 20 msec power up

    for (i = 0; i < 6; i++)
    {
        LCD_Command(setup[i]);
        Delay10KTCYx(10);                               // 5 msec
```

```
    }
}

/*****************************************************
 * LCD_Command: Modify data to set RS (Register              *
 *  Select) to command mode.  Sends data to the LCD.*
 *****************************************************/
void LCD_Command(int command)
{
    int temp;

    RS = 0;                                                    // PORTDbits.RD1 =
    0

    temp = command & 0xF0;                          // Mask off lower nibble
    lcdport &= 0x0F;                                // Clear upper nibble
    lcdport |= temp;
    LCD_En();

    Delay10TCYx(110);                                    // 55 usec

    temp = command << 4;
    temp &= 0xF0;                                        // Clear upper nibble
    lcdport &= 0x0F;
    lcdport |= temp;
    LCD_En();
}

/*****************************************************
 * LCD_Write: Reads in the argument and displays it  *
 *   on the LCD.                                                    *
 *****************************************************/
void LCD_Write(char data)
{
    int temp;

    RS = 1;                                                    // PORTDbits.RD1 =
    1

    temp = data & 0xF0;                                  // Mask off lower nibble
    lcdport &= 0x0F;                                // Clear upper nibble
    lcdport |= temp;
    LCD_En();

    Delay10TCYx(110);                                    // 55 usec

    temp = data << 4;
    temp &= 0xF0;
```

```
    lcdport &= 0x0F;                        // Clear upper nibble
    lcdport |= temp;
    LCD_En();

}
```

```
/*****************************************************
* ANALOG TO DIGITAL CONVERSION                       *
*                                                    *
* Written for the PIC18F8720 @ 20 MHz                *
* Interfaces: Analog inputs on ANO:5                 *
* Copyright 2005 Trevor Skipp                        *
*                                                    *
* NOTES:                                             *
* The followig code is for 8 analog channels, and    *
*   the remaining pins are set to digital I/O.  If   *
*   more analog channels are desired, adjust         *
*   ADCON1 in each function, and follow the          *
*   pattern in the functions.                        *
* A delay is required before a subsequent sample.    *
*   This is the delay before the return statement    *
*   in each function.  This delay could be moved     *
*   or eliminated (i.e. you will not be taking       *
*   samples back to back) to free up processor       *
*   cycles.                                          *
*****************************************************/

#include <p18f8720.h>
#include <delays.h>
#include "adconv.h"

int Read_AD0();
int Read_AD1();
int Read_AD2();
int Read_AD3();
int Read_AD4();
int Read_AD5();
void Poll_AD_Done(void);

/*****************************************************
* Poll_AD_Done: Poll until the conversion is         *
*   complete.                                        *
*****************************************************/
void Poll_AD_Done(void)
{
    for(;;)
    {
        if (ADCON0 & 0b00000001 == 0b00000001)
        {
                return;
        }
```

```
    }
}

/*****************************************************
 * Read_ADx: Select "x" analog channel and sample    *
 *   it.  Returns the 8 bit value of the analog      *
 *   channel.                                                                    *
 *****************************************************/
int Read_AD0(void)
{
    ADCON1 = 0b00000111;        //Vref+ = External Vref + (5V to pin 27), Vref- =
    Vss, ANO:5 analog in, AN6:15 digital I/O (PAGE 214)

    ADCON0 = 0b00000000;        //select ADO, set "GO", and ADON off (PAGE
    213)

    ADCON2 = 0b00000010;        //left justify, conversion time = 64 * Tosc (PAGE
    215)
    ADCON0 = 0b00000001;        //turn on A/D module

    Delay10TCYx(13);            //6.5us

    ADCON0 = 0b00000011;        //GO

    Poll_AD_Done();

    Delay10TCYx(35);            //23us

    return ADRESH;
}

int Read_AD1(void)
{
    ADCON1 = 0b00000111;        //Vref+ = External Vref + (5V to pin 27), Vref- =
    Vss, ANO:5 analog in, AN6:15 digital I/O (PAGE 214)

    ADCON0 = 0b00000100;        //select AD1, set "GO", and ADON off (PAGE
    213)

    ADCON2 = 0b00000010;        //left justify, conversion time = 64 * Tosc (PAGE
    215)
    ADCON0 = 0b00000101;        //turn on A/D module

    Delay10TCYx(13);            //6.5us

    ADCON0 = 0b00000111;        //GO

    Poll_AD_Done();
```

```
    Delay10TCYx(35);                //23us

    return ADRESH;
}

int Read_AD2(void)
{
    ADCON1 = 0b00000111;            //Vref+ = External Vref + (5V to pin 27), Vref- =
    Vss, ANO:5 analog in, AN6:15 digital I/O (PAGE 214)

    ADCON0 = 0b00001000;            //select AD2, set "GO", and ADON off (PAGE
    213)

    ADCON2 = 0b00000010;            //left justify, conversion time = 64 * Tosc (PAGE
    215)
    ADCON0 = 0b00001001;            //turn on A/D module

    Delay10TCYx(13);                //6.5us

    ADCON0 = 0b00001011;            //GO

    Poll_AD_Done();

    Delay10TCYx(35);                //23us

    return ADRESH;
}

int Read_AD3(void)
{
    ADCON1 = 0b00000111;            //Vref+ = External Vref + (5V to pin 27), Vref- =
    Vss, ANO:5 analog in, AN6:15 digital I/O (PAGE 214)

    ADCON0 = 0b00001100;            //select AD3, set "GO", and ADON off (PAGE
    213)

    ADCON2 = 0b00000010;            //left justify, conversion time = 64 * Tosc (PAGE
    215)
    ADCON0 = 0b00001101;            //turn on A/D module

    Delay10TCYx(13);                //6.5us

    ADCON0 = 0b00001111;            //GO

    Poll_AD_Done();

    Delay10TCYx(35);                //23us
```

```
    return ADRESH;
}

int Read_AD4(void)
{
    ADCON1 = 0b00000111;        //Vref+ = External Vref + (5V to pin 27), Vref- =
    Vss, ANO:5 analog in, AN6:15 digital I/O (PAGE 214)

    ADCON0 = 0b00010000;        //select AD4, set "GO", and ADON off (PAGE
    213)

    ADCON2 = 0b00000010;        //left justify, conversion time = 64 * Tosc (PAGE
    215)
    ADCON0 = 0b00010001;        //turn on A/D module

    Delay10TCYx(13);            //6.5 us

    ADCON0 = 0b00010011;        //GO

    Poll_AD_Done();

    Delay10TCYx(35);            //23us

    return ADRESH;
}

int Read_AD5(void)
{
    ADCON1 = 0b00000111;        //Vref+ = External Vref + (5V to pin 27), Vref- =
    Vss, ANO:5 analog in, AN6:15 digital I/O (PAGE 214)

    ADCON0 = 0b00010100;        //select AD5, set "GO", and ADON off (PAGE
    213)

    ADCON2 = 0b00000010;        //left justify, conversion time = 64 * Tosc (PAGE
    215)
    ADCON0 = 0b00010101;        //turn on A/D module

    Delay10TCYx(13);            //6.5us

    ADCON0 = 0b00010111;        //GO

    Poll_AD_Done();

    Delay10TCYx(35);            //23us

    return ADRESH;
```

```
}


int Read_AD6(void)
{
    ADCON1 = 0b00000111;        //Vref+ = External Vref + (5V to pin 27), Vref- =
    Vss, ANO:5 analog in, AN6:15 digital I/O (PAGE 214)

    ADCON0 = 0b00011000;        //select AD6, set "GO", and ADON off (PAGE
    213)

    ADCON2 = 0b00000010;        //left justify, conversion time = 64 * Tosc (PAGE
    215)
    ADCON0 = 0b00011001;        //turn on A/D module

    Delay10TCYx(13);            //6.5us

    ADCON0 = 0b00011011;        //GO

    Poll_AD_Done();

    Delay10TCYx(35);            //23us

    return ADRESH;
}

int Read_AD7(void)
{
    ADCON1 = 0b00000111;        //Vref+ = External Vref + (5V to pin 27), Vref- =
    Vss, ANO:5 analog in, AN6:15 digital I/O (PAGE 214)

    ADCON0 = 0b00011100;        //select AD7, set "GO", and ADON off (PAGE
    213)

    ADCON2 = 0b00000010;        //left justify, conversion time = 64 * Tosc (PAGE
    215)
    ADCON0 = 0b00011101;        //turn on A/D module

    Delay10TCYx(13);            //6.5us

    ADCON0 = 0b00011111;        //GO

    Poll_AD_Done();

    Delay10TCYx(35);            //23us

    return ADRESH;
}
```

```
/*********************************************************
* Interrupt Polling                                                    *
*
        *
* Written for the PIC18F8720                                           *
* Interfaces:
        *
* High Priority interrupts: INT0 -> Lower Fork Limit          *
*                                          INT3 -> Upper Fork Limit    *
* Low Priority interrupts:      INT1 -> TRF-24G RD1                    *
*                                          INT2 -> Rear Bump switches  *
*                                                          (falling edge)   *
*
        *
* Credits: Microchip                                                   *
* Modified by Albert Chung                                             *
*********************************************************/

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

#define driver2_en      PORTDbits.RD3

void low_isr(void);
void high_isr(void);
void Init_Int(void);
void Disable_TMR0_IRQ(void);
void Init_TMR1_Overflow_IRQ(void);

// RF Stop and Wait ARQ
#define TIMEOUT_SIZE 3
unsigned char timeout_ctr = 0;
unsigned char num_timeouts = 0;


// IR Detector variables
unsigned char count;
unsigned char data[5];
unsigned char remote;
unsigned char num_overflows;
unsigned char go = 1;
unsigned char skip = 5; // Used to skip the first 5 IR samples on startup, subsequent IR
    samples must skip 6
unsigned char manual_mode = 0;
```

```
// Manual Drive Mode Motor Speeds
int forward = 100;
int reverse = -100;


// Servo compare code
unsigned char present = SHORT;

/*****************************************************************
* 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)
{
/*
    unsigned char temp1, temp2;

    if ( INTCON3bits.INT2IF == 1)  // IR triggered interrupt
    {
        T3CON = 0b00000001;

        // Skip the first 5 samples

        for(count = 0; count < skip; count++)
        {
                while (PORTBbits.RB2 == 0)          //wait for rising edge
                {}

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

        // Grab the length of the next 5 high pulses

        for(count = 0; count < 5; count++)
        {
```

```
        TMR3H = 0x00;
        TMR3L = 0x00;

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

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

        if (TMR3H > 20)
        {
                data[count] = 1;
        }
        else
        {
                data[count] = 0;
        }

}

if (skip == 5)
{
        skip = 6;
}

// Convert data[] into an integer

count = 0;
remote = 0;

if (data[count++] == 1)
{
        remote |= 0b00001;
}
if (data[count++] == 1)
{
        remote |= 0b00010;
}
if (data[count++] == 1)
{
        remote |= 0b00100;
}
if (data[count++] == 1)
{
        remote |= 0b01000;
}
if (data[count] == 1)
{
```

```
            remote |= 0b10000;
        }

        // "remote" now contains the hex code for the remote button

        TMR3H = 0;
        TMR3L = 0;

        Init_TMR3_Overflow_IRQ();           // Enable TMR3 overflow interupt

        INTCON3bits.INT2IE = 0;                     // Disable INT1 XIRQ

        INTCON3bits.INT2IF = 0;                             // Clear INT2 flag
    }
*/

    if ( INTCON3bits.INT1IF == 1)  // RF Rx data ready
    {
        Receive();

        INTCON3bits.INT1IF = 0;             // Clear the INT1 Flag
    }

    if (INTCONbits.TMR0IF == 1)             // TMR0 overflowed (no ACK recieved)
    {
        if( num_timeouts == 4 )
        {
                if( timeout_ctr < TIMEOUT_SIZE )
                {
                        Disable_TMR0_IRQ();
                        Transmit(tx_buffer);            // Other terminal missed the packet /
    resend
                        timeout_ctr++;
                }
                else                                                    // Give up sending
    data
                {
                        Disable_TMR0_IRQ();
                        timeout_ctr = 0;
                }

                num_timeouts = 0;
        }

        num_timeouts++;

        INTCONbits.TMR0IF = 0;              // Clear the TMR0 overflow 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 ( PIR2bits.CCP2IF == 1)                  // Compare Module
    {
        if (present == SHORT)
        {
                PORTEbits.RE3 = 0;                      // Servo low
                CCPR2L = 0b11010100;                    // New compare time = 20ms
                CCPR2H = 0b00110000;
                present = LONG;
        }
        else            // (present == LONG)
        {
                PORTEbits.RE3 = 1;                      // Servo high

                TMR1H = 0x00;
                TMR1L = 0x00;

                if (fork_direction == DOWN)
                {
                        CCPR2L = 0b11101110;        // 1.2 Ms 1011101110
                        CCPR2H = 0b010;

//CCP2CON = 0b00000000;             // Turn off CCP module

                }
                else    // fork direction == DOWN
                {
// 1110101001 straight up
                        CCPR2L = 0b10100100;        // 2 Ms 10010100100
                        CCPR2H = 0b100;
```

```
                }

                present = SHORT;
        }

        PIR2bits.CCP2IF = 0;            // Clear the INT0 Flag
    }

    if ( PIR2bits.TMR3IF == 1)                  // Timer 3 overflowed
    {
        if (++num_overflows == 30)
        {
                INTCON3bits.INT2IE = 1;             // Enable INT1 XIRQ
                PIE2bits.TMR3IE = 0;                // Disable interupt
                num_overflows = 0;
        }

        PIR2bits.TMR3IF = 0;            // Clear flag
    }

    if ( PIR2bits.CCP2IF == 1)                  // Capture pin triggered
    {
        // ...
        PIR2bits.CCP2IF = 0;            // Clear the INT0 Flag
    }

    if ( PIR1bits.TMR1IF == 0)                  // Timer 1 overflowed
    {
        // ...
        PIR1bits.TMR1IF = 0;
    }
}


/*************************************************************
* 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 Disable_Global_IRQ(void)
{
    INTCONbits.GIEH = 0;          // Enable all high priority interrupts
    INTCONbits.GIEL       = 0;          // Enable all low priority interrupts
}

void Init_TMR3_Overflow_IRQ(void)
{
    IPR2bits.TMR3IP = 1;          // High priority
    PIR2bits.TMR3IF = 0;          // Clear flag
    PIE2bits.TMR3IE = 1;          // Enable interrupt
}

void Init_RF_IRQ(void)
{
    INTCON2bits.INTEDG1 = 1;    // INT1 = rising edge interrupt
    INTCON3bits.INT1IP = 0;           // INT1 = low priority interrupt
    INTCON3bits.INT1IF = 0;           // Clear the INT1 Flag
    INTCON3bits.INT1IE = 1;           // Enable INT1 XIRQ
}

void Disable_RF_IRQ(void)
{
    INTCON3bits.INT1IE = 0;           // Disable INT1 XIRQ
}

void Init_TMR0_IRQ(void)
{
    INTCON2bits.TMR0IP = 0;           // TMR0 = low priority interrupt

// 400 ms  timer flag
    T0CON = 0b10000101;           // Timer0 on, 16 bit, instruction clk (5MHz), low to
    high transition increment, prescale on, 5MHz/64
    TMR0H = 0;                                  // Clear the Timer
    TMR0L = 1;

    INTCONbits.TMR0IF = 0;            // Clear the TMR0 overflow Flag
    INTCONbits.TMR0IE = 1;            // Enable TMR0 overflow IRQ
    INTCONbits.TMR0IF = 0;            // Clear the TMR0 overflow Flag
}

void Disable_TMR0_IRQ(void)
{
    T0CON = 0;                                  // Turn off the timer
    INTCONbits.TMR0IE = 0;            // Disable TMR0 overflow IRQ
}

/*
```

```
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

    // Configure Timer1 which is used in subroutine

    T1CON = 0b00000001;
}
*/

void Init_Servo_Comp(void)
{
    IPR2bits.CCP2IP = 1;            // High priority
    PIR2bits.CCP2IF = 0;            // Clears flag
    PIE2bits.CCP2IE = 1;            // Enables interupt for compare
}

void Disable_Servo_Comp(void)
{
    PIE2bits.CCP2IE = 0;            // Disables compare
}
```

```
/*****************************************************
* MOTOR CONTROLLER                                   *
*                                                    *
*                                                    *
* Written for the PIC18F8720 @ 20 MHz                *
* Interfaces: Digital outputs on CCP1,3,4,5 that     *
*   connect to a JRC Dual H-Bridge NJM2670.          *
* Copyright (2005) Trevor Skipp & Albert Chung       *
*                                                    *
*                                                    *
* NOTES:                                             *
*                                                    *
* The PWM output pins CCP1,3,4,5 must be set to      *
*   outputs in the main function.                    *
*     DDRC = DDRC & 0b11111011;                      *
*     DDRG = DDRG & 0b11100110;                      *
* 8 bit resolution                                   *
* The JRC chip uses 2 PWM inputs to control each     *
*   motor.  No direction pin is used.                *
*   CCP1: Left motor (-)         / Fork motor (+)    *
*   CCP3: Left motor (+)         / Fork motor (-)    *
*   CCP4: Right motor (+)                            *
*   CCP5: Right motor (-)                            *
*   RD2:  Motor Driver 1 Enable                      *
*   RD3:  Motor Driver 2 Enable                      *
*****************************************************/

#include <p18f8720.h>
#include <delays.h>
#include "interrupts.h"
#include "motors.h"

#define driver1_en      PORTDbits.RD2
#define driver2_en      PORTDbits.RD3
#define lower_fork_limit PORTEbits.RE2
#define upper_fork_limit PORTEbits.RE6
#define step_size 5

int left_old_speed = 0;
int left_new_speed = 0;

int right_old_speed = 0;
int right_new_speed = 0;

unsigned char fork_direction;

/*****************************************************
```

```
 * PWM_Init: Initialize the PWM module for 4 DC                *
 *   motors using Timer 2 and 1 servo using Timer 1.  *
 ***************************************************/
void PWM_Init(void)
{
    PR2 = 0xFF;                                 //Period

    CCPR1L = 0;                                 //Duty cycle = 0
    CCPR3L = 0;
    CCPR4L = 0;
    CCPR5L = 0;

    T3CONbits.T3CCP1 = 0; // Timer1 for compare
    T3CONbits.T3CCP2 = 0; // Timer2 for PWM
    T2CON = 0b00000101;             //Timer 2: no postscale, module on, and 4 prescaler

    CCP1CON = 0b00001100;           //PWM mode and Duty cycle's LSB1:0 = 0
    CCP3CON = 0b00001100;
    CCP4CON = 0b00001100;
    CCP5CON = 0b00001100;
}

/***************************************************
 * Fork: Raise or lower the fork.                                  *
 ***************************************************/
void Fork(unsigned char direction)
{
    // Timer1 16 bit write, 1:8 prescaler, internal clock, module on
    T1CON = 0b00110001;

    // Reset Timer 1

    TMR1H = 0x00;
    TMR1L = 0x00;

    present = SHORT;

    if (direction == DOWN)
    {
        fork_direction = DOWN;

        CCPR2L = 0b11101110;        // 1.2 Ms 1011101110
        CCPR2H = 0b010;

        CCP2CON = 0b00001010;               // CCP pin generate software interupt
    }
    else if (direction == UP)           // Set initial compare time to 1.3ms
    {
```

```
        fork_direction = UP;

        CCPR2L = 0b10100100;        // 2 Ms 10010100100
        CCPR2H = 0b100;

        CCP2CON = 0b00001010;               // CCP pin generate software interupt
    }
    else        // Servo OFF
    {
        CCP2CON = 0b00000000;               // Turn off CCP module
        T1CON = 0b00000000;                         // Timer1 module off
        PORTEbits.RE3 = 0;                  // Servo always low
    }

    return;
}

/*****************************************************
 * Motors: Adjust the DC motor speed.  Implement    *
 *   a smoothing function to improve driving.       *
 *****************************************************/
void Motors(int left_desired_speed, int right_desired_speed)
{
    int i = 1;

    driver1_en = 1;
    driver2_en = 0;

    while( (left_new_speed != left_desired_speed) || (right_new_speed !=
    right_desired_speed) )
    {
        if (left_desired_speed > left_old_speed)
        {
                left_new_speed = left_old_speed + step_size;
        }
                else if( left_desired_speed < left_old_speed )
                {
                        left_new_speed = left_old_speed - step_size;
                }

        if( right_desired_speed > right_old_speed )
        {
                right_new_speed = right_old_speed + step_size;
        }
                else if( right_desired_speed < right_old_speed )
                {
                        right_new_speed = right_old_speed - step_size;
                }
```

```
        if ( left_new_speed >= 0 )
        {
                CCPR1L = left_new_speed;                        // left motor forward
                CCPR3L = 0;
        }
                else
                {
                        CCPR3L = -1*left_new_speed;
        // left motor reverse
                        CCPR1L = 0;
                }

        if ( right_new_speed >= 0 )
        {
                CCPR4L = right_new_speed;                       // right motor forward
                CCPR5L = 0;
        }
                else
                {
                        CCPR5L = -1*right_new_speed;
        // right motor reverse
                        CCPR4L = 0;
                }

        left_old_speed = left_new_speed;
        right_old_speed = right_new_speed;

        if ((255/i) > 0 && i > 0)
        {
                Delay100TCYx(255/i);
        }

        i++;
    }
}
```

```
/*****************************************************
* LINE FOLLOWING                                     *
*                                                    *
*                                                    *
* Written for the PIC18F8720 @ 20 MHz                *
* Interfaces: Digital I/O on RH0, RH1, RH2, RH3,     *
*   and INT1, that connect to a TRF-24G RF module    *
* Copywrite 2005 Albert Chung and Trevor Skipp       *
*                                                    *
*                                                    *
* NOTES:                                             *
*                                                    *
* Hardware Connections:                              *
* LEFT_IR:              AN4                          *
* MID_LEFT_IR:          AN5                          *
* MID_RIGHT_IR:         AN6                          *
* RIGHT_IR:             AN7                          *
*****************************************************/

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

// Navigation definitions
#define DIST_THRESHOLD 160
#define rear_bump PORTBbits.RB4
#define TURN_TIME 110

// Line follower declarations
#define LINE_THRESHOLD 20
int black_l, black_ml, black_mr, black_r;
int line_data;
int prev_line_data;

int rd_l, rd_ml, rd_mr, rd_r;

// Mapping Declarations
unsigned char x_cur = 0xFF;
unsigned char x_dest, y_dest;
signed char y_cur = 0;
signed char direction = 0;

unsigned char Map_Intersection(void);
```

```
/*********************************************************
 * Calibrate_LF: IR sensors in line follower can be      *
 *       affected by ambient light.  This function       *
 *       allows for the dynamic calibration of the       *
 *       line follower module.                                  *
 *********************************************************/
void Calibrate_LF (void)
{
    int i;
    int LEFT_RD[3];
    int MID_LEFT_RD[3];
    int MID_RIGHT_RD[3];
    int RIGHT_RD[3];

    while(PORTBbits.RB0 == 0)
    {
    }

    for (i = 0; i < 3; i++)
    {
        LEFT_RD[i] = LEFT_IR;
        MID_LEFT_RD[i] = MID_LEFT_IR;
        MID_RIGHT_RD[i] = MID_RIGHT_IR;
        RIGHT_RD[i] = RIGHT_IR;

        Delay10KTCYx(10);
    }

    black_l = (LEFT_RD[0] + LEFT_RD[1] + LEFT_RD[2]) / 3;
    black_ml = (MID_LEFT_RD[0] + MID_LEFT_RD[1] + MID_LEFT_RD[2]) / 3;
    black_mr = (MID_RIGHT_RD[0] + MID_RIGHT_RD[1] + MID_RIGHT_RD[2]) /
    3;
    black_r = (RIGHT_RD[0] + RIGHT_RD[1] + RIGHT_RD[2]) / 3;

    black_l += LINE_THRESHOLD;
    black_ml += LINE_THRESHOLD;
    black_mr += LINE_THRESHOLD;
    black_r += LINE_THRESHOLD;

    sprintf(message, "%3d %3d %3d %3d", black_l, black_ml, black_mr, black_r);
    UPDATE_LCD;

    Delay10KTCYx(0);
    Delay10KTCYx(0);
}

/*********************************************************
 * Navigate: Reads the IR line following sensors and   *
```

```
 *        converts the array into an integer                              *
 ***********************************************************/
unsigned char Read_IR (void)
{
    line_data = 0x00;

    if (LEFT_IR < black_l)
    {
        line_data |= 0b1000;
    }
    if (MID_LEFT_IR < black_ml)
    {
        line_data |= 0b0100;
    }
    if (MID_RIGHT_IR < black_mr)
    {
        line_data |= 0b0010;
    }
    if (RIGHT_IR < black_r)
    {
        line_data |= 0b0001;
    }

}

/***********************************************************
 * Turn_Left: Sets motors and delays processor so the          *
 *        vehicle can get off of the current line and          *
 *        begin polling for the next line.                     *
 ***********************************************************/
void Turn_Left(void)
{
//  sprintf(message,"Turning Left");
//  UPDATE_LCD;

    direction++;
    if( direction > 3)
    {
        direction = 0;
    }

    Motors(L_MF_FORWARD, R_MF_FORWARD);                 // Move forward
    before turning to center rear wheels on the line
    Delay10KTCYx(TURN_TIME);

    Motors(S_REVERSE, S_FORWARD);
    Delay10KTCYx(0);
```

```
    do
    {
        Read_IR();
    } while( line_data!= 0b0110 );

    return;
}

/*********************************************************
 * Turn_Right: Sets motors and delays processor so the   *
 *       vehicle can get off of the current line and     *
 *       begin polling for the next line.                *
 *********************************************************/
void Turn_Right(void)
{
    direction--;
    if( direction < 0)
    {
        direction = 3;
    }

//  sprintf(message,"Turning Right");
//  UPDATE_LCD;

    Motors(L_MF_FORWARD, R_MF_FORWARD);                 // Move forward
    before turning to center rear wheels on the line
    Delay10KTCYx(TURN_TIME);

    Motors(S_FORWARD, S_REVERSE);
    Delay10KTCYx(0);

    do
    {
        Read_IR();
    } while( line_data != 0b0110 );
}

/*********************************************************
 * Turn_Around: Sets motors and delays processor so the *
 *       vehicle can get off of the current line and     *
 *       begin polling for the next line.  Do to the     *
 *       operating environment, special considerations *
 *       where taken into the direction of the turn.     *
 *********************************************************/
void Turn_Around(void)
{
    direction ++;
    if( direction > 3)
```

```
    {
        direction = 0;
    }
    direction ++;
    if( direction > 3)
    {
        direction = 0;
    }

//  sprintf(message,"Turning Around");
//  UPDATE_LCD;

    if( (x_cur == 0) || (direction == 2) )
    {
        Motors(S_FORWARD, S_REVERSE);
    }
    else
    {
        Motors(S_REVERSE, S_FORWARD);
    }

    Delay10KTCYx(0);
    do
    {
        Read_IR();
    } while( line_data != 0b0110 );
}

/*********************************************************
 * Back_Up: Reverses the motors and delays for a set *
 *        amount of time.  No consideration for line        *
 *        tracking is needed.  Polls rear bumper to         *
 *        determine if a collision has occurred.            *
 *********************************************************/
void Back_Up(void)
{
    unsigned char temp, i;

    Motors(L_M_REVERSE, R_M_REVERSE);

    Delay10KTCYx(0);
    Delay10KTCYx(0);

    Motors(0,0);
}

/*********************************************************
 * Navigate: Use value from line following sensors to  *
```

```
*       determine what action to take.  Stops robot if *
*       something is in its forward collision path.                *
* External variables: prev_line_data                                    *
* Makes changes to: prev_line_data                                          *
*************************************************/
void Navigate(void)
{
    unsigned char counter = 0;

    while(1)
    {
        Read_IR();

        if( line_data == 0b0000 )                       // No Line
        {
                if( prev_line_data == 0b0100 )
                {
                        Motors(0,R_M_FORWARD);
                }
                else if( prev_line_data == 0b0010 )
                {
                        Motors(L_M_FORWARD,0);
                }
                else
                {
                        Motors(0,0);
                }
        }
        else if( line_data == 0b0010 )          // Line to right
        {
                Motors(L_MF_FORWARD, R_M_FORWARD);
                prev_line_data = line_data;
        }
        else if( line_data == 0b0100 )          // Line to left
        {
                Motors(L_M_FORWARD, R_MF_FORWARD);
                prev_line_data = line_data;
        }
        else if( line_data == 0b0110 )          // Centered on line
        {
                Motors(L_MF_FORWARD, R_MF_FORWARD);
        }
        else if( (line_data & 0b1000) == 0b1000 || (line_data & 0b0001) == 0b0001 )
        // Intersection or stop marker
        {
                if( direction == 1 && y_cur == 1 )              // Heading North (facing
    shelves)
                {
```

```
                Motors(0,0);
                return;
            }
            if( direction == 3 && y_cur == -1 )          // Heading North (facing
    shelves)
            {
                Motors(0,0);
                return;
            }
            else
            {
                Map_Intersection();
            }
        }
    }
}


/*********************************************************
 * Map_Intersection: Uses current location and                    *
 *   destination location to determine whether to go          *
 *   forward, left, right, or to turn around.                 *
 *********************************************************/
unsigned char Map_Intersection(void)
{
    if( direction == 0 )                    // Heading East
    {
        x_cur++;                                          // Increment Lane Number

        if( x_cur < x_dest )
        {
            Motors(L_MF_FORWARD,R_MF_FORWARD);
            Delay10KTCYx(175);                            // Go Forward & wait before
    reading next line data
        }
        else if( x_cur > x_dest )
        {
            Turn_Around();                                // Turn Around
        }
        else                                              // Arrived at destination
        {
            if( y_dest == 1 )                       // Shelf
            {
                y_cur = 1;
                Turn_Left();
            }
            else                                          // Dock
            {
```

```
                y_cur = -1;
                Turn_Right();
            }
        }
    }
    else if( direction == 1 )                   // Heading North (must be coming from
dock)
    {
        if( x_cur < x_dest )
        {
            Turn_Right();
            y_cur = 0;
        }
        else if( x_cur > x_dest )
        {
            Turn_Left();
            y_cur = 0;
        }
        else    // x_cur == x_dest
        {
            Motors(L_MF_FORWARD, R_MF_FORWARD);
            Delay10KTCYx(175);                  // Go Forward & wait before
reading next line data
            y_cur = 1;
        }
    }
    else if( direction == 2 )   // direction == 2                   // Heading West
    {
        x_cur--;                                                    // Decrement Lane
Number

        if( x_cur > x_dest )                    //
        {
            Motors(L_MF_FORWARD, R_MF_FORWARD);
            Delay10KTCYx(175);                  // Go Forward & wait before
reading next line data
        }
        else if( x_cur < x_dest )
        {
            Turn_Around();                      // Turn Around
        }
        else                                    // Arrived at destination
        {
            if( y_dest == 1 )           // Heading West
            {
                y_cur = 1;
                Turn_Right();
            }
```

```
                else
                {
                        y_cur = -1;
                        Turn_Left();
                }
        }
    }
    else        // direction == 3                       // Heading South (must be coming
    from shelves)
    {

        if( x_cur < x_dest )
        {
                Turn_Left();
                y_cur = 0;
        }
        else if( x_cur > x_dest )
        {
                Turn_Right();
                y_cur = 0;
        }
        else    // x_cur == x_dest
        {
                Motors(L_MF_FORWARD, R_MF_FORWARD);
                Delay10KTCYx(175);                      // Go Forward & wait before
    reading next line data
                y_cur = -1;
        }
    }
}
```

```
/************************************************************
* RF LINK                                                   *
*                                                           *
*                                                           *
* Written for the PIC18F8720 @ 20 MHz (5MHz Instruction)    *
* Interfaces: Digital I/O on RH0, RH1, RH2, RH3,            *
*   and INT1, that connect to a TRF-24G RF module           *
* Copyright (2005) Albert Chung                             *
*                                                           *
*                                                           *
* NOTES:                                                    *
*                                                           *
* Hardware Connections:                                     *
*                                                           *
*    RH0 (I/O):      TRF-24G Data                           *
*                                                           *
*   H1 (Output):     TRF-24G CLK1                           *
*   H2 (Output):     TRF-24G CS                             *
*   H3 (Output):     TRF-24G CE                             *
*   NT1 (I/O):       TRF-24G DR1                            *
*                                                           *
************************************************************/

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

unsigned char tx_buffer = 0;
unsigned char rx_buffer = 0;
unsigned char rx_buffer_synch = 0;

unsigned char tx_frame_num = 0b10000000;
unsigned char rx_frame_num = 0b00000000;
unsigned char synch_ctr = 0;

#define ACK 0xAA                    // Acknowledgement for stop and wait protocol (not
    used yet)
#define NCK 0xFF                    // Reject for stop and wait protocol

void CLK (void)
{
    Delay10TCYx(10);          // 500 nsec (tsetup)
    CLK1 = 1;                       // clock in the value
```

```
    Delay10TCYx(10);            // 500 nsec (thold)
    CLK1 = 0;
}

void Tx_En(void)                        // Set module to active transmit mode
{
    signed char i;
    char Tx_2500MHz = (rf_ch << 1);

    DDRHbits.RH0 = 0;
    DDRHbits.RH1 = 0;
    DDRHbits.RH2 = 0;
    DDRHbits.RH3 = 0;

    CE = 0;                                         // Set configuration mode
    CS = 1;                                         // Select configuration register

    Delay10TCYx(40);                    // 10 usec (tcs2data)

    for ( i = 7; i >= 0; i--)
    {
        DATA = Tx_2500MHz >> i;             // Shift out RF channel and set to RX_EN =
    0
        CLK();                                      // Clock in the data
    }

    CS = 0;                                             // shift the configuration
    word into the module

    RF_IRQ_OFF;                                         // Disallow RD1 to interrupt
    MCU

    Delay100TCYx(100);                          // 250 usec (tsettling)

    CE = 1;                                             // Turn on ACTIVE TX
    Mode

    Delay10TCYx(40);                    // 10 usec (tce2data)
}

void Rx_En(void)                                // Set module to active receive mode
{
    signed char i;
    char Rx_2500MHz = (rf_ch << 1) | 1;             // Last byte in configuration word

    DDRHbits.RH0 = 0;                           // Set Outputs
    DDRHbits.RH1 = 0;
    DDRHbits.RH2 = 0;
```

```
    DDRHbits.RH3 = 0;

    CE = 0;                                                 // Set configuration mode
    CS = 1;                                                 // Select configuration
    register

    Delay10TCYx(40);                        // 10 usec (tcs2data)

    for ( i = 7; i >= 0; i--)
    {
        DATA = Rx_2500MHz >> i;             // Shift out RF channel and set to RX_EN =
    1
        CLK();                                      // Clock in the data
    }

    CS = 0;                                 // shift the configuration word into the
    module

    DDRHbits.RH0 = 1;               // Set Data as Input

    Delay100TCYx(100);              // 250 usec (tsettling)

    RF_IRQ_ON;                              // Allow RD1 to interrupt MCU

    CE = 1;                                 // Turn on ACTIVE TX Mode

    Delay10TCYx(40);            // 10 usec (tce2data)
}

/*********************************************************
 * Transmit: Adds header error control and clocks out*
 *   data to the RF chip.                                                  *
 *********************************************************/
void Transmit (int tx_payload)
{
    int parity;
    signed char i;

    tx_payload = tx_payload >> 8;

    tx_payload = tx_payload << 1;
    tx_payload &= 0b00011110;
    tx_payload |= 0b10100000;

    parity = tx_payload % 2;

    if (parity == 0)                // Even
    {
```

```
        tx_payload |= 0b00000001;
    }

    Tx_En();

    for ( i = 7; i >= 0; i--)
    {
        DATA = rx_addr >> i;                    // Shift out Rx address (MSB first)
        CLK();                                  // Clock in the data
    }

    for ( i = 7 ; i >=  0; i--)
    {
        DATA = tx_payload >> i;                 // Shift out payload (MSB first)
        CLK();                                  // Clock in the data
    }

    CE = 0;                                     // Activate
    Shockburst Tx

    Rx_En();

    return;
}
/*************************************************
 * Receive: Clock in data from the RF chip and   *
 *   header error control such as parity to determine if *
 *   data is valid.
        *
 *************************************************/
void Receive (void)
{
    signed char i;
    int temp_payload = 0;
    int parity;
    int temp;

//  T0CONbits.TMR0ON = 0;                       // Pause the
    timer to process logic

    for ( i = 7; i >=  0; i--)
    {
        CLK1 = 1;                               // Clock in the
    data
        Delay10TCYx(10);                        // 500 nsec (tsetup)
        temp_payload |=  DATA << i;             // Shift in payload
    (MSB first)
        CLK1 = 0;
```

```
        Delay10TCYx(10);                                        // 500 nsec (tsetup)
    }

    parity = temp_payload % 2;
    if (parity == 0)                // Even
    {
        sprintf(message, "Parity Error    %d", temp_payload);
        UPDATE_LCD;
        return;                             // Bad data
    }

// AGV COMMAND
    if( (temp_payload & 0b11100000) == 0b10100000 )         // Check header
    {
        temp_payload = temp_payload >> 1;
        temp = temp_payload & 0b00000111;

        if( temp <= 4)
        {
                temp_payload &= 0b00001111;
                Queue_Push(temp_payload);                                   // Place into
    queue
        }
    }
// End AGV COMMAND

}

/*********************************************************
 * Init_RF: Configure the RF module.                     *
 *********************************************************/
void Init_RF (void)
{

// Set up the configuration packet in segments of 8 bits
// {data_w, addr2 not used = 5x"0", redundant address bits exceeding address width =
    3x"0", asrs_addr, addr_w[bit7:2] crc[bit1:0], mode, rf_ch & receive mode}

    char addr_w_crc = ( addr_w << 2 ) | crc;
    char config[15] = {0,data_w, 0, 0, 0, 0, 0, 0, 0, 0, 0, tx_addr, addr_w_crc, mode, (rf_ch
    << 1)};
    unsigned char i;
    signed char j;

    DDRHbits.RH0 = 0;
    DDRHbits.RH1 = 0;
    DDRHbits.RH2 = 0;
    DDRHbits.RH3 = 0;
```

```
    Delay10KTCYx(10);       // 5 msec (tpd2sby)

    CE = 0;                         // Set configuration mode
    CS = 1;                         // Select configuration register

    Delay10TCYx(40); // 10 usec (tcs2data)

    for ( i = 0; i < 15; i++)
    {
        for ( j = 7; j >= 0; j--)       // send the configuration word MSB first
        {
                DATA = config[i] >> j;              // shift config word 1 bit at a time
                CLK();                                      // Clock in the data
        }
    }

    CS = 0;                                 // Shift the configuration word into the
    module

    Rx_En();                        // Set the module to active Rx mode

    INTCONbits.TMR0IF = 0;              // Clear the TMR0 overflow Flag
}

unsigned char Frame (unsigned char dock_num)
{
    unsigned char data_frame = 0;

    if( tx_frame_num == 0b10000000 )
    {
        tx_frame_num = 0;
    }
    else
    {
        tx_frame_num = 0b10000000;
    }

    data_frame &= 0b00001111;

    data_frame |= dock_num;
    data_frame |= header;                                   // Add the header
    data_frame |= tx_frame_num;                             // Add the frame number

return data_frame;
}
```

```
/*****************************************************
 * ANALOG TO DIGITAL CONVERSION                                    *
 *
        *
 * Written for the PIC18F8720 @ 20 MHz                  *
 * Interfaces: Analog inputs on ANO:7                   *
 * Copyright 2005 Trevor Skipp                                 *
 *****************************************************/

#define ADCH0                   Read_AD0()
#define RIGHT_DIST              Read_AD1()
#define LEFT_DIST       Read_AD2()
#define ADCH3                   Read_AD3()
#define LEFT_IR                 Read_AD4()
#define MID_LEFT_IR     Read_AD5()
#define MID_RIGHT_IR    Read_AD6()
#define RIGHT_IR        Read_AD7()

extern int Read_AD0(void);
extern int Read_AD1(void);
extern int Read_AD2(void);
extern int Read_AD3(void);
extern int Read_AD4(void);
extern int Read_AD5(void);
extern int Read_AD6(void);
extern int Read_AD7(void);
```

```
/*************************************************
 * AUTOMATED STORAGE & RETRIEVAL SYSTEM                          *
 *
        *
 * EEL5666 Intelligent Machines Design Laboratory   *
 * University of Florida                                        *
 * Written for the PIC18F8720 @ 20 MHz                          *
 * Copyright (2005) Albert Chung 2.12.2005                  *
 *************************************************/

extern unsigned char IO;
extern unsigned char first;
extern unsigned char last;

// Function declarations
extern unsigned char Queue_Pull(void);
extern void Queue_Push(unsigned char);
extern void Move_Pallet(void);
```

```
/************************************************
 * HITACHI LCD CONTROLLER                                        *
 *
         *
 * Written for the PIC18F8720 @ 20 MHz                          *
 * Interfaces: Enable on RD0, RS on RD1, DB4:7 on *
 *   RD4:7
         *
 * Copyright 2005 Trevor Skipp & Albert Chung                *
 ************************************************/

#define INIT_LCD LCD_Init();
#define UPDATE_LCD Update_LCD();

extern char message[];

extern void LCD_Init(void);
extern void Update_LCD(void);
```

```
/*****************************************************
* Interrupt Polling                                                 *
*                                                                   *
*                                                                   *
* Written for the PIC18F8720                                        *
* Interfaces:                                                       *
* Low Priority interrupts: INT0 -> NJM2670 Alarm  *
*                                                   INT1 -> TRF-24G RD1             *
*                                                                   *
*                                                                   *
* Credits: Microchip                                                *
* Modified by Albert Chung                                          *
*****************************************************/

#define LONG 1
#define SHORT 0

#define GLOBAL_IRQ_ON  Init_Global_IRQ()
#define INIT_CCP_IRQ       Init_Capture_IRQ()
#define RF_IRQ_ON          Init_RF_IRQ()
#define RF_IRQ_OFF                 Disable_RF_IRQ()
#define TMR0_IRQ_ON                Init_TMR0_IRQ()
#define TMR0_IRQ_OFF     Disable_TMR0_IRQ()

extern void Init_Global_IRQ(void);
extern void Disable_Global_IRQ(void);
extern void Init_RF_IRQ(void);
extern void Disable_RF_IRQ(void);
extern void Init_TMR0_IRQ(void);
extern void Disable_TMR0_IRQ(void);
extern void Init_IR_IRQ(void);
extern void Init_TMR3_Overflow_IRQ(void);
extern void Init_Fork_Limit_IRQ(void);

extern unsigned char timeout_ctr;

extern unsigned char present;
```

```
/*****************************************************
 * MOTOR CONTROLLER                                  *
 *                                                   *
 *                                                   *
 * Written for the PIC18F8720 @ 20 MHz               *
 * Interfaces: Digital outputs on CCP1,3,4,5 that    *
 *   connect to a JRC Dual H-Bridge NJM2670.         *
 * Copyright (2005) Trevor Skipp                     *
 *****************************************************/


#define     L_MF_FORWARD 175
#define     R_MF_FORWARD 170
#define     L_M_FORWARD 130
#define     R_M_FORWARD 125
#define     S_FORWARD 85
#define     S_REVERSE -85
#define     L_M_REVERSE -155
#define     R_M_REVERSE -135
#define     MF_REVERSE -170


extern int left_old_speed;
extern int left_new_speed;

extern int right_old_speed;
extern int right_new_speed;
extern unsigned char fork_direction;

extern void PWM_Init(void);
extern void Fork(unsigned char direction);
extern void Motors(int left_desired_speed, int right_desired_speed);

// Servo declarations
#define UP 1
#define DOWN 2
#define DISABLE 0
```

```
/*****************************************************
* LINE FOLLOWING                                     *
*                                                    *
*                                                    *
* Written for the PIC18F8720 @ 20 MHz                *
* Interfaces: Digital I/O on RH0, RH1, RH2, RH3,     *
*   and INT1, that connect to a TRF-24G RF module    *
* Credit:        William Dubel original code         *
*                                                    *
*                                                    *
* NOTES:                                             *
*                                                    *
* Hardware Connections:                              *
* LEFT_IR:                AN4                        *
* MID_LEFT_IR:            AN5                        *
* MID_RIGHT_IR:           AN6                        *
* RIGHT_IR:               AN7                        *
*****************************************************/

extern void Calibrate_LF (void);
extern void Turn_Left(void);
extern void Turn_Right(void);
extern void Turn_Around(void);
extern void Back_Up(void);
extern void Navigate (void);

extern unsigned char direction;
extern unsigned char x_dest;
extern signed char y_dest;
extern unsigned char x_cur;
extern signed char y_cur;
```

```
/*****************************************************
 * RF LINK                                           *
 *                                                   *
 *                                                   *
 * Written for the PIC18F8720 @ 20 MHz               *
 * Interfaces: Digital I/O on RH0, RH1, RH2, RH3,    *
 *   and INT1, that connect to a TRF-24G RF module   *
 * Copyright (2005) Albert Chung                     *
 *****************************************************/

#define DATA PORTHbits.RH0
#define CLK1 PORTHbits.RH1
#define CS PORTHbits.RH2
#define CE PORTHbits.RH3

#define tx_addr 0b11011101
#define rx_addr 0b11100110
#define data_w 8
#define addr_w 8
#define crc 0b11                    // CRC enable
#define mode 0b01001111             // Rx2En = 0,  Shockburst Mode, 250kbps,16 MHz
    module crystal,  0db Power
#define rf_ch 0x64                  // 2500 MHz frequency channel
#define header 0b01010000

extern unsigned char tx_buffer;
extern unsigned char rx_buffer;

extern void Transmit(int tx_payload);
extern void Receive(void);
extern void Init_RF(void);
extern unsigned char Frame(unsigned char dock_num);
```