Line Tracing Robot (Name: TBD)

Final Project Report
December 12, 2002
E155 – Microprocessor Design

Morgan Cross
Raymond Fong

Abstract:

TBD is a robot that follows a path composed of black electric tape against a white background. While line following path robots have been designed before, TBD uses a new and robust algorithm that allows it to turn up to 180º and work even in a non-closed path. The robot is a self-powered individual entity unattached to other objects. The mobilizing units on TBD are two modified RC servos. The algorithm is implemented in an HC908 microcontroller which takes inputs from seven strategically placed phototransistors and sends appropriate instructions to the RC servos. TBD is able to stay on the path but is not fully capable of continuing in the same direction in certain situations.

Introduction

The team designed a robot (TBD) capable of following a pre-set path composed of electric tape on a relatively white ground. The robot is an individual entity not attached to any other objects. It contains its own power source and is controlled by an HC908 microcontroller that is mounted onboard. The robot, once powered on, would run for infinitely long (or until it is turned off or its onboard power source is depleted) and is capable of turning up to 180 degrees so that it is able to return along the path when it reaches the end of a line.

Algorithm

In order to follow the preset path, TBD is able to detect the path and determine which direction the path is headed. This is accomplished with the use of seven reflective infrared sensors and by ensuring that the path is composed of black electric tape against a relatively white ground. The sensors take advantage of the fact that the tape has a different reflectivity than the white background and depending on the signals the sensors get back, TBD is able to tell which of the sensors are directly above the path. The sensors are strategically placed on the base of the robot (Fig. 1) such that sensor D will always be over the path (this will be an axiom which the robot will function on). TBD scans sensors B and C frequently to see if either one has detected the path, if so, it stores the appropriate values that will command the robot to spin in the direction of the sensor into four memory locations. The robot moves forward while sensors D and G detect the path, however, as soon as G does not detect the path, it goes into spin mode. In the spin mode, it reads the four memory locations and determines which way to spin. It will continue to spin until A detects the path.

In an ideal world, since D is positioned in the halfway point between the wheels on the axis and the servos are told to rotate at the same speed, TBD should spin about sensor D. However, this is not a perfect world and this is why sensors E and F are in place. E and F serve only to correct TBD's position and attempt to realign it such that D is on the path again. This is done so by checking E and F whenever D does not detect the tape; if E senses the tape, TBD turns right (stop the right servo while making the left servo rotate in a forward direction); if F senses the tape, TBD turns left.

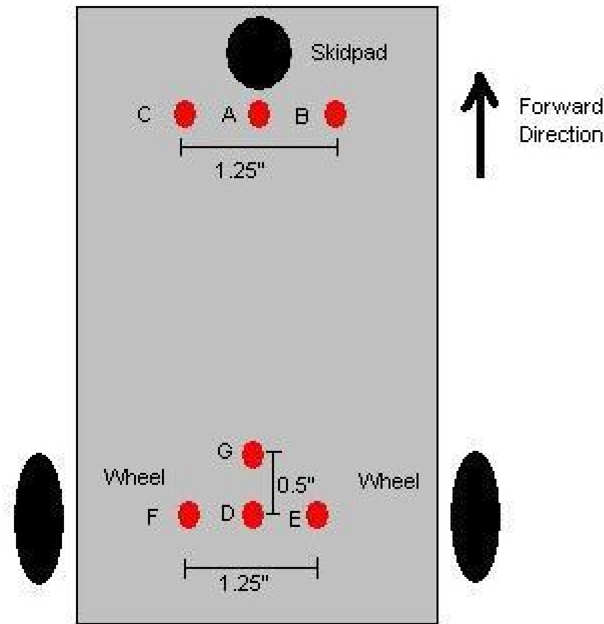


Figure 1: Infrared Phototransistor Layout

As mentioned before, TBD operates on the axiom that D needs to be kept over the path; this implies that no matter what it is doing, it always check D to make sure this is true. If TBD is in a spin and D somehow gets off track, it stops the spin, but keep checking to see if B and C ever detect that path and if so, write the corresponding values to the memory locations, and make the appropriate turn. Once D is back on track, it goes back into the spin mode, reading from the four memory locations. The values in these memory locations might have been changed depending on whether in the turning process B and C ever sense the path.

This algorithm is more easily understood with the aid of the finite state machine diagram found in Appendix A.

New Hardware

Power Supply

The power that is driving TBD is a battery pack containing eight AA batteries for a total of 12V and a sufficient amount of current to run our system. Since the components used in this project require 5V, a voltage regulator is used. The purpose of a voltage regulator is to take a voltage higher than the desired output voltage and output the desired voltage. In this case, the regulator takes in 12V and outputs 5V. The advantage of this process is that while voltage is kept to 5V, the available current is fairly high (as compared to a standard 9V battery). The chosen regulator for this project is the LM7805 (http://www.national.com/pf/LM/LM7805C.html).

HC908

The HC908 microcontroller has sixteen pins – one ground pin, one power pin, one IRQ pin, and thirteen input/output pins. The key features of this microcontroller relevant to the project are the fact that it can output two independent pulse-width modulated (PWM) signals and that it may take in seven inputs.

While the operation of the HC908 is similar to the HC11 microcontroller with which the team is familiar with, there are some subtle differences between their instruction sets. The team studied these differences carefully with the aid of the on-line manual on Motorola's website[•] and adjusted accordingly.

RC Servo

Two Futaba S-148 RC servos (obtainable from such hobby stores as Pegasus and Tower Hobbies, http://www.towerhobbies.com) are selected to act as the motors that mobilize TBD. Whereas a conventional motor takes in only power and ground to operate, the servos need an additional PWM input signal. The signal dictates the direction and speed of rotation of the servo by adjusting the pulse width. For example, a PWM signal whose period is 20 ms and pulse width is 1.5 ms makes the servo hold its

[•] Data sheet: http://e-www.motorola.com/brdata/PDFDB/docs/MC68HC908KX8.pdf
and instruction set: http://e-www.motorola.com/brdata/PDFDB/docs/MC68HC908QY4SM.pdf

position, a pulse width of 2 ms makes it rotate clockwise, and a pulse width of 1 ms makes it rotate counter-clockwise.

Conventional RC servos such as the ones used are limited in their range of motion, typically 0 ~ 210 degrees depending on the manufacturer. In order for these servos to act as motors, they must be capable of continuous rotation; this is accomplished by "hacking" the servos. The instruction for doing this is well documented and can be found on such websites as http://www.rdrop.com/~marvin/explore/servhack.htm. Essentially, this process removes the mechanism that halts the servos at the designated degree of rotation.

When selecting servos to use, one must be careful to select ones which are capable of being dissembled without ruining the servos. Some servos, especially ones involving ball bearings, have a plastic covering over some of the internal screws, therefore preventing one from taking out the electrical parts. The Futaba S-148 is an ideal servo to hack and is the one used in the presented website. Some pre-hacked servos may also be purchased from Parallax (http://www.parallax.com).

Phototransistor Sensors

The phototransistor sensor chosen for this project is the QRB1114 available from Digi-Key. It has four pins and how they are to be hooked up are diagramed in Figure 3. The phototransistors implement an infrared LED that shines down onto an object and a phototransistor configured to accept incoming light from an object. According to the amount of light reflected back to phototransistor, it outputs a signal that varies from zero to five volts.

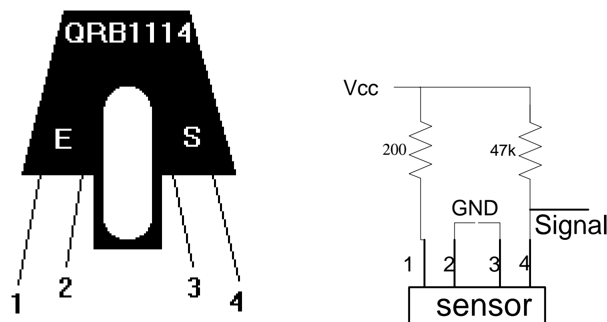


Figure 2: Phototransistor Sensor and setup

Both the LED and the phototransistor are angled inward so that the light reflects of the object directly towards the phototransistor when the object is placed about an inch away.

Comparators

The purpose of comparators is to compare an input voltage with a set reference voltage and depending on which is greater, output either 5V or 0.1V. This chip is ideal for use with the phototransistors since their output varies anywhere from 0 to 5V. By selecting an appropriate reference voltage (which is created through implementing the voltage divider rule and using the available 5V power source and two resistors), all values output by the phototransistor lower than this voltage is translated to 0.1V before being sent to the HC908 and all values higher is translated to 5V. This ensures that the HC908 is getting clean high or low signals. In this project, the reference voltage is set at 3.2V and is obtained by using an 820? and a 470? resistor.

Schematics

Many of the components required additional circuitry in order to function properly.

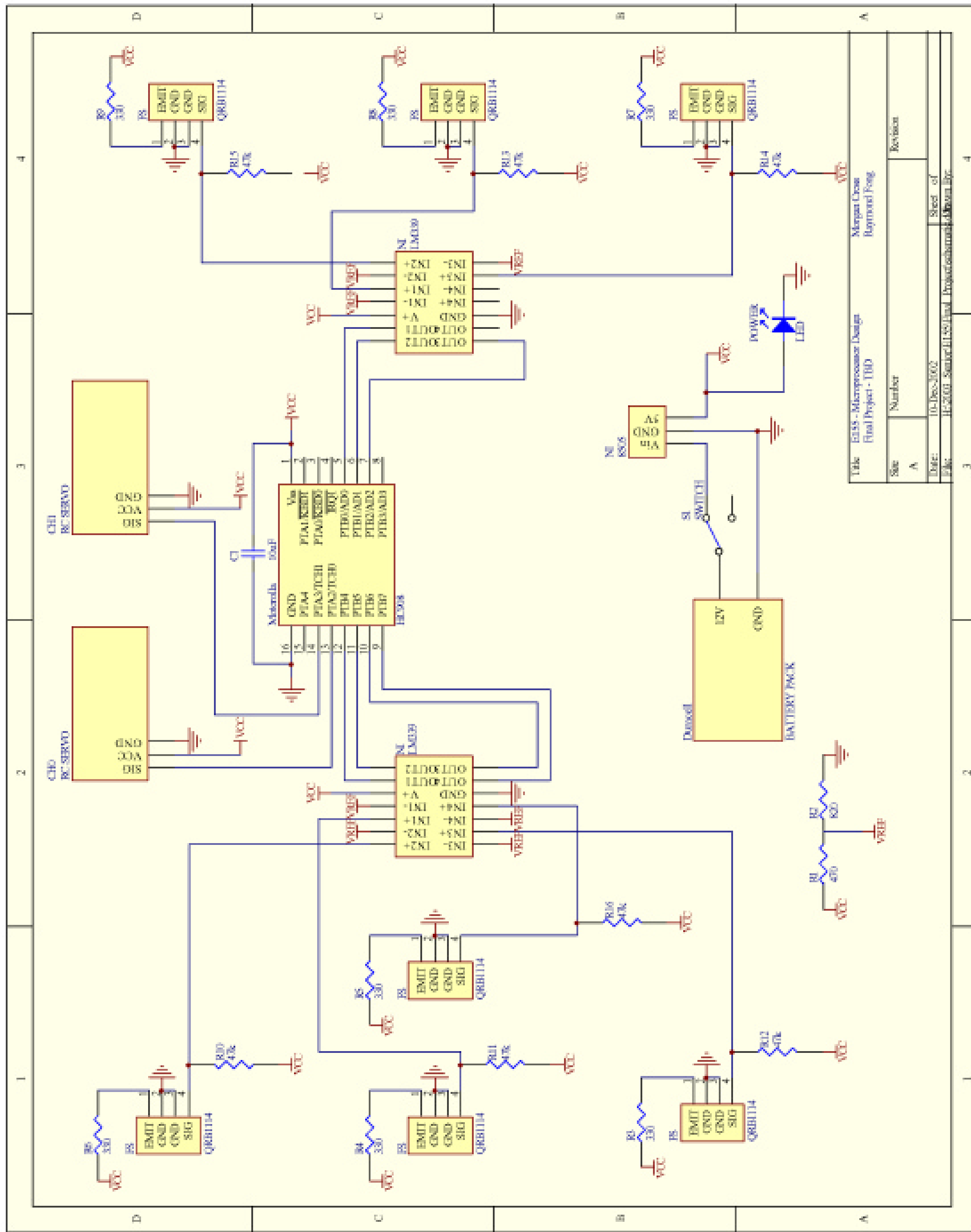


Figure 3: Schematic

Voltage levels produced from the IR photo sensors are passed to the comparators which translate that voltage into a stable logic HI or LOW voltage according the reference voltage. The output signal from the comparators then goes to the microcontroller running the algorithm. According to the algorithm, the microcontroller sends out proper pulse width modulated signals to the RC servos.

The IR photo sensors require a resistor in series with the emitter diode in order to limit the incoming current. A pull-up resistor is also needed on the signal line in order to provide the proper output level referenced to Vcc. The comparator chips require a reference voltage in order to determine a HI or LOW logic level. This is fabricated with the implementation of a voltage divider to drop the operating voltage of five volts down to the required 3.2 volts. The phototransistors produce outputs with voltage levels between 2.25V and 5V. Choosing the reference voltage was through trial and error because the output voltages varied between phototransistors. A ten-microfarad capacitor is placed between the Vcc and ground pins in order to prevent any drops in voltage when the chip demanded more current, ensuring clean PWM signals. A switch is implemented as a convenience to the users. A power LED is used to determine whether the power is on or not to reduce the risk of mistaking the power to be off when attempting to debug.

Microcontroller Design

The microcontroller code is divided into five main sections: the declaration of the constants, initializing the PWM channels, the main program, and the subroutines all resemble the setup for the HC11 code, and the section that instructs the chip what to do upon power up is new to the HC908.

In setting up the chip upon power up, the program needs to know where to start reading the memory location. "Org ROMStart " declares that all code from that point on will be placed at the memory location declared by ROMStart. “Main_Init:” is a label so that the program knows where to start reading upon power up. The RSP (reset stack pointer) command that follows will be at memory location ROMStart.

"Dummy_ISR" is a routine that does nothing. It is simply there so that on any interrupt, the CPU will go to this dummy routine instead of some undefined location in memory. The location that the PC goes to after an interrupt is defined by the vector table (which is a set of registers in the 0xFFF0 area of memory, starting at the location VectorStart). Each of these vectors is given a particular piece of memory location to jump to. For example, if the IRQ pin goes low, the interrupt vector Dummy_ISR is jumped to. Since it jumps to Dummy_ISR, effectively nothing happens on an active IRQ line.

The "dw" lines after the "ORG VectorStart" are what actually fill up the vector registers. These commands place the memory location of “Dummy_ISR” into each of the registers starting at memory location VectorStart and then Main_Init in the 18th register.

After configuring the chip according to the codes discussed above, the chip initializes the PWM channels 0 and 1. First it stops the timer counter, resets it, and selects the appropriate prescalar by writing to the Timer Status and Control (TSC). The prescalar is used for slowing down the clock and helping to create a PWM of the desired period. With the internal clock operating at 2.5 MHz, a prescalar of ÷16 is appropriate to ultimately obtain a period close to 20 ms. Values are loaded into the Timer Counter Modulo Register High/Low (TMODH/L) to further declare the period of the PWM. Both Timer Channel 0 and 1 Status and Control Registers (TSC0/1) are configured so that the unbuffered PWM outputs are cleared on compare, the timer counter toggle on overflow is

enabled, and max 100% duty cycle is disabled. Finally, the STOP bit of TSC is cleared to start the timer, thus enabling PWM.

The main program contains the algorithm and represents a finite state machine (FSM). It sequentially scans the sensors following the logic demonstrated in Figure 1. CHECK is a variable that keeps track of what the servos are doing. Before putting new values in the Timer Channel 0/1 Registers (TCH 0/1 H/L), the program determines if this is redundant by comparing CHECK. If CHECK is the same value as the CHECK that corresponds to the new values that are to be stored in the TCH, the program avoids storing the new values into the TCH since this would result in no change. The purpose of this is to reduce the number of wasted cycles and ensure cleaner PWM signals. To demonstrate, imagine that TBD is on a straight path, if CHECK were not in place, TBD would constantly be writing the same signals to the TCH to move forward. Every time these values are written to TCH, a slight glitch occurs because the "new" PWM signals interrupt the "old" PWM signals, causing it to go high for the appropriate amount of time before settling into the appropriate PWM signals.

There are two subroutines, "ChkBC" and "Servo". ChkBC scans sensors B and C to see which one detects the path and store the corresponding values into the four dedicated memory locations. Servo is used to copy the values of these four memory locations into actual TCH registers and direct the servos to behave as dictated.

Implementation

The final design of our robot consisted of a chassis produced from sheet metal, two protoboards, two servos, an eight-battery power source, a voltage regulator, an HC908 sixteen pin package, a ten-microfarad capacitor, two voltage comparators, seven IR reflective photo sensors, and various resistors.

The chassis was built from sheet metal and was constructed using a bender. The chassis had holes to hold the large protoboard that held the sensors and the comparators. The bends were necessary to set the heights of the phototransistors at the proper distance from the ground. The chassis also cradled the servos and allowed them to be screwed in. A hole was poked at the front of the robot in order to use the skid pad available made from Lego's. Another hole was made at the front of the robot to allow the voltage regulator to be screwed down to the chassis to help dissipate heat.

As mentioned earlier, the large protoboard held the sensors and its necessary circuitry, and the voltage comparators along with the voltage dividers providing the reference voltage. This protoboard was screwed into the chassis to keep it in place on the underbelly of the robot. The smaller protoboard held the microcontroller, the voltage regulator, the power switch and LED. This protoboard was held in place on the top of the robot by poster mounts. The power and ground wires coming from the battery pack was connected to this board. The common ground came from the battery pack and the power wire from the batteries went to the voltage regulator. The output of the voltage regulator was then sent to long rails of both the protoboard. The power was observed to be "bald" in certain areas along the long rail of the large protoboard. This required extra wires to supply power to every spot on the board.

Implementation of the IR photo sensors was very difficult because they had to be placed a certain distance from the ground and all of them required being the same distance if the same reference voltage wanted to be used. A lot of debugging work was necessary to find the proper distance from both the ground and from each other.

Results

TBD requires a path composed of electric tape laid out against white paper to operate optimally; this is so that there is a significant change between the amount of light the sensors sense while positioned over the electric tape and the white paper. TBD demonstrates that it is very well capable of making turns up to, and even over, 180°. If placed on a path that does not loop upon itself, it is able to turn around when it reaches the end of the path and retrace the path. However, while TBD is able to stay on a path, it sometimes arbitrarily chooses to turn 180° and go the opposite way instead of continuing down a path even though the path does not end. This "feature" of TBD occurs due to two main reasons: the fact that the servos are not spinning at the same rate and the width of the electric tape.

TBD is incapable of moving exactly straight (perpendicular to the wheels' axis) because the servos do not spin at the same rate. The reason why the servos are not calibrated to the same speed is because the resolution in which to control the PWM signals within the code are restricted by the fact that TBD operates better at a slower speed. This difference in servo speed causes TBD to steer off track more often than it should, even when the path is obviously a straight line. This is especially problematic when it approaches a turn because just before it reaches the turn, it may find itself off track and start the spin prematurely, causing it sometimes to miss the turn completely.

The path that TBD is supposed to follow is composed of electric tape; however, the thickness of this path also plays a role in determining TBD's performance. The tape used is half an inch in width; this implies that the sensors detect the tape even though they are not necessarily positioned directly over the middle of the tape. Therefore, whenever TBD attempts to reposition itself relative to the tape (i.e. in the midst of a spin, TBD stops when sensor A picks up the path), it may start going forward even though it is positioned at an angle relative to the path, causing it to get off path again.

These two main issues with TBD are the main reasons why it sometimes behaves unexpectedly, making spins when it is not suppose to (albeit it still remains on the path). Attempts have been made to make the path thinner so that when TBD repositions after a spin or turn, it may better realign itself. However, because of the thinness of the tape, the

robot now has a more difficult time detecting the path and also, due to its inability to go straight forward, falls off the path even more often.

Difficulties

Some difficulties were encountered throughout the course of this project. The regulator caused one such difficulty. The regulator was originally affixed into a protoboard by its three pins. It behaves as expected, outputting five volts, but after a small amount of time, about thirty seconds, it gets heated up rather significantly. It was not detected until later that as it heats up, it behaves erratically, outputting voltage well below the expected five volts. This proved to be a problem that caused TBD to behave abnormally. The solution was to use the chassis as a heatsink and attach the regulator to it. Some thermal compound was used between these two mediums to aid the heat transferring process.

The IR reflective phototransistor sensors were very difficult to work with because of how much variance in output it would generate. The output signal was very dependent on the position of the sensor relative to the ground. Changes in this distance on the range of millimeters would cause the "no tape" signal voltage to shift either up or down around a volt. This was a large problem because if the reference voltage were too close, the output would be unstable causing the output of the comparators to bounce HI and LOW. This confuses the microcontroller and TBD would often end up in the wrong place. Debugging of the mechanical system mostly consisted of sensor placement. The sensors ended up having a radius of sight that caused the placement of the sensors relative to each other, and to the ground, another large factor. If the sensors were placed too close to each other, this would create the problem where certain signals that should not be on would be on causing the algorithm to act as though the tape was present directly under the sensor. One example of this is in the algorithm where after sensor A falls of the tape and is now checking sensor G until that falls off before it spins. The problem here is that if sensor G stays on long enough to allow sensor D to fall off as well, the algorithm will then cause the robot to turn instead of spin. The distances between the sensors were eventually spaced enough to not overlap their radii of sight.

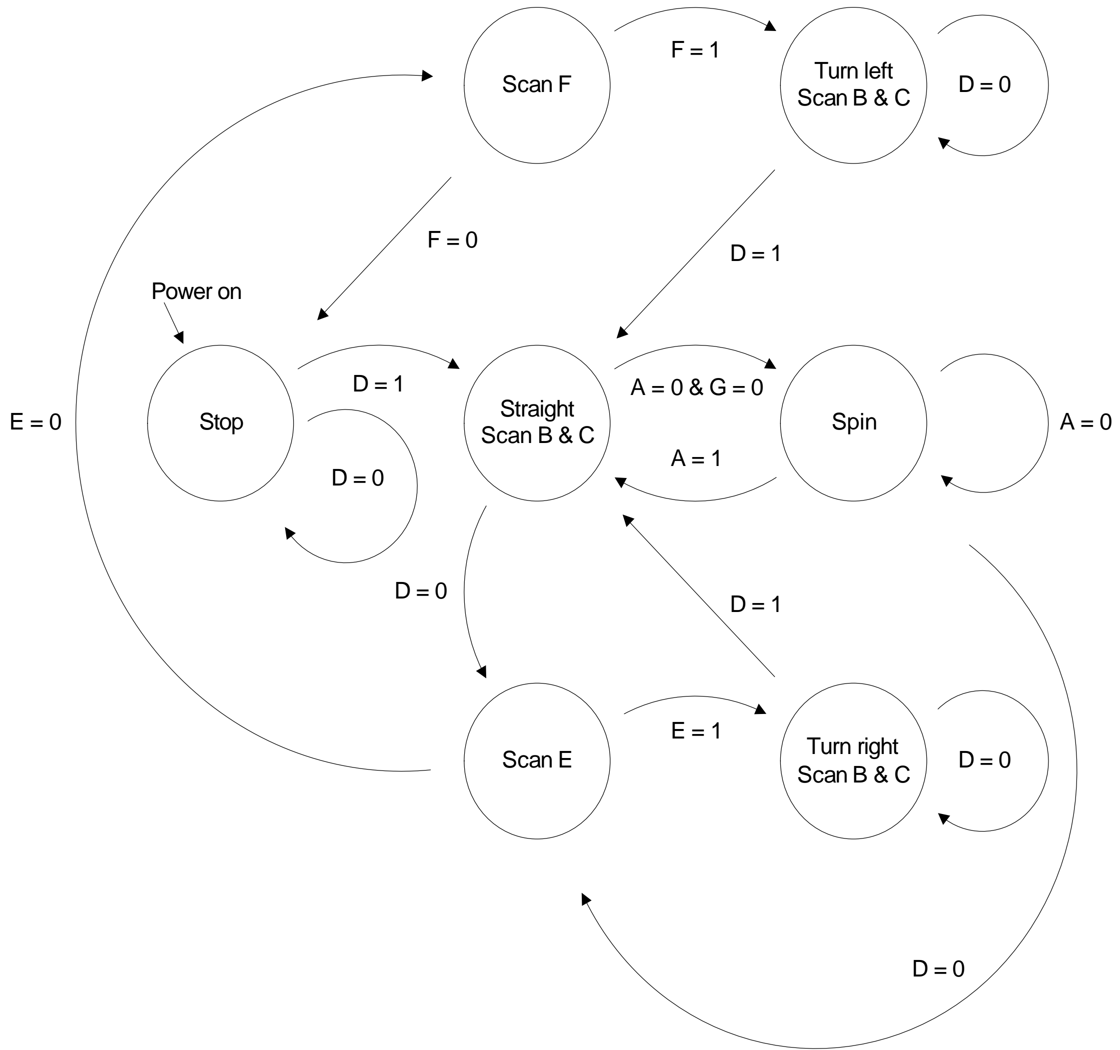
References

[1] Motorola, http://e-www.motorola.com/brdata/PDFDB/docs/MC68HC908KX8.pdf
and http://e-www.motorola.com/brdata/PDFDB/docs/MC68HC908QY4SM.pdf
[2] National Semiconductor Products, http://www.national.com/pf/LM/LM7805C.html
[3] Ross, Kevin, "Hacking a Servo,"
http://www.rdrop.com/~marvin/explore/servhack.htm

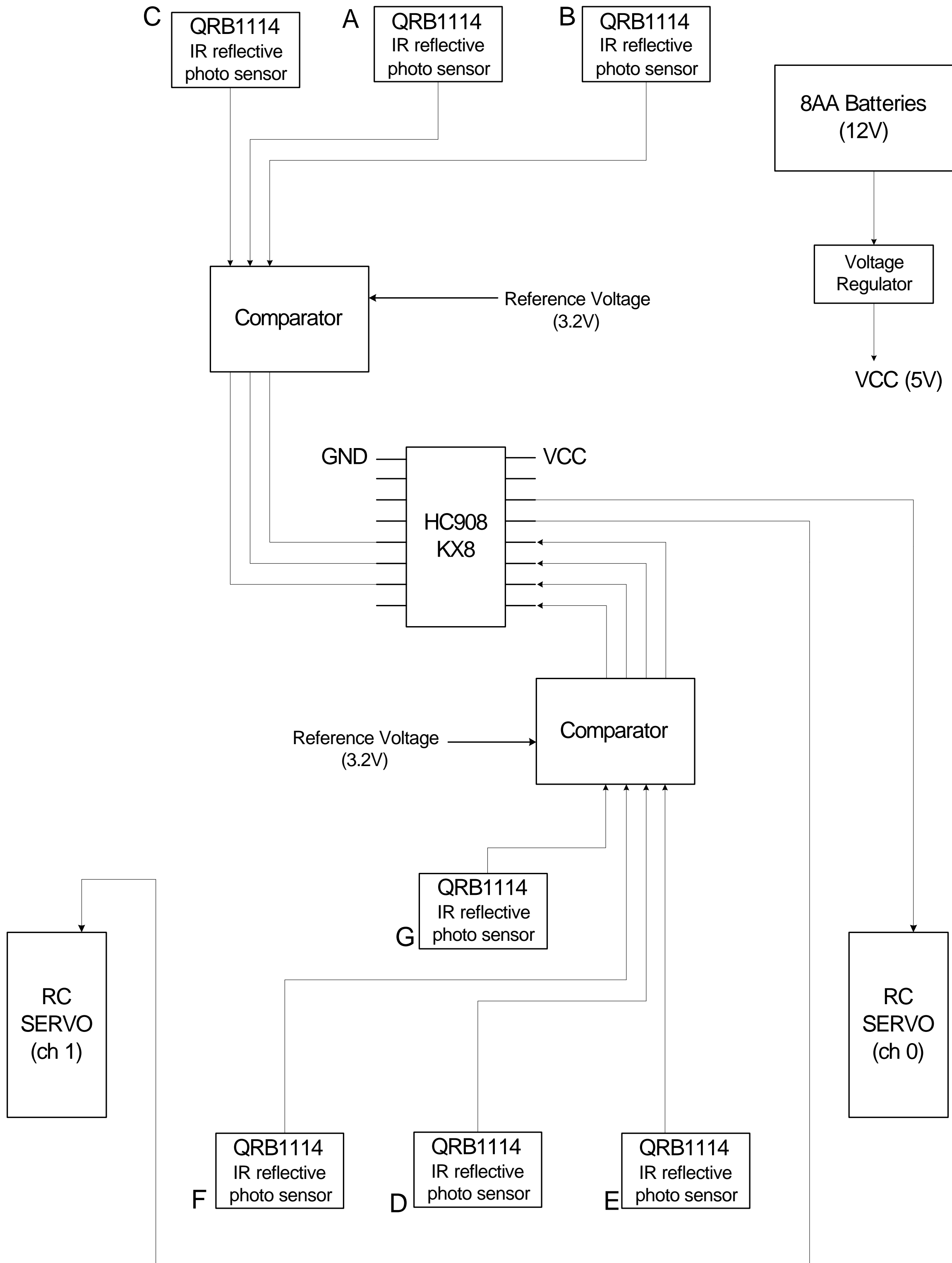
Parts

Part	Source	Vendor Part #	Price
MCHC908KX8 Microcontroller	VLSI Design Lab	-------------	-------------
Duracell Batteries (x8)	Vons	-------------	$10.06
Futaba FP-S148 RC Servo (x2)	Pegasus Hobbies	-------------	$31.15
LM339 Comparator (2x)	MarVac Electronics	LM339	$2.21
8505 Voltage Regulator	VLSI Design Lab	-------------	-------------
10microFarad Capacitor	Engineering Stockroom	-------------	-------------
IR Reflective Phototransistor Sensors (x7)	Digikey	QRB1114-ND	$12.51
Protoboard (x2)	Engineering Stockroom	-------------	-------------
Lego Skidpad	Professor Harris	-------------	-------------
Lego Wheels	Professor Harris	-------------	-------------
Sheet Metal	Tool Shop	-------------	$1.00
TOTAL			$56.93

Appendix A – Finite State Machine



Appendix B – Block Diagram



Appendix C – HC908 Code

```
* TBD.asm
*
*Written 12/12/02 Raymond Fong and Morgan Cross
*
*Line following robot that uses six photosensors, two RC Servos, and a
*HC908.

************
* Constants*
************
AON     EQU     $01
BON     EQU     $02
CON     EQU     $04
DON     EQU     $10
EON     EQU     $20
FON     EQU     $40
GON     EQU     $80
CCWH0   EQU     $00
CCWL0   EQU     $80
CWH0    EQU     $00
CWL0    EQU     $7A
CCWH1   EQU     $00
CCWL1   EQU     $81
CWH1    EQU     $00
CWL1    EQU     $7B
LOOKA   EQU     $0000
DDRA    EQU     $0004
DDRB    EQU     $0005
SENSA   EQU     $0001
SENSB   EQU     $0001
SENSC   EQU     $0001
SENSD   EQU     $0001
SENSE   EQU     $0001
SENSF   EQU     $0001
SENSG   EQU     $0001
TSC     EQU     $0020
TMODH   EQU     $0023
TMODL   EQU     $0024
TSC0    EQU     $0025
TSC1    EQU     $0028
TCH0H   EQU     $0026
TCH0L   EQU     $0027
TCH1H   EQU     $0029
TCH1L   EQU     $002A
XTCH0H  EQU     $00F2
XTCH0L  EQU     $00F3
XTCH1H  EQU     $00F0
XTCH1L  EQU     $00F1
STOPH0  EQU     $00
STOPL0  EQU     $7D
STOPH1  EQU     $00
STOPL1  EQU     $7E
COPCTL  EQU     $001F
CHECK   EQU     $00E8
```

```
RAMStart        EQU     $0040
RomStart        EQU     $E000
DataStart       EQU     $E300
VectorStart     EQU     $FFDC


*CHECK is so the program knows what the servos are doing
*$00 = Stop
*$01 = Straight
*$02 = Turn right
*$03 = Turn left
*$04 = Spin

**************
*Main Program*
**************
        ORG ROMStart

Main_Init:
        rsp

*Make port B pins to be inputs

        LDA     #$00
        STA     DDRB

        LDA     #01
        STA     DDRA

*Turn off COP so that the program does not reset when it thinks it is
*in an infinite loop

        LDA     #$01
        STA     COPCTL

*Start RC Servo, but do not make them spin (Make CHECK = $00)
*Initialize PWM by stopping TSC, and putting in the desired time scale;
*the desired period values are inputted into the
*TMODH/L and the desired high times are put into the PWM channels
*1 and 0 with TCH1/0H/L, the TSC0/1 declares the PWMs as
*unbuffered PWMs; the fifth bit of TSC is cleared to start the PWM

        MOV     #$00,CHECK

        MOV     #%00110100,TSC

        MOV     #$07,TMODH
        MOV     #$D4,TMODL

        MOV     #STOPH1,TCH1H
        MOV     #STOPH0,TCH0H

        MOV     #STOPL1,TCH1L
        MOV     #STOPL0,TCH0L

        MOV     #%00011010,TSC0
        MOV     #%00011010,TSC1
```

```
        BCLR    5,TSC

*Setup the values needed for spinning left, in case the robot
*goes on a straight line with no turns and hits the end... it will
*know to spin right 180

        MOV     #CCWH1,XTCH1H
        MOV     #CCWH0,XTCH0H

        MOV     #CCWL1,XTCH1L
        MOV     #CCWL0,XTCH0L

*Check CHECK and see if it is $00, if so, then there is no need to
*write the same STOP values into the PWMs

Stop    LDA     CHECK
        CMP     #$00
        BEQ     Loop

*Write $00 into CHECK

        MOV     #$00,CHECK

        MOV     #STOPH1,TCH1H
        MOV     #STOPH0,TCH0H

        MOV     #STOPL1,TCH1L
        MOV     #STOPL0,TCH0L

*Check sensor D and see if robot is on track
*If not, go to Offtk
*Check sensor B and C, if either is true, store the
*appropriate values in dummy variables XTCH1 and XTCH0

Loop    LDA     SENSD
        AND     #DON
        CMP     #DON
        BNE     Offtk
        JSR     ChkBC

*Check sensor A, if A senses the track, make robot go
*straight and restart Loop, if not, go to Spn

        LDA     SENSA
        AND     #AON
        CMP     #AON
        BNE     Spn

*****Make robot go straight*****

*Check CHECK and verify the need to write the straight values
*into the PWMs

Go      LDA     CHECK
        CMP     #$01
        BEQ     Loop
```

```
        MOV     #$01,CHECK

        MOV     #CCWH1,TCH1H
        MOV     #CCWL1,TCH1L

        MOV     #CWH0,TCH0H
        MOV     #CWL0,TCH0L

        BRA     Loop

*Check sensor G, if G is off the path
*Make robot spin until sensor A picks up tape, when that
*happens, make robot go straight and restart Loop
*this whole time, check D in case D is off the path,
*if D is off the path, go to Offtrk
*The whole time, it is still scanning B and C

Spn     JSR     ChkBC
        LDA     SENSG
        AND     #GON
        CMP     #GON
        BEQ     Spn
        JSR     Servo

ChkA    LDA     SENSD
        AND     #DON
        CMP     #DON
        BNE     Offtk
        LDA     SENSA
        AND     #AON
        CMP     #AON
        BEQ     Go
        BRA     ChkA

*If D is off the track, then
*Check E, if E senses the path, turn right, if E
*does not sense the path, check F

Offtk   LDA     SENSE
        AND     #EON
        CMP     #EON
        BNE     Right

*****Store the appropraite values needed to make the robot turn
*right*****

*Check CHECK and verify the need to write the turn left values
*into the PWMs

        LDA     CHECK
        CMP     #$03
        BEQ     Dchk

        MOV     #$03,CHECK

        MOV     #STOPH0,TCH0H
```

```
        MOV     #STOPL0,TCH0L

        MOV     #CCWH1,TCH1H
        MOV     #CCWL1,TCH1L

*Now check D, if D is on, go to Loop
*If D is not on the path yet, check B and C and store
*appropriate spinning values if need to

Dchk    LDA     SENSD
        AND     #DON
        CMP     #DON
        BEQ     Spn1
        JSR     ChkBC
        BRA     Dchk

*Check F, if F senses the path, start the turn, and then check D, B,
*and C if need to, if F does not sense the path, HALT!

Right   LDA     SENSF
        AND     #FON
        CMP     #FON
        BNE     Mkjmp

*****Store the appropraite values needed to make the robot turn
*left*****

*Check CHECK and verify the need to write the turn left values
*into the PWMs

        LDA     CHECK
        CMP     #$02
        BEQ     Dchk

        MOV     #$02,CHECK

        MOV     #STOPH1,TCH1H
        MOV     #STOPL1,TCH1L

        MOV     #CWH0,TCH0H
        MOV     #CWL0,TCH0L

        BRA     Dchk

Mkjmp   JMP     Stop

*Spn1 is used after TBD gets out of a turn, it is going to start
*spinning while checking D, if D is on the path, it will keep spinning
*until A is back on the path, if D is off the path, it will go to
*Offtrk

Spn1    JSR     Servo
ChkD    LDA     SENSD
        AND     #DON
        CMP     #DON
        BNE     Offtk
```

```
        LDA     SENSA
        AND     #AON
        CMP     #AON
        BEQ     Goo

        BRA     ChkD

Goo     JMP     Loop

************************************************************

*Subroutine for spinning
*If the values in the PWMs are the same as the appropriate
*memory location, there is no need to make any changes
*Make the CHECK know that it is spinning

Servo   MOV     #$04,CHECK

        LDA     TCH1L
        CMP     XTCH1L
        BEQ     Other

        MOV     XTCH1H,TCH1H
        MOV     XTCH1L,TCH1L

Other   LDA     TCH0L
        CMP     XTCH0L
        BEQ     Bah

        MOV     XTCH0H,TCH0H
        MOV     XTCH0L,TCH0L

Bah     RTS

*Subroutine for checking B and C

ChkBC   LDA     SENSB
        AND     #BON
        CMP     #BON
        BNE     Left

***Setup the values needed for spinning right***

        MOV     #CCWH1,XTCH1H
        MOV     #CCWH0,XTCH0H

        MOV     #CCWL1,XTCH1L
        MOV     #CCWL0,XTCH0L

Left    LDA     SENSC
        AND     #CON
        CMP     #CON
        BNE     End

***Setup the values needed for spinning left***

        MOV     #CWH1,XTCH1H
```

```
        MOV     #CWH0,XTCH0H

        MOV     #CWL1,XTCH1L
        MOV     #CWL0,XTCH0L

End     RTS

*Setup the vectors so the chip knows what to do at powerup

dummy_isr:
 rti

 org VectorStart

 dw dummy_isr    ; Time Base Vector
 dw dummy_isr    ; ADC Conversion Complete
 dw dummy_isr    ; Keyboard Vector
 dw dummy_isr    ; SCI Transmit Vector
 dw dummy_isr    ; SCI Receive Vector
 dw dummy_isr    ; SCI Error Vector
 dw dummy_isr    ; Reserved
 dw dummy_isr    ; Reserved
 dw dummy_isr    ; Reserved
 dw dummy_isr    ; Reserved
 dw dummy_isr    ; Reserved
 dw dummy_isr    ; TIM Overflow Vector
 dw dummy_isr    ; TIM Channel 1 Vector
 dw dummy_isr    ; TIM Channel 0 Vector
 dw dummy_isr    ; CMIREQ Vector
 dw dummy_isr    ; ~IRQ1 Vector
 dw dummy_isr    ; SWI Vector
 dw Main_Init    ; Reset Vector
```